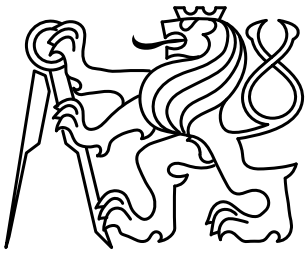
Czech Technical University in Prague
Faculty of Electrical Engineering

Habilitation Thesis

April 2019

Branislav Bošanský

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science



Equilibrium Computation in Dynamic Games

Habilitation Thesis

Branislav Bošanský

Prague, April 2019

Copyright

Dedicated to my daughter Zuzka, my wife Pavlína, and all my family.

Acknowledgments

I am grateful for having an opportunity to work with great researches on papers presented in this thesis. First of all, I thank my students, Jiří Čermák, Karel Horák, and Jakub Černý for all their hard work over the years (not only) on the papers presented in this work. Secondly, I thank Christopher Kiekintveld and Viliam Lisý for their collaboration on many research projects and extensive discussions and brainstorming sessions that pushed me always a bit further. I also thank prof. Michal Pěchouček for his endless support at the Department of Computer Science and as the head of Artificial Intelligence Center and for giving me this opportunity to work on dynamic games and game theory.

I am also grateful to all my coauthors on all the publications that were published over the years. Namely, I would like to thank Peter Bro Miltersen for the opportunity to spend great postdoc at Aarhus University and I also thank Simina Brânzei, Kristoffer Arnsfelt Hansen and Troels Bjerre Lund for their work on Stackelberg equilibrium. I also thank Milind Tambe, Krishnendu Chatterjee, Marc Lanctot, Albert Xin Jiang, Nicolla Gatti, Bo An, Qingyu Guo, Karel Durkota, and Tomáš Pevný for their collaboration on joint papers.

Finally, I want to thank all master and bachelor students that I supervised and all other colleagues at Artificial Intelligence Center for making this center a great place for research.

The work in this thesis was supported by Czech Science Foundation grants P202/12/2054 and 15-23235S, by the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16 019/0000765 "Research Center for Informatics", by the Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). Finally, many of the experimental results presented in the papers were obtained using CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures". During the postdoc, I was supported by the Danish National Research Foundation and the National Science Foundation of China (under the grant 61361136003) for the Sino-Danish Center for the Theory of Interactive Computation, and by the Center for Research in Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council.

Abstract

This habilitation thesis presents advancements in computing exact and approximate solution concepts in dynamic games. Dynamic games model scenarios that evolve over time, players are able to perform actions that modify the environment, however, the players do not have perfect information about the environment and receive only partial information as observations. We consider strictly competitive (or zero-sum) games where a gain of one player is a loss of the opponent as well as general-sum games. Similarly, we consider both games with a finite, pre-defined number of moves (horizon) after which the game terminates, as well as games where the number of moves is not fixed.

There are several key contributions. For zero-sum games, we provide algorithmic contributions for games with both finite and with infinite horizon. For finite games, we adopted the incremental strategy-generation technique in order to scale-up to larger domains and also provided the first algorithm for approximately solving games where players have imperfect memory (imperfect recall). For games with infinite horizon, we provide the first algorithms for approximately solving games where at least one player has partial information about the environment.

For general-sum games, we provide several theoretical results determining the complexity of computing a Stackelberg Equilibrium and novel algorithms for its computation in finite dynamic games. Moreover, we formally define a novel solution concept, a variant of Stackelberg Equilibrium termed Stackelberg Extensive-Form Correlated Equilibrium, and we show that this solution concept is important both from the theoretical perspective, since the computational complexity is often lower compared to Stackelberg Equilibrium, as well as from the practical perspective. To this end, we propose an algorithm that uses this new solution concept in order to quickly compute a Stackelberg Equilibrium.

Abstrakt

Tato habilitační práce shrnuje nové poznatky v oblasti algoritmické a výpočetní teorie her pro dynamické hry. Dynamickými hrami rozumíme situace, které se rozvíjejí v čase, hráči vykonávají akce, které modifikují prostředí a zároveň nemají hráči o prostředí plnou informaci a pozorují jej pouze částečně. V rámci práce uvažujeme jak striktně kompetitivní hry, ve kterých zisk jednoho hráče odpovídá ztrátě oponenta, tak i obecnější hry s nenulovým součtem. Rovněž, uvažujeme jak hry s konečným, pevně daným počtem tahů (tzv. horizontem), tak i hry, kde počet tahů může být nekonečný.

Habilitační práce má několik přínosů. Pro hry s nulovým součtem přináší nové algoritmické výsledky v hrách s konečným i nekonečným horizontem. Pro hry s konečným horizontem jsme adoptovali algoritmus inkrementálního přidávání strategií s cílem zlepšení škálovatelnosti a umožnění řešení větších her. Takisto popisujeme první praktické algoritmy pro řešení her, ve kterých mají hráči nedokonalou paměť. V hrách s nekonečným horizontem představujeme vůbec první algoritmus pro aproximativní řešení her, pokud alespoň jeden z hráčů má neúplnou informaci.

Pro hry s nenulovým součtem jsme provedli teoretickou analýzu několika problémů výpočetní složitosti výpočtu Stackelbergova ekvilibria a představili první algoritmy pro jeho výpočet v dynamických hrách s konečným horizontem. Navíc definujeme novou variantu Stackelbergova ekvilibria, nazvanou Stackelberg Extensive-Form Correlated Equilibrium, a ukazujeme její jak teoretické výhody (složitost výpočtu této varianty je často nižší v porovnání s originálním konceptem) tak praktické výhody, které demonstrujeme novým algoritmem, který pro výpočet Stackelbergova ekvilibria využívá právě tuto korelovanou variantu.

Contents

1 Introduction **1**

2 Dynamic Games **5**
2.1 Formal Models of Dynamic Games 5
2.1.1 Extensive-Form Games 5
2.1.2 Partially Observable Stochastic Games 6
2.2 Solving a Dynamic Game 7
2.2.1 Solution Concepts 7
2.2.2 Complexity of Computing Solution Concepts 8

3 Algorithms for Solving Dynamic Games **9**
3.1 Computational and Algorithmic Results for Zero-Sum Games 9
3.1.1 Results for Extensive-Form Games 10
3.1.2 Results for Partially Observable Stochastic Games 14
3.2 Computational and Algorithmic Results for General-Sum Games 16
3.2.1 Computing Stackelberg Equilibrium in EFGs 16
3.2.2 Using Correlation in Computing Stackelberg Equilibrium 17
3.2.3 Using Incremental Strategy Generation for Stackelberg Equilibrium Computation 18

4 Conclusions and Future Work **21**

Appendix A An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information **29**

Appendix B Algorithms for computing strategies in two-player simultaneous move games **69**

Appendix C Approximating maxmin strategies in imperfect recall games using A-loss recall property **111**

Appendix D Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games **149**

Appendix E Solving Partially Observable Stochastic Games with Public Observations 157

Appendix F Sequence-Form Algorithm for Computing Stackelberg Equilibria in Extensive-Form Games 167

Appendix G Computation of Stackelberg Equilibria of Finite Sequential Games 175

Appendix H Using Correlated Strategies for Computing Stackelberg Equilibria in Extensive-Form Games 201

Appendix I Incremental Strategy Generation for Stackelberg Equilibria in Extensive-Form Games 211

Chapter 1

Introduction

> "Nothing in life is to be feared, it is only to be understood.
> Now is the time to understand more, so that we may fear less."
>
> – *Marie Curie*

Recent years have seen a massive deployment of algorithms and techniques of artificial intelligence (AI) into every-day life. AI, mostly models based on machine or reinforcement learning, empower autonomous cars, they translate text or speech to different languages, and they are more and more becoming an inherent part of our lives. However, our dependence on AI models can turn into a significant risk if such models are deceived or directly attacked by an adversary. While there is a significant effort devoted to make such models robust, all of the typically used improvements are still vulnerable to attacks [Athalye et al., 2018]. To guarantee robustness and reliability of AI models, explicit reasoning about the adversary and their plans has to be used. To this end, *game theory*, that formally defines optimal behavior under a presence of an adversary, can be applied.

Game-theoretic strategies have already found their place in many real-world applications where competitive situations between interacting parties (or *agents*) naturally arise. The examples mostly include security, where a defense agency needs to allocate scares resources to protect valuable targets (e.g., airport terminals, ports, or wildlife animals, computer network) against an attacker (a thief, a terrorist, poachers) [Tambe, 2011, Yin et al., 2012, Fang et al., 2015]. Another recently emerging applications relate to the problem of machine/reinforcement learning in a presence of an adversary. Consider the problem of securing a computer network with a classification system that is used to identify anomalies or suspicious behavior. The classifier adapts from the previous and/or current data that can be poisoned by a strategic attacker [Durkota et al., 2017]. Therefore, in order to provide true robustness even against unseen samples, the game-theoretic reasoning and game-theoretic algorithms must be used.

Many of the mentioned real-world scenarios are *dynamic* in nature – agents act in an environment and they are able to (imperfectly) observe the changes in the environment that are caused by the actions of other agents. In network security, for example, the defending agent is able to react to intelligence and current situation (e.g., increased risk level, suspicious activity on a computer network), while the attacking agent can observe the current

allocation of the resources and adapt its plan accordingly. In some scenarios, we can specify that there is a certain number of actions after which the interaction terminates. However, in many real-world cases, there is *no fixed horizon* of the dynamic interaction. For example, the attacker can weigh the trade-off between the length of reconnaissance and estimate how much information they can learn, and between the costs for gathering the intelligence (e.g., in [An et al., 2012]) and such a trade-off is typically not restricted with a fixed deadline. For Advanced Persistent Threats (e.g.,in [Rass et al., 2017]), the attack can take up to several months and the attacker can patiently and strategically wait for the best moment to execute their attack and then cover their steps. Solving such dynamic games is challenging in general due to players' uncertainty and exponentially many possibilities that can arise during the game.

There are two main questions that drive algorithmic and computational research for dynamic games:

1. *How difficult it is to compute (approximate) optimal strategies for different classes of dynamic games?*
2. *Do there exist scalable, practical algorithms that allow us to compute (approximate) optimal strategies in dynamic games?*

To describe the results answering these questions, basic concepts of game theory must be described. First of all, we must specify what does it mean to solve the game or to find optimal strategies. Optimal strategies are defined by *solution concepts* (equilibria) and there are various solution concepts that are used in practical applications. First, there are the *maxmin* strategies that guarantee a player the best expected outcome in the worst case. Maxmin strategies are particularly useful as robust strategies for protecting critical infrastructures and they turn the game into a strictly competitive one (called a *zero-sum game*). If the game is zero-sum, maxmin strategies coincide with well-known Nash Equilibrium [Nash, 1950]. Second solution concept, that is often used, is *Stackelberg Equilibrium* [von Stackelberg, 1934]. In this solution concept, one player (typically the defender) commits to a strategy, while the other player (the attacker) plays the best response to this commitment. Stackelberg Equilibrium is widely used in asymmetric scenarios where the defender (or a policy maker, a market leader) has the power to commit to a strategy and announce this strategy so that the other player(s) can react to this commitment.

Secondly, we must specify what kind of dynamic games we consider. As mentioned above, there are two main subclasses of dynamic games – games with a fixed, finite number of possible moves (termed *Extensive-Form Games*) and games with an infinite or indefinite number of possible moves (termed *Partially Observable Stochastic Games*). We consider both strictly competitive (or *zero-sum* games where a gain of one player is a loss of another player) as well as more general *general-sum games*.

Main Contributions:

This habilitation thesis summarizes extensive research advancements made in algorithmic and computational game theory for dynamic games. For **zero-sum dynamic games**, the contributions are made for both games with finite as well as infinite horizon:

1. Extensive-Form Games
   (a) Fundamentally new algorithm for computing Nash strategies in general extensive-form games as well in games with perfect information and simultaneous moves based on incremental strategy-generation technique.
   (b) Novel computational complexity results and first algorithm for computing approximate maxmin strategies in extensive-form games with imperfect recall (players have imperfect memory).
2. Partially Observable Stochastic Games
   (a) First algorithm for computing approximate maxmin strategies in partially observable stochastic games with one-sided partial observability (one player has perfect information).
   (b) First algorithm for computing approximate maxmin strategies in partially observable stochastic games with public observations (both players have imperfect information).

For general-sum games, this thesis summarizes contributions for computing a Stackelberg Equilibrium in finite extensive-form games:

1. First algorithm for computing a Stackelberg Equilibrium in general extensive-form games.
2. A formal definition of a novel variant of Stackelberg Equilibrium termed Stackelberg Extensive-Form Correlated Equilibrium.
   (a) Novel computational complexity result for computing Stackelberg Extensive-Form Correlated Equilibrium.
   (b) Novel algorithm for computing a Stackelberg Equilibrium using Stackelberg Extensive-Form Correlated Equilibrium.
3. First exact and heuristic algorithm for using the incremental strategy-generation technique for computing a Stackelberg Equilibrium in general extensive-form games.

Chapter 2

Dynamic Games

This chapter introduces basic concepts and definitions used in computational game theory[1]. We introduce two formal representations of games used for reasoning about dynamic games – *extensive-form games* that model games with a finite and known number of moves in the game (termed *horizon*) and *partially observable stochastic games* that do not have a fixed horizon.

2.1 Formal Models of Dynamic Games

The baseline formal model for reasoning about games are *normal-form games (NFG)*, also known as strategic or matrix games. Formally, a NFG G is defined as a tuple $G = (N, A, u)$, where N is a set of players, A_i is a set of pure strategies (or actions) for player $i \in N$, and $u_i : A \to \mathbb{R}$. In all of the discussed research results, only two-player games have been considered, hence $N = \{1, 2\}$. We say that the game is *zero-sum* if gains of one player are the losses of the other player (formally, $u_1(a) = -u_2(a)$ for any $a \in A$). If this assumption does not hold, the game is *non-zero-sum* (or *general-sum*).

NFGs are suited for reasoning about one-shot games that end immediately after playing one action. On the other hand, NFGs are impractical for studying dynamic games since they can be exponentially larger (or infinite) compared to models defined specifically for modeling dynamic strategic interaction.

2.1.1 Extensive-Form Games

Extensive-form games (EFGs) model games with a finite and predetermined horizon. An EFG can be visualized as a tree where each node of the tree corresponds to a state of the game where one player can make a decision – i.e., to choose from one of the applicable actions (edges in the tree outgoing from this node) that changes the state of the game and a new state (node) is reached. EFGs are general enough to model stochastic events – in specific nodes, termed *chance nodes* an action to be played is chosen according to a known probability

[1] Definitions of concepts from game theory are based on books [Shoham and Leyton-Brown, 2009, Maschler et al., 2013].

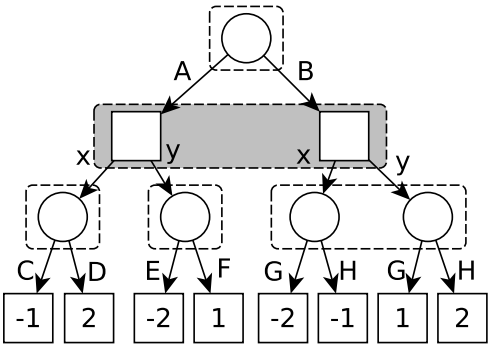


Figure 2.1: Example of a two-player zero-sum extensive-form game depicted as a game tree between player 1 (circle) and player 2 (box). Dashed boxes visualize the information sets. Utility values are for player 1, player 2 minimizes the value.

distribution. In EFGs, the players do not have to have perfect information about the state of the game – if a player cannot distinguish between a set of states, all of these states are grouped together in an *information set*. Finally, the outcomes (utility function) of the game are defined in leafs of the game tree. Example of a simple game is visualized in Figure 2.1.

The main advantage of using an EFG is that it offers an exponentially smaller representation of a game compared to the NFG representation. The main reason for this is that a pure strategy in an EFG corresponds to an assignment of an action that should be played for each information set. As a consequence, the number of pure strategies is exponential in the number of information sets in an EFG.

To reduce the size of strategies, players may forget certain information and thus create larger information sets (more states are considered to be indistinguishable). If all players in a game perfectly remember the history of their actions as well as all information gained during the course of the game, we say that the game has *perfect recall*. Otherwise, we say that the game has *imperfect recall*.

2.1.2 Partially Observable Stochastic Games

As discussed in the introduction, having a fixed known horizon for the game is not always satisfied in real-world scenarios. Therefore, *stochastic games* model dynamic interaction among players that can take an infinite number of moves/turns. Visually, one can imagine the game as an infinitely large EFG tree for which there exist infinitely long branches. This representation poses two main issues – *(i)* specification of the utility function for players cannot be defined for leafs (as there may not be any) and *(ii)* the size of a pure strategy can be infinite (there can be an infinite number of information sets).

Due to the first issue, the utility in stochastic games is defined over runs in the game (i.e., the sequence of actions played by both players). There are several options when defining the utility for players. The most common approach is that the game has specified immediate rewards over states and joint actions and the players optimize the (discounted) sum of rewards over the runs in the game. Alternatively, players may want to optimize average re-

ward, or one player might want to optimize run such that a certain subset of states is either reached or avoided (so-called reachability/safety objective). The papers presented in this thesis assume that players optimize the discounted sum of rewards.

Due to the second issue, solving a stochastic game often requires finding a compact finite representation of strategies. Consequently, the existing focus of research has been given on computing strategies from a restricted class of strategies (e.g., memoryless or stationary strategies) that are not guaranteed to reach (near) optimum rewards. The second approach is to restrict to a subclass of POSGs where one can show that there exist a compact representation of strategies that is sufficient for computing (approximately) optimal solutions.

2.2 Solving a Dynamic Game

There are two notions for solving a game. Either the game can be solved *quantitatively* where, given a game, the algorithm computes an (ϵ-approximation of) expected value of an equilibrium, or *strategically*, where the algorithm computes an (ϵ-approximation of) equilibrium strategies. For games with a finite horizon, both notions coincide, however, for stochastic games, some algorithms work in a quantitative manner and solving the game strategically requires additional computation.

2.2.1 Solution Concepts

The best known solution concept is the *Nash Equilibrium (NE)*. A pair of strategies is in a NE if neither of the players can gain by unilaterally deviating to a different strategy. Alternatively, one can say that in a NE, both players are playing a best response to the strategy of the opponent. Note that NE is descriptive equilibrium since it only describes which pairs of strategies are stable but NE does not give an answer for players which strategy to adopt.

In *Stackelberg Equilibrium (SE)*, the roles of the players are asymmetric. One player, called *the leader*, commits to a (possibly randomized) strategy (from now on we will assume that player 1 is always the leader). The opponent, called *the follower*, observes this commitment and plays a best response that maximizes her utility. In case there are multiple best responses of the follower, she split ties using a pre-determined rule. The follower can either split the ties in favor of the leader (so-called Strong SE, or the optimistic case) or against the leader (so-called Weak SE, or pessimistic case). In the literature, it is most common to assume the first approach that we have also adopted in our works and whenever we talk about Stackelberg Equilibrium we mean it in the strong sense. There are only a few existing works that focus on computing weak SE [Coniglio et al., 2017] or finding its approximation since it has been shown the WSE does not have to exist [von Stengel and Zamir, 2010]. Contrary to NE, SE the prescribes what strategy the leader should adopt.

Finally, in the presented works we have exploited a connection between SE and a different solution concept called *Correlated equilibrium* (CE). A probability distribution over outcomes in a game is in CE, if an external device (a trusted mediator) samples from this distribution and recommends the players which action to play; following this recommendation is a best response for the players knowing the initial probability distribution. As demon-

strated by Conitzer and Korzhyk [Conitzer and Korzhyk, 2011], algorithms for computing CE can be easily adapted for computing SE in one-shot games and we have established similar connections also for the dynamic games.

In the works discussed by this thesis, the goal was to either compute a NE or a maxmin strategy in zero-sum games or to compute a variant of a SE in non-zero-sum games. For many zero-sum games, many of the solution concepts coincide. Due to Von Neumann's minmax theorem, we know that quantitatively *maxmin*, *minmax*, NE, and SE share all the same value (the expected utility of player 1 in an equilibrium) that is called *the value of the game*. Therefore, we often refer to computing the value of the game in zero-sum games as *solving the game* and *computing optimal strategy* without specifying which solution concept is computed. This is no longer true for *general-sum games* where the equilibrium strategies differ for different solution concepts. Since NE is a descriptive solution concept, for general-sum games we aim to compute a (variant of) SE that specifically *prescribes* the strategy the leader should commit to playing.

2.2.2 Complexity of Computing Solution Concepts

The difficulty of the problem of computing equilibria depends on the class of the game and the solution concept. For zero-sum games, solving a game is a polynomial problem for both one-shot as well as extensive-form games with finite horizon. Solving stochastic games is tractable only in the perfect-information case (e.g., solving simple stochastic games is in PLS [Yannakakis, 1990, Etessami and Yannakakis, 2007]), however, many even single-player problems with imperfect information and infinite horizon are undecidable [Madani et al., 1999].

Moreover, there are additional complications that make computing optimal strategies in POSGs highly intractable even in the two-player zero-sum setting. Since the players do not perfectly observe the environment, each player has a belief over possible states of the environment. However, the reward the player receives for choosing some action(s) also depends on the action of the other player who decides based on their belief. Therefore, player 1 has to consider also the belief of player 2 and belief that player 2 has about player 1, and so on. This reasoning is called *nested beliefs* (e.g., in [MacDermed, 2013]) and it causes a doubly-exponential number of histories to consider for each agent. Therefore, we focus on approximate algorithms that solve the game in the sense of weak approximation (i.e., computed expected utility value is ϵ close to the optimal expected values) and our goal is to consider subclasses of POSGs where strategies with finite memory are sufficient to approximate value of the game.

For general-sum games, the complexity classes are more diverse. Computing a NE is PPAD-complete in both finite representations – i.e., in NFGs [Chen et al., 2006] as well as in EFGs [Daskalakis et al., 2006]. The computational complexity for computing a SE depends on the representation – in NFGs, computing a SE is polynomial [Conitzer and Sandholm, 2006], however, computing a SE in EFGs is typically NP-hard [Letchford and Conitzer, 2010].

Chapter 3

Algorithms for Solving Dynamic Games

This chapter summarizes the novel results presented in referenced papers. First, we focus on the results for strictly competitive (zero-sum) games, following by results for general-sum games.

3.1 Computational and Algorithmic Results for Zero-Sum Games

Our results are discussed separately for different classes of games. Most of the zero-sum games can be solved in polynomial time using linear programming. For normal-form games, a simple linear program can be constructed where variables correspond to a mixed strategy of one player and the constraints correspond to a best-responding opponent. For EFGs, it is also possible to construct a single linear program that computes the value of the game and equilibrium strategy due to von Stengel and Koller [von Stengel, 1996, Koller et al., 1996]. The linear program for EFGs, termed *sequence-form linear program*, has a linear number of variables and constraints in the size of the game tree and exploits different representation of strategies in EFGs known as *realization plans*. In this representation, the strategy is represented as a probability that certain sequence of actions of player i will be executed conditioned the opponent allows these actions to be executed (technically, the opponent chooses such actions that lead to information sets where the actions of a sequence can be applied). Due to this representation, this linear program is applicable only for EFGs with perfect recall.

The main challenge when solving EFGs is to tackle the size of the game tree. The game tree grows exponentially with the horizon (or a number of the moves in the game) – even a simple game where each player chooses from two actions, has more than 10^6 states after 10 moves of each player.

When moving to games with infinite horizon, there is the problem with nested beliefs that prevents one from designing an (approximate) optimal algorithm for fully general settings, as we discussed in Section 2.2.2. Nested beliefs can be tackled directly with histories – one of the few such approaches is a bottom-up dynamic programming for construct-

ing relevant finite-horizon policy trees for individual players while pruning-out dominated strategies [Hansen et al., 2004, Kumar and Zilberstein, 2009]. However, due to the explicit dependence on history, the scalability in the horizon is very limited leading to similar problems as in EFGs.

A more common approach is to focus on a subclass of POSGs where approximate optimal strategies do not need to depend on history. In [Ghosh et al., 2004], zero-sum POSGs with public actions and observations are considered. The authors show that the game has a value and present an algorithm that exploits the transformation of such a model into a game with complete information. Another significant subclass of POSGs are One-Sided POSGs where one player has perfect information [Chatterjee and Doyen, 2014, Basu and Stettner, 2015]. This subclass is particularly important for security applications since it provides naturally robust strategies against the worst-case fully-informed opponent. It is this subclass of POSGs that we investigate and for which we have designed the first approximate algorithm that can solve non-trivial games. Moreover, in a follow-up work, we have shown that this algorithm and approach can be generalized even to settings where both players have some partial information, but at the same time, they are able to infer the belief of the opponent. This is ensured by assuming that all observations that are received by players are public.

3.1.1 Results for Extensive-Form Games

For EFGs, the main challenge is to address the exponentially large input of the game – both the size of a strategy and the size of the game tree is exponential in the number of moves in the game. Dealing with this exponential size can be done in multiple ways. We have investigated two possible directions for improving scalability and thus allowing solving larger games. First, we describe how it is possible to find an exact solution without necessarily constructing the complete game. Second, we describe a way for solving EFGs where we allow players to forget the history of their actions.

In the first case, we adopted the incremental strategy generation methodology, known as the *double-oracle algorithm*, originally introduced for normal-form games by McMahan et al. [McMahan and Gordon, 2003]. The idea of the double-oracle algorithm is as follows (see Figure 3.1). The algorithm forms a restricted variant of a game to be solved by restricting the number of possible actions the players can choose from. The algorithm then operates in iterations and in each iteration, the restricted game is solved using a standard algorithm (e.g., the linear program) and the algorithm computes optimal strategy in the restricted game. Now, for each player, the algorithm computes a best response to the strategy of the opponent from the restricted game. However, this best response is selected from the unrestricted set of all possible actions in the original game. If the expected value for playing this best response is better for a player compared to the expected value gained in the restricted game, the best response strategy is allowed in subsequent iterations and the restricted game is expanded with this strategy. Otherwise, if neither of the players wants to expand the restricted game with additional strategy, the double-oracle algorithm computed a solution of the original game without necessarily constructing the complete game and considering all possible pure strategies.

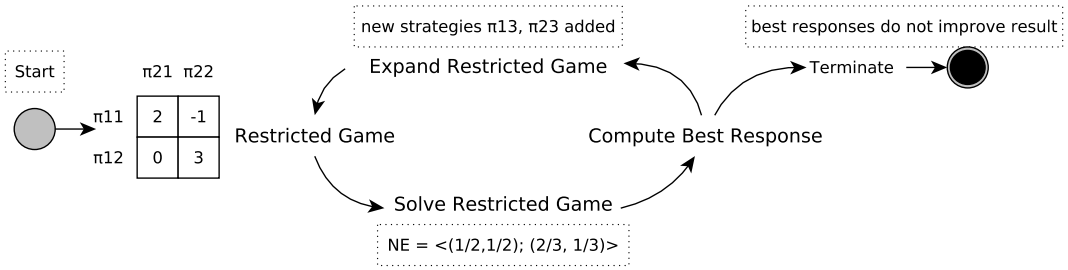


Figure 3.1: Schematic of the double-oracle algorithm for a normal-form game.

This method has been successfully used for solving large normal-form games, mostly in the security domain (e.g., in [Kiekintveld et al., 2009, Jain et al., 2011a, Jain et al., 2013]) due to an exponential number of strategies. The only previous attempt for using double oracle principle for EFGs has been using a transformation to normal-form pure strategies that are incrementally added [Zinkevich et al., 2007]. The main disadvantage of this approach is an exponential number of required iterations since there are exponentially many pure strategies in an EFG. In our work, we have demonstrated that the ideas of the double-oracle algorithm can be generalized to compact strategy spaces (i.e., realization plans in EFGs) and thus can be used to scale-up algorithms for many classes of dynamic games. We describe our main contributions in the next section, but we also use this idea in other algorithms as well.

As the second approach, we investigate a method for solving EFGs where players are allowed to forget the history of their own actions – technically, we focused on solving EFGs with imperfect recall. The main benefit of imperfect recall games is that the size of a strategy of a player can be exponentially smaller compared to the perfect recall case. An imperfect recall game can, for example, be a result of an abstraction algorithm applied on an EFG. The abstraction algorithm can identify that certain information is not required or necessary for finding (approximate) optimal strategies and thus merges two information sets into a single one, thus transforming the perfect recall game into an imperfect recall game with (exponentially) fewer information sets. However, there were no known practical algorithms for solving games with imperfect recall. Therefore, as our second major contribution for zero-sum EFGs, we describe first novel algorithms for solving games with imperfect recall.

Double-Oracle Methods for Solving Sequential Games with Finite Horizon

We have designed double-oracle algorithm for EFGs to operate directly on game trees. The restricted game is defined as a subset of sequences that players can play in the game and the algorithm uses the sequence-form linear program for solving the restricted game. The main technical contribution when translating double oracle for EFGs is to specify the restricted game and design the methodology for expanding the restricted game. In normal-form games, adding a new pure strategy into the restricted game is straightforward – for the algorithm, it is only necessary to calculate all utility values for combinations of the newly added pure strategies of player i and all already added pure strategies of the opponent $-i$.

In EFGs, allowing a player to play a new sequence of actions is not sufficient to formulate a well-defined restricted EFG since this particular sequence of actions does not have to be executable in the restricted game – the opponent does not play actions that allow this sequence of actions to be played.

We have made the following key technical contributions in order to define the double-oracle algorithm for EFGs:

- Formal definition of a valid restricted EFG defined as using a subset of allowed sequences of actions to be played.
- Algorithms for expanding and maintaining the validity of the restricted game.
- Domain-independent search for computing best-response sequences in EFGs.

We have formally proven that the algorithm converges to a Nash Equilibrium (Theorem 5.5 in [Bošanský et al., 2014]) and demonstrated that the double-oracle algorithm can find an exact NE adding only small fractions of all possible sequences. We have compared the double-oracle algorithm with exact and approximate existing algorithms on several games, including a search game, poker, and phantom variant of Tic-Tac-Toe, and showed that it is able to find the exact solution of much larger games, often using only as few as 1% sequences of the original game.

All discussed contributions describing double oracle for general EFGs are summarized in the following journal publication (see Appendix A).

B. Bošanský, C. Kiekintveld, V. Lisý, and M. Pěchouček. An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information. *Journal of Artificial Intelligence Research (JAIR), pp. 829–866, 2014.* **(65%)**

We have also investigated a possibility for using double-oracle approach for selected subclasses of EFGs. Specifically, we have examined EFGs with perfect information and simultaneous moves that generalize many simple turn-based scenarios – e.g., pursuit-evasion games and many card and board games. We have combined double-oracle algorithm with the alpha-beta pruning known for perfect information game and introduced an algorithm that scales orders of magnitude better compared to the existing state of the art.

All discussed contributions describing double oracle for EFGs for simultaneous games are summarized in the following journal publication (see Appendix B).

B. Bošanský, V. Lisý, M. Lanctot, J. Čermák, and M. M. H. Winands. Algorithms for Computing Strategies in Two-player Simultaneous Move Games. *Artificial Intelligence (AIJ), pp. 1–40, 2016.* **(30%)**

Solving EFGs with Imperfect Recall

Solving imperfect recall games is known to be a difficult problem (see, e.g., [Wichardt, 2008, Koller and Megiddo, 1992, Hansen et al., 2007]). We are interested in solving imperfect recall games created by an abstraction algorithm. Therefore, we focus on finding an

efficiently solvable subclass of imperfect recall games. Previous approaches create very specific abstracted games, so that perfect recall algorithms are still applicable: e.g., in *chance relaxed skew well-formed games* [Kroer and Sandholm, 2016, Lanctot et al., 2012] or in *normal-form games with sequential strategies* [Bošanský et al., 2015, Lisý et al., 2016]. The restrictions posed by these classes are unnecessarily strict, which can prevent us from fully exploiting the possibilities of abstractions and compact representation of dynamic games. We focus on a much larger subclass of imperfect recall games called *A-loss recall games* [Kaneko and Kline, 1995, Kline, 2002] where each loss of information of a player can be traced back to forgetting his own actions.

The contributions for solving imperfect recall games are both theoretical as well as practical. First, we present a complete picture of the problem of solving imperfect recall games and show which computational tasks become easier when restricting to A-loss recall. Second, we use the properties of the A-loss recall to provide the first family of algorithms capable of approximating the strategies with the best worst-case expected outcome (*maxmin strategies*[1]). Note that we require only one of the player (the opponent, or the minimizing player) to have A-loss recall. The player for which we compute the (approximate) optimal robust strategy is allowed to have a general imperfect recall.

Our theoretical results show that by restricting to A-loss recall opponent, the problem of computing maxmin strategies does not become significantly easier from the theoretical perspective. Specifically, we show that computing maxmin strategies is still NP-hard (Theorem 4 in [Čermák et al., 2018]), determining whether a NE exists is NP-hard (Theorem 5 in [Čermák et al., 2018]), and the optimal NE strategies may require irrational numbers even if all utility values are rational (Theorem 3 in [Čermák et al., 2018]). As an important positive result, we have identified necessary and sufficient (i.e., if and only if) condition for the existence of a Nash Equilibrium (NE) in behavioral strategies in A-loss recall games, making A-loss recall games the only subclass of imperfect recall games for which such condition is known (Theorem 1 in [Čermák et al., 2018]).

From the computational perspective, we exploit the fact that the best response of a player with A-loss recall can be computed in polynomial time [Kaneko and Kline, 1995, Kline, 2002]. We thus provide a novel approximate algorithm, denoted IRABNB (Imperfect Recall Abstraction Branch-and-Bound algorithm), for computing maxmin strategies in imperfect recall games where the maximizing player has imperfect recall and the minimizing player has A-loss recall. We base the algorithm on the sequence-form linear program for computing maxmin strategies in perfect recall games [von Stengel, 1996, Koller et al., 1996] extended by bilinear constraints necessary for the correct representation of strategies of the maximizing player in imperfect recall games. We approximate the bilinear terms using Multiparametric Disaggregation Technique (MDT) [Kolodziej et al., 2013] and provide a mixed-integer linear program (MILP) for approximating maxmin strategies. Next, we propose a novel branch-and-bound algorithm that repeatedly solves the linear relaxation of the MILP. The main novelty is that the algorithm simultaneously tightens the constraints

[1]We compute (approximate) maxmin strategies since Nash Equilibrium does not have to exist for this class of games and for the class of behavioral strategies that allow reducing size of strategies in imperfect recall EFGs.

that approximate bilinear terms and searches for the optimal assignment to the relaxed binary variables from the MILP. We prove that the algorithm has guaranteed convergence to maxmin strategy and we provide a bound on the number of steps needed.

Finally, we extend the IRABNB algorithm by incremental strategy generation technique. The resulting algorithm is denoted DOIRABNB (Double Oracle Imperfect Recall Abstraction Branch-and-Bound Algorithm). Compared to the double-oracle algorithm for perfect recall EFGs [Bošanský et al., 2014], there are several fundamental challenges that need to be addressed when the double-oracle algorithm is used for imperfect recall EFGs. First, the algorithm for solving the restricted game is a more complex (approximate) search algorithm based on a branch-and-bound scheme. Second, the problem is that for incremental computation of maxmin strategies in imperfect recall EFGs, adding best response sequences of actions is not sufficient for convergence (see Example 2 in [Čermák et al., 2018]). Therefore, we had to design more general rules to guarantee the convergence and add all possible actions that can improve the outcome for the maximizing player. The experimental evaluation shows that DOIRABNB is capable of solving some games with up to $5 \cdot 10^9$ states in approximately 1 hour. We also experimentally demonstrated the effectiveness of the use of imperfect recall abstractions to reduce the size of strategies to be stored. We show that employing simple abstractions which still allow us to compute the maxmin strategy of the original game can lead to strategies with the relative size as low as 0.03% of the size of the strategy in the original unabstracted game.

All discussed contributions for computing maxmin strategies including the double oracle extension are summarized in the following journal publication (see Appendix C).

Jiří Čermák, **Branislav Bošanský** Karel Horák, Viliam Lisý and Michal Pěchouček. Approximating maxmin strategies in imperfect recall games using A-loss recall property. *International Journal of Approximate Reasoning, pp. 290–326, 2018.* **(25%)**

3.1.2 Results for Partially Observable Stochastic Games

Our main contribution is the first domain-independent algorithm that has guarantees to approximate optimal strategies in one-sided POSGs. Our algorithm is a generalization of the heuristic search value iteration algorithm (HSVI) originally proposed for Partially Observable Markov Decision Processes (POMDPs) [Smith and Simmons, 2004, Smith and Simmons, 2012]. Similarly to POMDPs, One-Sided POSGs allow us to compactly represent strategies and value functions representing values of the game based on the belief the first player has about the state of the game. Contrary to POMDPs, the presence of the opponent player causes significant technical challenges that we resolve in our contribution. First, we show that the assumption of the one-sided partial observability guarantees that the value functions are convex. Second, we define a value backup operator and show that an iterative application of this operator converges to the optimal values. Third, we generalize the ideas behind HSVI towards one-sided POSGs and show that our algorithm approximates optimal strategies. Finally, we demonstrate the applicability and scalability of our algorithm on three different domains – patrolling games (including the variant with alarms),

pursuit-evasion games, and search games. The results show that our algorithm can closely approximate solutions of large games with more than 4000 states.

All discussed contributions describing One-Sided POSGs and HSVI algorithm for this class of stochastic games are summarized in the following A* publication (see Appendix D).

Karel Horák, **Branislav Bošanský** and Michal Pěchouček. Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games *In Proceedings of AAAI Conference on Artificial Intelligence.* pp. 558–564, 2017. **(45%)**

While One-Sided POSGs have great motivation for deployment to scenarios where the robust strategies are necessary, the computed strategies may be unnecessarily pessimistic and the algorithm cannot properly evaluate the value of disclosing some information to the opponent (since it assumes that the opponent has perfect information). Relaxing these assumptions into a fully general setting is not possible due to the problem of nested beliefs. Therefore, we relaxed the subclass of One-Sided POSGs to games where both players have partial information, but we assume that observations that affect the private beliefs of the players are *public* and thus each player is able to exactly reconstruct the belief of the opponent [Horák and Bošanský, 2019]. The key characteristics of our model, termed POSGs with public observations (PO-POSGs), are: (1) the state space is factored – each player observes his private state, but the state of the other player is not observed; (2) each observation that modifies belief about the state of the other player is public (both players are aware of this observation); (3) the true state of the player is observed privately by that player.

The contributions for the class of PO-POSGs: (1) We show that games in this class have a value; (2) We show that the value function of PO-POSGs is convex in the belief of the maximizing player and concave in the belief of the minimizing player; (3) We introduce a novel algorithm based on Heuristic Value Iteration Search (HSVI) for One-Sided POSGs [Horák et al., 2017a, Smith and Simmons, 2004] and show that this algorithm converges to (approximate) optimal values.

We demonstrate our algorithm on two different domains – a patrolling game, where the attacker has imprecise information about the position of the defender [Basilico et al., 2009], and a lasertag game based on a single-player variant [Pineau et al., 2003]. The results show that, for the first time, there is a practical domain-independent algorithm able to closely approximate optimal values of non-trivial POSGs with hundreds of states where both players have partial information about the environment.

All discussed contributions describing PO-POSGs and HSVI algorithm for this class of stochastic games are summarized in the following A* publication (see Appendix E).

Karel Horák and **Branislav Bošanský**. Solving Partially Observable Stochastic Games with Public Observations In *Proceedings of AAAI Conference on Artificial Intelligence.* 2019 **(40%)**

3.2 Computational and Algorithmic Results for General-Sum Games

In general-sum games, the algorithmic and computational work focuses mostly on computing the Stackelberg Equilibrium (SE) and its most common variant Strong Stackelberg Equilibrium where the follower break ties in favor of the leader. While there is a large volume of works focusing on computing SE in one-shot games, mostly in security domain (e.g., in [Tambe, 2011]), the algorithms for computing SE in EFGs were not developed prior to our work. Another significant difference compared to the zero-sum case, the complexity of computing SE differs in EFGs (even with perfect recall) compared to NFGs. As shown by Letchford and Conitzer [Letchford and Conitzer, 2010], computing SE is NP-hard for most of the variants of EFGs. This is in contrast to a positive result for computing SE in NFGs where the problem is polynomial [Conitzer and Sandholm, 2006].

Therefore, over several works, we have focused on algorithmic and computational aspects when computing SE in dynamic games. Most importantly, we have formalized the first algorithm for computing SE in EFGs by extending the sequence-form linear program in order to compute SE [Bošanský and Čermák, 2015]. The scalability of the first algorithm has been limited and thus several of our follow-up works focused on improving the scalability. There are two notable contributions that allowed us to push the scalability further. First, we have formally defined a new variant of SE for EFGs, termed Stackelberg Extensive-Form Correlated Equilibrium (SEFCE) [Bošanský et al., 2017], where the leader is allowed to commit to correlated strategies and send signals to the follower (following the signals must be the best response for the follower). In two papers, we showed that SEFCE has not only lower computational complexity for certain subclasses of EFGs [Bošanský et al., 2017] but also that SEFCE can be computationally used for computing standard SE [Čermák et al., 2016].

Second, we have also explored the possibility for incremental strategy generation for computing SE in EFGs. Compared to zero-sum EFGs, building a restricted game is more challenging since the abstracted parts of the game tree cannot be represented using a single value as it is done in the zero-sum case. We have overcome these challenges and proposed two variants of an algorithm based on incremental strategy generation that does not have to expand (and thus consider) the entire game tree to compute (approximate) SE.

3.2.1 Computing Stackelberg Equilibrium in EFGs

The sequence form mixed integer linear program (MILP) for computing SE in EFGs is a direct extension of the sequence form linear programs for solving zero-sum games. The main extensions are in the representation of strategies of the follower – binary variables are used in order to represent the best response of the follower. Next, the expected outcome is calculated based on the joint probability that a certain terminal state of the game is reached. Since the follower plays a pure strategy, the joint probability can be expressed with linear constraints. As a consequence, we formulate a MILP that has a linear size in the size of the game tree. Therefore, we have provided a constructive proof that computing SE in EFGs is

NP-complete (NP-hardness has been shown before [Letchford and Conitzer, 2010]) and the first algorithm specifically designed for computing SE in EFGs.

The scalability of our MILP algorithm was significantly better compared to algorithms based on NFG representation that has an exponential size in the size of the game tree. The algorithm was able to compute an exact SE within a few hours of games with 10^4 nodes or 10^5 if the number of pure strategies of the follower has been small.

All discussed contributions describing sequence-form MILP for computing SE in EFGs are summarized in the following A* publication (see Appendix F).

Branislav Bošanský and Jiří Čermák. Sequence-Form Algorithm for Computing Stackelberg Equilibria in Extensive-Form Games. *Proceedings of AAAI Conference on Artificial Intelligence, pp. 805–811, 2015.* **(70%)**

3.2.2 Using Correlation in Computing Stackelberg Equilibrium

As mentioned above, we have formally defined a novel variant of SE, termed Stackelberg Extensive-Form Correlated Equilibrium (SEFCE), where the leader is allowed to commit to correlated strategies and send signals to the follower (following the signals must be the best response for the follower). In EFGs, this concept is closely related to Extensive-Form Correlated Equilibrium [von Stengel and Forges, 2008]. The key characteristic is that the follower receives a particular signal which action to play after reaching certain information set, however, the follower does not know which actions are going to be recommended afterwards. The follower only knows the probability distribution from which the actions are sampled. This is in contrast to standard correlated equilibrium, where the complete strategy in the game is received by players.

We analyzed the computational complexity of this new solution concept. We showed that for certain subclasses of EFGs, computing a SEFCE is polynomial while computing SE is NP-hard (e.g., for finite EFGs with perfect information and simultaneous moves) [Bošanský et al., 2017]. Moreover, we have shown that the expected utility of the leader in SEFCE forms an upper bound on the expected utility of the leader in SE. This proposition is important since constructing a tight upper bound is important for the optimization problem like Stackelberg Equilibrium. From the optimization perspective, finding an SE corresponds to operating over a non-continuous piece-wise linear function with exponentially many parts. SEFCE, however, forms a tight convex hull over this function and thus can be allowed to guide the search when computing SE.

All discussed theoretical contributions regarding SEFCE are summarized in the following journal publication (see Appendix G).

Branislav Bošanský, Simina Brânzei, Kristoffer Arnsfelt Hansen, Troels Bjerre Lund, Peter Bro Miltersen. Computation of Stackelberg Equilibria of Finite Sequential Games. *ACM Transactions on Economics and Computation, Vol. 5, No. 4, Article 23, 2017.* **(35%)**

We exploited this fact and designed a novel algorithm that uses SEFCE for computing SE [Čermák et al., 2016]. The algorithm first computes a correlated variant of SE and then

examines whether the signals the follower receives are all pure. If this is indeed the case, the algorithm has, in fact, found a SE. Otherwise, the algorithm selects some information set where the follower can receive multiple signals, adds a constraint that makes this no longer possible and resolves the problem. There are several variants of our algorithm – there is a choice in which information set is selected and whether the constraints are compatible with linear programming or whether they use binary variables. Any of the variants, however, significantly outperforms the first algorithm for computing SE are to this day present the exact state-of-art domain-independent algorithm for computing SE in EFGs.

All discussed contributions describing sequence-form MILP for computing SE in EFGs are summarized in the following A* publication (see Appendix H).

Jiří Čermák, **Branislav Bošanský**, Karel Durkota, Viliam Lisý and Christopher Kiekintveld. Using Correlated Strategies for Computing Stackelberg Equilibria in Extensive-Form Games. In *Proceedings of AAAI Conference on Artificial Intelligence, pp. 439–445, 2016.* **(30%)**

3.2.3 Using Incremental Strategy Generation for Stackelberg Equilibrium Computation

Since our first algorithm based on sequence-form mathematical program, we aimed at possibility exploiting incremental strategy-generation technique for scaling-up the performance of our algorithms for computing SE in EFGs. While incremental strategy generation works well for computing SE in other classes of games (for example, for Bayesian games [Jain et al., 2011b]), translating these ideas to EFGs is not straightforward. Similarly, despite the fact that we have successfully designed a double-oracle algorithm for zero-sum EFGs, adapting it for computing SE in general-sum games is again not straightforward. The main complication is that in zero-sum EFGs, the not-expanded parts of the game tree (i.e., the branches of the game tree that use sequences that were not added to the restricted game yet) are in the restricted game represented using a single temporary leaf with a single temporary value. For SE, however, this is not sufficient since there is no such single value. The reason is that the leader can commit to a sequentially irrational strategy in a certain part of the game tree, just to force the follower to play differently (i.e., to deliberately use threats; since the commitment is not modifiable by the leader, the threats are credible).

We have solved this issue with a smaller temporary gadget-game that represent several utility points from the abstracted game and let the leader choose from these possible outcomes. The outcomes are chosen such that we may preserve guarantees that SE will eventually be found. However, in order to guarantee convergence, complete sub-games have to be expanded in such an algorithm. In imperfect-information EFGs, however, sub-games represent rather large parts of the game tree and thus the algorithm in the form with guarantees does not scale well. Removing this requirement loses the theoretical guarantees, however, allows us to scale to much larger game trees. We were able to find near-optimal solutions with error from true SE typically less than 4% while constructing a mathematical program with size less than 5% compared to the full programs. Moreover, we have found these near-optimal strategies an order of magnitude faster.

All discussed contributions describing sequence-form MILP for computing SE in EFGs are summarized in the following A* publication (see Appendix I).

Jakub Černý, **Branislav Bošanský** and Christopher Kiekintveld. Incremental Strategy Generation for Stackelberg Equilibria in Extensive-Form Games. In *Proceedings of ACM Conference on Economics and Computation, EC, pp. 151–168, 2018.* **(40%)**

Chapter 4

Conclusions and Future Work

This thesis summarizes results of research in equilibrium computation in dynamic games. The discussed result advance state of the art in algorithmic and computational game theory. We have provided new definitions of solution concepts, determined the computational complexity of several open problems, and most importantly designed, implemented, and experimentally evaluated a collection of novel algorithms for computing (approximate) solutions in dynamic games. Most of our algorithmic approaches are domain-independent and thus can act as baseline methods for many possible real-world applications. Moreover, all our algorithms can be further enhanced with various heuristics to further improve scalability and thus solve real-world instances.

There are several possible directions for possible future research in dynamic games. One challenge is to push forward algorithms for solving Partially Observable Stochastic Games (POSGs). There are three interesting directions for POSGs. First, the existing algorithms presented in this thesis are the first of their kind. Hence, the scalability is limited and further improvements in the algorithm can dramatically improve the practical applicability of these algorithms. Indeed, our preliminary results show that there is a room for further improvements and our algorithms for solving subclasses of POSGs can be used for computing approximate optimal strategies in real-world scenarios, especially in network security and autonomous defense mechanisms [Horák et al., 2017b, Horák et al., 2019].

The second direction is to further generalize the concept of POSGs with public observations (PO-POSGs) and identify the largest subclass of POSGs where using a finitely-bounded histories (or memory) is sufficient for approximately solving the game. We have shown that public observations allow players to derive belief of the opponent thus avoiding the problem of nested beliefs. However, there are two open questions: (1) Is it possible to generalize the subclass of PO-POSGs even further? Do all observations need to be publicly observable? (2) What is the quality of strategies computed for a One-Sided POSG or a PO-POSG variant of a general POSG?

Third, we have designed algorithms for zero-sum POSGs. Another possible direction is to generalize these algorithms to general-sum games and compute a Stackelberg Equilibrium (SE) instead of maxmin. The main challenge in this direction is the prerequisite of having a dynamic-programming operator for computing SE in a bottom-up fashion sim-

ilarly as it is done in HSVI-based algorithms for POSGs [Horák et al., 2017a, Horák and Bošanský, 2019]. Despite that this is again computationally more challenging, the preliminary results show that it is possible to design such a dynamic-programming algorithm and its approximate variant can be a valid basis for the first algorithm for computing SE in POSGs [Rindt, 2019].

Another completely different direction is to design algorithms for dynamic games for other classes of games. Among all, the biggest challenge would be to design solution concepts and practical algorithms for solving dynamic games with many players. For example, for Stackelberg Equilibrium, there already exist some algorithms with multiple followers [Basilico et al., 2016], however, they focus on normal-form games. Even more challenging would be to generalize concepts of dynamic games to succinctly represented games that model interactions with many (hundreds) players. A typical example is a congestion game where a set of agents is selecting a route through a network and their choices affect utilities of other agents (e.g., if all agents are using the same road, the reward for each agent will be low due to congestion at that road). Adding a dynamic aspect would be highly desirable since many of the modeled scenarios are in fact dynamic (e.g., people change their decision about which route to take). While there are again some results demonstrating that it is possible to design variants of succinct games with dynamic aspect [Hoefer et al., 2009], a fully analyzed model and tailored algorithms for dynamic succinct games are missing.

Bibliography

[An et al., 2012] An, B., Kempe, D., Kiekintveld, C., Shieh, E., Singh, S., Tambe, M., and Vorobeychik, Y. (2012). Security games with limited surveillance. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, pages 1241–1248.

[Athalye et al., 2018] Athalye, A., Carlini, N., and Wagner, D. (2018). Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 274–283, Stockholmsmässan, Stockholm Sweden. PMLR.

[Basilico et al., 2009] Basilico, N., Gatti, N., Rossi, T., Ceppi, S., and Amigoni, F. (2009). Extending Algorithms for Mobile Robot Patrolling in the Presence of Adversaries to More Realistic Settings. In *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*.

[Basilico et al., 2016] Basilico, N., Nittis, G. D., and G, N. (2016). A Security Game Combining Patrolling and Alarm–Triggered Responses Under Spatial and Detection Uncertainties. In *AAAI Conference on Artificial Intelligence*.

[Basu and Stettner, 2015] Basu, A. and Stettner, L. (2015). Finite- and infinite-horizon shapley games with nonsymmetric partial observation. *SIAM Journal on Control and Optimization*, 53(6):3584–3619.

[Bošanský et al., 2017] Bošanský, B., Brânzei, S., Hansen, K. A., Lund, T. B., and Miltersen, P. B. (2017). Computation of Stackelberg Equilibria of Finite Sequential Games. *ACM Transactions on Economics and Computation*, 5(4):23:1–23:24.

[Bošanský et al., 2015] Bošanský, B., Jiang, A. X., Tambe, M., and Kiekintveld, C. (2015). Combining Compact Representation and Incremental Generation in Large Games with Sequential Strategies. In *AAAI Conference on Artificial Intelligence*.

[Bošanský et al., 2014] Bošanský, B., Kiekintveld, C., Lisý, V., and Pěchouček, M. (2014). An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information. *Journal of Artificial Intelligence Research*, 51:829–866.

[Bošanský and Čermák, 2015] Bošanský, B. and Čermák, J. (2015). Sequence-Form Algorithm for Computing Stackelberg Equilibria in Extensive-Form Games. In *AAAI Conference on Artificial Intelligence*.

[Chatterjee and Doyen, 2014] Chatterjee, K. and Doyen, L. (2014). Partial-observation stochastic games: How to win when belief fails. *ACM Transactions on Computational Logic (TOCL)*, 15(2):16.

[Chen et al., 2006] Chen, X., Deng, X., and Teng, S.-H. (2006). Computing Nash equilibria: Approximation and smoothed complexity. In *Proc. 47th IEEE FOCS*.

[Coniglio et al., 2017] Coniglio, S., Gatti, N., and Marchesi, A. (2017). Pessimistic leader-follower equilibria with multiple followers. In *Proceedings of the 27th International Conference on Artificial Intelligence (IJCAI)*.

[Conitzer and Korzhyk, 2011] Conitzer, V. and Korzhyk, D. (2011). Commitment to Correlated Strategies. In *Proceedings of AAAI Conference on Artificial Intelligence*.

[Conitzer and Sandholm, 2006] Conitzer, V. and Sandholm, T. (2006). Computing the optimal strategy to commit to. In *Proceedings of the 7th ACM conference on Electronic commerce*, pages 82–90. ACM.

[Daskalakis et al., 2006] Daskalakis, C., Fabrikant, A., and Papadimitriou, C. H. (2006). The Game World Is Flat: The Complexity of Nash Equilibria in Succinct Games. In *ICALP*, pages 513–524.

[Durkota et al., 2017] Durkota, K., Lisý, V., Kiekintveld, C., Horák, K., Bošanský, B., and Pevný, T. (2017). Optimal Strategies for Detecting Data Exfiltration by Internal and External Attackers. In *GameSec 2017, LNCS 10575*, pages 171–192.

[Etessami and Yannakakis, 2007] Etessami, K. and Yannakakis, M. (2007). On the complexity of nash equilibria and other fixed points. In *FOCS*.

[Fang et al., 2015] Fang, F., Stone, P., and Tambe, M. (2015). When Security Games Go Green: Designing Defender Strategies to Prevent Poaching and Illegal Fishing. In *In Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*.

[Ghosh et al., 2004] Ghosh, M. K., McDonald, D., and Sinha, S. (2004). Zero-Sum Stochastic Games with Partial Information. *Journal of Optimization Theory and Applications*, 121(1):99–118.

[Hansen et al., 2004] Hansen, E. A., Bernstein, D. S., and Zilberstein, S. (2004). Dynamic Programming for Partially Observable Stochastic Games. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*.

[Hansen et al., 2007] Hansen, K., Miltersen, P., and Sørensen, T. (2007). Finding equilibria in games of no chance. In *Computing and Combinatorics*, volume 4598 of *Lecture Notes in Computer Science*, pages 274–284. Springer Berlin Heidelberg.

[Hoefer et al., 2009] Hoefer, M., Mirrokni, V. S., Röglin, H., and Teng, S.-H. (2009). Competitive routing over time. In Leonardi, S., editor, *Internet and Network Economics*, pages 18–29, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Horák and Bošanský, 2019] Horák, K. and Bošanský, B. (2019). Solving Partially Observable Stochastic Games with Public Observations. In *Proceedings of AAAI Conference on Artificial Intelligence.*

[Horák et al., 2017a] Horák, K., Bošanský, B., and Pěchouček, M. (2017a). Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games. In *In Proceedings of AAAI Conference on Artificial Intelligence*, pages 558–564.

[Horák et al., 2017b] Horák, K., Zhu, Q., and Bošanský, B. (2017b). Manipulating Adversary's Belief: A Dynamic Game Approach to Deception by Design for Proactive Network Security. In *GameSec 2017, LNCS 10575*, pages 273–294.

[Horák et al., 2019] Horák, K., Bošanský, B., Kiekintveld, C., and Kamhoua, C. (2019). Compact Representation of Value Function in Partially Observable Stochastic Games. https://arxiv.org/abs/1903.05511.

[Jain et al., 2013] Jain, M., Conitzer, V., and Tambe, M. (2013). Security Scheduling for Real-world Networks. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 215–222.

[Jain et al., 2011a] Jain, M., Korzhyk, D., Vanek, O., Conitzer, V., Tambe, M., and Pechoucek, M. (2011a). Double Oracle Algorithm for Zero-Sum Security Games on Graph. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 327–334.

[Jain et al., 2011b] Jain, M., Tambe, M., and Kiekintveld, C. (2011b). Quality-bounded solutions for finite Bayesian Stackelberg games: Scaling up. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, pages 997–1004.

[Kaneko and Kline, 1995] Kaneko, M. and Kline, J. J. (1995). Behavior Strategies, Mixed Strategies and Perfect Recall. *International Journal of Game Theory*, 24:127–145.

[Kiekintveld et al., 2009] Kiekintveld, C., Jain, M., Tsai, J., Pita, J., Ordóñez, F., and Tambe, M. (2009). Computing optimal randomized resource allocations for massive security games. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 689–696.

[Kline, 2002] Kline, J. J. (2002). Minimum Memory for Equivalence between Ex Ante Optimality and Time-Consistency. *Games and Economic Behavior*, 38:278–305.

[Koller and Megiddo, 1992] Koller, D. and Megiddo, N. (1992). The Complexity of Two-Person Zero-Sum Games in Extensive Form. *Games and Economic Behavior*, 4:528–552.

[Koller et al., 1996] Koller, D., Megiddo, N., and von Stengel, B. (1996). Efficient Computation of Equilibria for Extensive Two-Person Games. *Games and Economic Behavior*, 14(2):247–259.

[Kolodziej et al., 2013] Kolodziej, S., Castro, P. M., and Grossmann, I. E. (2013). Global optimization of bilinear programs with a multiparametric disaggregation technique. *Journal of Global Optimization*, 57(4):1039–1063.

[Kroer and Sandholm, 2016] Kroer, C. and Sandholm, T. (2016). Imperfect-recall abstractions with bounds in games. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 459–476. ACM.

[Kumar and Zilberstein, 2009] Kumar, A. and Zilberstein, S. (2009). Dynamic programming approximations for partially observable stochastic games.

[Lanctot et al., 2012] Lanctot, M., Gibson, R., Burch, N., Zinkevich, M., and Bowling, M. (2012). No-Regret Learning in Extensive-Form Games with Imperfect Recall. In *Proceedings of the 29th International Conference on Machine Learning (ICML 2012)*, pages 1–21.

[Letchford and Conitzer, 2010] Letchford, J. and Conitzer, V. (2010). Computing optimal strategies to commit to in extensive-form games. In *Proceedings of the 11th ACM conference on Electronic commerce*, pages 83–92, New York, NY, USA. ACM.

[Lisý et al., 2016] Lisý, V., Davis, T., and Bowling, M. (2016). Counterfactual Regret Minimization in Sequential Security Games. In *Proceedings of AAAI Conference on Artificial Intelligence*.

[MacDermed, 2013] MacDermed, L. C. (2013). *Value Methods for Efficiently Solving Stochastic Games of Complete and Incomplete Information*. PhD thesis, Georgia Institute of Technology.

[Madani et al., 1999] Madani, O., Hanks, S., and Condon, A. (1999). On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *AAAI/IAAI*, pages 541–548.

[Maschler et al., 2013] Maschler, M., Zamir, S., and Solan, E. (2013). *Game Theory*. Cambridge University Press.

[McMahan and Gordon, 2003] McMahan, H. B. and Gordon, G. J. (2003). Planning in cost-paired markov decision process games. In *NIPS Workshop: Planning for the Real-World*, volume 3. Citeseer.

[Nash, 1950] Nash, J. (1950). Equilibrium points in n-person games. In *National Academy of Sciences*, volume 36, pages 48–49.

[Pineau et al., 2003] Pineau, J., Gordon, G., and Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

[Rass et al., 2017] Rass, S., König, S., and Schauer, S. (2017). Defending Against Advanced Persistent Threats Using Game-Theory. *PLoS ONE*, 12(1).

[Rindt, 2019] Rindt, E. (2019). Dynamic Programming for Computing Stackelberg equilibrium in Sequential Games. Master's thesis, Faculty of Electrical Engineering, Czech Technical University in Prague.

[Shoham and Leyton-Brown, 2009] Shoham, Y. and Leyton-Brown, K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.

[Smith and Simmons, 2004] Smith, T. and Simmons, R. (2004). Heuristic search value iteration for POMDPs. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 520–527. AUAI Press.

[Smith and Simmons, 2012] Smith, T. and Simmons, R. (2012). Point-based POMDP algorithms: Improved analysis and implementation. *arXiv preprint arXiv:1207.1412*.

[Tambe, 2011] Tambe, M. (2011). *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press.

[Čermák et al., 2016] Čermák, J., Bošanský, B., Durkota, K., Lisý, V., and Kiekintveld, C. (2016). Using Correlated Strategies for Computing Stackelberg Equilibria in Extensive-Form Games. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 439–445.

[von Stackelberg, 1934] von Stackelberg, H. (1934). Marktform und gleichgewicht.

[von Stengel, 1996] von Stengel, B. (1996). Efficient Computation of Behavior Strategies. *Games and Economic Behavior*, 14:220–246.

[von Stengel and Forges, 2008] von Stengel, B. and Forges, F. (2008). Extensive-Form Correlated Equilibrium: Definition and Computational Complexity. *Mathematics of Operations Research*, 33(4):1002–1022.

[von Stengel and Zamir, 2010] von Stengel, B. and Zamir, S. (2010). Leadership games with convex strategy sets. *Games and Economic Behavior*, 69(2):446 – 457.

[Wichardt, 2008] Wichardt, P. C. (2008). Existence of Nash equilibria in finite extensive form games with imperfect recall: A counterexample. *Games and Economic Behavior*, 63(1):366–369.

[Yannakakis, 1990] Yannakakis, M. (1990). The analysis of local search problems and their heuristics. In *STACS*, pages 298–311.

[Yin et al., 2012] Yin, Z., Jiang, A. X., Johnson, M. P., Tambe, M., Kiekintveld, C., Leyton-Brown, K., Sandholm, T., and Sullivan, J. P. (2012). TRUSTS: Scheduling Randomized Patrols for Fare Inspection in Transit Systems. In *Proceedings of 24th Conference on Innovative Applications of Artificial Intelligence (IAAI)*.

[Zinkevich et al., 2007] Zinkevich, M., Bowling, M., and Burch, N. (2007). A New Algorithm for Generating Equilibria in Massive Zero-Sum Games. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 788–793.

[Čermák et al., 2018] Čermák, J., Bošanský, B., Horák, K., Lisý, V., and Pěchouček, M. (2018). Approximating maxmin strategies in imperfect recall games using A-loss recall property. *International Journal of Approximate Reasoning*, pages 290–326.

Appendix A

An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information

Journal of Artificial Intelligence Research 51 (2014) 829-866 Submitted 06/14; published 12/14

An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information

Branislav Bošanský BRANISLAV.BOSANSKY@AGENTS.FEL.CVUT.CZ
Agent Technology Center
Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague

Christopher Kiekintveld CDKIEKINTVELD@UTEP.EDU
Computer Science Department
University of Texas at El Paso, USA

Viliam Lisý VILIAM.LISY@AGENTS.FEL.CVUT.CZ
Michal Pěchouček MICHAL.PECHOUCEK@AGENTS.FEL.CVUT.CZ
Agent Technology Center
Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague

Abstract

Developing scalable solution algorithms is one of the central problems in computational game theory. We present an iterative algorithm for computing an exact Nash equilibrium for two-player zero-sum extensive-form games with imperfect information. Our approach combines two key elements: (1) the compact sequence-form representation of extensive-form games and (2) the algorithmic framework of double-oracle methods. The main idea of our algorithm is to restrict the game by allowing the players to play only selected sequences of available actions. After solving the restricted game, new sequences are added by finding best responses to the current solution using fast algorithms.

We experimentally evaluate our algorithm on a set of games inspired by patrolling scenarios, board, and card games. The results show significant runtime improvements in games admitting an equilibrium with small support, and substantial improvement in memory use even on games with large support. The improvement in memory use is particularly important because it allows our algorithm to solve much larger game instances than existing linear programming methods.

Our main contributions include (1) a generic sequence-form double-oracle algorithm for solving zero-sum extensive-form games; (2) fast methods for maintaining a valid restricted game model when adding new sequences; (3) a search algorithm and pruning methods for computing best-response sequences; (4) theoretical guarantees about the convergence of the algorithm to a Nash equilibrium; (5) experimental analysis of our algorithm on several games, including an approximate version of the algorithm.

1. Introduction

Game theory is a widely used methodology for analyzing multi-agent systems by applying formal mathematical models and solution concepts. One focus of computational game theory is the development of scalable algorithms for reasoning about very large games. The

need for continued algorithmic advances is driven by a growing number of applications of game theory that require solving very large game instances. For example, several decision support systems have recently been deployed in homeland security domains to recommend policies based on game-theoretic models for placing checkpoints at airports (Pita, Jain, Western, Portway, Tambe, Ordonez, Kraus, & Parachuri, 2008), scheduling Federal Air Marshals (Tsai, Rathi, Kiekintveld, Ordóñez, & Tambe, 2009), and patrolling ports (Shieh, An, Yang, Tambe, Baldwin, Direnzo, Meyer, Baldwin, Maule, & Meyer, 2012). The capabilities of these systems are based on a large amount of research in fast algorithms for security games (Tambe, 2011). Another notable example is the algorithmic progress that has led to game-theoretic Poker agents that are competitive with highly skilled human opponents (e.g., see Zinkevich, Bowling, & Burch, 2007; Sandholm, 2010).

We focus on developing new algorithms for an important general class of games that includes security games and Poker, as well as many other familiar games. More precisely, we study two-player zero-sum extensive-form games (EFGs) with imperfect information. This class of games captures sequential interactions between two strictly competitive players in situations where they make decisions under uncertainty. Uncertainty can be caused either by having a stochastic environment or by having opponent actions that are not directly observable. We consider general models for both sequential interactions and uncertainty, while many of the fast algorithms that have been developed for Poker and security domains rely on more specific game structure.

We propose a new class of algorithms for finding exact (or approximate) Nash equilibrium solutions for the class of EFGs with imperfect information. The leading exact algorithm in the literature uses the compact *sequence-form* representation and linear programming optimization techniques to solve games of this type (Koller, Megiddo, & von Stengel, 1996; von Stengel, 1996). Our approach exploits the same compact representation, but we improve the solution methods by adopting the algorithmic framework based on decompositions known in the computational game theory literature as *oracle algorithms* (McMahan, Gordon, & Blum, 2003). Oracle algorithms are related to the methods of constraint/column generation used for solving large-scale optimization problems (Dantzig & Wolfe, 1960; Barnhart, Johnson, Nemhauser, Savelsbergh, & Vance, 1998) and exploit two characteristics commonly found in games. First, in many cases finding a solution to a game only requires using a small fraction of the possible strategies, so it is not necessary to enumerate all of the strategies to find a solution (Wilson, 1972; Koller & Megiddo, 1996). Second, finding a best response to a specific opponent strategy in a game is computationally much less expensive than solving for an equilibrium. In addition, best response algorithms can often make use of domain-specific knowledge or heuristics to speed up the calculations even further.

Our sequence-form double-oracle algorithm integrates the decomposition ideas of oracle algorithms with the compact sequence-form representation for EFGs with imperfect information. This results in an iterative algorithm that does not always need to generate the complete linear program for the game to find a Nash equilibrium solution. The main idea of the algorithm is to create a restricted game in which the players choose from a limited space of possible strategies (represented as sequences of actions). The algorithm solves the restricted game and then uses a fast best-response algorithm to find strategies in the original unrestricted game that perform well against the current solution of the restricted

game. These strategies are added to the restricted game and the process iterates until no best response can be found to improve the solution. In this case, the current solution is an equilibrium of the original game. Typically, a solution can be found by adding only a small fraction of the strategies to the restricted game.

We begin by presenting related work, technical background, and our notation. We then describe our main algorithm in three parts: (1) methods for creating, solving, and expanding a valid restricted game, (2) the algorithm for finding the best-response strategies to be added to the restricted game, and (3) variants of the main loop controlling the iterative process of solving restricted games and adding new strategies. We present a formal analysis and prove that our algorithm converges to a Nash equilibrium of the original game. Finally, we provide an experimental evaluation of the runtime performance and convergence behavior of our algorithm on several realistic games with different characteristics including a border patrolling scenario, Phantom Tic-Tac-Toe, and a simplified variant of Poker. We compare our results with state-of-the-art algorithms for finding both exact and approximate solutions: linear programming using the sequence form, and Counterfactual Regret Minimization (CFR, Zinkevich, Johanson, Bowling, & Piccione, 2008; Lanctot, 2013).

The experimental results confirm that our algorithm requires only a fraction of all possible sequences to solve a game in practice and significantly reduces memory requirements when solving large games. This advances the state of the art and allows us to exactly solve much larger games compared to the existing algorithms. Moreover, in games admitting an equilibrium with small support (i.e., only a few sequences have non-zero probability in an equilibrium), our algorithm also achieves significant improvements in computation time and finds an equilibrium after only few iterations. These result hold without using any domain-specific knowledge, but we also show that incorporating domain-specific heuristics and bounds into the algorithm in a straightforward way can lead to even more significant performance improvements. Analysis of the convergence rate shows that the approximative bounds on the value of the game are either similar or a bit worse during the early stages compared to CFR. However, the convergence behavior of CFR algorithm has a very long tail and our algorithm always finds an exact solution much faster than CFR.

2. Related Work

Solving imperfect-information EFGs is a computationally challenging task, primarily due to uncertainty about the actions of the opponent and/or a stochastic environment. The leading exact algorithm (Koller et al., 1996; von Stengel, 1996) is based on formulating the problem of finding an optimal strategy to play as a linear program. This algorithm exploits a compact representation of strategies as sequences of individual actions (called *the sequence form*) and results in a linear program of linear size in the size of the game tree. However, this approach has limited applicability since the game tree grows exponentially with the number of sequential actions in the game. A common practice for overcoming the limited scalability of sequence-form linear programming is to use an approximation method. The best known approximative algorithms include counterfactual regret minimization (CFR, Zinkevich et al., 2008), improved versions of CFR with sampling methods (Lanctot, Waugh, Zinkevich, & Bowling, 2009; Gibson, Lanctot, Burch, Szafron, & Bowling, 2012); Nesterov's Excessive Gap Technique (EGT, Hoda, Gilpin, Peña, & Sandholm, 2010); and variants of

Monte Carlo Tree Search (MCTS) algorithms applied to imperfect-information games (e.g., see Ponsen, de Jong, & Lanctot, 2011).

The family of counterfactual regret minimization algorithms is based on learning methods that can be informally described as follows. The algorithm repeatedly traverses the game tree and learns a strategy to play by applying a no-regret learning rule that minimizes a specific variant of regret (counterfactual regret) in each information set. The no-regret learning converges to an optimal strategy in each information set. The overall regret is bounded by the sum of the regret in each information set; hence, the strategy as a whole converges to a Nash equilibrium. The main benefits of this approach include simplicity and robustness, as it can be adapted for more generic games (e.g., see Lanctot, Gibson, Burch, Zinkevich, & Bowling, 2012, where CFR is applied on games with imperfect recall). However, the algorithm operates on the complete game tree and therefore requires convergence in all information sets, which can be very slow for large games when one desires a solution with small error.

Another popular method is Excessive Gap Technique that exploits the convex properties of the sequence-form representation and uses recent mathematical results on finding extreme points of smooth functions (see Hoda et al., 2010, for the details). The main idea is to approximate the problem of finding a pair of equilibrium strategies by two smoothed functions and guiding them to find an approximate solution. Although this approach achieves faster convergence in comparison with CFR, the algorithm is less robust (it is not known whether a similar approach can be used for more general classes of games) and less used in practice. Like CFR, EGT also operates in the complete strategy space of all sequences.

Monte Carlo Tree Search (MCTS) is another family of methods that has shown promise for solving very large games, in particular perfect information board games such as Go (e.g., Lee et al., 2009). While the CFR and EGT algorithms are guaranteed to find an ε-Nash equilibrium, convergence to an equilibrium solution has not been formally shown for any of the variants of MCTS in imperfect-information games. On the contrary, the most common version of MCTS based on the Upper Confidence Bounds (UCB) selection function can converge to incorrect solutions even in simultaneous-move games (Shafiei, Sturtevant, & Schaeffer, 2009) that are the simplest class of imperfect-information EFGs. MCTS algorithms therefore do not (in general) guarantee finding an (approximate) optimal solution in imperfect-information games. One exception is the recent proof of convergence of MCTS with certain selection methods for simultaneous-move games (Lisy, Kovarik, Lanctot, & Bosansky, 2013). Still, using MCTS is sometimes a reasonable choice since it can produce good strategies in practice (Ponsen et al., 2011).

Contrary to the existing approximative approaches, our algorithm aims to find an exact solution without explicitly considering the strategy in the complete game tree. Our work combines the compact sequence-form representation and the double-oracle algorithmic framework. Previous work on the double-oracle framework has focused primarily on applications in normal-form games, where the restricted game was expanded by adding pure best-response strategies in each iteration. One of the first examples of solving games using the double-oracle principle was by McMahan et al. (2003). They introduced the double-oracle algorithm, proved the convergence to a Nash equilibrium, and experimentally verified that the algorithm achieves computation time improvements on a search game where an evader was trying to cross an environment without being detected by sensors placed by the

opponent. The double-oracle algorithm reduced the computation time from several hours to tens of seconds and allowed to solve much larger instances of this game. Similar success with the domain-specific double-oracle methods has been demonstrated on a variety of different domains inspired by pursuit-evasion games (Halvorson, Conitzer, & Parr, 2009) and security games played on a graph (Jain, Korzhyk, Vanek, Conitzer, Tambe, & Pechoucek, 2011; Letchford & Vorobeychik, 2013; Jain, Conitzer, & Tambe, 2013).

Only a few works have tried to apply the iterative framework of oracle algorithms to EFGs, primarily using pure and mixed strategies in EFGs. The first work that exploited this iterative principle is the predecessor of the sequence-form linear-program formulation (Koller & Megiddo, 1992). In this algorithm, the authors use a representation similar to the sequence form only for a single player, while the strategies for the opponent are iteratively added as constraints into the linear program (there is an exponential number of constraints in their formulation). This approach can be seen as a specific variant of the oracle algorithms, where the strategy space is expanded gradually for a single player. Our algorithm is a generalization of this work, since our algorithm uses the sequence-form representation for both players and it also incrementally expands the strategy space for both players.

More recent work has been done by McMahan in his thesis (McMahan, 2006) and follow-up work (McMahan & Gordon, 2007). In these works the authors investigated an extension of the double-oracle algorithm for normal-form games to the extensive-form case. Their double-oracle algorithm for EFGs operates very similarly to the normal-form variant and uses pure and mixed strategies defined for EFGs. The main disadvantage of this approach is that in the basic version it still requires a large amount of memory since a pure strategy for an EFG is large (one action needs to be specified for each information set), and there is an exponential number of possible pure strategies. To overcome this disadvantage, the authors propose a modification of the double-oracle algorithm that keeps the number of the strategies in the restricted game bounded. The algorithm removes from the restricted game those strategies that are the least used in the current solution of the restricted game. In order to guarantee the convergence, the algorithm adds in each iteration into the restricted game a mixed strategy representing the mean of all removed strategies; convergence is then guaranteed similarly to fictitious play (see McMahan & Gordon, 2007, for the details). Bounding the size of the restricted game results in low memory requirements. However, the algorithm converges extremely slowly and it can take a very long time (several hours for a small game) for the algorithm to achieve a small error (see the experimental evaluation in McMahan, 2006; McMahan & Gordon, 2007).

A similar concept for using pure strategies in EFGs is used in an iterative algorithm designed for Poker in the work of Zinkevich et al. (2007). The algorithm in this work expands the restricted game with strategies found by *a generalized best response* instead of using pure best response strategies. Generalized best response is a Nash equilibrium in a partially restricted game – the player computing the best response can use any of the pure strategies in the original unrestricted game, while the opponent is restricted to use only the strategies from the restricted game. However, the main disadvantages of using pure and mixed strategies in EFGs are still present and result in large memory requirements and an exponential number of iterations.

In contrast, our algorithm directly uses the compact sequence-form representation of EFGs and uses the sequences as the building blocks (i.e., the restricted game is expanded

by allowing new sequences to be played in the next iteration). Using sequences and the sequence form for solving the restricted game reduces the size of the restricted game and the number of iterations, however, it also introduces new challenges when constructing and maintaining the restricted game, and ensuring the convergence to a Nash equilibrium, which we must solve for our algorithm to converge to a correct solution.

3. Technical Background

We begin by presenting the standard game-theoretic model of extensive-form games, followed by a discussion of the most common solution concepts and the algorithms for computing these solutions. Then we present the sequence-form representation and the state-of-the-art linear program for computing solutions using this representation. Finally, we describe oracle algorithms as they are used for solving normal-form games. A summary of the most common notation is provided in Table 1 for quick reference.

3.1 Extensive-Form Games

Extensive-form games (EFGs) model sequential interactions between players in a game. Games in the extensive form are visually represented as game trees (e.g., see Figure 2). Nodes in the game tree represent states of the game; each state of the game corresponds to a sequence of moves executed by all players in the game. Each node is assigned to a player that acts in the game state associated with this node. An edge in the game tree from a node corresponds to an action that can be performed by the player who acts in this node. Extensive-form games model limited observations of the players by grouping the nodes into *information sets*, so that a given player cannot distinguish between nodes that belong to the same information set when the player is choosing an action. The model also represents uncertainty about the environment and stochastic events by using a special *Nature player*.

Formally, a two-player EFG is defined as a tuple $G = (N, H, Z, A, p, u, \mathcal{C}, \mathcal{I})$: N is a set of two players $N = \{1, 2\}$. We use i to refer to one of the two players (either 1 or 2), and $-i$ to refer to the opponent of i. H denotes a finite set of *nodes* in the game tree. Each node corresponds to a unique *history* of actions taken by all players and Nature from the root of the game; hence, we use the terms history and node interchangeably. We denote by $Z \subseteq H$ the set of all *terminal nodes* of the game. A denotes the set of all actions and we overload the notation and use $A(h) \subseteq A$ to represent the set of actions available to the player acting in node $h \in H$. We specify $ha = h' \in H$ to be node h' reached from node h by executing action $a \in A(h)$. We say that h is a *prefix* of h' and denote it by $h \sqsubseteq h'$. For each terminal node $z \in Z$ we define a *utility function* for each player i ($u_i : Z \to \mathbb{R}$). We study zero-sum games, so $u_i(z) = -u_{-i}(z)$ holds for all $z \in Z$.

The function $p : H \to N \cup \{c\}$ assigns each node to a player who takes an action in the node, where c means that the Nature player selects an action in the node based on a fixed probability distribution known to all players. We use function $\mathcal{C} : H \to [0, 1]$ to denote the probability of reaching node h due to Nature (i.e., assuming that both players play all required actions to reach node h). The value of $\mathcal{C}(h)$ is the product of the probabilities assigned to all actions taken by the Nature player in history h. Imperfect observation of player i is modeled via *information sets* $\mathcal{I}_i$ that form a partition over the nodes assigned to player i $\{h \in H : p(h) = i\}$. Every information set contains at least one node and each

node belongs to exactly one information set. Nodes in an information set of a player are indistinguishable to the player. All nodes h in a single information set $I_i \in \mathcal{I}_i$ have the same set of possible actions $A(h)$. Action a from $A(h)$ uniquely identifies information set I_i and there cannot exist any other node $h' \in H$ that does not belong to information set I_i and for which a is allowed to be played (i.e., $a \in A(h')$). Therefore we overload notation and use $A(I_i)$ to denote the set of actions defined for each node h in this information set. We assume *perfect recall*, which means that players perfectly remember their own actions and all information gained during the course of the game. As a result, all nodes in any information set I_i have the same history of actions for player i.

3.2 Nash Equilibrium in Extensive-Form Games

Solving a game requires finding a strategy profile (i.e., one strategy for each player) that satisfies conditions defined by a specific solution concept. *Nash equilibrium* (NE) is the best known solution concept in game theory and it describes the behavior of players under certain assumptions about their rationality. In a Nash equilibrium, every player plays a best response to the strategies of the other players. Let Π_i be the set of *pure strategies* for player i. In EFGs, a pure strategy is an assignment of exactly one action to be played in each information set. A mixed strategy is a probability distribution over the set of all pure strategies of a player. We denote by Δ_i the set of all mixed strategies of player i. For any pair of strategies $\delta \in \Delta = (\Delta_1, \Delta_2)$ we use $u_i(\delta) = u_i(\delta_i, \delta_{-i})$ for the expected outcome of the game for player i when players follow strategies δ. A *best response* of player i to the opponent's strategy δ_{-i} is a strategy δ_i^{BR}, for which $u_i(\delta_i^{BR}, \delta_{-i}) \geq u_i(\delta_i', \delta_{-i})$ for all strategies $\delta_i' \in \Delta_i$. A strategy profile $\delta = (\delta_1, \delta_2)$ is a NE if and only if for each player i it holds that δ_i is a best response to δ_{-i}. A game can have multiple NEs; in the zero-sum setting, all of these equilibria have the same value (i.e., the expected utility for every player is the same). This is called the *value of the game*, denoted V^*. The problem of finding a NE in a zero-sum game has a polynomial computational complexity in the size of the game.

The NE solution concept is somewhat weak for extensive-form games. Nash equilibrium requires that both players act rationally. However, there can be irrational strategies selected for the parts of the game tree that are not reachable when both players follow the NE strategies (these parts are said to be *off the equilibrium path*). The reason is that NE does not expect this part of the game to be played and therefore does not sufficiently restrict strategies in these information sets. To overcome these drawbacks, a number of refinements of NE have been introduced imposing further restrictions with the intention of describing more sensible strategies. Examples include *subgame-perfect equilibrium* (Selten, 1965) used in perfect-information EFGs. The subgame-perfect equilibrium forces the strategy profile to be a Nash equilibrium in each sub-game (i.e., in each sub-tree rooted in some node h) of the original game. Unfortunately, sub-games are not particularly useful in imperfect-information EFGs; hence, here the refinements include *strategic-from perfect equilibrium* (Selten, 1975), *sequential equilibrium* (Kreps & Wilson, 1982), or *quasi-perfect equilibrium* (van Damme, 1984; Miltersen & Sørensen, 2010). The first refinement avoids using weakly dominated strategies in equilibrium strategies for two-player games (van Damme, 1991, p. 29) and it is also known as the *undominated equilibrium*. Sequential equilibrium tries to exploit the mistakes of the opponent by using the notion of beliefs consistent with the

strategy of the opponent even in information sets off the equilibrium path. The main intuitions behind the first two refinements are combined in quasi-perfect equilibrium.

Even though the solution described by NE does not always prescribe rational strategies off the equilibrium path, it is still valuable to compute exact NE of large extensive-form games for several reasons. We focus on zero-sum games, so the NE strategy guarantees the value of the game even off the equilibrium path. In other words, the strategy off the equilibrium path does not optimally exploit the mistakes of the opponent, but it still guarantees an outcome of at least value gained by following the equilibrium path. Moreover, a refined equilibrium is still a NE and calculating the value of the game is often a starting point for many of the algorithms that compute these refinements – for example it is used for computing undominated equilibrium (e.g., see Ganzfried & Sandholm, 2013; Cermak, Bosansky, & Lisy, 2014) and *normal-form proper equilibrium* (Miltersen & Sørensen, 2008).

3.3 Sequence-Form Linear Program

Extensive-form games with perfect recall can be compactly represented using the *sequence form* (Koller et al., 1996; von Stengel, 1996). A *sequence* σ_i is an ordered list of actions taken by a single player i in a history h. The number of actions (i.e., the length of sequence σ_i) is denoted by $|\sigma_i|$ and the empty sequence (i.e., sequence with no actions) is denoted by $\emptyset$. The set of all possible sequences for player i is denoted by Σ_i and the set of sequences for all players is $\Sigma = \Sigma_1 \times \Sigma_2$. A sequence $\sigma_i \in \Sigma_i$ can be extended by a single action a taken by player i, denoted by $\sigma_i a = \sigma_i'$ (we use $\sigma_i \sqsubseteq \sigma_i'$ to denote that σ_i is a prefix of σ_i'). In games with perfect recall, all nodes in an information set I_i share the same sequence of actions for player i and we use $\mathsf{seq}_i(I_i)$ to denote this sequence. We overload the notation and use $\mathsf{seq}_i(h)$ to denote the sequence of actions of player i leading to node h, and $\mathsf{seq}_i(H') \subseteq \Sigma_i$, where $\mathsf{seq}_i(H') = \bigcup_{h' \in H'} \mathsf{seq}_i(h')$ for some $H' \subseteq H$. Since action a uniquely identifies information set I_i and all nodes in an information set share the same history of actions of player i, each sequence uniquely identifies an information set. We use the function $\mathsf{inf}_i(\sigma_i')$ to denote the information set in which the last action of the sequence σ_i' is taken. For an empty sequence, function $\mathsf{inf}_i(\emptyset)$ is the information set of the root node.

Finally, we define the auxiliary payoff function $g_i : \Sigma \to \mathbb{R}$ that extends the utility function to all nodes in the game tree. The payoff function g_i represents the expected utility of all nodes reachable by sequentially executing the actions specified in a pair of sequences σ:

$$g_i(\sigma_i, \sigma_{-i}) = \sum_{h \in Z \,:\, \forall j \in N \;\sigma_j = \mathsf{seq}_j(h)} u_i(h) \cdot \mathcal{C}(h) \tag{1}$$

The value of the payoff function is defined to be 0 if no leaf is reachable by sequentially executing all of the actions in the sequences σ – either all actions from the pair of sequences σ are executed and an inner node ($h \in H \setminus Z$) is reached, or during the sequential execution of the actions node h is reached, for which the current action a to be executed from sequence $\sigma_{\rho(h)}$ is not defined (i.e., $a \notin A(h)$). Formally we define a pair of sequences σ to be *compatible* if there exists node $h \in H$ such that sequence σ_i of every player i equals to $\mathsf{seq}_i(h)$.

We can compute a Nash equilibrium of a two-player zero-sum extensive-form game using a linear program (LP) of a polynomial size in the size of the game tree using the

sequence form (Koller et al., 1996; von Stengel, 1996). The LP uses an equivalent compact representation of mixed strategies of players in a form of *realization plans*. A realization plan for a sequence σ_i is the probability that player i will play this sequence of actions under the assumption that the opponent will choose compatible sequences of actions that reach the information sets for which the actions specified in the sequence σ_i are defined. We denote the realization plan for player i by $r_i : \Sigma_i \to \mathbb{R}$. The equilibrium realization plans can be computed using the following LP (e.g., see Shoham & Leyton-Brown, 2009, p. 135):

$$\max_{r,v} v_{\mathsf{inf}_{-i}(\emptyset)}$$

$$v_{\mathsf{inf}_{-i}(\sigma_{-i})} - \sum_{I'_{-i} \in \mathcal{I}_{-i} : \mathsf{seq}_{-i}(I'_{-i}) = \sigma_{-i}} v_{I'_{-i}} \leq \sum_{\sigma_i \in \Sigma_i} g_i(\sigma_{-i}, \sigma_i) \cdot r_i(\sigma_i) \qquad \forall \sigma_{-i} \in \Sigma_{-i} \tag{2}$$

$$r_i(\emptyset) = 1 \tag{3}$$

$$\sum_{\forall a \in A(I_i)} r_i(\sigma_i a) = r_i(\sigma_i) \qquad \forall I_i \in \mathcal{I}_i,\ \sigma_i = \mathsf{seq}_i(I_i) \tag{4}$$

$$r_i(\sigma_i) \geq 0 \qquad \forall \sigma_i \in \Sigma_i \tag{5}$$

Solving the LP yields a realization plan for player i using variables r_i, and expected values for the information sets of player $-i$ (variables $v_{I_{-i}}$). The LP works as follows: player i maximizes the expected utility value by selecting the values for the variables of realization plan that is constrained by Equations (3–5). The probability of playing the empty sequence is defined to be 1 (Equation 3), and the probability of playing a sequence σ_i is equal to the sum of the probabilities of playing sequences extended by exactly one action (Equation 4). Finding such a realization plan is also constrained by the best responding opponent, player $-i$. This is ensured by Equation (2), where player $-i$ selects in each information set I_{-i} such action that minimizes the expected utility value $v_{I_{-i}}$ in this information set. There is one constraint defined for each sequence σ_{-i}, where the last action of this sequence determines the best action to be played in information set $\mathsf{inf}_{-i}(\sigma_{-i}) = I_{-i}$. The expected utility is composed of the expected utilities of the information sets reachable after playing sequence σ_{-i} (sum of v variables on the left side) and of the expected utilities of leafs to which this sequence leads (sum of g values on the right side of the constraint).

3.4 Double-Oracle Algorithm for Normal-Form Games

We now describe the concept of column/constraint generation techniques applied previously in normal-form games and known as the double-oracle algorithm (McMahan et al., 2003). *Normal-form games* are represented using game matrices; rows of the matrix correspond to pure strategies of one player, columns correspond to pure strategies of the opponent, and values in the matrix cells represent the expected outcome of the game when players play corresponding pure strategies. Zero-sum normal-form games can be solved by linear programming in polynomial time in the size of the matrix (e.g., see Shoham & Leyton-Brown, 2009, p. 89).

Figure 1 shows the visualization of the main structure of the double-oracle algorithm for normal-form games. The algorithm consists of the following three steps that repeat until convergence:

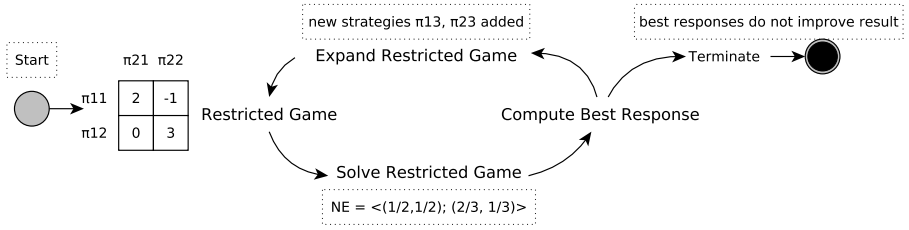


Figure 1: Schematic of the double-oracle algorithm for a normal-form game.

1. create a restricted game by limiting the set of pure strategies that each player is allowed to play
2. compute a pair of Nash equilibrium strategies in this restricted game using the LP for solving normal-form games
3. for each player, compute a pure best response strategy against the equilibrium strategy of the opponent found in the previous step; the best response may be *any* pure strategy in the original unrestricted game

The best response strategies computed in step 3 are added to the restricted game, the game matrix is expanded by adding new rows and columns, and the algorithm continues with the next iteration. The algorithm terminates if neither of the players can improve the outcome of the game by adding a new strategy to the restricted game. In this case both players play a best response to the strategy of the opponent in the original unrestricted game. The algorithm maintains the values of the expected utilities of the best-response strategies throughout the iterations of the algorithm. These values provide bounds on the value of the original unrestricted game V^* – from the perspective of player i, the minimal value of all of her past best-response calculations represents an upper bound of the value of the original game, V_i^{UB}, and the maximal value of all of past best-response calculations of the opponent represents the lower bound on the value of the original game, V_i^{LB}. Note that for the bounds it holds that the lower bound for player i is equal to the negative of the value of the upper bound for the opponent:

$$V_i^{LB} = -V_{-i}^{UB}$$

In general, computing best responses is computationally less demanding than solving the game, since the problem is reduced to a single-player optimization. Due to the fact that best-response algorithms can operate very quickly (e.g., also by exploiting additional domain-specific knowledge), they are called *oracles* in this context. If the algorithm incrementally adds strategies only for one player, the algorithm is called a *single-oracle algorithm*, if the algorithm incrementally adds the strategies for both players, the algorithm is called a *double-oracle algorithm*. Double-oracle algorithms are typically initialized by an arbitrary pair of strategies (one pure strategy for each player). However, we can also use a larger set of initial strategies selected based on a domain-specific knowledge.

The double-oracle algorithm for zero-sum normal-form games runs in a polynomial time in the size of the game matrix. Since each iteration adds at least one pure strategy to

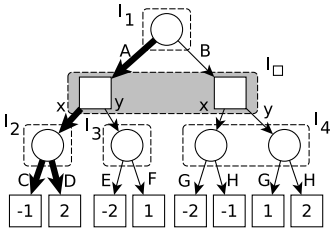


Figure 2: Example of a two-player extensive-form game visualized as a game tree. Circle player aims to maximize the utility value, box aims to minimize the utility value. The bold edges represent the sequences of actions added to the restricted game.

the restricted game and there are finite pure strategies, the algorithm stops after at most $|\Pi_i| + |\Pi_{-i}|$ iterations. Each iteration is also polynomial, since it consists of solving the linear program and computing best responses. The relative performance of the double-oracle algorithm compared to solving the linear program for the original unrestricted game closely depends on the number of iterations required for convergence. In the worst case, the algorithm adds all pure strategies and solves the original game, although this is rarely the case in practice. Estimating the expected number of iterations needed for the double-oracle algorithm to converge, however, remains an open problem.

3.4.1 Towards Extensive-Form Games

The straightforward method of applying the double-oracle algorithm for EFGs is to use pure strategies defined in EFGs (i.e., assignments of action for each information set, or realization plans) and apply exactly the algorithm described in this section – i.e., iteratively add pure strategies from the unrestricted extensive-form game into the restricted game matrix. However, this can result in an exponential number of iterations and an exponentially large restricted game in the worst case. Our algorithm differs significantly from this idea since it directly operates on (more compact) sequences instead of full strategies.

4. Sequence-Form Double-Oracle Algorithm for Extensive-Form Games

We now describe our sequence-form double-oracle algorithm for solving extensive-form games with imperfect information. First, we give an informal overview of our algorithm. We use an example game depicted in Figure 2 to illustrate some of the key concepts. Afterwards, we formally define the restricted game and describe the key components of the algorithm, following by a full example run of our algorithm.

The overall scheme of our algorithm is based on the double-oracle framework described in the previous section. The main difference is that our algorithm uses the sequences to define the restrictions in the game tree. The restricted game in our model is defined by allowing players to use (i.e., to play with non-zero probability) only a subset of the sequences from the original unrestricted game. This restricted subset of sequences defines the subsets of reachable actions, nodes, and information sets from the original game tree. Consider our example in Figure 2. A restricted game can be defined by sequences $\emptyset, A, AC, AD$ for the circle player, and $\emptyset, x$ for the box player. These sequences represent actions allowed in the game,

they define reachable nodes (using history we can reference them as $\emptyset, A, Ax, AxC, AxD$), and reachable information sets (I_1, I_2 for the circle player and the only information set $I_\square$ for the box player).

The algorithm iteratively adds new sequences of allowed actions into the restricted game, similarly to the double-oracle algorithm for normal-form games. The restricted game is solved as a standard zero-sum extensive-form game using the sequence-form linear program. Then a best response algorithm searches the original unrestricted game to find new sequences to add to the restricted game. When the sequences are added, the restricted game tree is expanded by adding all new actions, nodes, and information sets that are now reachable based on the new sets of allowed sequences. The process of solving the restricted game and adding new sequences iterates until no new sequences that improve the solution can be added.

There are two primary complications that arise when we use sequences instead of full strategies in the double-oracle algorithm, both due to the fact that sequences do not necessarily define actions in all information sets: (1) a strategy computed in the restricted game may not be a complete strategy in the original game, because it does not define behavior for information sets that are not in the restricted game, and (2) it may not be possible to play every action from a sequence that is allowed in the restricted game, because playing a sequence can depend on having a compatible sequence of actions for the opponent. In our example game tree in Figure 2, no strategy of the circle player in the restricted game specifies what to play in information sets I_3 and I_4. The consequence of the second issue is that some inner nodes of the original unrestricted game can (temporarily) become leafs in the restricted game. For example, the box player can add sequence y into the restricted game making node Ay a leaf in the restricted game, since there are no other actions of the circle player in the restricted game applicable in this node.

Our algorithm solves these complications using two novel ideas. The first idea is the concept of a *default pure strategy* (denoted $\pi_i^{\text{DEF}} \in \Pi_i$). Informally speaking, the algorithm assumes that each player has a fixed implicit behavior that defines what the player does by default in any information set that is not part of the restricted game. This is described by the default strategy π_i^{DEF}, which specifies an action for every information set. Note that this default strategy does not need to be represented explicitly (which could use a large amount of memory). Instead, it can be defined implicitly using rules, such as selecting the first action from a deterministic method for generating the ordered set of actions $A(h)$ in node h. We use the default pure strategies to map every strategy from the restricted game into a valid strategy in the full game. Specifically, the strategy in the original unrestricted game selects actions according to the probabilities specified by a strategy for the restricted game in every information set that is part of the restricted game, and for all other information sets it plays according to the default pure strategy. Recall our example in Figure 2, where the pure default strategy for the circle player can be $\langle A, C, E, G \rangle$ (i.e., selecting the leftmost action in each information set). Hence, a strategy in the original unrestricted game can use a strategy from the restricted game in information sets I_1 and I_2, and select pure actions in E, G in information sets I_3 and I_4 respectively.

The second key idea is to use *temporary utility values* for cases where there are no allowed actions that can be played in some node in the restricted game that is an inner node in the original game (so called *temporary leaf*). To ensure the correct convergence of

H	game-tree nodes / histories
$Z \subseteq H$	leafs / terminal states
π_i^{DEF}	implicit default pure strategy for player i
$r_i : \Sigma_i \mapsto \mathbb{R}$	realization plan of player i for a sequence
$\mathcal{C} : H \mapsto \mathbb{R}$	probability of reaching a node due to Nature play
$g_i : H \mapsto \mathbb{R}$	extension of the utility function to all nodes; $g_i(h) = u_i(h) \cdot \mathcal{C}(h)$ if $h \in Z$ and $g_i(h) = 0$ if h is not a terminal node ($h \notin Z$)
seq_i	sequence(s) of actions of player i leading to a node / a set of nodes / / an information set
$\mathsf{inf}_i : \Sigma_i \mapsto I_i$	an information set in which the last action of the sequence was executed

Table 1: An outline of the main symbols used in the paper.

the algorithm these temporary utilities must be assigned so that they provide a bound on the expected value gained by continuing the play from the given node. Our algorithm uses a value that corresponds to the expected outcome of continuing the game play, assuming the player making the choice in the temporary leaf uses the default strategy, while the opponent plays a best response. Assume we add sequence y for the box player into the restricted game in our example tree in Figure 2. The temporary utility value for node Ay would correspond to value -2, since the default strategy in information set I_3 is to play E for the circle player. In the next section we formally describe this method and prove the correctness of the algorithm given these temporary values.

We now describe in detail the key parts of our method. We first formally define the restricted game and methods for expanding the restricted game, including the details of both of the key ideas introduced above. Then we describe the algorithm for selecting the new sequences that are allowed in the next iteration. The decision of which sequences to add is based on calculating a best response in the original unrestricted game using game-tree search improved with additional pruning techniques. Finally, we discuss different variations of the main logic of the double-oracle algorithm that determines for which player(s) the algorithm adds new best-response sequences in the current iteration.

4.1 Restricted Game

This section formally defines the restricted game as a subset of the original unrestricted game. A restricted game can be fully specified by the set of allowed sequences. We define the sets of nodes, actions, and information sets as subsets of the original unrestricted sets based on the allowed sequences. We denote the original unrestricted game by a tuple $G = (N, H, Z, A, p, u, \mathcal{C}, \mathcal{I})$ and the restricted game by $G' = (N, H', Z', A', p, u', \mathcal{C}, \mathcal{I}')$. All sets and functions associated with the restricted game use prime in the notation; the set of players, and the functions p and $\mathcal{C}$ remain the same.

The restricted game is defined by a set of *allowed sequences* (denoted by $\Phi' \subseteq \Sigma$) that are returned by the best response algorithms. As indicated above, even an allowed sequence $\sigma_i \in \Phi'$ might not be playable to the full length due to missing compatible sequences of the opponent. Therefore, the restricted game is defined using the maximal compatible set of sequences $\Sigma' \subseteq \Phi'$ for a given set of allowed sequences Φ'. We define Σ' as the *maximal*

subset of the sequences from Φ' such that:

$$\Sigma'_i \leftarrow \{\sigma_i \in \Phi'_i \ : \ \exists \sigma_{-i} \in \Phi'_{-i} \exists h \in H \ \forall j \in N \ \mathsf{seq}_j(h) = \sigma_j\} \quad \forall i \in N \tag{6}$$

Equation (6) means that for each player i and every sequence σ_i in Σ'_i, there exists a compatible sequence of the opponent σ_{-i} that allows the sequence σ_i to be executed in full (i.e., by sequentially executing of all the actions in these sequences σ some node h can be reached such that $\mathsf{seq}_j(h) = \sigma_j$ for all players $j \in N$).

The set of sequences Σ' fully defines the restricted game, because all other sets in the tuple G' can be derived from Σ'. A node h is in the restricted game if and only if the sequences that must be played to reach h are in the set Σ' for *both* players:

$$H' \leftarrow \{h \in H \ : \ \forall i \in N \ \mathsf{seq}_i(h) \in \Sigma'\} \tag{7}$$

If a pair of sequences is in Σ', then *all* nodes reachable by executing this pair of sequences are included in H'. Actions defined for a node h are in the restricted game if and only if playing the action in this node leads to a node that is in the restricted game:

$$A'(h) \leftarrow \{a \in A(h) \ : \ ha \in H'\} \quad \forall h \in H' \tag{8}$$

Nodes from the restricted game corresponding to inner nodes in the original unrestricted game may not be inner nodes in the restricted game. Therefore, the set of leaves in the restricted game is a union of leaf nodes of the original game and inner nodes from the original game that currently do not have a valid continuation in the restricted game, based on the allowed sequences:

$$Z' \leftarrow \left(Z \cap H'\right) \cup \{h \in H' \setminus Z \ : \ A'(h) = \emptyset\} \tag{9}$$

We explicitly differentiate between leaves in the restricted game that correspond to leaves in the original unrestricted game (i.e., $Z' \cap Z$) and leaves in the restricted game that correspond to inner nodes in the original unrestricted game (i.e., $Z' \setminus Z$), since the algorithm assigns temporary utility values to nodes in the latter case.

The information sets in the restricted game correspond to information sets in the original unrestricted game. If some node h belongs to an information set $I_{p(h)}$ in the original game, then the same holds in the restricted game. We define an information set to be a part of the restricted game if and only if at least one inner node that belongs to this information set is included in the restricted game:

$$\mathcal{I}'_i \leftarrow \{I_i \in \mathcal{I}_i \ : \ \exists h \in I_i \ h \in H' \setminus Z'\} \tag{10}$$

An information set in the restricted game $I_i \in \mathcal{I}'_i$ consists only of nodes that are in the restricted game – i.e., $\forall h \in I_i \ : \ h \in H'$.

Finally, we define the modified utility function u' for the restricted game. The primary reason for the modified utility function is to define the temporary utility values for leaves in the set $Z' \setminus Z$. Consider $h \in Z' \setminus Z$ to be a temporary leaf and player i to be the player acting in this node ($i = p(h)$). Moreover, let $u^*_i(h)$ be the expected outcome of the game starting from this node assuming both players are playing NE strategies in the original unrestricted game. The modified utility function u'_i for this leaf must return a value that is a lower bound

on value $u_i^*(h)$. Due to the zero-sum assumption, this value represents an upper bound on value for the opponent $-i$. Setting the value this way ensures two things: (1) player $-i$ is likely to use sequences leading to node h in optimal strategies in the restricted game (since the modified utility value is an upper bound of an actual value), and (2) player i adds new sequences using best-response algorithms that prolong sequence $\mathsf{seq}_i(h)$ leading to node h if there are sequences that would yield better expected value than u'_i. Later we show a counterexample where setting the value otherwise can cause the algorithm to converge to an incorrect solution. We calculate the lower bound by setting the utility value so that it corresponds to the outcome in the original game if the player i continues by playing the default strategy π_i^{DEF} and the opponent plays a best response δ_{-i}^{BR} to this default strategy. This is a valid lower bound since we consider only a single strategy for the player acting in node h, which correspond to the default strategy; considering other strategies could allow this player to improve the value of continuing from the node h. For all other leaf nodes $h \in Z' \cap Z$ we set $u'_i(h) \leftarrow u_i(h)$.

4.1.1 Solving the Restricted Game

The restricted game defined in this section is a valid zero-sum extensive-form game and it can be solved using the sequence-form linear programming described in Section 3. The algorithm computes a NE of the restricted game by solving a pair of linear programs using the restricted sets Σ', H', Z', I', and the modified utility function u'.

Each strategy from the restricted game can be translated to the original game by using the pure default strategy to extend the restricted strategy where it is not defined. Formally, if r'_i is a mixed strategy represented as a realization plan of player i in the restricted game, then we define the *extended strategy* $\overline{r}'_i$ to be a strategy identical to the strategy in the restricted game for sequences included in the restricted game, and correspond to the default strategy π_i^{DEF} if a sequence is not included in the restricted game:

$$\overline{r}'_i(\sigma_i) \leftarrow \begin{cases} r'_i(\sigma_i) & \sigma_i \in \Sigma'_i \\ r'_i(\sigma'_i) \cdot \pi_i^{\text{DEF}}(\sigma_i \setminus \sigma'_i) & \sigma_i \notin \Sigma'_i;\ \sigma'_i = \arg\max_{\sigma''_i \in \Sigma'_i;\ \sigma''_i \sqsubseteq \sigma_i} |\sigma''_i| \end{cases} \tag{11}$$

The realization plan of a sequence σ_i not allowed in the restricted game (i.e., $\sigma_i \notin \Sigma'_i$) is equal to the realization probability of the longest prefix of the sequence allowed in the restricted game (denoted by σ'_i), and setting the remaining part of the sequence (i.e., $\sigma_i \setminus \sigma'_i$) to correspond to the default strategy of player i. This computation is expressed as a multiplication of two probabilities, where we overload the notation and use $\pi_i^{\text{DEF}}(\sigma_i \setminus \sigma'_i)$ to be 1 if the remaining part of the sequence σ_i corresponds to the default strategy of player i, and 0 otherwise.

In each iteration of the double-oracle algorithm one sequence-form LP is solved for each player to compute a pair of NE strategies in the restricted game. We denote these strategies as (r_i^*, r_{-i}^*) and $(\overline{r}_i^*, \overline{r}_{-i}^*)$ when they are extended to the original unrestricted game using the default strategies.

4.1.2 Expanding the Restricted Game

The restricted game is expanded by adding new sequences to the set Φ' and updating the remaining sets according to their definition. After adding new sequences, the algorithm

calculates and stores the temporary utility values for leaves in $Z' \setminus Z$ so they can be used in the sequence-form LP.

After updating the restricted game, the linear programs are modified so that they correspond to the new restricted game. For all newly added information sets and sequences, new variables are created in the linear programs and the constraints corresponding to these information sets/sequences are created (Equations 2 and 4). Moreover, some of the constraints already existing in the linear program need to be updated. If a sequence σ_i is added to the set Σ_i' and the immediate prefix sequence (i.e., sequence $\sigma_i' \sqsubseteq \sigma_i$ such that $|\sigma_i'| + 1 = |\sigma_i|$) was already a part of the restricted game, then we need to update the constraint for information sets I_i for which $\sigma_i' = \mathsf{seq}_i(I_i)$ to ensure the consistency of the strategies (Equation 4), and the constraint corresponding to sequence σ_i' (Equation 2). In addition, the algorithm updates Equations (2) assigned to sequences of the opponent σ_{-i} for which $g(\sigma_i, \sigma_{-i}) \neq 0$. Finally, the algorithm updates all constraints that previously used utilities for temporary leaf nodes that are no longer leaf nodes in the restricted game after adding the new sequences.

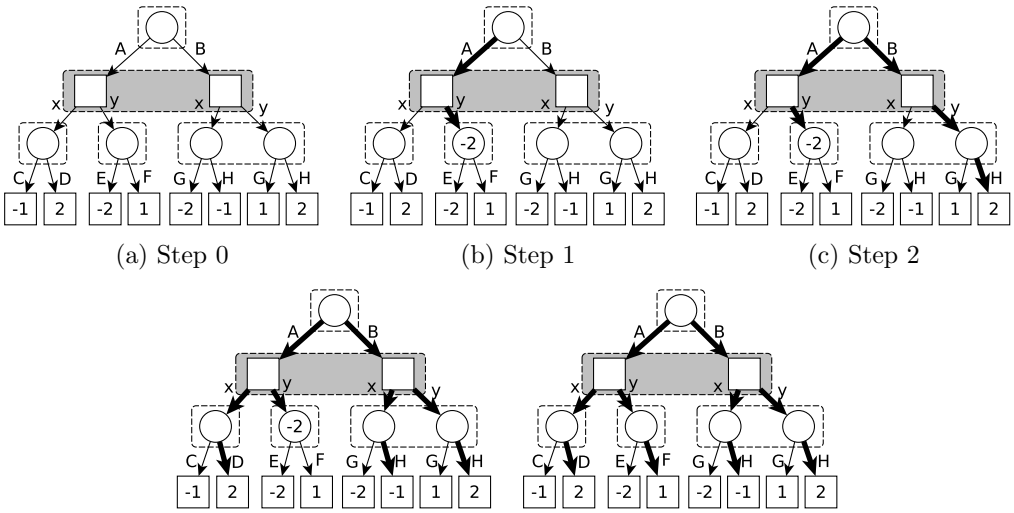
New sequences for each player are found using the best response sequence (BRS) algorithms described in Section 4.2. From the perspective of the sequence-form double-oracle algorithm, the BRS algorithm calculates a pure best response for player i against a fixed strategy of the opponent in the original unrestricted game. This pure best response specifies an action to play in each information set that is currently reachable given the opponent's extended strategy $\overline{r}^*_{-i}$. The best response can be formally defined as a pure realization plan r_i^{BR} that assigns only integer values 0 or 1 to the sequences. This realization plan is not necessarily a pure strategy in the original unrestricted game because there may not be an action specified for every information set. Specifically, there is no action specified for information sets that are not reachable (1) due to choices of player i, and (2) due to zero probability in the realization plan of the opponent $\overline{r}^*_{-i}$. Omitting these actions does not affect the value of the best response because these information sets are never reached; hence, for r_i^{BR} it holds that $\forall \overline{r}_i' \in \Delta_i \; u_i(r_i^{\mathrm{BR}}, \overline{r}^*_{-i}) \geq u_i(\overline{r}_i', \overline{r}^*_{-i})$ and there exists a pure best response strategy $\pi_i^{\mathrm{BR}} \in \Pi_i$ such that $u_i(r_i^{\mathrm{BR}}, \overline{r}^*_{-i}) = u_i(\pi_i^{\mathrm{BR}}, \overline{r}^*_{-i})$. The sequences that are used in the best-response pure realization plan with probability 1 are returned by BRS algorithm and we call these the *best-response sequences*:

$$\{\sigma_i \in \Sigma_i \; : \; r_i^{\mathrm{BR}}(\sigma_i) = 1\} \tag{12}$$

4.1.3 Example Run of the Algorithm

We now demonstrate the sequence-form double-oracle algorithm on an example game depicted in Figure 3a. In our example, there are two players: circle and box. Circle aims to maximize the utility value in the leafs, box aims to minimize the utility value. We assume that choosing the leftmost action in each information set is the default strategy for both players in this game.

The algorithm starts with an empty set of allowed sequences in the restricted game $\Phi' \leftarrow \emptyset$; hence, the algorithm sets the current pair of $(\overline{r}_i^*, \overline{r}^*_{-i})$ strategies to be equivalent to $(\pi_i^{\mathrm{DEF}}, \pi_{-i}^{\mathrm{DEF}})$. Next, the algorithm adds new sequences that correspond to the best response to the default strategy of the opponent; in our example the best response sequences for the circle player are $\{\emptyset, A, AD\}$, and $\{\emptyset, y\}$ for the box player. These sequences are added



(a) Step 0 (b) Step 1 (c) Step 2

(d) Step 3 (e) Step 4

Figure 3: Example of the steps of the sequence-form double-oracle algorithm in a two-player zero-sum game, where circle player aims to maximize the utility value, box aims to minimize the utility value. Bold edges correspond to the sequences of actions added into the restricted game. The dashed boxes indicate the information sets.

to the set of allowed sequences Φ'. Next, the set of sequences of the restricted game Σ' is updated. The maximal compatible set of sequences from set Φ' cannot contain sequence AD because the compatible sequence of the box player (i.e., x in this case) is not allowed in the restricted game yet and sequence AD cannot be fully executed. Moreover, by adding sequences A and y, the restricted game will contain node Ay for which actions E and F are defined in the original unrestricted game. However, there is no continuation in the current restricted game yet; hence, this node is a temporary leaf, belongs to $Z' \setminus Z$, and the algorithm needs to define a new value for a modified utility function u' for this node. The value $u'(Ay)$ is equal to -2 and corresponds to the outcome of the game if the circle player continues by playing the default strategy and the box player plays the best response. To complete the first step of the algorithm we summarize the nodes and information sets included in the restricted game; H' contains 3 nodes (the root, the node after playing an action A and the node Ay), and two information sets (the information set for node Ay is not added into the restricted game, because this node is now a leaf in the restricted game). Playing the sequences A and y with probability 1 is the Nash equilibrium of the restricted game. The situation is depicted in Figure 3b, the sequences in Σ' are shown as bold edges.

The algorithm proceeds further and the complete list of steps of the algorithm is summarized in Table 2. In the second iteration, new sequences B and BH are added into the restricted game. The box player does not add new sequences in this iteration because y is the best response to the extended equilibrium strategy of the circle player – i.e., playing sequences A, AC, AE with probability 1. NE in the updated restricted game changes to playing sequences B, BH and sequence y, all with probability 1. In the third iteration the situation changes and the box player adds sequence x, while there are no new sequences

added for the circle player. After adding sequence x, sequence AD also becomes a part of the set $\Sigma'_{\bigcirc}$ as it can now be fully executed due to adding the compatible sequence x. NE in the restricted game is now fully mixed, the sequences starting with A and with B are played in a ratio of $3:4$, x and y in a ratio of $4:3$. In the fourth iteration, the algorithm adds sequence AF to the restricted game (the best response for the circle player), which removes the assigned value $u'(Ay)$ since the node no longer belongs to set Z'. The algorithm stops after four iterations. No other sequences are added into the restricted game, the solution of the restricted game (r^*_i, r^*_{-i}) can be translated to the solution in the original unrestricted game, and $(\overline{r}^*_i, \overline{r}^*_{-i})$ is Nash equilibrium of the original game.

Iteration	$r^{\mathrm{BR}}_{\bigcirc}$	$r^{\mathrm{BR}}_{\square}$	$\Sigma'_{\bigcirc}$	$\Sigma'_{\square}$
1.	$\emptyset, A, AD$	$\emptyset, y$	$\emptyset, A$	$\emptyset, y$
2.	$\emptyset, B, BH$	$\emptyset, y$	$\emptyset, A, B, BH$	$\emptyset, y$
3.	$\emptyset, B, BH$	$\emptyset, x$	$\emptyset, A, AD, B, BH$	$\emptyset, y, x$
4.	$\emptyset, A, AF$	$\emptyset, y$	$\emptyset, A, AD, AF, B, BH$	$\emptyset, y, x$

Table 2: Steps of the sequence-form double-oracle algorithm applied to the example.

Consider now a small modification of the example game where there is a utility value of -3 in the leaf following action F (i.e., node AyF). In this case, the algorithm does not need to add sequence AF (nor AE) to the restricted game because it does not improve the value of the restricted game. Note that this modified example game shows why the algorithm needs to set the utility values for nodes in $Z' \setminus Z$. If the algorithm simply uses the unmodified utility function, then the node Ay will be treated as if it had zero utility value. This value overestimates the outcome of any actual continuation following this node in the original game for the circle player and since sequences AE or AF will never be a part of the best response for the circle player, the algorithm can converge to an incorrect solution.

4.2 Best-Response Sequence Algorithm

The purpose of the best-response sequence (BRS) algorithm is to generate new sequences that will be added to the restricted game in the next iteration, or to prove that there is no best response with better expected value that uses sequences currently not allowed in the restricted game. Throughout this section we use the term *searching player* to represent the player for whom the algorithm computes the best response sequences. We refer to this player as i.

The BRS algorithm calculates the expected value of a pure best response to the opponent's strategy $\overline{r}^*_{-i}$. The algorithm returns both the set of best-response sequences as well as the expected value of the strategy against the extended strategy of the opponent.

The algorithm is based on a depth-first search that traverses the original unrestricted game tree. The behavior of the opponent $-i$ is fixed to the strategy given by the extended realization plan $\overline{r}^*_{-i}$. To save computation time, the best-response algorithms use branch and bound during the search for best-response sequences. The algorithm uses a bound on the expected value for each inner node, denoted by λ. This bound represents the minimal utility value that the node currently being evaluated needs to gain in order to be a part

Require: i - searching player, h - current node, I_i^k - current information set, $\overline{r}'_{-i}$ - opponent's strategy, Min/MaxUtility - bounds on utility values, λ - lower bound for a node h

1: $w \leftarrow \overline{r}_{-i}(\mathsf{seq}_{-i}(h)) \cdot \mathcal{C}(h)$
2: **if** $h \in Z$ **then**
3: **return** $u_i(h) \cdot w$
4: **else if** $h \in Z' \setminus Z$ **then**
5: **return** $u'_i(h) \cdot w$
6: **end if**
7: sort $a \in A(h)$ based on probability $w_a \leftarrow \overline{r}'_{-i}\left(\mathsf{seq}_{-i}(ha)\right) \cdot \mathcal{C}(ha)$
8: $v^h \leftarrow 0$
9: **for** $a \in A(h),\ w_a > 0$ **do**
10: $\lambda' \leftarrow \lambda - \left[v^h + (w - w_a) \cdot \text{MaxUtility}\right]$
11: **if** $\lambda' \leq w_a \cdot \text{MaxUtility}$ **then**
12: $v' \leftarrow \text{BRS}_i(ha, \lambda')$
13: **if** $v' = -\infty$ **then**
14: **return** $-\infty$
15: **end if**
16: $v^h \leftarrow v^h + v'$
17: $w \leftarrow w - w_a$
18: **else**
19: **return** $-\infty$
20: **end if**
21: **end for**
22: **return** v^h

Figure 4: BRS_i in the nodes of other players.

of a best-response sequence. Using this bound during the search, the algorithm is able to prune branches that will certainly not be part of any best-response sequence. The bound λ is set to *MinUtility* for the root node.

We distinguish 2 cases in the search algorithm: either the algorithm is evaluating an information set (or more specifically a node h) assigned to the searching player i, or the node is assigned to one of the other players (either to the opponent, player $-i$, or it is a chance node). The pseudocode for these two cases is depicted in Figures 4 and 5.

4.2.1 Nodes of the Opponent

We first describe the case used when the algorithm evaluates node h assigned to either the opponent of the searching player or to Nature (see Figure 4). The main idea is to calculate the expected utility for this node according to the (fixed) strategy of the player. The strategy is known because it is either given by the extended realization plan $\overline{r}^*_{-i}$, or by the stochastic environment ($\mathcal{C}$). Throughout the algorithm, the variable w represents the probability of this node based on the realization probability of the opponent and stochastic environment (line 1). This value is iteratively decreased by values w_a that represent realization probabilities of the currently evaluated action $a \in A(h)$. Finally, v_h is the expected utility value for this node.

The algorithm evaluates actions in the descending order according to the probability of being played (based on $\overline{r}'_{-i}$ and $\mathcal{C}$; lines 9–21). First, we calculate a new lower bound

λ' for the successor ha (line 10). The new lower bound is the minimal value that must be returned from the recursive call $\text{BRS}_i(ha)$ under the optimistic assumption that all the remaining actions will yield the maximum possible utility. If the lower bound does not exceed the maximum possible utility in the game, the algorithm is executed recursively on the successors (line 12). Note that the algorithm does not evaluate branches with zero realization probability (line 9).

There are 3 possibilities for pruning in this part of the search algorithm. The first pruning is possible if the currently evaluated node is a leaf in the restricted game, but this node is an inner node in the original node (i.e., $h \in Z' \setminus Z$; line 5). The algorithm can directly use the value from the modified utility function u' in this case, since it is calculated as a best response of the searching player against the default strategy of the opponent that will be applied in the successors of node h since $h \in Z'$. Secondly, a cut-off also occurs if the new lower bound for a successor is larger than the maximum possible utility in the game, since this value can never be obtained in the successor (line 19). Finally, a cut-off occurs if there was a cut-off in one of the successors (line 14).

4.2.2 Nodes of the Searching Player

In nodes assigned to the searching player, the algorithm evaluates every action in each state that belongs to the current information set. The algorithm traverses the states in the descending order according to the probability of occurrence given the strategies of the opponent and Nature (line 8). Similar to the previous case, in each iteration the algorithm calculates a new lower bound for the successor node (line 17). The new lower bound λ' is the minimal value that must be returned from the recursive call $\text{BRS}_i(h'a)$ in order for the action a to be selected as the best action for this information set under the optimistic assumption that this action yields the maximum possible utility value after applying it in each of the remaining states in this information set. The algorithm performs a recursive call (line 20) only for an action that still could be the best in this information set (i.e., the lower bound does not exceed the maximal possible utility in the game). Note that if a cut-off occurs in one of the successors, the currently evaluated action a can no longer be the best action in this information set. Hence, v_a is set to $-\infty$ and action a will not be evaluated for any of the remaining nodes. When the algorithm determines which action will be selected as the best one in an information set, it evaluates only this action for all remaining nodes in the information set. Finally, the algorithm stores the values for the best action for all nodes in this information set (line 30). These are reused if the same information set is visited again (i.e., the algorithm reaches a different node h' from the same information set I_i; line 5).

A cut-off occurs in this part of the search algorithm if the maximal possible value v_a^h is smaller than the lower bound λ after evaluating node h. This means that regardless of which action will be selected as the best action in this information set, the lower bound for node h will not be reached; hence, the cut-off occurs (line 27). If a cut-off occurs in an information set, this information set cannot be reached again and the sequences of the searching player leading to this information set cannot be a part of the best response. This is due to propagating the cut-off to at least one previous information set of the searching player, otherwise there will be no tight lower bound set (the bound is first set only in the

Require: i - searching player, h - current node, I_i^k - current information set, r_{-i} - opponent's strategy, Min/MaxUtility - bounds on utility values, $\lambda-$ lower bound for a node h

if $h \in Z$ **then**
  return $u_i(h) \cdot \overline{r}'_{-i}(\mathsf{seq}_{-i}(h)) \cdot \mathcal{C}(h)$
end if
if v^h is already calculated **then**
  return v^h
end if
$H' \leftarrow \{h';\ h' \in I_i\}$
sort H' descending according to value $\overline{r}_{-i}(\mathsf{seq}_{-i}(h')) \cdot \mathcal{C}(h')$
$w \leftarrow \sum_{h' \in H'} \overline{r}_{-i}(\mathsf{seq}_{-i}(h')) \cdot \mathcal{C}(h')$
$v_a \leftarrow 0\ \ \forall a \in A(h)$; $maxAction \leftarrow \emptyset$
for $h' \in H'$ **do**
  $w_{h'} \leftarrow \overline{r}'_{-i}(\mathsf{seq}_{-i}(h')) \cdot \mathcal{C}(h')$
  for $a \in A(h')$ **do**
    if $maxAction$ is empty **then**
      $\lambda' \leftarrow w_{h'} \cdot$MinUtility
    else
      $\lambda' \leftarrow (v_{maxAction} + w \cdot \text{MinUtility}) - (v_a + (w - w_{h'}) \cdot \text{MaxUtility})$
    end if
    if $\lambda' \leq w_{h'} \cdot \text{MaxUtility}$ **then**
      $v_a^{h'} \leftarrow \text{BRS}_i(h'a, \lambda')$
      $v_a \leftarrow v_a + v_a^{h'}$
    end if
  end for
  $maxAction \leftarrow \arg\max_{a \in A(h')} v_a$
  $w \leftarrow w - w_{h'}$
  if h was evaluated $\wedge\ \left(\max_{a \in A(h)} v_a^h < \lambda\right)$ **then**
    return $-\infty$
  end if
end for
store $v_{maxAction}^{h'}$ as $v^{h'}\ \ \forall h' \in H'$
return $v_{maxAction}^h$

Figure 5: BRS_i in the nodes of the searching player.

information sets of the searching player). Therefore, there exists at least one action of the searching player that will never be evaluated again (after a cut-off, the value v_a for this action is set to $-\infty$) and cannot be selected as the best action in the information set. Since we assume perfect recall, all nodes in information set I_i share the same sequence of actions $\mathsf{seq}_i(I_i)$; hence, no node $h' \in I_i$ can be reached again.

4.3 Main Loop Alternatives

We now introduce several alternative formulations for the main loop of the sequence-form double-oracle algorithm. The general approach in the double-oracle algorithm is to solve the restricted game to find the equilibrium strategy for each player, compute the best responses in the original game for both of the players, and continue with the next iteration. However, the sequence-form LP is formulated in our double-oracle scheme in such a way that on each

iteration the algorithm can solve the restricted game only from the perspective of a single player i. In other words, we formulate a single LP as described in Section 3.3 that computes the optimal strategy of the opponent in the restricted game (player $-i$), and then compute the best response of player i to this strategy. This means that on each iteration we can select a specific player i, for whom we compute the best response in this iteration. We call this selection process the *player-selection policy*.

There are several alternatives for the player-selection policy that act as a domain-independent heuristics in double-oracle algorithm. We consider three possible policies: (1) the *standard double-oracle player-selection policy* of selecting both players on each iteration, (2) an *alternating policy*, where the algorithm selects only one player and switches between the players regularly (player i is selected in one iteration, player $-i$ is selected in the following iteration), and finally (3) a *worse-player-selection policy* that selects the player who currently has the worse bound on the solution quality. At the end of an iteration the algorithm selects the player i for whom the upper bound on utility value is further away from the current value of the restricted game. More formally,

$$\arg\max_{i \in N} \left| V_i^{UB} - V_i^{LP} \right| \tag{13}$$

where V_i^{LP} is the last calculated value of the restricted game for player i. The intuition behind this choice is that either this bound is precise and there are some missing sequences of this player in the restricted game that need to be added, or the upper bound is overestimated. In either case, the best-response sequence algorithm should be run for this player in the next iteration, either to add new sequences or to tighten the bound. In case of a tie, the alternating policy is applied in order to guarantee regular switching of the players. We experimentally compare these policies to show their impact on the overall performance of the sequence-form double-oracle algorithm (see Section 6).

5. Theoretical Results

In this section we prove that our sequence-form double-oracle algorithm will always converge to a Nash equilibrium of the original unrestricted game. First, we formally define the strategy computed by the best-response sequence (BRS) algorithm, then we prove lemmas about the characteristics of the BRS strategies, and finally we prove the main convergence result. Note that variations of the main loop described in Section 4.3 do not affect the correctness of the algorithm as long as the player-selection policy ensures that if no improvement is made by the BRS algorithm for one player that the BRS algorithm is run for the opponent on the next iteration.

Lemma 5.1 *Let r'_{-i} be a realization plan of player $-i$ in some restricted game G'. $BRS(r'_{-i})$ returns sequences corresponding to a realization plan r_i^{BR} in the unrestricted game, such that r_i^{BR} is part of a pure best response strategy to $\overline{r}'_{-i}$. The value returned by the algorithm is the value of executing the pair of strategies $u_i(\overline{r}'_{-i}, r_i^{BR})$.*

Proof $BRS(r'_{-i})$ searches the game tree and selects the action that maximizes the value of the game for player i in all information sets I_i assigned to player i reachable given the strategy of the opponent $\overline{r}'_{-i}$. In the opponent's nodes, it calculates the expected value

according to r'_{-i} where it is defined and the value according to the pure action of the default strategy π^{Def}_{-i} where r'_{-i} is not defined. In chance nodes, it returns the expected value of the node as the sum of the values of the successor nodes weighted by their probabilities. In each node h, if the successors have the maximal possible value for i then node h also has the maximal possible value for i (when playing against $\overline{r}'_{-i}$). The selections in the nodes that belong to i achieves this maximal value; hence, they form a best response to strategy $\overline{r}'_{-i}$. □

For brevity we use $v(BRS(r'_{-i}))$ to denote the value returned by the BRS algorithm, which is equal to $u_i(\overline{r}'_{-i}, r_i^{BR})$.

Lemma 5.2 *Let r'_{-i} be a realization plan of player $-i$ in some restricted game G' and let V_i^* be the value of the original unrestricted game G for player i, then*

$$v(BRS(r'_{-i})) \geq V_i^*. \tag{14}$$

Proof Lemma 5.1 showed that $v(BRS(r'_{-i}))$ is a value of the best response against $\overline{r}'_{-i}$ which is a valid strategy in the original unrestricted game G. If $v(BRS(r'_{-i})) < V_i^*$ then V_i^* cannot be the value of the game since player $-i$ has a strategy $\overline{r}'_{-i}$ that achieves better utility, which is a contradiction. □

Lemma 5.3 *Let r'_{-i} be a realization plan of player $-i$ that is returned by the LP for some restricted game G' and let V_i^{LP} be the value of the restricted game returned by the LP, then*

$$v(BRS(r'_{-i})) \geq V_i^{LP}. \tag{15}$$

Proof The realization plan r'_{-i} is part of the Nash equilibrium strategy in a zero-sum game that guarantees value V_i^{LP} in G'. If the best response computation in the original unrestricted game G selects only the actions from restricted game G', it creates the best response in game G' as well obtaining value V_i^{LP}. If the best response selects an action that is not allowed in the restricted game G', there are two cases.

Case 1: The best response strategy uses an action in a temporary leaf $h \in Z' \setminus Z$. Player i makes the decision in the leaf, because otherwise the value of the temporary leaf would be directly returned by BRS. The value of the temporary leaf has been under-estimated for player i in the restricted game by the modified utility function u' and it is over-estimated in the BRS computation as the best response to the default strategy π^{Def}_{-i}. The value of the best response can only increase by including this action.

Case 2: The best response strategy uses an action not allowed in G' in an internal node of the restricted game $H' \setminus Z'$. This can occur in nodes assigned to player i, because the actions of player $-i$ going out of G' have probability zero in $\overline{r}'_{-i}$. BRS takes the action with maximum value in the nodes assigned to player i, so the reason for selecting an action leading outside G' is that it has greater or equal value to the best action in G'. □

Lemma 5.4 *Under the assumptions of the previous lemma, if $v(BRS(r'_{-i})) > V_i^{LP}$ then it returns sequences that are added to the restricted game G' in the next iteration.*

Proof Based on the proof of the previous Lemma, BRS for player i can improve over the value of the LP (V_i^{LP}) only by selecting an action a that is not present in G' but is performed in a node h that is included in G' (in which i makes decision). Let (σ_i, σ_{-i}) be the pair of sequences leading to h. Then in the construction of the restricted game for the next iteration, sequence σ_{-i} is the sequence that ensures that $\sigma_i a$ can be executed in full and will be part of the new restricted game. □

Note, that Lemmas 5.2 and 5.4 would not hold if the utility values u' for temporary leaves ($h \in Z' \setminus Z$) are set arbitrarily. The algorithm sets the values in temporary leaf h as if the player $p(h)$ continues by playing the default strategy and the opponent ($-p(h)$) is playing the best response. If the utility values for the temporary leaves are set arbitrarily and used in the BRS algorithms to speed-up the calculation as proposed (see the algorithm in Figure 4, line 5), then Lemma 5.2 does not need to hold in cases where the value in node h strictly overestimates the optimal expected value for player $p(h)$. In this case, the best-response value of the opponent may be lower than the optimal outcome,

$$v\left(BRS(r_{p(h)})\right) < V^*_{-p(h)} \tag{16}$$

On the other hand, if the BRS algorithm does not use the temporary values u' for such a node, then Lemma 5.4 is violated because the best-response value will be strictly higher for player $-p(h)$ even though no new sequences are to be added into the restricted game.

Theorem 5.5 *The sequence-form double-oracle algorithm for extensive-form games described in the previous section terminates if and only if*

$$v(BRS(r'_{-i})) = -v(BRS(r'_i)) = V_i^{LP} = V_i^*, \tag{17}$$

which always happens after a finite number of iterations (because the game is finite), and strategies $(\overline{r}'_i, \overline{r}'_{-i})$ are a Nash equilibrium of the original unrestricted game.

Proof First we show that the algorithm continues until all equalities (17) hold. If $v(BRS(r'_{-i})) \neq -v(BRS(r'_i))$ then from Lemma 5.2 and Lemma 5.4 we know that for some player i it holds that $BRS(r'_{-i}) > V_i^{LP}$, so the restricted game in the following iteration is larger by at least one action and the algorithm continues. In the worst case, the restricted game equals the complete game $G' = G$, and it cannot be extended any further. In this case the BRS cannot find a better response then V_i^* and the algorithm stops due to Lemma 5.4.

If the condition in the theorem holds the algorithm has found a NE in the complete game, because from Lemma 5.1 we know that $r_{-i}^{BR} = BRS(r'_i)$ is the best response to $\overline{r}'_i$ in the complete game. However, if the value of the best response to a strategy in a zero-sum game is the value of the game, then the strategy $\overline{r}'_i$ is optimal and it is part of a Nash equilibrium of the game. □

6. Experiments

We now present our experimental evaluation of the performance of the sequence-form double-oracle algorithm for EFGs. We compare our algorithm against two state-of-the-art

baselines, the full sequence-form LP (referred to as FullLP from now on), and Counterfactual Regret Minimization (CFR). The first baseline is the standard exact method for solving sequence-form EFG, while CFR is one of the leading approximate algorithms applied to EFG. Our experimental results demonstrate the advantages of the double-oracle algorithm on three different classes of realistic EFGs. We also test the impact of the different variants of the main loop of the algorithm described in Section 4.3.

We compare three variants of the sequence-form double-oracle algorithm: (1) DO-b is a variant in which the best-responses are calculated for both players in each iteration; (2) DO-sa calculates the best-response for a single player on each iteration according to a simple alternating policy; and (3) DO-swp is a variant in which the best-response is calculated for a single player according to the worse-player selection policy. For all of the variants of the double-oracle algorithm we use the same default strategy where the first action applicable in a state is played by default.

Since there is no standardized collection of zero-sum extensive-form games for benchmark purposes, we use several specific games to evaluate the double-oracle algorithm and to identify the strengths and weaknesses of the algorithm. The games were selected to evaluate the performance under different conditions, so the games differ in the maximal utility the players can gain, in the causes of the imperfect information, and in the structure of the information sets. One of the key characteristics that affects the performance of the double-oracle algorithm is the relative size of the support of Nash equilibria (i.e., the number of sequences used in a NE with non-zero probability). If there does not exist a NE with small support, the algorithm must necessarily add a large fraction of the sequences into the restricted game to find a solution, mitigating the advantages of the double-oracle approach.

We present results for two types of games where the double-oracle significantly outperforms the FullLP on all instances: a search game motivated by border patrol and Phantom Tic-Tac-Toe. We also present results on a simplified version of poker for which the double-oracle algorithm does not always improve the computation time. However, the FullLP also has limited scalability due to larger memory requirements and cannot find solutions for larger variants of poker, while the double-oracle algorithm is able to solve these instances.

Our principal interest is in developing new generic methods for solving extensive-form games. Therefore, we implemented the algorithm in a generic framework for modeling arbitrary extensive-form games.[1] The algorithms do not use any domain-specific knowledge in the implementation, and do not rely on any specific ordering of the actions. The drawbacks of this generic implementation are higher memory requirements and additional overhead for the algorithms. A domain-specific implementation could improve the performance by eliminating some of the auxiliary data structures. We run all of the experiments using a single thread on an Intel i7 CPU running at 2.8 GHz. Each of the algorithms was given a maximum of 10 GB of memory for Java heap space. We used IBM CPLEX 12.5 for solving the linear programs, with parameter settings to use a single thread and the barrier solution algorithm.

In addition to runtimes, we analyze the speed of convergence of the double-oracle algorithms and compare it to one of the state-of-the-art approximative algorithms, Counterfactual Regret Minimization (CFR). We implemented CFR in a domain independent way

1. Source code is available at the home pages of the authors.

based on the pseudocode in the work of Lanctot (2013, p. 22). In principle, it is sufficient for CFR to maintain only a set of information sets and apply the no-regret learning rule in each information set. However, maintaining and traversing such a set effectively in a domain independent manner could be affected by our implementation of generic extensive-form games data structures (i.e., generating applicable actions in the states of the game, applying the actions, etc.). Therefore we use an implementation where CFR traverses the complete game tree that is held in memory to maintain the fairness of the comparison, and to guarantee the maximal possible speed of convergence of the CFR algorithm. The time necessary to build the game tree is *not* included in the computation time of CFR.

6.1 Test Domains

Search Games Our first test belongs to the class of search (or pursuit-evasion) games, often used in experimental evaluation of double-oracle algorithms (McMahan et al., 2003; Halvorson et al., 2009). The search game has two players: the *patroller* (or the defender) and the *evader* (or the attacker). The game is played on a directed graph (see Figure 6), where the evader aims to cross safely from a starting node (E) to a destination node (D). The defender controls two units that move in the intermediate nodes (the shaded areas) trying to capture the evader by occupying the same node as the evader. During each turn both players move their units simultaneously from the current node to an adjacent node, or the units stay in the same location. The only exception is that the evader cannot stay in the two leftmost nodes. If a pre-determined number of turns is made without either player winning, the game is a draw. This is an example of a win-tie-loss game and the utility values are from the set $\{-1, 0, 1\}$.

Players are unaware of the location and the actions of the other player with one exception – the evader leaves tracks in the visited nodes that can be discovered if the defender visits the nodes later. The game also includes an option for the evader to avoid leaving the tracks using a special move (a *slow move*) that requires two turns to simulate the evader covering the tracks.

Figure 6 shows examples of the graphs used in the experiments. The patrolling units can move only in the shaded areas (P1,P2), and they start at any node in the shaded areas. Even though the graph is small, the concurrent movement of all units implies a large branching factor (up to ≈ 50 for one turn) and thus large game trees (up to $\approx 10^{11}$ nodes). In the experiments we used three different graphs, varied the maximum number of turns of the game (from 3 to 7), and we altered the ability of the attacker to perform the slow moves (labeled *SA* if the slow moves are allowed, *SD* otherwise).

Phantom Tic-Tac-Toe The second game is a blind variant of the well-known game of Tic-Tac-Toe (e.g., used in Lanctot et al., 2012). The game is played on a 3×3 board, where two players (cross and circle) attempt to place 3 identical marks in a horizontal, vertical, or diagonal row to win the game. In the blind variant, the players are unable to observe the opponent's moves and each player only knows that the opponent made a move and it is her turn. Moreover, if a player tries to place her mark on a square that is already occupied by an opponent's mark, the player learns this information and can place the mark in some other square. Again, the utility values of this game are from the set $\{-1, 0, 1\}$.

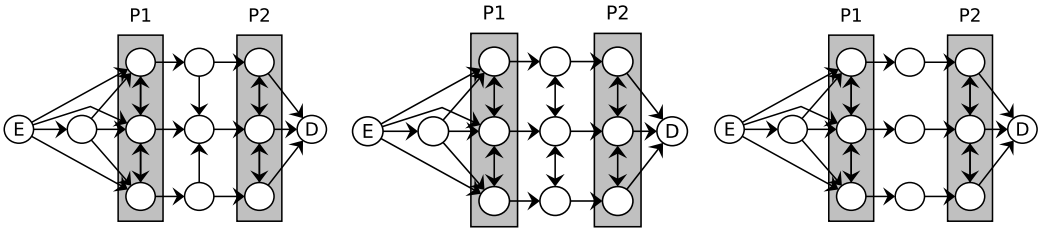


Figure 6: Three variants of the graph used in the experiments on the search game; we refer to them as G1 (left), G2 (middle), and G3 (right).

The uncertainty in phantom Tic-Tac-Toe makes the game large ($\approx 10^9$ nodes). In addition, since one player can try several squares before her move is successful, the players do not necessarily alternate in making their moves. This rule makes the structure of the information sets rather complicated and since the opponent never learns how many attempts the first player actually performed, a single information set can contain nodes at different depths in the game tree.

Poker Games Poker is frequently studied in the literature as an example of a large extensive-form game with imperfect information. We include experiments with a simplified two-player poker game inspired by Leduc Hold'em.

In our version of poker, each player starts with the same amount of chips and both players are required to put some number of chips in the pot (called the *ante*). In the next step, the Nature player deals a single card to each player (the opponent is unaware of the card) and the betting round begins. A player can either *fold* (the opponent wins the pot), *check* (let the opponent make the next move), *bet* (being the first to add some amount of chips to the pot), *call* (add the amount of chips equal to the last bet of the opponent into the pot), or *raise* (match and increase the bet of the opponent). If no further raise is made by any of the players, the betting round ends, the Nature player deals one card on the table, and the second betting round with the same rules begins. After the second betting round ends, the outcome of the game is determined – a player wins if: (1) her private card matches the table card and the opponent's card does not match, (2) none of the players' cards matches the table card and her private card is higher than the private card of the opponent, or (3) the opponent folds. The utility value is the amount of chips the player has won or lost. If no player wins, the game is a draw and the pot is split.

In the experiments we alter the number of types of the cards (from 3 to 4; there are 3 types of cards in Leduc), the number of cards of each type (from 2 to 3; set to 2 in Leduc), the maximum length of sequence of raises in a betting round (ranging from 1 to 4; set to 1 in Leduc), and the number of different sizes of bets (i.e., amount of chips added to the pot) for *bet*/*raise* actions (ranging from 1 to 4; set to 1 in Leduc).

6.2 Results

Search Games The results for the search game scenarios show that the sequence-form double-oracle algorithm is particularly successful when applied to games where NEs with small support exist. Figure 7 shows a comparison of the running times for FullLP and variants of the double-oracle algorithm (note the logarithmic y-scale). All variants of the

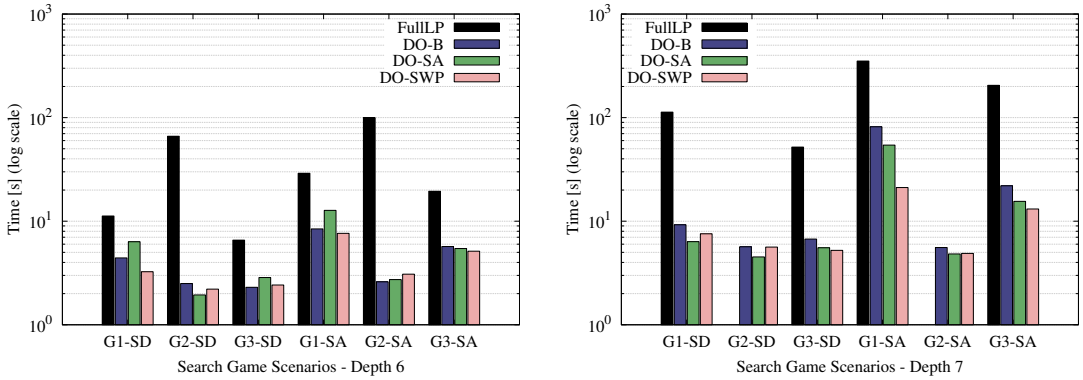


Figure 7: Comparison of the running times on 3 different graphs with either slow moves allowed (SA) or disallowed (SD), the depth is set to 6 (left subfigure) or 7 (right subfigure). Missing values for the FullLP algorithm indicate that the algorithm runs out of memory.

double-oracle algorithm are several orders of magnitude faster than FullLP. This is most apparent on the fully-connected graph (G2) that generates the largest game tree. When slow moves are allowed and the depth is set to 6, it takes almost 100 seconds for FullLP to solve the instance of the game but all variants of the double-oracle algorithms solve the game in less than 3 seconds. Moreover, when the depth is increased to 7, FullLP was unable to solve the game due to the memory constraints, while the fastest variant DO-swp solved the game in less than 5 seconds. Similar results were obtained for the other graphs.

The graph G1 induced a game that was the most difficult for the double-oracle algorithm: when the depth is set to 7, it takes almost 6 minutes for FullLP to solve the instance, while the fastest variant DO-swp solved the game in 21 seconds. The reason is that even though the game tree is not the largest, there is a more complex structure of the information sets. This is due to limited compatibility among the sequences of the players; when the patrolling unit P1 observes the tracks in the top-row node, the second patrolling unit P2 can capture the evader only in the top-row node, or in the middle-row node.

Comparing the different variants of the sequence-form double-oracle algorithm does not show consistent results. There is no variant consistently better in this game since all the double-oracle variants are typically able to compute a Nash equilibrium very quickly. However, DO-swp is often the fastest and for some settings the difference is quite significant. The speed-up this variant offers is most apparent on the G1 graph. On average through all instances of the search game, DO-sa uses 92.59% of the computation time of DO-b, and DO-swp uses 88.25%.

Table 3 shows a breakdown of the cumulative computation time spent in different components of the double-oracle algorithm: solving the restricted game (LP), calculating best responses (BR), and creating a valid restricted game after selecting new sequences to add (Validity). The results show that due to the size of the game, the computation of the best-response sequences takes the majority of the time (typically around 75% on larger instances), while creating the restricted game and solving it takes only a small fraction of the total time. It is also noticeable that the size of the final restricted game is very small

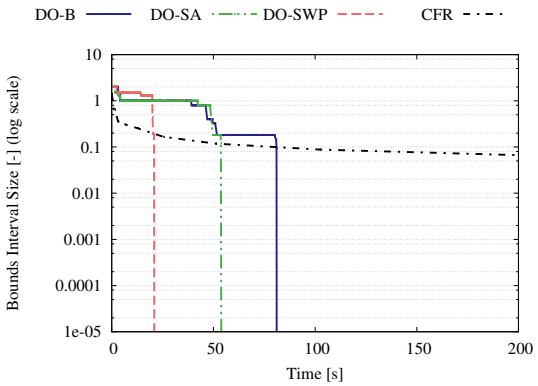


Figure 8: Convergence of variants of the double-oracle algorithm and CFR on the search game domain: y-axis displays the current approximation error.

Algorithm	Overall [s]	LP [s]	BR [s]	Validity [s]	Iterations	$\lvert\Sigma'_1\rvert(\frac{\lvert\Sigma'_1\rvert}{\lvert\Sigma_1\rvert})$	$\lvert\Sigma'_2\rvert(\frac{\lvert\Sigma'_2\rvert}{\lvert\Sigma_2\rvert})$
FullLP	351.98	–	–	–	–	–	–
DO-B	81.51	6.97	63.39	10.58	187	252 (17.22%)	711 (0.26%)
DO-SA	54.32	5.5	39.11	9.09	344	264 (18.05%)	649 (0.24%)
DO-SWP	21.15	1.93	16.28	2.47	209	193 (13.19%)	692 (0.25%)

Table 3: Cumulative running times for different components of the double-oracle algorithm, iterations, and size of the restricted game in terms of the number of sequences compared to the size of the complete game. The results are shown for scenario G1, depth 7, and allowed slow moves.

compared to the original game, since the number of sequences for the second player (the defender) is less than 1% (there are 273,099 sequences for the defender).

Finally, we analyze the convergence rate of the variants of the double-oracle algorithm. The results are depicted in Figure 8, where the size of the interval given by the bounds V_i^{UB} and V_i^{LB} defines the current error of the double-oracle algorithm as $|V_i^{UB} - V_i^{LB}|$. The convergence rate of the CFR algorithm is also depicted. The error of CFR is calculated in the same way, as a sum of the best-response values to the current mean strategies from the CFR algorithm. We can see that all variants of the double-oracle algorithm perform similarly – the error drops very quickly to 1 and a few iterations later each version of the algorithm quickly converges to an exact solution. These results show that in this game the double-oracle algorithm can very quickly find the correct sequences of actions and compute an exact solution, in spite of the size of the game. In comparison, the CFR algorithm can also quickly learn the correct strategies in most of the information sets, but the convergence has a very long tail. After 200 seconds, the error of CFR is equal to 0.0657 and it is dropping very slowly (0.0158 after 1 hour). The error of CFR is quite significant considering the value of the game in this case ($-0.\overline{3333}$).

Phantom Tic-Tac-Toe The results on Phantom Tic-Tac-Toe confirm that this game is also suitable for the sequence-form double-oracle algorithm. Due to the size of the game, both baseline algorithms (the FullLP and CFR) ran out of memory and were not able

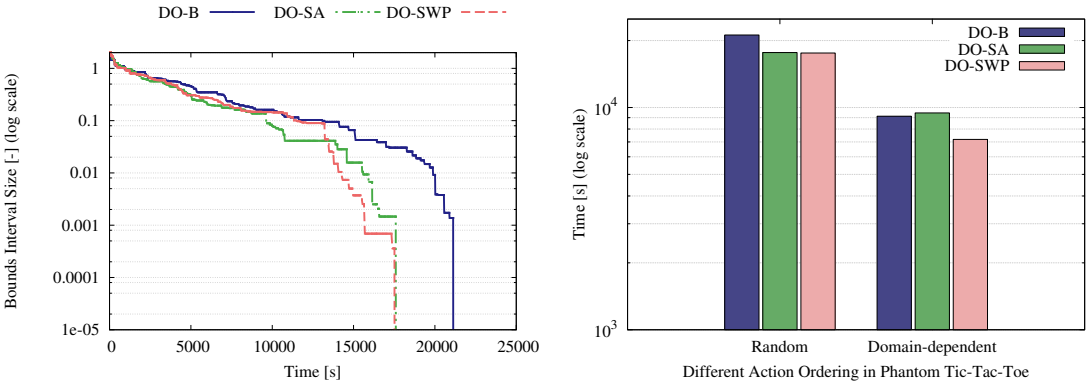


Figure 9: (left) Comparison of the convergence rate of the double-oracle variants for Phantom Tic-Tac-Toe; (right) Comparison of the performance of the double-oracle variants for Phantom Tic-Tac-Toe when domain-specific move ordering and default strategy is used.

Algorithm	Overall [s]	LP [s]	BR [s]	Validity [s]	Iterations	$\lvert\Sigma_1'\rvert(\frac{\lvert\Sigma_1'\rvert}{\lvert\Sigma_1\rvert})$	$\lvert\Sigma_2'\rvert(\frac{\lvert\Sigma_2'\rvert}{\lvert\Sigma_2\rvert})$
FullLP	*N/A*	–	–	–	–	–	–
DO-B	21,197	2,635	17,562	999	335	7,676 (0.60%)	10,095 (0.23%)
DO-SA	17,667	2,206	14,560	900	671	7,518 (0.59%)	9,648 (0.22%)
DO-SWP	17,589	2,143	14,582	864	591	8,242 (0.65%)	8,832 (0.20%)

Table 4: Cumulative running times for different components of the double-oracle algorithm for the game of Phantom Tic-Tac-Toe.

to solve the game. Therefore, we only compare the times for different variants of the double-oracle algorithm. Figure 9 (left subfigure) shows the overall performance of all three variants of the double-oracle algorithm in the form of a convergence graph. We see that the performance of two of the variants is similar, with the performance of DO-SA and DO-SWP almost identical. On the other hand, the results show that DO-B converges significantly slower.

The time breakdown of the variants of the double-oracle algorithm is shown in Table 4. Similarly to the previous case, the majority of the time ($\approx$ 83%) is spent in calculating the best responses. Out of all variants of the double-oracle algorithm, the DO-SWP variant is the fastest one. It converged in significantly fewer iterations compared to the DO-SA variant (iterations are twice as expensive in the DO-B variant).

We now present the results that demonstrate the potential of combining the sequence-form double-oracle algorithm with domain-specific knowledge. Every variant of the double-oracle algorithm can use a move ordering based on domain-specific heuristics. The move ordering determines the default strategy (recall that our algorithm uses the first action as the default strategy for each player), and the direction of the search in the best response algorithms. By replacing the randomly generated move ordering with a heuristic one that chooses better actions first, the results show a significant improvement in the performance of all of the variants (see Figure 9, right subfigure), even though there are no changes to the rest of the algorithm. Each variant was able to solve the game in less than 3 hours, and it took 2 hours for the fastest DO-SWP variant.

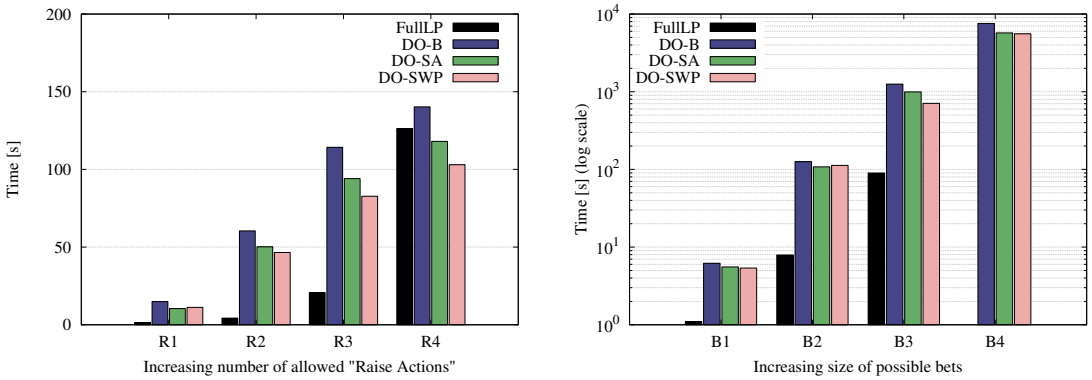


Figure 10: Comparison of the running times on different variants of the simplified poker game. The left subfigure shows the computation times with an increasing number of raise actions allowed, the right subfigure shows the computation times with an increasing number of different bet sizes for raise/bet actions.

Poker Games Poker represents a game where the double-oracle algorithms do not perform as well and the sequence-form LP is often faster on smaller instances. One significant difference compared to the previous games is that the size of the NE support is larger (around 5% of sequences for larger instances). Secondly, the game trees of poker games are relatively shallow and the only imperfect information in the game is due to Nature. As a result, the double-oracle algorithms require a larger number of iterations to add more sequences into the restricted game (up to 10% of all sequences for a player are added even for the largest poker scenarios) in order to find the exact solution. However, with increasing depth and/or branching factor, the size of the game grows exponentially and FullLP is not able to solve the largest instances due to the memory constraints.

Figure 10 shows the selected results for simplified poker variants. The results in the left subfigure show the computation times with increasing depth of the game by allowing the players to re-raise (players are allowed to re-raise their opponent a certain number of times). The remaining parameters are fixed to 3 types of cards, 2 cards of each type, and 2 different betting sizes. The size of the game grows exponentially, with the number of possible sequences increasing to 210,937 for each player for the $R4$ scenario. The computation time for FullLP is directly related to the size of the tree and increases exponentially with the increasing depth (note that there is a standard y scale). On the other hand, the increase is less dramatic for all of the variants of the double-oracle algorithm. The DO-swp variant is the fastest for the largest scenario – while FullLP solved this instance in 126 seconds, it took only 103 seconds for DO-swp. Finally, FullLP is not able to solve the games if we increase the length to $R5$ due to memory constraints, while the computation time of all of the double-oracle algorithms increases only marginally.

The right subfigure of Figure 10 shows the increase in computation time with an increasing number of different bet sizes for raise/bet actions. The remaining parameters were fixed to 4 types of cards, 3 cards of each type, and 2 raise actions allowed. Again, the game grows exponentially with the increasing branching factor. The number of sequences increases up to 685,125 for each player for the $B4$ scenario, and the computation time of

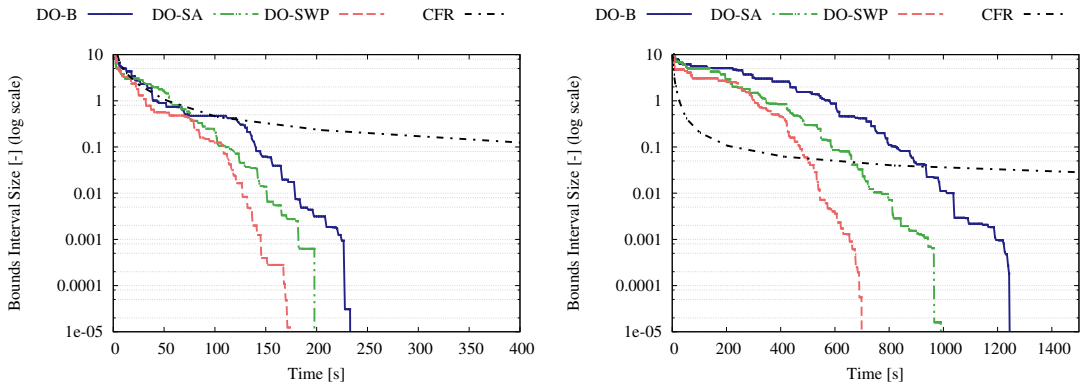


Figure 11: Comparison of the convergence of the variants of the double-oracle algorithm and CFR for two variants of the simplified poker with 4 types of cards, and 3 cards of each type. There are 4 raise actions allowed, 2 different bet sizes in the left subfigure; there are 2 raise actions allowed, 3 different bet sizes in the right subfigure.

Algorithm	Overall [s]	LP [s]	BR [s]	Validity [s]	Iterations	$\lvert\Sigma'_1\rvert(\frac{\lvert\Sigma'_1\rvert}{\lvert\Sigma_1\rvert})$	$\lvert\Sigma'_2\rvert(\frac{\lvert\Sigma'_2\rvert}{\lvert\Sigma_2\rvert})$
FullLP	278.18	–	–	–	–	–	–
DO-B	234.60	149.32	56.04	28.61	152	6,799 (1.81%)	6,854 (1.83%)
DO-SA	199.24	117.71	51.25	29.59	289	6,762 (1.80%)	6,673 (1.78%)
DO-SWP	182.68	108.95	48.25	24.8	267	6,572 (1.75%)	6,599 (1.76%)

Table 5: Cumulative running times for different components of the double-oracle algorithm, iterations, and sizes of the restricted game in terms of the number of sequences compared to the size of the complete game. The results are shown for poker scenario with 4 raise actions allowed, 2 different betting values, 4 types of cards, and 3 cards of each type.

all algorithms increases exponentially as well (note logarithmic y scale). The results show that even with the increasing branching factor, the double-oracle variants tend to be slower than solving the FullLP. However, while the FullLP ran out of memory for the largest $B4$ setting, all of the double-oracle variants were able to find the exact solution using less memory.

Comparing the different variants of the double-oracle algorithm using the convergence graph (see Figure 11) and the decomposition of the computation times (see Table 5) shows that DO-SWP is the fastest variant in the selected scenario (and in nearly all of poker scenarios). Decomposition of the overall time shows that the majority of the computation time is spent in solving the restricted game LP (up to 65%). The decomposition also shows that DO-SWP is typically faster due to the lower number of iterations. In addition, the final size of the restricted game is typically the smallest for this variant. On average over all instances of the poker games, DO-SA uses 86.57% of the computation time of DO-B, and DO-SWP uses 82.3% of the computation time.

Convergence in poker games is slower compared to search games of similar size (note the logarithmic scale in Figure 11). Comparing the double-oracle algorithm variants with CFR shows an interesting result in the left subfigure. Due to the size of the game, the speed of the CFR convergence is nearly the same as for the double-oracle algorithms during the first

iterations. However, while the double-oracle algorithms continue to converge at roughly the same rate and are able to find an exact solution, the error of the CFR algorithm decreases very slowly. In the scenario depicted in the left subfigure, the CFR algorithm converged to an error of 0.1212 (the value of the game in this case is ≈ -0.09963) after 400 seconds. After 1 hour, the error dropped to 0.0268. For scenarios with more shallow game trees and larger branching factor, the convergence of CFR is faster at the beginning compared to the double-oracle algorithms (right subfigure of Figure 11). However, the main disadvantage of CFR having a long tail for convergence is still the case and the error after 1600 seconds is still over 0.0266 (the value of this game is ≈ -0.09828).

6.3 Discussion of the Results

The experimental results support several conclusions. The results demonstrate that the sequence-form double-oracle algorithm is able to compute an exact solution for much larger games compared to the state-of-the-art exact algorithm based on the sequence-form linear program. Moreover, we have experimentally shown that there are realistic games where only a small fraction of sequences are necessary to find a solution of the game. In these cases, the double-oracle algorithms also significantly speed up the computation time. Our results indicate that the DO-SWP variant is typically the fastest, but not in all cases. By selecting the player that currently has the worse bound on performance, the DO-SWP version can add more important sequences, or prove that there are not any better sequences and adjust the upper bound on the value faster.

Comparing the speed of convergence of the double-oracle algorithms with the state-of-the-art approximative algorithm CFR showed that CFR quickly approximates the solution during the first iterations. However, the convergence of CFR has a very long tail and CFR is not able to find an exact solution for larger games in a reasonable time. Another interesting observation is that for some games the convergence rate of the double-oracle algorithms and CFR is similar in the first iterations, and while the double-oracle algorithms continue at this rate and find an exact solution, the long tail convergence remains for CFR. This is despite the fact that our implementation of CFR has an advantage of having the complete game tree including the states for all histories in memory.

Unfortunately, it is difficult to characterize the exact properties of the games for which the double-oracle algorithms perform better in terms of computation time compared to the other algorithms. Certainly, the double-oracle algorithm is not suitable for games were the only equilibria have large support due to the necessity of large number of iterations. However, having a small support equilibrium is not a sufficient condition. This is apparent due to two graphs shown in the poker experiments, where either the depth of the game tree or the branching factor was increased. Even though the game grows exponentially and the size of the support decreases to $\approx 2.5\%$ in both cases, the behavior of the double-oracle algorithms is quite different. Our conjecture is that games with longer sequences suit the double-oracle algorithms better, since several actions that form the best-response sequences can be added during a single iteration. This contrasts with shallow game trees with large branching factors, where more iterations are necessary to add multiple actions. However, a deeper analysis to identify the exact properties of the games that are suitable is an open question that must be analyzed for normal-form games first.

7. Conclusion

We present a novel exact algorithm for solving two player zero-sum extensive-form games with imperfect information. Our approach combines the compact sequence-form representation for extensive-form games with the iterative algorithmic framework of double-oracle methods. This integrates two successful approaches for solving large scale games that have not yet been brought together for the general class of games that our algorithm addresses. The main idea of our algorithm is to restrict the game by allowing players to play only a restricted set of sequences from the available sequences of actions, and to iteratively expand the restricted game over time using fast best-response algorithms. Although in the worst case the double-oracle algorithm may need to add all possible sequences, the experimental results on different domains prove that the double-oracle algorithm can find an exact Nash equilibrium prior to constructing the full linear program for the complete game. Therefore, the sequence-form double-oracle algorithm reduces the main limitation of the sequence-form linear program—memory requirements—and it is able to solve much larger games compared to state-of-the-art methods. Moreover, since our algorithm is able to identify the sequences of promising actions without any domain-specific knowledge, it can also provide a significant runtime improvements.

The proposed algorithm also has another crucial advantage compared to the current state of the art. The double-oracle framework offers a decomposition of the problem of computing a Nash equilibrium into separate sub-problems, including the best-response algorithms, the choice of the default strategy, and the algorithms for constructing and solving the restricted game. We developed solutions for all of these sub-problems in a domain-independent manner. However, we can also view our algorithm as a more general framework that can be specialized with domain-specific components that take advantage of the structure of specific problems to improve the performance of these sub-problems. This can lead to substantial improvements in the speed of the algorithm, the number of iterations, as well as reducing the final size of the restricted game. We demonstrated the potential of the domain-specific approach on the game of Phantom Tic-Tac-Toe. Another example is that fast best-response algorithms that operate on the public tree (i.e., a compact representation of games with publicly observable actions; see Johanson, Bowling, Waugh, & Zinkevich, 2011) can be exploited for games like poker. Finally, our formal analysis identifies the key properties that these domain-specific implementations need to satisfy to guarantee the convergence to the correct solution of the game.

Our algorithm opens up a large number of directions for future work. It represents a new class of methods for solving extensive-form games with imperfect information that operates very differently than other common approaches (e.g., counterfactual regret minimization), and many possible alternatives to improve the performance of the algorithm remain to be investigated. Examples include more sophisticated calculation of utility values for the temporary leaves, alternative strategies for expanding the restricted game, and removing unused sequences from the restricted game. A broader analysis of using the sequence-form double-oracle algorithm as an approximation technique should be performed, possibly by exploring alternative approximative best-response algorithms based on sampling (e.g., Monte Carlo) techniques.

There are also several theoretical questions that could be investigated. First, the performance of the double-oracle algorithm depends strongly on the number of iterations and sequences that need to be added. However, the theoretical question regarding the expected number of iterations and thus the speed of the convergence of the double-oracle algorithm have not been explored even for simpler game models (e.g., games in the normal form). An analysis of these simpler models is needed to identify the general properties of games where the double-oracle methods tend to be faster and to identify the optimal way of expanding the restricted game.

Acknowledgements

Earlier versions of this paper were published at the European Conference on Artificial Intelligence (ECAI) (Bosansky, Kiekintveld, Lisy, & Pechoucek, 2012) and the conference on Autonomous Agents and Multi Agent Systems (AAMAS) (Bosansky, Kiekintveld, Lisy, Cermak, & Pechoucek, 2013). The major additions to this full version include (1) a novel, more detailed description of all parts of the algorithm, (2) introduction and analysis of different policies for the player selection in the main loop of the double-oracle algorithm, (3) new experiments on the phantom tic-tac-toe domain together with a more thorough analysis of the experimental results on all domains, including the analysis of the convergence of the algorithm, (4) experimental comparison with CFR, and finally (5) extended analysis of related work.

This research was supported by the Czech Science Foundation (grant no. P202/12/2054) and by U.S. Army Research Office (award no. W911NF-13-1-0467).

References

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., & Vance, P. H. (1998). Branch-And-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, *46*, 316–329.

Bosansky, B., Kiekintveld, C., Lisy, V., Cermak, J., & Pechoucek, M. (2013). Double-oracle Algorithm for Computing an Exact Nash Equilibrium in Zero-sum Extensive-form Games. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 335–342.

Bosansky, B., Kiekintveld, C., Lisy, V., & Pechoucek, M. (2012). Iterative Algorithm for Solving Two-player Zero-sum Extensive-form Games with Imperfect Information. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*, pp. 193–198.

Cermak, J., Bosansky, B., & Lisy, V. (2014). Practical Performance of Refinements of Nash Equilibria in Extensive-Form Zero-Sum Games. In *Proceedings of European Conference on Artificial Intelligence (ECAI)*, pp. 201–206.

Dantzig, G., & Wolfe, P. (1960). Decomposition Principle for Linear Programs. *Operations Research*, *8*, 101–111.

Ganzfried, S., & Sandholm, T. (2013). Improving Performance in Imperfect-Information Games with Large State and Action Spaces by Solving Endgames. In *Computer*

Poker and Imperfect Information Workshop at the National Conference on Artificial Intelligence (AAAI).

Gibson, R., Lanctot, M., Burch, N., Szafron, D., & Bowling, M. (2012). Generalized Sampling and Variance in Counterfactual Regret Minimization. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, pp. 1355–1361.

Halvorson, E., Conitzer, V., & Parr, R. (2009). Multi-step Multi-sensor Hider-seeker Games. In *Proceedings of the Joint International Conference on Artificial Intelligence (IJCAI)*, pp. 159–166.

Hoda, S., Gilpin, A., Peña, J., & Sandholm, T. (2010). Smoothing Techniques for Computing Nash Equilibria of Sequential Games. *Mathematics of Operations Research*, *35*(2), 494–512.

Jain, M., Conitzer, V., & Tambe, M. (2013). Security Scheduling for Real-world Networks. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 215–222.

Jain, M., Korzhyk, D., Vanek, O., Conitzer, V., Tambe, M., & Pechoucek, M. (2011). Double Oracle Algorithm for Zero-Sum Security Games on Graph. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 327–334.

Johanson, M., Bowling, M., Waugh, K., & Zinkevich, M. (2011). Accelerating Best Response Calculation in Large Extensive Games. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 258–265.

Koller, D., & Megiddo, N. (1992). The Complexity of Two-Person Zero-Sum Games in Extensive Form. *Games and Economic Behavior*, *4*, 528–552.

Koller, D., Megiddo, N., & von Stengel, B. (1996). Efficient Computation of Equilibria for Extensive Two-Person Games. *Games and Economic Behavior*, *14*(2), 247–259.

Koller, D., & Megiddo, N. (1996). Finding Mixed Strategies with Small Supports in Extensive Form Games. *International Journal of Game Theory*, *25*, 73–92.

Kreps, D. M., & Wilson, R. (1982). Sequential Equilibria. *Econometrica*, *50*(4), 863–94.

Lanctot, M. (2013). *Monte Carlo Sampling and Regret Minimization for Equilibrium Computation and Decision Making in Large Extensive-Form Games*. Ph.D. thesis, University of Alberta.

Lanctot, M., Gibson, R., Burch, N., Zinkevich, M., & Bowling, M. (2012). No-Regret Learning in Extensive-Form Games with Imperfect Recall. In *Proceedings of the 29th International Conference on Machine Learning (ICML 2012)*, pp. 1–21.

Lanctot, M., Waugh, K., Zinkevich, M., & Bowling, M. (2009). Monte Carlo Sampling for Regret Minimization in Extensive Games. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1078–1086.

Lee, C.-S., Wang, M.-H., Chaslot, G., Hoock, J.-B., Rimmel, A., Teytaud, O., Tsai, S.-R., Hsu, S.-C., & Hong, T.-P. (2009). The Computational Intelligence of Mogo Revealed in Taiwans Computer Go Tournaments. *IEEE Transactions on Computational Intelligence and AI in Games*, *1*, 73–89.

Letchford, J., & Vorobeychik, Y. (2013). Optimal Interdiction of Attack Plans. In *Proceedings of the 12th International Conference on Automonous Agents and Multiagent Systems (AAMAS)*, pp. 199–206.

Lisy, V., Kovarik, V., Lanctot, M., & Bosansky, B. (2013). Convergence of Monte Carlo Tree Search in Simultaneous Move Games. In *Advances in Neural Information Processing Systems (NIPS)*, Vol. 26, pp. 2112–2120.

McMahan, H. B. (2006). *Robust Planning in Domains with Stochastic Outcomes, Adversaries, and Partial Observability*. Ph.D. thesis, Carnegie Mellon University.

McMahan, H. B., & Gordon, G. J. (2007). A Fast Bundle-based Anytime Algorithm for Poker and other Convex Games. *Journal of Machine Learning Research - Proceedings Track*, *2*, 323–330.

McMahan, H. B., Gordon, G. J., & Blum, A. (2003). Planning in the Presence of Cost Functions Controlled by an Adversary. In *Proceedings of the International Conference on Machine Learning*, pp. 536–543.

Miltersen, P. B., & Sørensen, T. B. (2008). Fast Algorithms for Finding Proper Strategies in Game Trees. In *Proceedings of Symposium on Discrete Algorithms (SODA)*, pp. 874–883.

Miltersen, P. B., & Sørensen, T. B. (2010). Computing a Quasi-Perfect Equilibrium of a Two-Player Game. *Economic Theory*, *42*(1), 175–192.

Pita, J., Jain, M., Western, C., Portway, C., Tambe, M., Ordonez, F., Kraus, S., & Parachuri, P. (2008). Deployed ARMOR protection: The Application of a Game-Theoretic Model for Security at the Los Angeles International Airport. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 125–132.

Ponsen, M. J. V., de Jong, S., & Lanctot, M. (2011). Computing Approximate Nash Equilibria and Robust Best-Responses Using Sampling. *Journal of Artificial Intelligence Research (JAIR)*, *42*, 575–605.

Sandholm, T. (2010). The State of Solving Large Incomplete-Information Games, and Application to Poker. *AI Magazine*, *special issue on Algorithmic Game Theory*, 13–32.

Selten, R. (1975). Reexamination of the Perfectness Concept for Equilibrium Points in Extensive Games. *International Journal of Game Theory*, *4*, 25–55.

Selten, R. (1965). Spieltheoretische Behandlung eines Oligopolmodells mit Nachfragetrgheit [An oligopoly model with demand inertia]. *Zeitschrift für die Gesamte Staatswissenschaft*, *121*, 301–324.

Shafiei, M., Sturtevant, N., & Schaeffer, J. (2009). Comparing UCT versus CFR in Simultaneous Games. In *IJCAI Workshop on General Game Playing*.

Shieh, E., An, B., Yang, R., Tambe, M., Baldwin, C., Direnzo, J., Meyer, G., Baldwin, C. W., Maule, B. J., & Meyer, G. R. (2012). PROTECT : A Deployed Game Theoretic System to Protect the Ports of the United States. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 13–20.

Shoham, Y., & Leyton-Brown, K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.

Tambe, M. (2011). *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press.

Tsai, J., Rathi, S., Kiekintveld, C., Ordóñez, F., & Tambe, M. (2009). IRIS - A Tool for Strategic Security Allocation in Transportation Networks Categories and Subject Descriptors. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 37–44.

van Damme, E. (1984). A Relation Between Perfect Equilibria in Extensive Form Games and Proper Equilibria in Normal Form Games. *Game Theory*, *13*, 1–13.

van Damme, E. (1991). *Stability and Perfection of Nash Equilibria*. Springer-Verlag.

von Stengel, B. (1996). Efficient Computation of Behavior Strategies. *Games and Economic Behavior*, *14*, 220–246.

Wilson, R. (1972). Computing Equilibria of Two-Person Games From the Extensive Form. *Management Science*, *18*(7), 448–460.

Zinkevich, M., Johanson, M., Bowling, M., & Piccione, C. (2008). Regret Minimization in Games with Incomplete Information. *Advances in Neural Information Processing Systems (NIPS)*, *20*, 1729–1736.

Zinkevich, M., Bowling, M., & Burch, N. (2007). A New Algorithm for Generating Equilibria in Massive Zero-Sum Games. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pp. 788–793.

Appendix B

Algorithms for computing strategies in two-player simultaneous move games

Artificial Intelligence 237 (2016) 1–40

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Algorithms for computing strategies in two-player simultaneous move games

Branislav Bošanský [a,*], Viliam Lisý [a], Marc Lanctot [b,1], Jiří Čermák [a], Mark H.M. Winands [b]

[a] Agent Technology Center, Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague, Technicka 2, 166 27 Prague 6, Czech Republic

[b] Games and AI Group, Department of Data Science and Knowledge Engineering, Maastricht University, P.O. Box 616, 6200 MD, Maastricht, The Netherlands

ARTICLE INFO

Article history:
Received 14 July 2014
Received in revised form 9 January 2016
Accepted 22 March 2016
Available online 1 April 2016

Keywords:
Simultaneous move games
Markov games
Backward induction
Monte Carlo Tree Search
Alpha-beta pruning
Double-oracle algorithm
Regret matching
Counterfactual regret minimization
Game playing
Nash equilibrium

ABSTRACT

Simultaneous move games model discrete, multistage interactions where at each stage players simultaneously choose their actions. At each stage, a player does not know what action the other player will take, but otherwise knows the full state of the game. This formalism has been used to express games in general game playing and can also model many discrete approximations of real-world scenarios. In this paper, we describe both novel and existing algorithms that compute strategies for the class of two-player zero-sum simultaneous move games. The algorithms include exact backward induction methods with efficient pruning, as well as Monte Carlo sampling algorithms. We evaluate the algorithms in two different settings: the offline case, where computational resources are abundant and closely approximating the optimal strategy is a priority, and the online search case, where computational resources are limited and acting quickly is necessary. We perform a thorough experimental evaluation on six substantially different games for both settings. For the exact algorithms, the results show that our pruning techniques for backward induction dramatically improve the computation time required by the previous exact algorithms. For the sampling algorithms, the results provide unique insights into their performance and identify favorable settings and domains for different sampling algorithms.

1. Introduction

Strategic decision-making in multiagent environments is an important problem in artificial intelligence. With the growing number of agents interacting with humans and with each other, the need to understand these strategic interactions at a fundamental level is becoming increasingly important. Today, agent interactions occur in many diverse situations, such as e-commerce, social networking, and general-purpose robotics, each of which creates complex problems that arise from conflicting agent preferences.

* Corresponding author. Tel.: +420 22435 7581.
E-mail addresses: branislav.bosansky@agents.fel.cvut.cz (B. Bošanský), viliam.lisy@agents.fel.cvut.cz (V. Lisý), marc.lanctot@maastrichtuniversity.nl (M. Lanctot), jiri.cermak@agents.fel.cvut.cz (J. Čermák), m.winands@maastrichtuniversity.nl (M.H.M. Winands).
[1] This author has a new affiliation: Google DeepMind, London, United Kingdom.

http://dx.doi.org/10.1016/j.artint.2016.03.005
0004-3702/

Much research has been devoted to developing algorithms that reason about or learn in sequential (multi-step) interactions. As an example, adversarial search has been a central topic of artificial intelligence since the inception of the field itself, leading to very strong rational behaviors in Chess [1] and Checkers [2]. Advances in machine learning for multi-step interactions (e.g., reinforcement learning) have led to self-play learning of evaluation functions achieving master level play in Backgammon [3] and super-human level in Atari [4].

The most common model for these multistage environments is one with strictly sequential interactions. This model is sufficient in many settings [5], such as in the examples used above. On the other hand, it is not a good representation of the environment when agents are allowed to act simultaneously. These situations occur in many real-world scenarios such as auctions (e.g., [6]), autonomous driving, and many video and board games in the expanding gaming industry (e.g., [7, 8], including games we use for our experiments). In all of these scenarios, the simultaneity of the decision-making is crucial and we have to include it directly into the model when computing strategies. One of the fundamental differences of simultaneous move games versus strictly sequential games is that the agents may need to use randomized (or *mixed*) strategies in order to play optimally [9], i.e., to maximize their worst-case expected utility. This means that agents may need to randomize over several actions in some states of the game to guarantee the worst-case expected utility, even though the only information that is hidden is each player's action as they play it.

This paper focuses specifically on algorithms for decision-making in simultaneous move games. We cover the offline case, where the computation time is abundant and the optimal strategies are computed and stored, as well as the online case, where the computation time is limited and agents must choose an action quickly. We are concerned both with the quality of strategies based on their worst-case expected performance in theory and their observed performance in practice. We compare and contrast the algorithms and parameter choices in the offline and the online cases, and thoroughly evaluate each algorithm on a suite of games. Our collection covers Biased Rock–Paper–Scissors, Goofspiel, Oshi-Zumo, Pursuit–Evasion Games, and Tron, all of which have been used for benchmark purposes in previous work. We also perform experiments on randomly generated games. These games differ in the number of possible actions, the number of moves before the game ends, the variance of the utility values, and the proportion of states in which mixed strategies are required for optimal play.

Our experimental comparison shows that the algorithms perform differently in each case. The exact algorithms based on the backward induction are significantly better in the offline setting, where they are able to find the optimal strategy very quickly compared to the sampling algorithms. In some cases, our novel algorithm (DO$\alpha\beta$) solves the game in less than 2% of the time required by the standard backward induction algorithm. However, the exact algorithms are less competitive in the online setting. In contrast, the approximative sampling algorithms can perform very well in the online setting and find good strategies to play within a few seconds, however, they are not well-suited for offline solving of games.

The paper is structured as follows. First, we make explicit the contributions of the paper in Subsection 1.1. In Section 2, we present a formal introduction of the simultaneous move games that we will use throughout the paper. Section 3 follows with a list and discussion of the existing algorithms in the related work. In Section 4, we describe in detail selected exact and approximative algorithms. We first describe the algorithms in the offline setting, followed by the necessary modifications used in the online case described in Section 5. In Section 6, we present our experimental results comparing the algorithms. Finally, we conclude in Section 7.

1.1. Novel contributions

This paper presents detailed descriptions and analysis of recent state-of-the-art exact [10] and approximative algorithms [11–13] that compute strategies for the class of two-player simultaneous move games. Furthermore, it presents the following original contributions:

- We present the latest variants of state-of-the-art algorithms under a single unified framework and combine the offline and online computation perspectives that have been previously analyzed separately.
- We describe the first adaptation of backward induction and the double-oracle algorithm with serialized bounds (DO$\alpha\beta$) [10] to the online search setting in simultaneous move games using iterative deepening and heuristic evaluation functions.
- We describe a novel variant of Online Outcome Sampling [13] tailored for two-player simultaneous move games (SM-OOS) and provide its formal analysis.
- We provide a wide experimental analysis and a comparison of these and other algorithms on five different specific games and on randomly generated games.
- We replicate an experimental convergence analysis for approximative algorithms that is often used in the literature as a demonstration that sampling-based algorithms are not guaranteed to converge to an optimal solution [14], and we identify the sensitivity of the existing approximative algorithms to tie-breaking rules.

Our algorithms thus allow computing offline strategies in larger games than previously possible (using DO$\alpha\beta$). In online game-playing, our algorithms are less sensitive to chosen parameters (SM-MCTS-RM) or guarantee to closely approximate the optimal strategies given enough time (SM-OOS). Since we describe each algorithm in a domain-independent manner, they can be further tailored to specific domains to achieve additional improvements in the scalability and/or game-playing performance.

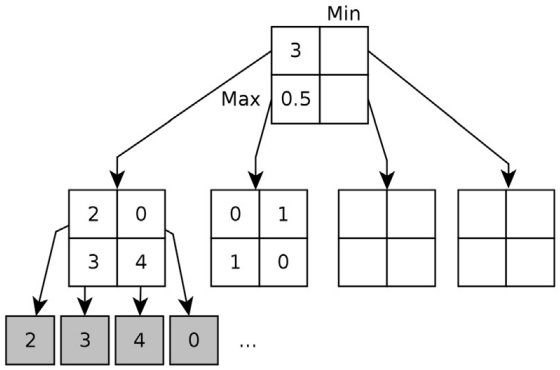


Fig. 1. An example of a two-player simultaneous move game. Each white matrix corresponds to a state of the game where both players (a maximizing player with actions in rows and a minimizing player with actions in columns) act simultaneously. The dark squares are terminal states. The values shown in the matrices correspond to the values of subgames (e.g., calculated by backward induction).

	H	T
H	1	0
T	0	1

	A	B
a	0	1
b	-1	0

Fig. 2. Matrix games of Matching Pennies (left), and one with a pure Nash equilibrium (right). Payoffs for the row player are shown.

2. Simultaneous move games

A finite two-player game with simultaneous moves and chance events (also called *Markov games*, or *stacked matrix games*) is a tuple $(\mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{T}, \Delta_\star, u_i, s_0)$, where $\mathcal{S} = \mathcal{D} \cup \mathcal{C} \cup \mathcal{Z}$. The player set $\mathcal{N} = \{1, 2, \star\}$ contains player labels, where $\star$ denotes the chance player, and by convention a player is denoted $i \in \mathcal{N}$. $\mathcal{S}$ is a set of states, with $\mathcal{Z}$ denoting the terminal states, $\mathcal{D}$ the states where players make decisions, and $\mathcal{C}$ the possibly empty set of states where chance events occur. $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ is the set of joint actions of individual players. We denote by $\mathcal{A}_i(s)$ the actions available to player i in state $s \in \mathcal{S}$. The number of actions available to player i, $|\mathcal{A}_i(s)|$, is called the *branching factor for player i*. When the player is not specified, we mean the joint branching factor $|\mathcal{A}(s)|$. The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \mapsto \mathcal{S}$ is a partial function that defines the successor state given a current state and actions for both players. $\Delta_\star : \mathcal{C} \mapsto \Delta(\mathcal{S})$ describes a probability distribution over possible successor states of the chance event. Induced by $\Delta_\star$, we also define $\mathcal{P}_\star(s, r, c, s')$ as the probability of transitioning to s' after choosing joint action (r, c) from s, or simply 1 when $\mathcal{T}(s, r, c) \notin \mathcal{C}$. The utility function $u_i : \mathcal{Z} \mapsto [v_{\min}, v_{\max}] \subseteq \mathbb{R}$ gives the utility of player i, with $v_{\min}$ and $v_{\max}$ denoting the minimum and maximum possible utility respectively. We assume zero-sum games: $\forall z \in \mathcal{Z}, u_1(z) = -u_2(z)$. The game begins in an initial state s_0 and a subset of a game that starts in some node s is called a *subgame*. An example of such a game is depicted in Fig. 1, more examples can be found in [15, Chapter 5].

In two-player zero-sum games, a (subgame perfect) Nash equilibrium strategy is often considered to be optimal (the formal definition follows). It guarantees an expected payoff of at least V against any opponent. Any non-equilibrium strategy has its nemesis, which makes it gain less than V in expectation. Moreover, a subgame perfect Nash equilibrium strategy can earn more than V against weak opponents. After the opponent makes a sub-optimal move, the strategy will never allow it to gain the loss back. The value V is known as the *value of the game* and it is the same for every equilibrium strategy profile by von Neumann's minimax theorem [16].

A *matrix game* is a single step simultaneous move game with action sets $\mathcal{A}_1$ and $\mathcal{A}_2$ (see Fig. 2). Each entry in the matrix A_{rc} where $(r, c) \in \mathcal{A}_1 \times \mathcal{A}_2$ corresponds to a utility value reached if row r is chosen by player 1 and column c by player 2. For example, in Matching Pennies in the left side of Fig. 2, each player has two actions (heads or tails). The row player receives a payoff of 1 if both players choose the same action and 0 if they do not match. In simultaneous move games, at every decision state $s \in \mathcal{D}$ there is a joint action set $\mathcal{A}_1(s) \times \mathcal{A}_2(s)$. Each joint action (r, c) leads to another state $\mathcal{T}(s, r, c)$ that is either a terminal state or a subgame which is itself another simultaneous move game. A chance event is a state $s \in \mathcal{C}$ with a fixed set of outcomes, each of which leads to a possible successor state. In simultaneous move games, A_{rc} refers to the value of the subgame rooted in state $\mathcal{T}(s, r, c)$.

A *behavioral strategy* for player i is a mapping from states $s \in \mathcal{S}$ to a probability distribution over the actions $\mathcal{A}_i(s)$, denoted $\sigma_i(s)$. We denote by $\sigma_i(s, a)$ the probability that strategy σ_i assigns to a in s. These strategies are often called *randomized*, or *mixed* because they represent a mixture over *pure* strategies, each of which is a point in the Cartesian product space $\prod_{s \in \mathcal{S}} \mathcal{A}_i(s)$.[2] Let $\mathcal{H}$ be a global set of histories (sequences of actions from the start of the game). Given

[2] Notice that a pure strategy is also a mixed strategy that assigns probability 1 to a single pure strategy and probability 0 to every other pure strategy. However, as it is common in the literature, we sometimes refer to a mixed strategy to specifically mean not a pure strategy. This is mostly clear from the context, but we clarify where necessary.

	r	p	s
R	0	−1	1
P	1	0	−1
S	−1	1	0

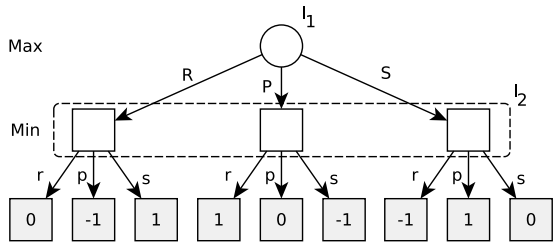


Fig. 3. The matrix game of Rock, Paper, Scissors (left) and its equivalent extensive-form game representation (right). The extensive game has four states, two information sets (I_1 and I_2), and nine terminal histories: $\{Rr, Rp, Rs, Pr, Pp, Ps, Sr, Sp, Ss\}$.

a strategy profile $\sigma = (\sigma_1, \sigma_2)$, we define the probability of reaching a history h under σ as $\pi^\sigma(h) = \pi_1^\sigma(h)\pi_2^\sigma(h)\pi_\star^\sigma(h)$, where each $\pi_i^\sigma(h)$ is a product of probabilities of the actions taken by player i along the path to h ($\pi_\star$ being chance's probabilities). Finally, we define Σ_i to be the set of all behavioral strategies for player i. We adopt a standard convention that the index $-i$ refers to the opponent of player i.

In order to define optimal behavior for this class of games, we now provide definitions of some fundamental concepts.

Definition 2.1 *(Strictly dominated action).* In a matrix game, an action $a_i \in \mathcal{A}_i$ is strictly dominated if $\forall a_{-i} \in \mathcal{A}_{-i}, \exists a_i' \in \mathcal{A}_i \setminus \{a_i\} : u_i(a_i, a_{-i}) < u_i(a_i', a_{-i})$.

No rational player would want to play a strictly dominated action, because there is always a better action to play independent of the opponent's action. The concept also extends naturally to behavioral strategies. For example, in the game on the right of Fig. 2, both b and B are strictly dominated. In this paper we refer to the dominance always in this strict sense.

Definition 2.2 *(Best response).* Suppose $\sigma_{-i} \in \Sigma_{-i}$ is a fixed strategy of player $-i$. Define the set of best response strategies $BR_i(\sigma_{-i}) = \{\sigma_i \mid u_i(\sigma_i, \sigma_{-i}) = \max_{\sigma_i' \in \Sigma_i} u_i(\sigma_i', \sigma_{-i})\}$. A single strategy in this set, e.g., $\sigma_i \in BR_i(\sigma_{-i})$, is called a best response strategy to σ_{-i}.

Note that a best response can be a mixed strategy, but a pure best response always exists [9] and it is often easier to compute.

Definition 2.3 *(Nash equilibrium).* A strategy profile (σ_i, σ_{-i}) is a Nash equilibrium profile if and only if $\sigma_i \in BR_i(\sigma_{-i})$ and $\sigma_{-i} \in BR_{-i}(\sigma_i)$.

In other words, in a Nash equilibrium profile each strategy is a best response to the opponent's strategy. In two-player zero-sum games, the set of Nash equilibria corresponds to the set of minimax-optimal strategies. That is, a Nash equilibrium profile is also a pair of behavioral strategies optimizing

$$V = \max_{\sigma_1 \in \Sigma_1} \min_{\sigma_2 \in \Sigma_2} \mathbb{E}_{z \sim \sigma}[u_1(z)] = \max_{\sigma_1 \in \Sigma_1} \min_{\sigma_2 \in \Sigma_2} \sum_{z \in \mathcal{Z}} \pi^\sigma(z) u_1(z). \tag{1}$$

None of the players can improve their utility by deviating unilaterally. For example, the game of Rock, Paper, Scissors (depicted in Fig. 3) modeled as a matrix game has a single state and the only equilibrium strategy is to mix equally between all actions, i.e., both players play with a mixed strategy $\sigma_i = \sigma_{-i} = (1/3, 1/3, 1/3)$ giving the expected payoff of $V = 0$. Note that using a mixed strategy is necessary in this game to achieve the guaranteed payoff of V. Any pure strategy of one player can be exploited by the opponent; so while a pure best response to a fixed strategy always exists, it is not always possible to find a Nash equilibrium for which both strategies are pure. For the same reason, randomized strategies are often necessary also in the multi-step simultaneous move games. If the strategies also optimize Equation (1) in every subgame, the equilibrium strategy is termed *subgame perfect*.

Finally, a two-player simultaneous move game is a specific type of two-player extensive-form game with imperfect information. In imperfect information games, states are grouped into *information sets*: two states s, s' are in an information set I if the player to act at I cannot distinguish whether she is in s or s'. Any simultaneous move game can be modeled using information sets to represent half-completed transitions, i.e., $\mathcal{T}(s, a_1, ?)$ or $\mathcal{T}(s, ?, a_2)$. The matrix game of Rock, Paper, Scissors can also be thought of as a two-step process where the first player commits to a choice, writing it on a face-down piece of paper, and then the second player responds. Fig. 3 shows this transformation, which can generally be applied to every state in a simultaneous move game. Therefore, algorithms intended for two-player zero-sum imperfect information games may also be applied to the simultaneous move game using this equivalent form.

3. Related work

There has been a number of algorithms designed for simultaneous move games. They can be classified into three categories: (1) iterative learning algorithms, (2) exact backward induction algorithms, (3) approximative sampling algorithms. The first type computes strategies through iterated self-play. The second type computes strategies in a game state recursively based on the values of its successors. The third type computes strategies by approximating utilities using sampling.

3.1. Iterative learning algorithms

A significant amount of interest in simultaneous move games was generated by initial work on multiagent reinforcement learning. In multiagent reinforcement learning, each agent acts simultaneously and the joint action determines how the state changes. Littman introduced Markov games to model these interactions as well as a variant of Q-learning called Minimax-Q to compute strategies [17,18]. Minimax-Q modifies the learning rule so that the value of the next state (the subgame) is obtained by solving a linear program using the estimated values of that subgame's root. As it is common in these settings, the goal of each agent is to maximize their expected utility. In two-player zero-sum Markov games, an optimal policy corresponds to a Nash equilibrium strategy, which assures the agent the highest worst-case expected payoff. Initial results provided conditions under which approximate dynamic programming could be used to guarantee convergence to the optimal value function and policies [19]. Later, in [20], Lagoudakis and Parr provided stronger bounds and convergence guarantees for least squares temporal difference learning using linear function approximation. Bounds on the approximation error for sampling techniques in discounted Markov games are presented in [21], and new bounds for approximate dynamic programming have also been recently shown [22].

In early 2000s, gradient ascent methods were introduced for playing repeated games [23,24]. These algorithms update strategies in a direction of the strategy space that increases the expected payoff with respect to the opponent's strategy. These were then generalized and combined, and shown to minimize regret over time [25,26], leading to strong convergence guarantees in multiagent learning. More no-regret algorithms followed and were applied to imperfect information games in sequence-form (One-Card Poker) [27]. Later, counterfactual regret (CFR) minimization was introduced for large imperfect information games [28]. CFR has gained much attention due to its success in computing Poker AI strategies, and recently an application of CFR has solved Heads Up Limit Texas Hold'em Poker [29]. In this paper we analyze the effectiveness of a specific form of Monte Carlo CFR for the first time in simultaneous move games.

As we focus on zero-sum simultaneous move games in this paper, the work on multiagent learning in general-sum and cooperative games has been omitted. For surveys of the relevant previous work in multiagent reinforcement learning and game theory (including the zero-sum case), see [30–32].

3.2. Exact backward induction algorithms

The techniques in this section are based on the backward induction algorithm (cf. [33]), a form of dynamic programming [34] often presented for purely sequential games. A modified variant of the algorithm can also be applied to simultaneous move games (e.g., see [35–37]). The algorithm enumerates states of the game tree in a depth-first manner and after computing the values of all the succeeding subgames of state $s \in \mathcal{S}$, it solves the normal-form game corresponding to s (i.e., computes a NE of the matrix game in s), and propagates the calculated game value to the predecessor. Backward induction then outputs a subgame perfect NE.

There are two notable algorithms that improve the standard backward induction in simultaneous move games. First is an algorithm by Saffidine et al. [38] termed simultaneous move alpha-beta algorithm (SMAB). The main idea of the algorithm is to reduce the number of the recursive calls of the backward induction algorithm by removing dominated actions in every state of the game. The algorithm keeps bounds on the utility value for each successor in a game state. The lower and upper bounds represent the threshold values, for which neither of the actions of the player is dominated by any other action in the current matrix game. These bounds are calculated by linear programs in the state given existing exact values (or appropriate bounds) of the utility values of all the other successors of the state. If they form an empty interval (the lower bound is greater than the upper bound), pruning takes place and the dominated action is no longer considered in this state afterward.

While SMAB outperforms classical backward induction, the speed-up is less significant in comparison to the second exact algorithm introduced in [10], a description of which is given in detail in Subsection 4.3.1. The main idea is to integrate two key components: (1) instead of evaluating all successors in each state of the game and solving a normal-form game, the algorithm exploits the iterative framework known in game theory as double-oracle algorithm [39]; (2) the algorithm computes bounds on the utility values of the successors by serializing the subgames and running the classic alpha-beta algorithm.

Finally, since simultaneous move games can be seen as extensive-form games with imperfect information, one can use techniques designed for large imperfect information games. An algorithm that is also built on double-oracle is the Range-of-Skill algorithm [40]. However, the number of iterations required by this algorithm in the worst case can be large [41]. There are also state-of-the-art algorithms for solving generic extensive-form games with imperfect information, based on

sequence-form optimization problems [42–44]. However, these algorithms do not exploit the specific structure of simultaneous move games and could require memory that is linear in the size of the game tree. In practice, this prohibits scaling to larger games (see, e.g., [38]) and causes weak performance compared to tailored algorithms.

3.3. Approximative sampling algorithms

Monte Carlo Tree Search (MCTS) is a simulation-based state space search technique often used in extensive-form games [45,46]. Having first seen practical success in computer Go [47,48], MCTS has since been applied successfully to simultaneous move games and to imperfect information games [13,49,50]. Most of the successful applications use the Upper Confidence Bounds (UCB) formula [51] as a selection strategy. These variants of MCTS are also known as UCT (UCB applied to trees). The first application of MCTS to simultaneous move games was in general game playing (GGP) [52] programs: CadiaPlayer [53,54] uses UCB selection strategy for each player in a single game tree. The success of MCTS was demonstrated by the success of CadiaPlayer which was the top-ranked player of the GGP competition between 2007 and 2009, and also in 2012.

Despite this success, Shafiei et al. in [14] provide a counter-example showing that this straightforward application of UCT does not converge to an equilibrium even in the simplest simultaneous move games and that a player playing a NE can exploit this strategy. Another variant of UCT, which has been applied to Tron [55], builds the tree as if the players were moving sequentially giving one of the players an informational advantage. This approach also cannot converge to an equilibrium in general. For this reason, other variants of MCTS were considered for simultaneous move games. Teytaud and Flory describe a search algorithm for games with short-term imperfect information [8], which are a generalization of simultaneous move games. Their algorithm uses a different selection strategy, called Exp3 [56], and was shown to work well in the Internet card game Urban Rivals. We provide details of these two main existing selection functions in Subsections 4.4.1 and 4.4.2. A more thorough experimental investigation of different selection policies including UCB, UCB1-Tuned, UCB1-greedy, Exp3, and more is reported in the game of Tron [57]. The work by Lanctot et al. [11] compares some of these variants and proposes Online Outcome Sampling, a search version of Monte Carlo CFR [58], which computes an approximate equilibrium strategy with high probability. We describe a new formulation of this algorithm in Subsection 4.5.1. Finally, [12,59] present variants of MCTS that provably converge to Nash equilibria in simultaneous move games, using any regret-minimizing algorithm at each stage. We elaborate on these results in Subsection 4.4.4.

There have been two recent studies that examine the head-to-head performance of these variants in practice. The first [60] builds on previous work in Tron by varying the shape of the initial board, comparing previous serialized variants of simultaneous move MCTS. The authors found that UCB1-Tuned worked particularly well in Tron when using knowledge-based playout policies. The success of UCB1-Tuned differed in a similar study of the same variants across nine domains [61] without domain knowledge. In this work, the chosen games were ones inspired by previous work in general game playing and did not include chance elements. Results indicate that parameter-tuning landscapes do not seem as smooth as in the purely sequential case.

3.3.1. Simulation-based search in real-time games

Real-time games are not turn-based and represent realistic physical situations where agents can move freely in space. The state of the game is a continuous function of time and the effect of some actions may only be realized some time after the decision is made. These games are often appropriately modeled as a simultaneous move game with very short delays (e.g., 40 milliseconds) between frames.

MCTS has enjoyed some success in these types of games, in the single-agent setting [62,63] and multiagent setting [64]. Much of this work is inspired by video games [65–67]. Few of these works have considered MCTS in the simultaneous move game directly. In one of the first papers on real-time strategy games, the authors used a randomized serialization of the game [68], or a strategy simulation from scripts was used to build a single matrix of values from which an equilibrium strategy was computed using linear programming [69]. This method can be extended to multiple nodes where internal nodes would correspond to scripts being interrupted to replan, similarly to [70]. MCTS-style multistage replanning was also applied to a real-time battle scenario which was also accurately represented as a discrete simultaneous move game [7]. Results of this work show that the multistage forward replanning can improve upon the single-stage forward planning, and can produce approximate Nash equilibrium strategies when mixed strategies are computed at each stage during the search. Around the same time, a serialized (sequential) version of the alpha-beta algorithm was proposed for simultaneous move games and run on combat scenarios [71]. This algorithm is described in greater detail in Subsection 4.2 as it forms the basis of the follow-up algorithm enhanced by double-oracle, presented in Subsection 4.3.

In this paper, we focus on the analysis of different algorithms for two-player simultaneous move games. Therefore, the problems arising from discrete modeling of continuous time and space remain outside the scope of this paper.

4. Offline strategy computation

This section focuses on algorithms that compute strategies for simultaneous move games. The baseline algorithm for solving simultaneous move games exactly is backward induction (BI) (Subsection 4.1). Afterwards we present a modification that exploits a fast computation of upper and lower bounds in a simultaneous move game (Subsection 4.2). Then, we further improve the algorithm by speeding up the computation of NE in matrix games, exploiting the iterative framework

```
   input : s – current matrix game; i – searching player
1  if s ∈ Z then
2      return u_i(s)
3  for r ∈ A_1(s) do
4      for c ∈ A_2(s) do
5          A_rc ← Σ_{s'∈S : P_⋆(s,r,c,s')>0} P_⋆(s, r, c, s') · BI(s', i)
6  ⟨v_s, σ_i(s)⟩ ← solve matrix game A
7  return v_s
```

Algorithm 1: Backward Induction algorithm (BI).

of double-oracle algorithms (Subsection 4.3). In Subsection 4.4 we describe Monte Carlo Tree Search for simultaneous move games. Finally, we present counterfactual regret minimization and its adaptation Online Outcome Sampling in Subsection 4.5.

4.1. Backward induction

The standard backward induction algorithm, first described for simultaneous move games in [35], enumerates the states in depth-first order. At each state of the game, it creates a matrix game for the current state using child subgame values, solves the matrix game, and propagates back the value of the matrix game. The pseudocode of the algorithm is given in Algorithm 1. If the successor node $\mathcal{T}(s, r, c)$ is a chance node, the algorithm directly evaluates all successors of this chance node and computes an expected utility: the value of each subgame rooted in node s' computed by the recursive call is weighted by the probability of the stochastic transition $\mathcal{P}_\star(s, r, c, s')$ (line 5).

Once the algorithm computes the value of each possible subgame following the current state s, matrix game A is well-defined and the algorithm solves matrix game A by solving the standard linear program (LP) for normal-form games[3]:

$$\max \quad v_s \tag{2}$$

$$\text{s.t.} \sum_{a_i \in \mathcal{A}_i} A_{a_i, a_{-i}} \cdot \sigma_i(s, a_i) \geq v_s \quad \forall a_{-i} \in \mathcal{A}_{-i}(s) \tag{3}$$

$$\sum_{a_i \in \mathcal{A}_i} \sigma_i(s, a_i) = 1 \tag{4}$$

$$\sigma_i(s, a_i) \geq 0 \quad \forall a_i \in \mathcal{A}_i(s) \tag{5}$$

A linear programming algorithm computes both the value v_s of the matrix game A, as well as the optimal strategy to play in this matrix game (variables $\sigma_i(s, a_i)$). Value v_s is then propagated to the predecessor (line 7 of Algorithm 1) and the optimal strategy $\sigma_i(s, a_i)$ is stored for this state. If the algorithm evaluates a terminal state, it directly returns the utility value of the state (line 1).

Evaluating each successor and solving an LP in each state of the game is the main computational bottleneck of the backward induction algorithm. The following algorithms try to prune some of the branches of the game tree in order to reduce this bottleneck even at the cost of multiple traversals of the game tree.

4.2. Backward induction with serialized alpha-beta bounds

Solving computationally expensive linear programs in the backward induction algorithm is necessary in game states that require mixed strategies. However, many realistic games include subgames where it is sufficient to use only pure strategies. These subgames can be found efficiently by transforming the simultaneous move game into a perfect information extensive-form game with sequential moves and subsequently using some of the algorithms developed for this more standard setting. We call this purely alternating form a *serialization* of the original simultaneous move game. Consider a matrix game representing a single joint action of both players. This matrix can be serialized by discarding the notion of information sets; hence, letting one player play first, followed by the second player. The difference between a serialized and a simultaneous move matrix game is that the second player to move has an advantage of knowing what action the first player chose.

Given this advantage, the value of a serialized game consisting of a single simultaneous move where player i is second to move is greater than or equal to the value of the original simultaneous move game from the perspective of player i, formally shown by the following lemma.

Lemma 4.1. *Let A be a single step simultaneous move game for state s with value v_s for player i. Let v_s^i be the value of the serialized game created from game A by letting player $-i$ move first and player i move second with the knowledge of the move played by the first player. Then*

$$v_s \leq v_s^i.$$

[3] By solving a game we mean computing both the optimal value and the strategy that achieves it.

	b_1	b_2
a_1	2	0
a_2	3	4

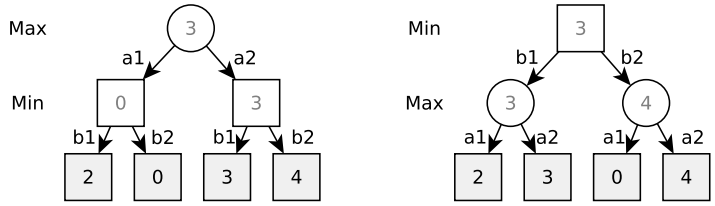


Fig. 4. Different serializations of a simple simultaneous move game. Utility values are in the leaf nodes, the gray values correspond to the value propagation when solving the serialized game.

Proof.

$$\begin{aligned} v_s &= \min_{\sigma_{-i}\in\Sigma_{-i}} \max_{\sigma_i\in\Sigma_i} \sum_{a_i\in\mathcal{A}_i(s)} \sum_{a_{-i}\in\mathcal{A}_{-i}(s)} \sigma_i(s,a_i)\sigma_{-i}(s,a_{-i})A_{a_i a_{-i}} \\ &= \min_{a_{-i}\in\mathcal{A}_{-i}(s)} \max_{\sigma_i\in\Sigma_i} \sum_{a_i\in\mathcal{A}_i(s)} \sigma_i(s,a_i)A_{a_i a_{-i}} \\ &\leq \min_{a_{-i}\in\mathcal{A}_{-i}(s)} \max_{a_i\in\mathcal{A}_i(s)} A_{a_i a_{-i}} = v_s^i. \end{aligned}$$

The first equality is the definition of the value of a zero-sum game. The second equality is from the fact that a best response can always be found in pure strategies: if there was a mixed strategy best response with expected utility v_s and some of the actions from its support would have lower expected utility, removing those actions from the support would increase the value of the best response, which is a contradiction. The inequality is due to the fact that a maximization over each action of player $-i$ can only increase the value. □

We can now generalize this lemma to game trees with multiple simultaneous moves.

Lemma 4.2. *Consider a simultaneous move subgame defined by state s and a serialized variant of this subgame, where in each state player i is second to move. The value of the serialized game is an upper bound on the value of the simultaneous move subgame for player i.*

Proof. We use Lemma 4.1 inductively. Let s be the current state of the game and let A be the exact matrix game corresponding to s with utilities of player i. By induction we assume that the algorithm computes for state s some A' so that each value in matrix A' is greater than or equal to A:

$$\forall a_i \in \mathcal{A}_i(s) \forall a_{-i} \in \mathcal{A}_{-i}(s)\ A'_{a_i a_{-i}} \geq A_{a_i a_{-i}}.$$

Therefore, the value of matrix game $v_{A'} \geq v_A$. Finally, by Lemma 4.1 the algorithm returns value $v^i_{A'} \geq v_{A'} \geq v_A$. □

An example of this serialization is depicted in Fig. 4. There is a simple matrix game for two players (the circle and the box player; the utility values are depicted for the circle player; the box player in the column is minimizing this value). There are two ways this game can be transformed into a serialized extensive-form game with perfect information. If the circle player moves first (the left game tree), then the value of this serialized game is the lower bound of the value of the game. If this player moves second (the right game tree), then the value of this serialized game is the upper bound of the value of the game. Since the serialized games are zero-sum perfect information games in the extensive form, they can be solved quite quickly by using some of the classic AI algorithms such as alpha-beta or Negascout [72]. If the values of both serialized games are equal, then this value is also equal to the value of the original simultaneous move game. This situation occurs in our example in Fig. 4, where both serialized games have value $V = 3$.

We can speed up the backward induction algorithm using bounds that are computed by the alpha-beta algorithm (denoted BI$\alpha\beta$). Algorithm 2 depicts the pseudocode. The BI$\alpha\beta$ algorithm first serializes the game and solves the serialized games using the standard alpha-beta algorithm; if the bounds are equal then this value is returned directly (line 3). Note that in Algorithm 2 the call alpha-beta(s, i), i is the second player to move in the serialized game rooted at s. If the bounds are not equal, the algorithm starts evaluating successors of the current state. As before, the algorithm computes upper and lower bounds using the alpha-beta algorithm on serialized variants of the subgame rooted at the successor s' (lines 9–10). Then, the algorithm uses the value directly if the bounds are equal (line 14), or performs a recursive call otherwise (line 12).

```
   input : s – current matrix game; i – searching player
 1 if s ∈ Z then
 2     return u_i(s)
 3 if (s is root) and (alpha-beta(s, i) = alpha-beta(s, −i)) then
 4     return alpha-beta(s, −i)
 5 for r ∈ A_1(s) do
 6     for c ∈ A_2(s) do
 7         A_rc ← 0
 8         for s' ∈ S : P_⋆(s, r, c, s') > 0 do
 9             v^i_{s'} ← alpha-beta(s', i)
10             v^{−i}_{s'} ← alpha-beta(s', −i)
11             if v^{−i}_{s'} < v^i_{s'} then
12                 A_rc ← A_rc + P_⋆(s, r, c, s') · BIαβ(s', i)
13             else
14                 A_rc ← A_rc + P_⋆(s, r, c, s') · v^i_{s'}
15 ⟨v_s, σ_i⟩ ← solve matrix game A
16 return v_s
```

Algorithm 2: Backward Induction with serialized bounds (BI$\alpha\beta$).

We distinguish two cases when extracting equilibrium strategies from the BI$\alpha\beta$ algorithm. In the first case, when a state is fully evaluated by the algorithm (i.e., an LP was built and solved for this state), we proceed as before and keep the pair of equilibrium strategies in this state. However, in the other case, the algorithms prunes certain branches and does not create an LP in some of the subgames. The algorithm then keeps the strategy computed by the serialized alpha-beta algorithm in those subgames. More precisely, for player i the algorithm keeps the pure strategy computed by alpha-beta$(s, -i)$, where the opponent has an advantage of knowing the moves of player i. Such a strategy provides a guarantees for player i (it is not exploitable) and due to the alpha-beta cut-offs we know that there is no better strategy for player i with a higher expected utility.

Theorem 4.3. *The algorithm BI$\alpha\beta(s, i)$ computes the value of the subgame from state s for player i.*

Proof. The correctness of the algorithm follows immediately from the correctness of the standard BI algorithm and the correctness of using the values computed by serialized alpha-beta (Lemma 4.2). Moreover, values computed by the serialized alpha-beta algorithm are used only if the upper bound equals the lower bound. □

The performance of BI$\alpha\beta$ depends on the existence of a pure NE in the simultaneous move game. In the best case (i.e., there exists a pure NE), the algorithm finds the solution by solving each serialization exactly once starting from the root state. In the worst case, all NE require mixed strategies in every state of the game. In this case, the algorithm not only solves the LP in each state similarly to BI, but also repeatedly attempts to solve serialized subgames by calling the alpha-beta algorithm. However, this case was very rarely encountered during our experiments.

4.3. Backward induction with double-oracle and serialized bounds

The computational complexity of solving a matrix game by linear programming can be reduced by their incremental construction using the iterative double-oracle algorithm [39]. The following algorithm incorporates this idea to BI$\alpha\beta$, which leads to additional pruning of the game tree. First of all, we describe the main principles of the double-oracle algorithm for matrix games, followed by the description of the integration of the double-oracle algorithm in simultaneous move games [10] (denoted DO$\alpha\beta$).

4.3.1. Double-oracle algorithm for matrix games

The goal of the double-oracle algorithm is to find a solution of a matrix game without necessarily constructing the complete LP that solves the game. The main idea is to create a restricted game where the players can choose only from a limited set of actions. The algorithm iteratively expands the restricted game by allowing the players to choose from new actions. The new actions are added incrementally: in each iteration, a best response (chosen from the unrestricted action set) to an optimal strategy of the opponent in the current restricted game, is added to restricted game.

Fig. 5 shows a visualization of the main structure of the algorithm, where the following three steps repeat until convergence:

1. Create a restricted matrix game by limiting the set of actions that each player is allowed to play.
2. Compute a pair of Nash equilibrium strategies in this restricted game using linear programming.
3. For each player, compute a pure best response strategy against the equilibrium strategy of the opponent; pure best response can be *any* action from the original unrestricted game.

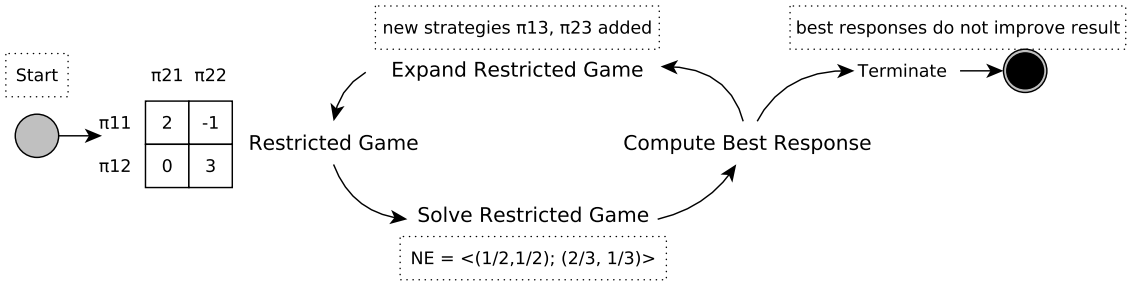


Fig. 5. Schematic of the double-oracle algorithm for a normal-form game.

The best response strategies computed in step 3 are added to the restricted game, the game matrix is expanded by adding new rows and columns, and the algorithm follows with the next iteration. The algorithm terminates if neither of the players can improve the outcome of the game by adding a new strategy to the restricted game; hence, both players play best response strategies to the strategy of the opponent. The algorithm maintains the values of the best expected utilities of the best-response strategies for each player throughout the iterations of the algorithm. These values provide bounds on the value of the original game V (from Equation (1)), and their sum represents the error of the algorithm which converges to zero.

4.3.2. Integrating double-oracle with backward induction

The double-oracle algorithm for matrix games can be directly incorporated into the backward induction algorithm: instead of immediately evaluating each of the successors of the current game state and solving the linear program, the algorithm can exploit the double-oracle algorithm. Pseudocode in Algorithm 3 details this integration.

Similarly to BI$\alpha\beta$, the algorithm first tests, whether the whole game can be solved by using the serialized variants of the game (line 3). If not, then in each state of the game the algorithm initializes the restricted game with an arbitrary action (line 5)[4] – A' represents the restricted matrix game, $\mathcal{A}'_i$ represents the restricted set of available actions to player i. The algorithm then starts the iterations of the double-oracle algorithm. First, the algorithm needs to compute the value for each of the successors of the restricted game, for which the current value is not known (lines 8–16). This evaluation is the same as in the case of BI$\alpha\beta$. Once all values for restricted game A' are known, the algorithm solves the restricted game and keeps the optimal strategies σ' of the restricted game (line 17). Next, the algorithm computes best responses for each of the player (lines 18, 19) using Algorithm 4 below, and updates the lower and upper bounds (line 20). Finally, the algorithm expands the restricted game with the new best response actions (line 21) until the lower and upper bound are equal. Once the bounds are equal, neither of the best responses improves the current solution from the restricted game; hence, the algorithm has found an equilibrium of the complete unrestricted matrix game corresponding to state s.

Now we describe the algorithm for computing the best responses on lines 18 and 19. The pseudocode of this step is depicted in Algorithm 4. The goal of the best response algorithm is to find the best action from the original unrestricted game against the current strategy of the opponent σ'_{-i}. Throughout the algorithm we use, as before, $v^i_{s'}$ to denote the upper bound of the value of the subgame rooted in state s' computed using alpha-beta(s', i). These values are computed on demand, i.e., they are computed once needed and cached until the game for state s is solved. Moreover, once the algorithm computes the exact value of a particular subgame, both upper and lower bounds are updated to be equal to the exact value of the game.

The best response algorithm iteratively examines all actions of player i from the unrestricted game (line 3). Every action a_i is evaluated against the actions of the opponent that are used in the optimal strategy from the restricted game (line 5). Before evaluating the successors, the algorithm determines whether the current action a_i of the searching player i can still be the best response action against the strategy of the opponent σ'_{-i}. In order to determine this, the algorithm computes value λ_{a_i} that represents the lower bound on the expected utility this action must gain against the current action of the opponent a_{-i} in order for action a_i to be a best response. λ_{a_i} is calculated (line 7) by subtracting the upper bound of the expected value against all other actions of the opponent ($v^i_{\mathcal{T}(s,a_i,a'_{-i})}$) from the current best response value (v^{BR}_i) and normalizing with the probability that the action a_{-i} is played by the opponent ($\sigma'_{-i}(a_{-i})$). This calculation corresponds to a situation where player i achieves the best possible utility by playing action a_i against all other actions from the strategy of the opponent and it needs to achieve at least λ_{a_i} against a_{-i} so that the expected value for playing a_i is at least v^{BR}_i. If λ_{a_i} is strictly higher than the upper bound on the value of the subgame rooted in the successor (i.e., $v^i_{\mathcal{T}(s,a_i,a_{-i})}$) then the algorithm knows that the action a_i can never be the best response action, and can proceed with the next action (line 9). Note that λ_{a_i} is recalculated for each action of the opponent since the upper bound values can become tighter when the exact values are computed for successor nodes s' (line 13).

[4] In practice we use the first action of a shuffled ordered set $\mathcal{A}_i$ for each player i. This initialization step can be improved with domain knowledge and by adding more actions.

input : s – current matrix game; i – searching player; α_s, β_s – bounds for the game value rooted in state s
if $s \in \mathcal{Z}$ **then**
  return $u_i(s)$
if (s *is root*) **and** (*alpha-beta*$(s,i) =$ *alpha-beta*$(s,-i)$) **then**
  return alpha-beta$(s,-i)$
initialize $\mathcal{A}'_i$, $\mathcal{A}'_{-i}$ with arbitrary actions from $\mathcal{A}_i, \mathcal{A}_{-i}$
repeat
  for $r \in \mathcal{A}'_i, c \in \mathcal{A}'_{-i}$ **do**
    if A'_{rc} *is not initialized* **then**
      $A'_{rc} \leftarrow 0$
      for $s' \in \mathcal{S} : \mathcal{P}_\star(s,r,c,s') > 0$ **do**
        $v^i_{s'} \leftarrow$ alpha-beta(s',i)
        $v^{-i}_{s'} \leftarrow$ alpha-beta$(s',-i)$
        if $v^{-i}_{s'} < v^i_{s'}$ **then**
          $A'_{rc} \leftarrow A'_{rc} + \mathcal{P}_\star(s,r,c,s') \cdot \text{DO}\alpha\beta(s', i, v^{-i}_{s'}, v^i_{s'})$
        else
          $A'_{rc} \leftarrow A'_{rc} + \mathcal{P}_\star(s,r,c,s') \cdot v^i_{s'}$
  $\langle v_s, \sigma' \rangle \leftarrow$ solve matrix game A'
  $\langle v^{BR}_i, a^{BR}_i \rangle \leftarrow \text{BR}(s, i, \sigma'_{-i}, \beta_s)$
  $\langle v^{BR}_{-i}, a^{BR}_{-i} \rangle \leftarrow \text{BR}(s, -i, \sigma'_i, -\alpha_s)$
  $\alpha_s \leftarrow \max(\alpha_s, -v^{BR}_{-i})$, $\beta_s \leftarrow \min(\beta_s, v^{BR}_i)$
  $\mathcal{A}'_i \leftarrow \mathcal{A}'_i \cup \{a^{BR}_i\}$, $\mathcal{A}'_{-i} \leftarrow \mathcal{A}'_{-i} \cup \{a^{BR}_{-i}\}$
until $\alpha_s = \beta_s$
return v_s

Algorithm 3: Double-Oracle with serialized bounds (DO$\alpha\beta$).

input : s – current matrix game; i – best-response player; σ'_{-i} – strategy of the opponent; λ – bound for the best-response value
1 $v^{BR}_i \leftarrow \lambda$
2 $a^{BR}_i \leftarrow$ **null**
3 **for** $a_i \in \mathcal{A}_i$ **do**
4 $v_{a_i} \leftarrow 0$
5 **for** $a_{-i} \in \mathcal{A}'_{-i} : \sigma'_{-i}(a_{-i}) > 0$ **do**
6 $v_{a_i,a_{-i}} \leftarrow 0$
7 $\lambda_{a_i} \leftarrow \frac{v^{BR}_i - \sum_{a'_{-i} \in \mathcal{A}'_{-i} \setminus \{a_{-i}\}} \sigma'_{-i}(a'_{-i}) \cdot v^i_{\mathcal{T}(s,a_i,a'_{-i})}}{\sigma'_{-i}(a_{-i})}$
8 **if** $\lambda_{a_i} > v^i_{\mathcal{T}(s,a_i,a_{-i})}$ **then**
9 continue from line 3 with next a_i
10 **else**
11 **for** $s' \in \mathcal{S} : \mathcal{P}_\star(s,a_i,a_{-i},s') > 0$ **do**
12 **if** $v^{-i}_{s'} < v^i_{s'}$ **then**
13 $v_{a_i,a_{-i}} \leftarrow v_{a_i,a_{-i}} + \mathcal{P}_\star(s,a_i,a_{-i},s') \cdot \text{DO}\alpha\beta(s', i, v^{-i}_{s'}, v^i_{s'})$
14 **else**
15 $v_{a_i,a_{-i}} \leftarrow v_{a_i,a_{-i}} + \mathcal{P}_\star(s,a_i,a_{-i},s') \cdot v^i_{s'}$
16 $v_{a_i} \leftarrow v_{a_i} + \sigma'_{-i}(a_{-i}) \cdot v_{a_i,a_{-i}}$
17 **if** $v_{a_i} \geq v^{BR}_i$ **then**
18 $v^{BR}_i \leftarrow v_{a_i}$
19 $a^{BR}_i \leftarrow a_i$
20 **return** $\langle v^{BR}_i, a^{BR}_i \rangle$

Algorithm 4: Best Response with serialized bounds (BR).

If the currently evaluated action a_i can still be a best response, the value of the successor is determined (first by comparing the bounds). Once the expected outcome against all actions of the opponent is known, the expected value of action a_i is compared against the current best response value (line 17) and saved if the expected utility is higher (line 19). These best response actions are allowed in the next iteration of the double-oracle algorithm and the algorithm progresses further as described.

When extracting strategies from DO$\alpha\beta$, we proceed exactly as in the case of BI$\alpha\beta$: either a double-oracle is initialized and solved for a certain matrix game and we keep the equilibrium strategies from the final restricted game, or the strategy is extracted from the serialized alpha-beta algorithms as before.

Theorem 4.4. *The DO$\alpha\beta(s, i, \alpha_s, \beta_s)$ algorithm computes the value of the subgame defined by state s for player i.*

Proof. The correctness of the algorithm follows from the correctness of the standard BI algorithm, Lemma 4.2, and the correctness of the double-oracle algorithm for matrix games [39]. We use them inductively for state s and assume DO$\alpha\beta$ for all the children of s returned correct values when called. Since we are using the classical double-oracle on a matrix

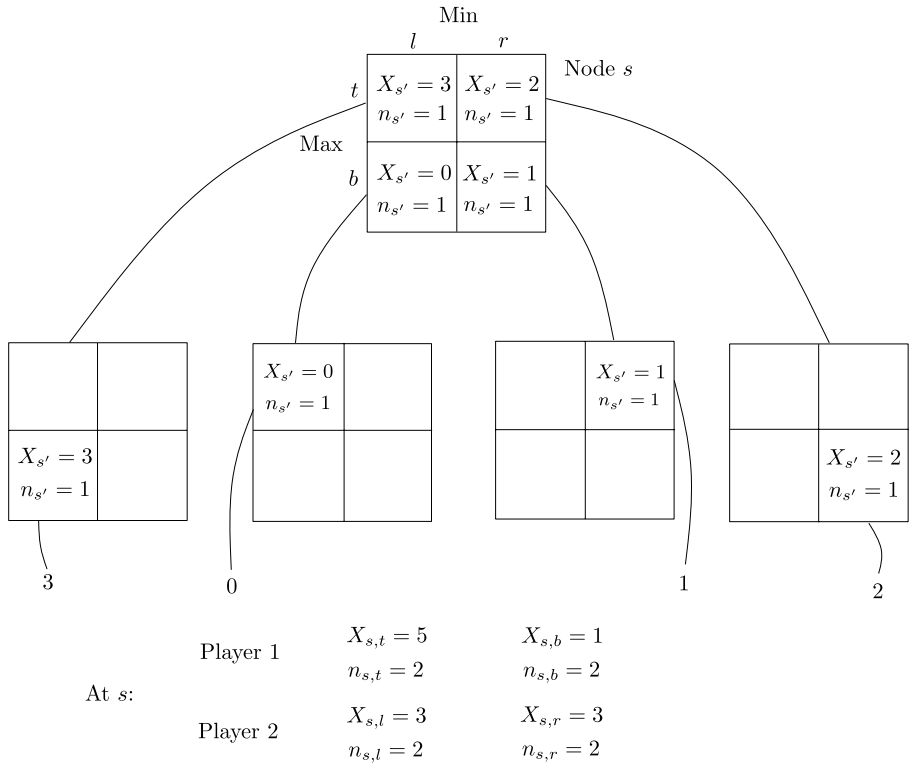


Fig. 6. Simultaneous Move MCTS example. Here, $X_{s'}$ represents the cumulative payoff of all simulations that have passed through the cell, while $n_{s'}$ represents the number of simulations that have passed through the cell.

game corresponding to state s with correct values, we only need to show that the best-response algorithm with serialized bounds cannot return **null** action due to setting the bounds incorrectly.

Without loss of generality, consider a lower bound $-\alpha_s$ for state s to be λ in the best response algorithm. Value λ thus corresponds either to a value calculated by serialized alpha-beta and propagated via bounds when calling DO$\alpha\beta(s, i, \alpha_s, \beta_s)$, or it was updated during the iterations of the double-oracle algorithm for state s (line 20). In either case there exists a pure best response strategy corresponding to this value; hence, the best response has to find the strategy that achieves this value and cannot return **null**. □

Similarly to BI$\alpha\beta$, the performance of DO$\alpha\beta$ also depends on the existence of a pure NE in the simultaneous move game. The best case is identical to BI$\alpha\beta$ and the algorithm finds the solution by solving each serialization exactly once starting from the root state. In the worst case, neither of the serialized games yield useful bounds and the algorithm needs to call the double-oracle algorithm in every state. Moreover, the worst case for the double-oracle algorithm occurs when all actions in this state must be added and an action for only a single player is added in each iteration causing the largest number of iterations repeatedly resolving the linear program. Again in practical games used for benchmark purposes, or in real-world applications this is rarely the case. Moreover, the computational overhead from repeatedly solving an LP is relatively small. This is due to the size of each LP that is determined by the number of actions in each state (the number of constraints and variables is bounded by the number of actions in each state). Therefore, the size of each LP is small compared to the number of states DO$\alpha\beta$ can prune out, especially if the pruning occurs close to the root of the game tree.

4.4. Simultaneous Move Monte Carlo Tree Search (SM-MCTS)

In the following subsections we move to the approximative algorithms. Monte Carlo Tree Search (MCTS) is a simulation-based state space search algorithm often used in game trees. In its simplest form, the tree is initially empty and a single leaf is added each iteration. Each iteration starts by visiting nodes in the tree, selecting which actions to take based on a *selection function* and information maintained in the node. Consequently, the algorithm transitions to a successor state. When a node is visited whose immediate children are not all in the tree, the node is expanded by adding a new leaf to the tree. Then, *a rollout policy* (e.g., random action selection) is applied from the new leaf to a terminal state. The outcome of the simulation is then returned as a reward to the new leaf and the information stored in the tree is updated.

Consider again the game depicted in Fig. 1. We demonstrate how Monte Carlo Tree Search could progress in this game using the example shown in Fig. 6. This game has a root state, two subgames that are simple matrix games, and two arbitrarily large subgames. In the root state, player 1 (Max) has two actions: top (t) and bottom (b), and player 2 also has two actions: left (l) and right (r). The tree is initialized with a single empty state, s. On the first iteration, the first child

```
   input : s – current state of the game
 1 if s ∈ Z then
 2     return u_1(s)
 3 if s ∈ C is a chance node then
 4     Sample s' ~ Δ_⋆(s)
 5     return SM-MCTS(s')
 6 if s is in the MCTS tree then
 7     (a_1, a_2) ← SELECT(s)
 8     s' ← T(s, a_1, a_2)
 9     v_s' ← SM-MCTS(s')
10     UPDATE(s, a_1, a_2, v_s')
11     return v_s'
12 else
13     Add s as a new child in the MCTS tree
14     v_s ← Rollout(s)
15     return v_s
```

Algorithm 5: Simultaneous Move Monte Carlo Tree Search (SM-MCTS).

corresponding to (t, l) is added to the tree, giving a payoff $u_1 = 3$ at the terminal state which is backpropagated to each state visited on the simulation. Similarly, on the second iteration the second child corresponding to (b, l) is added to the tree, giving a payoff $u_1 = 1$, which is backpropagated up to all of its parents. After four simulations, every cell in the root state has a value estimate.

There are many possible ways to select actions based on the estimates stored in each cell which lead to different variants of the algorithm. We therefore first formally describe a generic template of MCTS algorithms for simultaneous move games (SM-MCTS) and then explain different instantiations derived from this template. Algorithm 5 describes a single iteration of SM-MCTS. The "MCTS tree" is an explicit tree data structure that stores the nodes of the search tree maintained in memory, e.g., the five-node tree shown in Fig. 6. Every node s in the tree maintains algorithm-specific statistics about the iterations that previously visited this node. The template can be instantiated by specific implementations of the updates of the statistics on line 10 and the selection based on these statistics on line 7. In the terminal states, the algorithm returns the value of the state for the first player (line 2). At chance nodes, the algorithm samples one of the possible next states based on its distribution (line 4). If the current state has a node in the current MCTS tree, the statistics in the node are used to select an action for each player (line 7). These actions are executed (line 8) and the algorithm is called recursively on the resulting state (line 9). The result of this call is used to update the statistics maintained for state s (line 10). If the current state is not stored in the tree, it is added to the tree (line 13) and its value is estimated using the rollout policy (line 14).

Several different algorithms (e.g., UCB [51], Exp3 [56], and regret matching [73]) can be used as the selection function. We now present the variants of SM-MCTS that were consistently the most successful in the previous works, though more variants can be found in [57,60,61].

4.4.1. Decoupled upper-confidence bound applied to trees

The most common selection function for SM-MCTS is the decoupled Upper-Confidence Bound applied to Trees (UCT). For the selection and updates, it executes the well-known UCT [46] algorithm independently for each of the players in each nodes. The statistics stored in the tree nodes are independently computed for each action of each player. For player $i \in \mathcal{N}$ and action $a_i \in \mathcal{A}_i(s)$ the reward sums X_{a_i} and the number of times the action was used n_{a_i} are maintained. When a joint action needs to be selected by the SELECT function, an action that maximizes the UCB value over their utility estimates is selected for each player independently (therefore it is called decoupled):

$$a_i = \underset{a_i \in \mathcal{A}_i(s)}{\operatorname{argmax}} \left\{ \bar{X}_{a_i} + C_i \sqrt{\frac{\log n_s}{n_{a_i}}} \right\}, \text{ where } \bar{X}_{a_i} = \frac{X_{a_i}}{n_{a_i}} \text{ and } n_s = \sum_{b_i \in \mathcal{A}_i(s)} n_{b_i}. \tag{6}$$

The UPDATE function increases the visit count and rewards for each player i and its selected action a_i using $X_{a_i} \leftarrow X_{a_i} + u_i$ and $n_{a_i} \leftarrow n_{a_i} + 1$.

Consider again the example shown in Fig. 6. Decoupled UCT now groups together all the payoffs obtained for an action. Therefore, at the root Max has $\bar{X}_t = 5/2 = 2.5$, $\bar{X}_b = 1/2 = 0.5$ and the exploration term for both is $C_i\sqrt{(\log 4)/2}$, and so top action is selected. For Min, $\bar{X}_l = 3/2 = 1.5 = \bar{X}_r$, so both actions have the same value. Therefore, Min must use a tie-breaking rule in this situation to decide which action to take. As we discuss later, the specific tie-breaking rule used here can lead to a significant effect on the quality of the strategy that UCT produces.

After all the simulations are done, there are two options for how to determine the resulting action to play. The more standard option is to choose for each state the action a_i that maximizes n_{a_i} for each player i. This is suitable mainly for games, in which using mixed strategy is not necessary. Alternatively, the action to play in each state can be determined based on the mixed strategy obtained by normalizing the visit counts of each action

$$\sigma_i(a_i) = \frac{n_{a_i}}{\sum_{b_i \in \mathcal{A}_i(s)} n_{b_i}}. \tag{7}$$

Using the first method certainly makes the algorithm not converge to a Nash equilibrium, because the game may require a mixed strategy. Therefore, unless stated otherwise, we only use the mixed form in Equation (7), which was called DUCT(mix) in [11,61].

Note, that it was shown that this latter variant also might not converge to a Nash equilibrium (a well-known counter-example in Rock, Paper, Scissors with biased payoffs [14]). However, one of the issues when using UCT in game trees is an unspecified behavior in case there are multiple actions with identical value in the maximization described in the UCT formula in Equation (6). This may have a significant impact on the performance of the UCT in simultaneous move games. Consider the matrix game at the right of Fig. 2. This game has only one NE: (a, A). However, if UCT selects the first or the last action among the options with the same value, it will always get only the utility 0 and the bias term will cause the players to round-robin over the diagonal indefinitely. This is clearly not optimal, as each player can then improve by playing first action with probability 1. However, if we choose the action to play randomly among the tied actions (where "tied" could be defined as being within a small tolerance gap), UCT will quickly converge to the optimal solution in this game. We experimentally analyze the impact of this randomization on the example used in [14] and show that if a randomized variant of UCT is used, the algorithm still does not converge to a NE but does converge to a strategy that is much closer to a NE than without randomization (see Subsection 6.3). Therefore, unless stated otherwise, we use the randomized variant in our implementation.

Even though UCT is not guaranteed to converge to the optimal solution, it is often very successful in practice. It has been used in general game playing [54], in the card game Urban Rivals [8], and in Tron [57].

4.4.2. Exponential-weight algorithm for exploration and exploitation

Another common choice of a selection function is to use the Exponential-weight algorithm for Exploration and Exploitation (Exp3) [56] independently for each of the players. Unlike with UCT, two players using Exp3 in a single stage matrix game are guaranteed to converge to a Nash equilibrium [56]; hence, we can expect a good performance of this selection function even in multi-stage games. In Exp3, each player maintains an estimate of the sum of rewards for each action, denoted $\hat{X}_{a_i}$. The joint action produced by SELECT is composed of an action independently selected for each player. An action is selected by sampling from a probability distribution over actions. Define γ to be the probability of exploring, i.e., choosing an action uniformly. The probability of selecting action a_i is proportional to the exponential of the reward estimates:

$$\sigma_i(a_i) = \frac{(1-\gamma)\exp(\eta \hat{X}_{a_i})}{\sum_{b_i \in \mathcal{A}_i(s)} \exp(\eta \hat{X}_{b_i})} + \frac{\gamma}{|\mathcal{A}_i(s)|}, \text{ where } \eta = \frac{\gamma}{|\mathcal{A}_i(s)|}. \tag{8}$$

This standard formulation of Exp3 is suitable for deriving its properties, but a straightforward implementation of this formula leads to problems with a numerical stability. Both the numerator and the denominator of the fraction can quickly become too large. For this reason, other formulations have been suggested, e.g., in [11] and [50] that are more numerically stable. We use the following equivalent formulation from [50]:

$$\sigma_i(a_i) = \frac{(1-\gamma)}{\sum_{b_i \in A_i(s)} \exp(\eta(\hat{X}_{b_i} - \hat{X}_{a_i}))} + \frac{\gamma}{|\mathcal{A}_i(s)|}. \tag{9}$$

The update after selecting actions (a_1, a_2) and obtaining a simulation result v_1 normalizes the result to the unit interval for each player by

$$u_1 \leftarrow \frac{(v_1 - v_{\min})}{v_{\max} - v_{\min}}; \;\; u_2 \leftarrow (1 - u_1), \tag{10}$$

and adds to the corresponding reward sum estimates the reward divided by the probability that the action was played by the player using

$$\hat{X}_{a_i} \leftarrow \hat{X}_{a_i} + \frac{u_i}{\sigma_i(a_i)}. \tag{11}$$

Dividing the value by the probability of selecting the corresponding action makes $\hat{X}_{a_i}$ estimate the sum of rewards over all iterations, not only the ones where a_i was selected.

As the final strategy, after all iterations are executed, the algorithm computes the *average strategy* of the Exp3 algorithm over all iterations for each player. Let σ_i^t be the strategy used at time t. After T iterations in a particular node, the average strategy is

$$\bar{\sigma}_i^T(a_i) = \frac{1}{T} \sum_{t=1}^{T} \sigma_i^t(a_i). \tag{12}$$

In our implementation, we maintain the cumulative sum and normalize it to obtain the average strategy.

Previous work [8] suggests removing the samples caused by the exploration first. This modification proved to be useful also in our experiments and it has been shown not to reduce the performance substantially in the worst case [59], so as the resulting final mixed strategy, we use

$$\bar{\sigma}_i(a_i) \leftarrow \max\left(0, \bar{\sigma}_i(a_i) - \frac{\gamma}{|\mathcal{A}_i(s)|}\right), \tag{13}$$

normalized to sum to one.

4.4.3. Regret matching

The last selection function we propose is inspired by regret matching [73], which forms the bases of the successful algorithms for solving imperfect information games [28]. This variant applies regret matching to the current estimated matrix game at each stage and was first used in [11]. The statistics stored by this algorithm in each node are the visit count of each joint action ($n_{a_1a_2}$) and the sum of rewards for each joint action ($X_{a_1a_2}$).[5] Furthermore, the algorithm for each player i maintains a cumulative regret $r^i_{a_i}$ for having played σ^t_i instead of $a_i \in \mathcal{A}_i(s)$ on iteration t, initially set to 0. The regret values $r^i_{a_i}$ are maintained separately by each player. However, the updates use a value that is a function of the joint action space.

On iteration t, function Select first builds each player's current strategies from the cumulative regrets. Define $x^+ = \max(x, 0)$,

$$\sigma_i(a_i) = \frac{r^{i+}_{a_i}}{R^+_{sum}} \text{ if } R^+_{sum} > 0 \text{ otherwise } \frac{1}{|\mathcal{A}_i(s)|}, \text{ where } R^+_{sum} = \sum_{b_i \in \mathcal{A}_i(s)} r^{i+}_{b_i}. \tag{14}$$

The main idea is to adjust the strategy by assigning the probability to actions proportionally to the regret of having not taken them over the long-term. To ensure exploration, a sampling procedure similar to Equation (8) is used to select action a_i with probability $\gamma/|\mathcal{A}_i(s)| + (1-\gamma)\sigma_i(a_i)$.

Update adds the regret accumulated at the iteration to the regret tables r^i. Suppose joint action (a_1, a_2) is sampled from the selection policy and utility u_1 is returned from the recursive call on line 9. Label $reward(b_1, b_2) = \frac{X_{b_1b_2}}{n_{b_1b_2}}$ if $(b_1, b_2) \neq (a_1, a_2)$, or u_1 otherwise. The updates to the regret are:

$$\forall b_1 \in \mathcal{A}_1(s), \;\; r^1_{b_1} \leftarrow r^1_{b_1} + (reward(b_1, a_2) - u_1), \tag{15}$$

$$\forall b_2 \in \mathcal{A}_2(s), \;\; r^2_{b_2} \leftarrow r^2_{b_2} - (reward(a_1, b_2) - u_1). \tag{16}$$

After all simulations, the strategy to play in state s is defined by the mean strategy used in the corresponding node (Equation (12)).

4.4.4. Theoretical properties

While the completeness of the exact algorithms is based on the Markov property and backward induction, the concept of the completeness is less clear for the sampling algorithms due to the randomization. Instead, we discuss a form of a probabilistic completeness. Unfortunately, none of the variants of this algorithm introduced above has been proven to eventually converge to a Nash equilibrium. If the algorithm is instantiated by UCT, Shafiei et al. [14] have shown that the algorithm converges to a stable strategy, which is not close to a Nash equilibrium. We replicate the experiment below and note that this is the case only for the deterministic version of UCT. A randomized version of UCT with a well selected exploration parameter empirically converges close to the equilibrium strategy, but then in some games oscillates and does not converge further.

The only known theoretical result about SM-MCTS directly applicable to the algorithms in this paper is negative, and it has been proven in [59].

Theorem 4.5. *There are games, in which SM-MCTS instantiated by any regret minimizing selection function with a constant exploration γ cannot converge to a strategy that would be an ϵ-Nash equilibrium for an $\epsilon < \gamma D$, where D is the depth of the game tree.*

The main idea of the proof is to define a specific class of games (see Example 2 in [59]), in which the exploration in a greater depth of the game tree causes a bias in the values observed in the higher levels of the tree, consequently leading to an incorrect decision in the root.

In order to obtain positive formal results about the convergence of SM-MCTS-like algorithms, the authors in [59] either add an additional averaging step to the algorithm (that makes it significantly slower in practical games used in benchmarks), or assume additional non-trivial technical properties about the selection function, which are not known to hold for any of the selection functions above.

As for computational complexity, the time cost per node is linear in $|\mathcal{A}_i|$ for UCT and RM. The time cost per node is quadratic in the case of Exp3 due to the numerically stable update rule (Equation (9)). The memory required per node is linear for UCT and Exp3, and quadratic in $|\mathcal{A}_i|$ for RM due to storing estimates of each child subgame. This can be easily avoided by storing the mean estimates directly in the children.

[5] Note that $n_{a_1a_2}$ and $X_{a_1a_2}$ correspond to $n_{s'}$ and $X_{s'}$ from Fig. 6.

4.5. Counterfactual regret minimization and outcome sampling

Finally, we describe algorithms based directly on Counterfactual Regret (CFR, a notion of regret at the information set level), first designed for extensive-form games with imperfect information [28].

Recall from Section 2 the set of histories $\mathcal{H}$. Here we also use $\mathcal{Z}$ defined previously as the set of terminal states, to refer to the set of *terminal histories* since there is a one-to-one correspondence between them. A *history* is a sequence of actions taken by all players (including chance) that starts from the beginning of the game. A history h' is a prefix of another history h, denoted $h' \sqsubset h$, if h contains h' as a prefix sequence of actions. The *counterfactual value* of reaching information set I is the expected payoff given that player i played to reach I, the opponent played σ_{-i} and both players played σ after I was reached:

$$v_i(I,\sigma) = \sum_{(h,z)\in\mathcal{Z}_I} \pi^{\sigma}_{-i}(h)\pi^{\sigma}(h,z)u_i(z), \tag{17}$$

where $\mathcal{Z}_I = \{(h,z) \mid z \in \mathcal{Z}, h \in I, h \sqsubset z\}$, $\pi^{\sigma}_{-i}(h)$ is the product of probabilities to reach h under σ excluding player i's (i.e., including chance) and $\pi^{\sigma}(h,h')$, where $h \sqsubset h'$, is the probability of all actions taken along the path from h to h'. Suppose, at time t, players play with strategy profile σ^t. Define $\sigma^t_{I\to a}$ as identical to σ^t_i except at I action a is taken with probability 1. Player i's counterfactual regret of not taking $a \in \mathcal{A}(I)$ at time t is $r^t_i(I,a) = v_i(I,\sigma^t_{I\to a}) - v_i(I,\sigma^t)$. The CFR algorithm maintains the cumulative regret $R^T_i(I,a) = \sum_{t=1}^{T} r^t_i(I,a)$, for every action at every information set. Then, the distribution at each information set for the next iteration $\sigma^{T+1}(I)$ is obtained individually using regret-matching [73]. The distribution is proportional to the positive portion of the individual actions' regret:

$$\sigma^{T+1}(I,a) = \begin{cases} R^{T,+}_i(I,a)/R^{T,+}_{i,sum}(I) & \text{if } R^{T,+}_{i,sum}(I) > 0 \\ 1/|\mathcal{A}(I)| & \text{otherwise,} \end{cases}$$

where $x^+ = \max(0,x)$ for any term x, and $R^{T,+}_{i,sum}(I) = \sum_{a'\in A(I)} R^{T,+}_i(I,a')$. Furthermore, the algorithm maintains for each information set the average strategy profile

$$\bar{\sigma}^T(I,a) = \frac{\sum_{t=1}^{T} \pi^{\sigma^t}_i(I)\sigma^t(I,a)}{\sum_{t=1}^{T} \pi^{\sigma^t}_i(I)}, \tag{18}$$

where $\pi^{\sigma^t}_i(I) = \sum_{h\in I} \pi^{\sigma^t}_i(h)$. The combination of the counterfactual regret minimizers in individual information sets also minimizes the overall average regret [28], and hence due to the Folk Theorem the average profile is a 2ϵ-equilibrium, with $\epsilon \to 0$ as $T \to \infty$.

Monte Carlo Counterfactual Regret Minimization (MCCFR) applies CFR to sampled portions of the games [58]. In the *outcome sampling* (OS) variant, a single terminal history $z \in \mathcal{Z}$ is sampled in each iteration. The algorithm updates the regret in the information sets visited along z using the *sampled counterfactual value*,

$$\tilde{v}_i(I,\sigma) = \begin{cases} \frac{1}{q(z)}\pi^{\sigma}_{-i}(h)\pi^{\sigma}(h,z)u_i(z) & \text{if } (h,z) \in \mathcal{Z}_I \\ 0 & \text{otherwise,} \end{cases} \tag{19}$$

where $q(z)$ is the probability of sampling z. As long as every $z \in Z$ has a non-zero probability of being sampled, $\tilde{v}_i(I,\sigma)$ is an unbiased estimate of $v(I,\sigma)$ due to the importance sampling correction $(1/q(z))$. For this reason, applying CFR updates using these sampled counterfactual regrets $\tilde{r}^t_i(I,a) = \tilde{v}_i(I,\sigma^t_{I\to a}) - \tilde{v}_i(I,\sigma^t)$ on the sampled information sets values also eventually converges to the approximate equilibrium of the game with high probability. The required number of iterations for convergence is much larger, but each iteration is much faster.

4.5.1. Online Outcome Sampling

We now present Online Outcome Sampling for simultaneous move games (SM-OOS). Note, importantly, that SM-OOS is different from the general SM-MCTS algorithms presented in Subsection 4.4. SM-OOS is an adaptation of a more general algorithm which has been proposed for search in imperfect information games [13]. However, since simultaneous move games are decomposable into subgames, the typical problems encountered in the fully imperfect information search setting are not present here. Hence, we present a simpler OOS specifically intended for simultaneous move games.

Online Outcome Sampling resembles MCTS in that it builds its tree incrementally. However, the algorithm is based on MCCFR, from Subsection 4.5, rather than on stochastic and adversarial bandit algorithms, such as UCB and Exp3. A previous version of this algorithm for simultaneous move games was presented by Lanctot et al. [11]. The version presented here is simpler for implementation and it further reduces the variance of the regret estimates, which leads to a faster convergence and better game play. The main novelty in this version is that in any state s, it defines the counterfactual values as if the game actually started in s. This is possible in simultaneous move games, because the optimal strategy in any state depends only on the part of the game below the state.

The pseudocode is given in Algorithm 6. The game tree is incrementally built, starting only with one node for the root game state. Each node stores for each player: $R_i(s,a)$ the cumulative regret (denoted $R^T_i(I,a)$ above) of player i in state s

input : s – current state of the game; i – regret updating player
output: (x_i, q_i, u_i): x_i – i's contribution to tail probability ($\pi^\sigma(h,z)$); q_i – i's contribution to sample probability ($q(z)$); u_i – utility of the sampled leaf

1 **if** $s \in \mathcal{Z}$ **then return** $(1, 1, u_i(s))$
2 **else if** $s \in \mathcal{C}$ *is a chance node* **then**
3 Sample s' from $\Delta_\star(s)$
4 **return** SM-OOS(s', i)
5 **if** s *is already in the OOS tree* **then**
6 $\sigma_i \leftarrow$ RegretMatching$(R_i(s))$
7 $\forall a \in \mathcal{A}_i(s) : \sigma'_i(s,a) \leftarrow (1-\epsilon)\sigma_i(s,a) + \frac{\epsilon}{|\mathcal{A}_i(s)|}$
8 Sample action a_i from σ'_i
9 $\sigma_{-i} \leftarrow$ RegretMatching$(R_{-i}(s))$
10 Sample action a_{-i} from σ_{-i}
11 $(x_i, q_i, u_i) \leftarrow$ SM-OOS$(\mathcal{T}(s, a_i, a_{-i}), i)$
12 **else**
13 Add s to the tree
14 $\forall a \in \mathcal{A}_i(s) : \sigma_i(s,a) \leftarrow \frac{1}{|\mathcal{A}_i(s)|}$
15 Sample action a_i from σ_i
16 $\forall a \in \mathcal{A}_{-i}(s) : \sigma_{-i}(s,a) \leftarrow \frac{1}{|\mathcal{A}_{-i}(s)|}$
17 Sample action a_{-i} from σ_{-i}
18 $(x_i, q_i, u_i) \leftarrow$ OOS-Rollout$(\mathcal{T}(s, a_i, a_{-i}))$
19 $W \leftarrow u_i \cdot x_i / q_i$
20 $R_i(s,a_i) \leftarrow R_i(s,a_i) + \frac{1-\sigma_i(s,a_i)}{\sigma'_i(a_i)} W$
21 $\forall a \in \mathcal{A}_i(s) \setminus \{a_i\} : R_i(s,a) \leftarrow R_i(s,a) - \frac{\sigma_i(s,a_i)}{\sigma'_i(s,a_i)} W$
22 $S_{-i}(s) \leftarrow S_{-i}(s) + \sigma_{-i}$
23 **return** $(x \cdot \sigma_i(s,a_i), q \cdot \sigma'_i(s,a_i), u_i)$

Algorithm 6: Simultaneous Move Online Outcome Sampling (SM-OOS).

and action a, and average strategy table $S_i(s)$, which stores the cumulative average strategy contribution for each action. Normalizing S_i gives the resulting strategy of the algorithm for player i.

The algorithm runs iterations from a starting state until it uses the given time limit. A single iteration is depicted in Algorithm 6, which recursively descends down the tree. In the root of the game, the function is run as SM-OOS$(root, i)$, alternating player $i \in \{1, 2\}$ in each iteration. If the function reaches a terminal history of the game (line 1), it returns the utility of the terminal node for player i, and 1 for both the tail and sample probability contribution of i. If it reaches a chance node, it recursively continues after a randomly selected chance outcome (lines 3–4). If none of the first two conditions holds, the algorithm reaches a state where the players make decisions. If this state is already included in the incrementally built tree (line 5), the following state is selected based on the cumulative regrets stored in the tree by regret matching with ϵ-on-policy sampling strategy for player i (lines 6–8) and the exact regret matching strategy for player $-i$ (lines 9–11). The recursive call on line 11 then continues the iteration until the end of the game tree. If the reached node is not in the tree, it is added (line 13) and an action for each player is selected based on the uniform distribution (lines 14–16). Afterwards, a random rollout of the game until a terminal node is initiated on line 18. The rollout is similar to the MCTS case, but in addition, it has to compute the tail probability x_i and the sampling probability q_i required to compute the sampled counterfactual value. For example, if in the rollout player i acts n_i times, and each time samples uniformly from exactly b actions, then $x_i = \frac{1}{b^{n_i}}$. Regardless of whether the current node was in the tree, the algorithm updates the regret table of player i based on the simplified definition of sampled counterfactual regret for simultaneous move games (lines 19–21) and the mean strategy of player $-i$ (line 22). Finally, the function returns the updated probabilities to the upper level of the tree.

SM-OOS appears similar to SM-MCTS using the RM selection mechanism (Subsection 4.4.3). However, there are a number of differences: SM-OOS uses importance sampling of a sequence of probabilities to keep its estimate unbiased, but will suffer a higher variance than RM which uses only a one-step correction. RM does not distinguish whether its utility comes from exploration or otherwise, whereas SM-OOS separates the two into the tail probabilities of the strategy for the sequence sampled (x_i) and the sampling probability of the sequence (q_i); when $\sigma_i(s,a) = 0$, due to exploration, then $x_i = 0$ and the value of the update increments are also 0. RM uses the means from the subgames as estimates of utility for those subgames, which could introduce some bias in the estimators. We further discuss the comparison in Subsection 6.6.

4.5.2. Theoretical properties

SM-OOS, contrary to the MCTS-based algorithms, has finite-time probabilistic convergence guarantees. Since SM-OOS is designed to update each node of the game in the same way as the root of the game, we present the following theorem from the perspective of the root of the entire game. It holds also for starting the algorithm in non-root nodes, but the values of $|S|$ and δ can be adapted to represent the subgame.

Theorem 4.6. *When SM-OOS is run from the root of the game, with probability* $(1-p)$ *an* ϵ*-NE is reached after* $O(\frac{|\mathcal{A}||\mathcal{S}|^2\Delta_{u,i}^2}{p\delta^2\epsilon^2})$ *iterations, where* $|\mathcal{A}| = \max_{s\in\mathcal{S},i\in\{1,2\}}|\mathcal{A}_i(s)|$, $\Delta_{u,i} = \max_{z,z'\in\mathcal{Z}}|u_i(z') - u_i(z)|$, *and* δ *is the smallest probability of sampling any single leaf in the subtree of the root node.*

Proof. The proof is composed of two observations. First, the whole game tree is eventually built by the algorithm. A direct consequence of [59, Lemma 40] is that the tree of depth D is built with probability $(1-p_1)$ in less than

$$16D\left(\frac{|\mathcal{A}|}{\gamma}\right)^{2D}\max(D, 4\log p_1^{-1}+4) \tag{20}$$

iterations by an algorithm with a fixed exploration γ. This is the number of iterations needed for each leaf in the game to be visited at least D times.

Second, during these and the following iterations, the algorithm performs exactly the same updates in the nodes contained in memory, as the Outcome Sampling (OS) MCCFR [58]. If some nodes below a state were not added to the tree yet, a uniform strategy is assumed in these states for the regret updates. Since CFR minimizes the counterfactual regret in an individual information set regardless of the strategies in other information sets, the samples acquired during the tree building cannot have a negative impact on the rate of regret minimization in individual states. Therefore, we can use [74, Theorem 4] that bounds the number of iterations needed for OS as an offline solver with the complete game in the memory, starting after the tree has been built with a high probability. It states that with probability $(1-p_2)$ an ϵ-NE is reached after $O(\frac{|\mathcal{A}||\mathcal{S}|^2\Delta_{u,i}^2}{p_2\delta^2\epsilon^2})$ iterations.

We can see that the OS bound dominates the time required to build the tree. A single explorative action is taken with probability $\gamma/|\mathcal{A}|$, and when sampling a terminal z only due to exploration, $\frac{1}{\delta} = (\frac{|\mathcal{A}|}{\gamma})^{2D}$, and $D^2 < |\mathcal{A}|^{2D} \in O(|\mathcal{S}|)$ for any $\mathcal{A}$, and we can set $p_1 = p_2 = p/2$. Then the probability that both the tree will be built and the convergence will be achieved can be bounded by $(1-p_1)(1-p_2) \geq (1-p)$. □

As for computational complexity, the time cost as well as the memory required per node is linear in $|\mathcal{A}_i|$ in SM-OOS.

5. Online search

In this section, we describe online adaptations of the algorithms described in the previous section and their application to any-time search given a limited time budget.

5.1. Iterative deepening backward induction algorithms

Minimax search [5] has been used with much success in sequential perfect information games, leading to super-human chess AI, one of the key advances of artificial intelligence [1]. Minimax search is an online application of backward induction run on a heuristically approximated game. The game is approximated by searching to a fixed depth limit d, treating the states at depth d as terminal states, evaluating their values using a heuristic evaluation function, $eval(s)$. The main focus is to compute an optimal strategy for this heuristic approximation of the original game.

Similarly to the perfect information case, we can modify our algorithms based on backward induction for simultaneous move games. Under the limited time settings, a search algorithm is given a fixed time budget to compute a strategy. We use the classic approach of *iterative deepening* [5] that runs several depth-limited searches, starting at a low depth and iteratively increasing the depth of each successive search. Note that the depth limit of d means that the algorithm evaluates d joint actions (i.e., pairs of simultaneous actions) possibly preceded by a chance outcome if present.

In iterative deepening, the algorithm by default starts at depth $d = 1$ and gradually increases d until there is no more time. In our implementation of iterative deepening we follow a natural observation that saves the computation time *between different searches*: a solution computed in state s by player i to depth d contains an optimal solution on $d-1$ approximation of subgames starting in possible next states $\mathcal{T}(s,r,c)$, where r is the action selected for the player performing the search and c is the action of the opponent. Therefore, when the iterative deepening algorithm starts a new search in state $s' \in \mathcal{T}(s,r,c)$, it can often begin at depth d. This can require space exponential in the depth d in the worst case, but it is beneficial in practical experiments. When information is missing due to pruning, then a search starts with the lowest possible depth $d = 1$.

5.2. Online search using sampling algorithms

Using sampling algorithms in the online settings is simpler than with the algorithms based on backward induction, since no significant changes are needed and the algorithms do not need an evaluation function. The algorithms are stopped after a given time limit and the move to play or the complete strategy is extracted as described for each sampling algorithm in Section 4.

There are two concepts that have to be discussed. First, the algorithms can re-use all information and statistics gained in the previous iterations; hence, after returning a move and advancing to a succeeding state of the game s', the subtree of the incrementally built tree rooted in s' is preserved and used in the next iterations. Note that reusing the previously gathered statistics in the sub-tree rooted in s' has no potentially negative effect on any variant of the MCTS algorithms since the behavior of the algorithms is exactly the same when the iteration is started in this node, and if this node is reached from its predecessor. This is also true in SM-OOS because of the structure of simultaneous move games; a similar adaptation of the algorithm is not possible in more general imperfect information games [13].

Second, even though the sampling algorithms do not require the use of domain-specific knowledge for online search, they often incorporate this type of knowledge to better guide the sampling and thus to evaluate more relevant parts of the state space [75–79]. When directly comparing approximative sampling algorithms with the backward induction algorithms using an evaluation function, the outcome of such a comparison strictly depends on the quality of the evaluation function. In a very large game, an accurate evaluation function greatly benefits the backward induction algorithm. Therefore, we also use sampling algorithms combined with an evaluation function. The integration is done via replacing the random rollout by directly using the value of the evaluation function in the current state for MCTS and OOS algorithms; i.e., Rollout(s) in line 14 of Algorithm 5 or line 18 of Algorithm 6 is replaced by *eval*(s). This has been commonly used in several previous works in Monte Carlo search [76,78–81].

Again, such a modification does not generally affect theoretical properties of the algorithms – the proofs of the convergence assume that a whole game tree is eventually built and any statistics in the nodes collected before (either by random rollouts or evaluation functions) can eventually be over-weighted. For MCTS algorithms, there is no reason to believe that a good evaluation function would give a worse estimate of the quality of a sub-tree using random play-outs. The only complication could be with the way the probabilities are computed in OOS. The weight of the sample in Equation (19) is multiplied by the probability of reaching the terminal state z from some history h, $\pi^{\sigma}(h, z)$. However, the "tail" probability is canceled because the rollout policy is fixed and so its contribution to $q(z)$ is identical to its contribution to $\pi^{\sigma}(h, z)$.

6. Empirical evaluation

We now present a thorough experimental evaluation of the described algorithms. We analyze both the offline and the online case on a collection of games inspired by previous work, and randomly generated games. After describing rules and properties of the games, we present the results for the offline strategy computation and we follow with the online game playing.

6.1. Experimental settings

We start with an experimental evaluation of a well-known example of Biased Rock, Paper, Scissors [14] that often serves as an example that MCTS with UCT selection function does not converge to a Nash equilibrium. We reproduce this experiment and show the differences in performance of the sampling algorithms – primarily the impact of randomization in UCT. Then, we compare the offline performance of the algorithms on other domains. For each domain, we first analyze the exact algorithms and measure the computation time taken to solve a particular instance of the game. Afterward, we analyze the convergence of the approximative algorithms. At a specified time step the algorithm produces strategies (σ_1, σ_2). Using best responses we compute $\text{error}(\sigma_1, \sigma_2) = \max_{\sigma'_1 \in \Sigma_1} \mathbb{E}_{z \sim (\sigma'_1, \sigma_2)}[u_1(z)] + \max_{\sigma'_2 \in \Sigma_2} \mathbb{E}_{z \sim (\sigma_1, \sigma'_2)}[u_2(z)]$, which is equal to 0 at a Nash equilibrium. In each offline convergence setting, the reported values are means over at least 20 runs of each sampling algorithm on a single instance of the game. We compared at least 3 different settings for each exploration parameter and present the result only for the best exploration parameter. For OOS, Exp3, and RM the best values for the parameters were almost always 0.6, 0.1, and 0.1, respectively. The only exception was Goofspiel with chance, where both Exp3 and RM converge faster with the parameter set to 0.3. We give the optimal value for UCT constant C in each setting.

Finally, we turn to the comparison of the algorithms in the online setting and we present results from head-to-head tournaments in each game. Here, we use larger instances of each game and compare the algorithms based on actual game play with a limited time for each move. The algorithms based on backward induction need to use a domain-specific evaluation function in the online setting. This may give these algorithms an advantage if the evaluation function is accurate. Therefore, we also run the sampling-based algorithms with an evaluation function for selected domains to compare the algorithms in a fairer setting. Moreover, we have also tuned parameters for the sampling algorithms specifically for each domain. Reported results are means over at least 1000 matches for each pair of algorithms.

Each of the described algorithms was implemented in a generic framework for modeling and solving extensive-form games.[6] We are interested in the performance of the algorithms and their ability to find or approximate the optimal behavior. Therefore, with the exception of the evaluation function used in selected online experiments, no algorithm uses any domain-specific knowledge.

[6] Source code is available at the web page of the authors. We use IBM CPLEX 12.5 to solve the linear programs.

	r	p	s
R	0	-25	50
P	25	0	-5
S	-50	5	0

Fig. 7. Biased Rock, Paper, Scissors matrix game from [14].

6.2. Domains

In this subsection, we describe the six domains used in our experiments. The games in our collection differ in characteristics, such as the number of available actions for each player (i.e., the branching factor), the maximal depth, and the number of possible utility values. Moreover, the games also differ in the *randomization factor* – i.e., how often it is necessary to use mixed strategies and whether this randomization occurs at the beginning of the game, near the end of the game, or is spread throughout the whole course of the game.

For each domain we also describe the evaluation function used in the online experiments. Note that we are not seeking the best-performing algorithm for a particular game; hence, we have not aimed for the most accurate evaluation functions for each game. We intentionally use evaluation functions of different quality that allow us to compare the differences between the algorithms from this perspective as well.

Biased Rock, Paper, Scissors. BRPS is a payoff-skewed version of the one-shot game Rock, Paper, Scissors shown in Fig. 7. This game was introduced in [14], and it was shown that the visit count distribution of UCT converges to a fixed balanced situation, but not one that corresponds to the optimal mixed strategy of $(\frac{1}{16}, \frac{10}{16}, \frac{5}{16})$.

Goofspiel. Goofspiel is a card game that appears as a common example of a simultaneous move game (e.g., [11,35,37,38]). There are 3 identical decks of d cards with values $\{0, \ldots, (d-1)\}$, one for chance and one for each player, where d is a parameter of the game. Standard Goofspiel is played with 13 cards. The game is played in rounds: at the beginning of each round, chance reveals one card from its deck and both players bid for the card by simultaneously selecting (and removing) a card from their hands. A player that selects a higher card wins the round and receives a number of points equal to the value of the chance's card. In case both players select the card with the same value, the chance's card is discarded. When there are no more cards to be played, the winner of the game is chosen based on the sum of card values he received during the whole game.

There are two parameters of the game that can be altered to create four different variants of Goofspiel. The first parameter determines whether or not the chance player is included. We can use an assumption made in the previous work that used Goofspiel as a benchmark for evaluation of the exact offline algorithms [38], where the sequence of the cards is randomly chosen at the beginning of the game and it is known to both players. We refer to this setting as the *fixed sequence* of cards. Alternatively, we can treat chance in the standard way, where chance nodes determine the card that gets drawn. We refer to this setting as the *stochastic sequence*. The games are fairly similar in terms of performance of the algorithms, however, the second variant induces a considerably larger game tree. The second parameter relates to the utility functions. Either we treat the game as a win–tie–lose game (i.e., the players receive utility from $\{-1, 0, 1\}$), or the utility values for the players are equal to the points they gain during the game.

Goofspiel forms game trees with interesting properties. First unique feature is that the number of actions for each player is uniformly decreasing by 1 with the depth. Secondly, algorithms must randomize in NE strategies, and this randomization is present throughout the whole course of the game. As an example, the following table depicts the number of states with pure strategies and mixed strategies for each depth in a subgame-perfect NE calculated by backward induction for Goofspiel with 5 cards and a fixed sequence of cards:

Depth	0	1	2	3	4
Pure	0	17	334	3,354	14,400
Mixed	1	8	66	246	0

We can see that the relative number of states with mixed strategies slowly decreases, however, players need to mix throughout the whole game. In the last round, each player has only a single card; hence, there cannot be any mixed strategy.

Our hand-tuned evaluation function used in Goofspiel takes into consideration the remaining cards in the deck weighted by a chance of winning these cards depending on the remaining cards in hand for each player. Moreover, if the position is clearly winning for one of the players (there is not enough cards to change the current score), the evaluation function is set to maximal (or minimal) value. The formal definition follows (c_i is the sum of values of the remaining cards of player i):

$$eval(s) = \begin{cases} u_1(s) & \text{if } c_1 + c_2 = 0\,; \\ \tanh\left(\frac{c_1 - c_2}{c_1 + c_2} \cdot \frac{c_\star}{0.5 \cdot d(d+1)}\right) & \text{otherwise.} \end{cases}$$

For the win–tie–lose case we use tanh to scale the evaluation function into the interval $[-1, 1]$; this function is omitted in the exact point case.

Oshi-Zumo. Oshi-Zumo (also called *Alesia* in [22]) is a board game that has been analyzed from the perspective of computational game theory in [36]. There are two players in the game, both starting with N coins, and there is a board represented as a one-dimensional playing field with $2K + 1$ locations (indexed $0, \ldots, 2K$). At the beginning, there is a stone (or a *wrestler*) located in the center of the playing field (i.e., at position K). During each move, both players simultaneously place their bid from the amount of coins they have (but at least M if they still have some coins). Afterward, the bids are revealed, both bids are subtracted from the number of coins of the players, and the highest bidder can push the wrestler one location towards the opponent's side. If the bids are the same, the wrestler does not move. The game proceeds until the money runs out for both players, or the wrestler is pushed out of the field. The winner is determined based on the position of the wrestler – the player in whose half the wrestler is located loses the game. If the final position of the wrestler is the center, the game is a draw. Again, we have examined two different settings of the utility values: they are either restricted to win–tie–lose values $\{-1, 0, 1\}$, or they correspond to the relative position of the wrestler $\{\text{wrestler} - K, K - \text{wrestler}\}$. In the experiments we varied the number of coins and parameter K.

Many instances of the Oshi-Zumo game have a pure Nash equilibrium. With the increasing number of the coins the players need to use mixed strategies, however, mixing is typically required only at the beginning of the game. As an example, the following table depicts the number of states with pure strategies and mixed strategies in a subgame-perfect NE calculated by backward induction for Oshi-Zumo with $N = 10$ coins, $K = 3$, and minimal bid $M = 1$. The results show that there are very few states where mixed strategies are required, and they are present only at the beginning of the game tree. Also note, that contrary to Goofspiel, not all branches have the same length.

Depth	0	1	2	3	4	5	6	7	8	9
Pure	1	98	2,012	14,767	48,538	79,926	69,938	33,538	8,351	861
Mixed	0	1	4	17	8	0	0	0	0	0

The evaluation function used in Oshi-Zumo takes into consideration two components: (1) the current position of the wrestler and, (2) the remaining coins for each player. Formally:

$$eval(s) = \tanh\left(\frac{b}{2} + \frac{1}{3}\left(\frac{\text{coins}_1 - \text{coins}_2}{M} + \text{wrestler} - K\right)\right),$$

where $b = 1$ if $\text{coins}_1 \geq \text{coins}_2$ and $\text{wrestler} \geq K$, and at least one of the inequalities is strict; or $b = -1$ if $\text{coins}_1 \leq \text{coins}_2$ and $\text{wrestler} \leq K$, and at least one of the inequalities is strict; $b = 0$ otherwise. Again, we use tanh to scale the value into the interval $[-1, 1]$ only in the win–tie–lose case.

Pursuit–evasion games. Another important class of games is pursuit–evasion games (for example, see [82]). There is a single evader and a pursuer that controls 2 pursuing units on a four-connected grid in our pursuit–evasion game. Since all units move simultaneously, the game has larger branching factor than Goofspiel (up to 16 actions for the pursuer). The evader wins if she successfully avoids the units of the pursuer for the whole game. The pursuer wins if her units successfully capture the evader. The evader is captured if either her position is the same as the position of a pursuing unit, or the evader used the same edge as a pursuing unit (in the opposite direction). The game is win–loss and the players receive utility from the set $\{-1, 1\}$. We use 3 different square four-connected grid-graphs (with the size of a side 4, 5, and 10 nodes) for the experiments without any obstacles or holes. In the experiments we varied the maximum length of the game d and we altered the starting positions of the players (the distance between the pursuers and the evader was always at most $\left\lfloor \frac{2}{3}d \right\rfloor$ moves, in order to provide a possibility for the pursuers to capture the evader).

Similarly to Oshi-Zumo, many instances of pursuit–evasion games have a pure Nash equilibrium. However, the randomization can be required towards the actual end of the game in order to capture the evader. Therefore, depending on the length of the game and the distance between the units, there might be many states that do not require mixed strategies (the units of the pursuers are simply going towards the evader). Once the units are close to each other, the game may require mixed strategies for the final coordination. This can be seen on our small example on a graph with 4×4 nodes and depth 5:

Depth	0	1	2	3	4
Pure	1	12	261	7,656	241,986
Mixed	0	0	63	1,008	6,726

The evaluation function used in pursuit–evasion games takes into consideration the distance between the units of the pursuer and the evader (denoted distance_j for the distance in moves of the game between the jth unit of the pursuer and the evader). Formally:

$$eval(s) = \frac{\min(\text{distance}_1, \text{distance}_2) + 0.01 \cdot \max(\text{distance}_1, \text{distance}_2)}{1.01 \cdot (w + l)},$$

where w and l are dimensions of the grid graph.

Random/synthetic games. Finally, we also use randomly generated games to be able to experiment with additional parameters of the game, mainly larger utility values and their correlation. In randomly generated games, we fixed the number of actions that the players can play in each stage to 4 and 5 (the results were similar for different branching factors) and we varied the depth of the game tree. We use 2 different methods for randomly assigning the utility values to the terminal states of the game: (1) the utility values are uniformly selected from the interval $[0, 1]$; (2) we randomly assign either -1, 0, or $+1$ value to each joint action (pair of actions) and the utility value in a leaf is a sum of all the values on the edges on the path from the root of the game tree to the leaf. The first method produces extremely difficult games for pruning using either alpha-beta, or the double-oracle algorithm, since there is no correlation between actions and utility values in sibling leaves. The latter method is based on random *P-games* [83] and creates more realistic games using the intuition of good and bad moves.

Randomly generated games represent games that require mixed strategies in most of the states. This holds even for the games of the second type with correlated utility values in the leaves. The following table shows the number of states depending on the depth for a randomly generated game of depth 5 with 4 actions available to both players in each state:

Depth	0	1	2	3	4
Pure	0	2	29	665	20,093
Mixed	1	14	227	3,431	45,443

Only the second type of randomly generated games is used in the online setting. The evaluation function used in this case is computed similarly to the utility value and it is equal to the sum of values on the edges from the root to the current node.

Tron. Tron is a two-player simultaneous move game played on a discrete grid, possibly obstructed by walls [55,57,60]. At each step, both players move to adjacent nodes and a wall is placed to the original positions of the players. If a player hits the wall or the opponent, the game ends. The goal of both players is to survive as long as possible. If both players move into a wall, off the board, or into each other on the same turn, the game ends in a draw. The utility is $+1$ for a win, 0 for a draw, and -1 for a loss. In the experiments, we used an empty grid with no obstacles and various sizes of the grid.

Similarly to pursuit–evasion games, there are many instances of Tron that have pure NE. However, even if mixed strategies are required, they appear in the middle of the game once both players reach the center of the board and compete over the advantage of possibly being able to occupy more squares. Once this is determined, the endgame can be solved in pure strategies since it typically consists of filling the available space in an optimal ordering one square at a time. The following table comparing the number of states demonstrates this characteristics of Tron on a 5×6 grid:

Depth	0	1	2	3	4	5	...
Pure	1	4	14	100	565	2,598	
Mixed	0	0	2	0	9	7	

...	6	7	8	9	10	11	12	13
	9,508	25,964	54,304	83,624	87,009	63,642	23,296	3,127
	51	92	106	121	74	0	0	0

The evaluation function is based on how much space is "owned" by each player, which is a more accurate version of the space estimation heuristic [84] that was used in [60]. A cell is owned by player i if it can be reached by player i before the opponent. These values are computed using an efficient flood-fill algorithm whose sources start from the two players' current positions:

$$eval(s) = \tanh\left(\frac{\text{owned}_1 - \text{owned}_2}{5}\right).$$

6.3. Non-convergence and random tie-breaking in UCT

We first revisit the counter-example given in [14] showing that UCT does not converge to an equilibrium strategy in Biased Rock, Paper, Scissors when using a mixed strategy created by normalizing the visit counts. We expand on this result, showing the effect of the synchronization occurring when the UCT selection mechanism is fully deterministic (see Subsection 4.4.1).

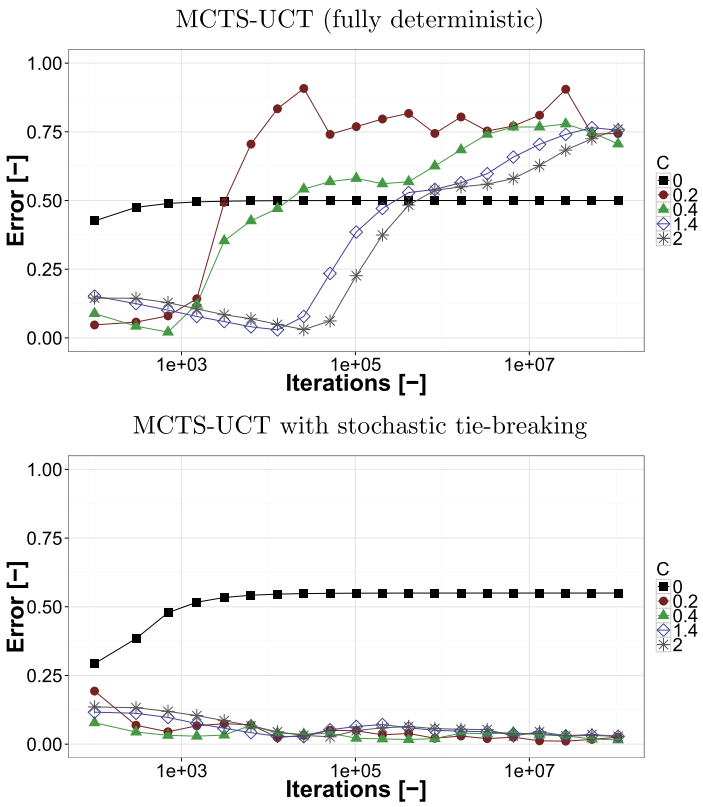


Fig. 8. Exploitability of strategies of recommended by MCTS-UCT over time in Biased Rock, Paper, Scissors. Vertical axis represents exploitability.

We run SM-MCTS with UCT, Exp3, and Regret Matching selection functions on Biased Rock, Paper, Scissors for 100 million (10^8) iterations, measuring the exploitability of the strategy recommended by each variant at regular intervals. The results are shown in Figs. 8 and 9.

The first observation is that deterministic UCT does not seem to converge to a low-exploitability strategy (see Fig. 8, top figure). The exploitability of the strategies of Exp3 and RM variants do converge to low-exploitability strategies (see Fig. 9), and the resulting approximation depends on the amount of exploration. If less exploration is used, then the resulting strategy is less exploitable, which is natural in the case of a single state. RM does seem to converge slightly faster than Exp3, as we will see in the remaining domains as well.

We then tried adding a stochastic tie-breaking rule to the UCT selection mechanism typically used in MCTS implementations, which chooses an action randomly when the scores of the best values are "tied" (less than 0.01 apart). The bottom figure in Fig. 8 shows the convergence. One particularly striking observation is that this simple addition leads to a large drop in the resulting exploitability, where the exploitability ranges from $[0.5, 0.8]$ in the deterministic case, compared to $[0.01, 0.05]$ with the stochastic tie-breaking. Therefore, the stochastic tie-breaking is enabled in all of our experiments.

In summary, with this randomization UCT appears to be converging to an approximate equilibrium in this game but not to an exact equilibrium, which is similar to results of a variant of UCT in Kuhn poker [85].

6.4. Offline equilibrium computation

We now compare the offline performance of the algorithm on all the remaining games. We measure the overall computation time for each of the algorithms and the number of evaluated nodes – i.e., the nodes for which the main method of the backward induction algorithm executed (nodes evaluated by serialized alpha-beta algorithms are not included in this count, since they may be evaluated repeatedly). Unless otherwise stated, each data point represents a mean over at least 30 runs.

6.4.1. Goofspiel

We now describe the results for the card game Goofspiel. First, we analyze the games with fixed sequences of the cards.

Exact algorithms with fixed sequences. The results are depicted in Fig. 10 (note the logarithmic vertical scale), where the left subfigure depicts the results for win–tie–lose utilities and the right subfigure depicts the results for point utilities. We present the mean results over 10 different fixed sequences. The comparison on the win–tie–lose variant shows that there is a significant number of subgames with a pure Nash equilibrium that can be computed using the serialized alpha-beta algorithms. Therefore, the performance of BI$\alpha\beta$ and DO$\alpha\beta$ is fairly similar and the gap only slowly increases in favor of

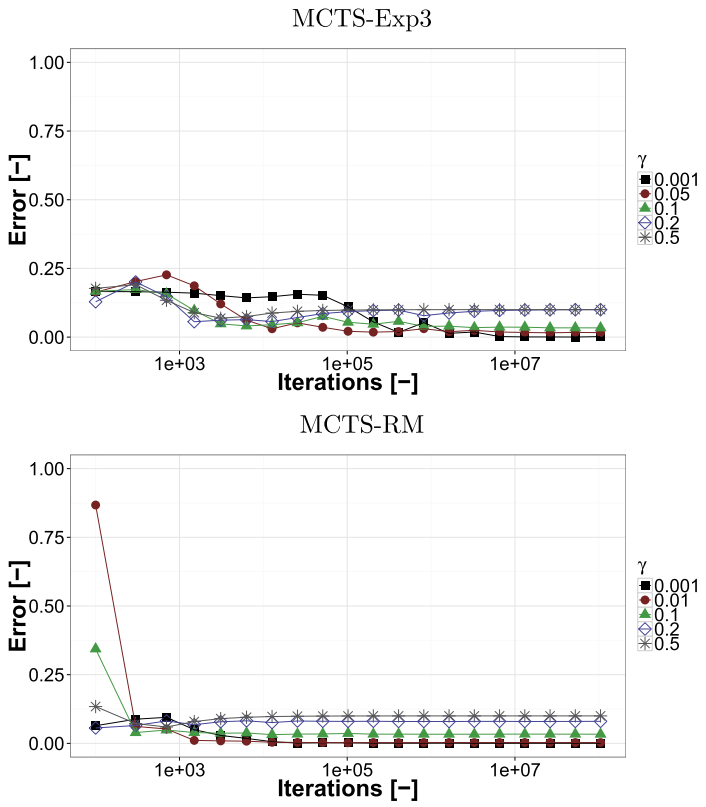


Fig. 9. Exploitability of strategies recommended by MCTS-Exp3 and MCTS-RM over time in Biased Rock, Paper, Scissors. Vertical axis represents exploitability.

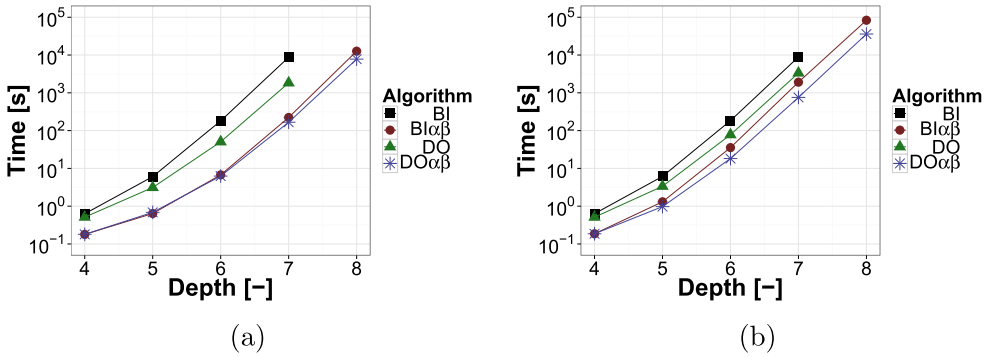


Fig. 10. Running times of the exact algorithms on Goofspiel with fixed sequences of cards for increasing size of the deck; subfigure (a) depicts the results with win–tie–lose utilities, (b) depicts the results with point difference utilities.

DO$\alpha\beta$ with the increasing size of the game. Since serialized alpha-beta is able to solve a large portion of subgames, both of these algorithms significantly reduce the number of the states visited by the backward induction algorithm. While BI evaluates 3.2×10^7 nodes in the setting with 7 cards in more than 2.5 hours, BI$\alpha\beta$ evaluates only 198,986 nodes in less than 4 minutes. The performance is further improved by DO$\alpha\beta$ that evaluates on average 79,105 nodes in less than 3 minutes. The overhead is slightly higher in case of DO$\alpha\beta$; hence, the time difference between DO$\alpha\beta$ and BI$\alpha\beta$ is relatively small compared to the difference in evaluated nodes. Finally, the results show that even the DO algorithm without the serialized alpha-beta search can improve the performance of BI. In the setting with 7 cards, DO evaluates more than 6×10^6 nodes which takes on average almost 30 minutes.

The results for the point utilities are the same for BI, while DO is slightly worse. On the other hand, the success of serialized alpha-beta algorithms is significantly lower and it takes both algorithms much more time to solve the games of the same size. With 7 cards, BI$\alpha\beta$ evaluates more than 2×10^6 nodes and it takes the algorithm on average 32 minutes to find the solution. DO$\alpha\beta$ is still the fastest and it evaluates more than 3×10^5 nodes in less than 13 minutes on average.

The performance of algorithms BI$\alpha\beta$ and DO$\alpha\beta$ represent a significant improvement over the results of the pruning algorithm SMAB presented in [38]. In their work, the number of evaluated nodes was at best around 29%, and the running time improvement was only marginal.

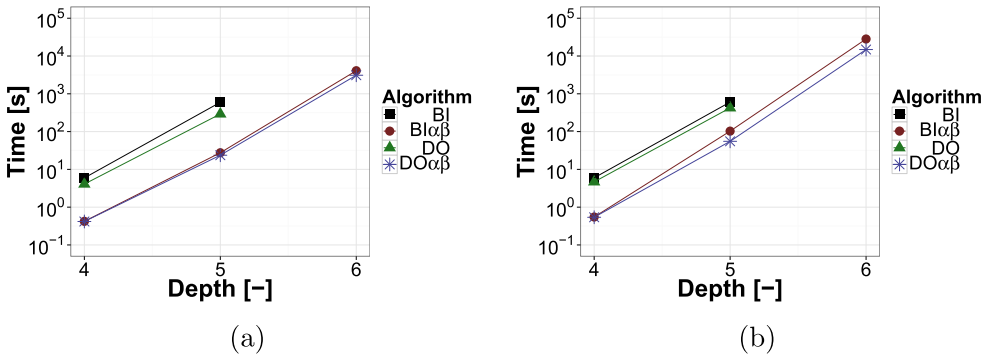


Fig. 11. Running times of exact algorithms on Goofspiel with chance nodes for increasing size of the deck; subfigure (a) depicts the results with win–tie–lose utilities, (b) depicts the results with point utilities.

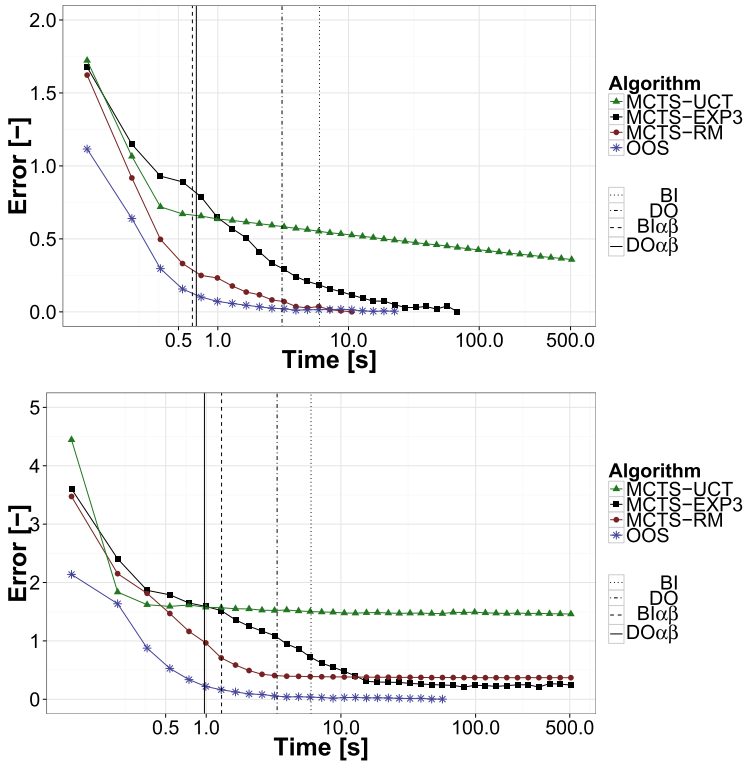


Fig. 12. Convergence of the sampling algorithms on Goofspiel with 5 cards and a fixed sequence of cards. The vertical lines correspond to the computation times for the exact algorithms. (Top) Goofspiel with win–tie–lose utility values; (bottom) Goofspiel with point utilities.

Exact algorithms with a stochastic sequence. Next we compare the exact algorithms in the variant of Goofspiel with standard chance nodes. Introducing another branching due to moves by chance causes a significant increase in the size of the game tree. For 7 cards, the game tree has more than 10^{11} nodes, which is 4 orders of magnitude more than in the case with fixed sequences of cards. The results depicted in Fig. 11 show that the games become quickly too large to solve exactly and the fastest algorithms solved games with at most 6 cards. Relative performance of the algorithms, however, is similar to the case with fixed sequences. With win–tie–lose utilities, serialized alpha-beta is again able to find pure NE in most of the subgames and prunes out a large fraction of the states. For the game with 5 cards, BI evaluates more than 2×10^6 nodes in almost 10 minutes, while BI$\alpha\beta$ evaluates only 17,315 nodes in 27 seconds and DO$\alpha\beta$ evaluates 6,980 nodes in 23 seconds. As before, the serialized alpha-beta algorithm is less helpful in the case with point utilities. Again with 5 cards, BI$\alpha\beta$ evaluates 91,419 nodes in more than 100 seconds and DO$\alpha\beta$ evaluates 14,536 nodes in almost 55 seconds.

Sampling algorithms with fixed sequences. We now turn to the analysis of the convergence of the sampling algorithms – i.e., their ability to approximate Nash equilibrium strategies of the complete game. Fig. 12 depicts the results for Goofspiel game with 5 cards with fixed sequence of cards (note the logarithmic horizontal scale). We compare MCTS algorithms with three different selection functions (UCT, Exp3, and RM), and OOS. The results are means over 30 runs of each algorithm. Due to the different selection and update functions, the algorithms differ in the number of iterations per second. RM is the fastest

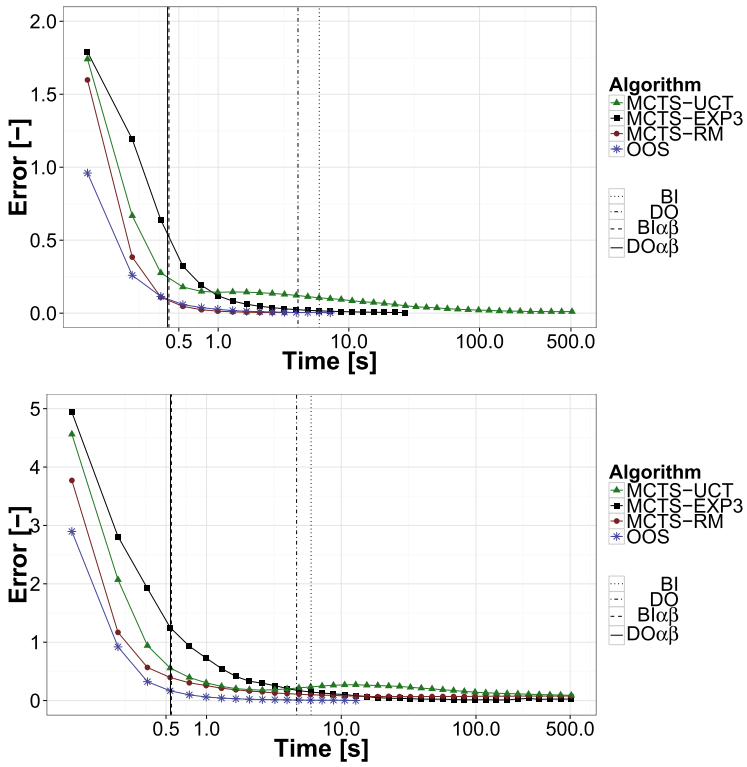


Fig. 13. Convergence of the sampling algorithms on Goofspiel with 4 cards and chance nodes. The vertical lines correspond to the computation times for the exact algorithms. (Top) Goofspiel with win–tie–lose utility values; (bottom) Goofspiel with point utilities.

with more than 2.6×10^5 iterations per second, OOS has around 2×10^5 iterations, UCT 1.9×10^5, and Exp3 only 5.4×10^4 iterations.

The results show that OOS converges the fastest out of all sampling algorithms. This is especially noticeable in the point-utility settings, where none of the other sampling algorithms were approaching zero error due to the exploration. MCTS with RM selection function is only slightly slower in the win–tie–lose case, however, the other two selection functions perform worse. While Exp3 eventually converges close to 0 in the win–tie–lose case, the exploitability of UCT decreases rather slowly and it was still over 0.35 at the time limit of 500 seconds. The best C constant for UCT was 5 in the win–tie–lose setting, and 10 in the point utility setting. While setting lower constant typically improves the convergence rate slightly during the first iterations, the final error was always larger. The vertical lines represent the times for the exact algorithms. In the win–tie–lose case, BI$\alpha\beta$ is slightly faster and finishes first in 0.64 seconds, followed by DO$\alpha\beta$ (0.69 seconds), DO (3.1 seconds), and BI (6 seconds). In the point case, DO$\alpha\beta$ is the fastest (0.97 seconds), followed by BI$\alpha\beta$ (1.3 seconds), followed by DO and BI with similar times as in the previous case.

Sampling algorithms with a stochastic sequence. We also performed the experiments in the setting with chance nodes. Due to the size of the game tree, we have reduced the number of cards to 4, since the size of this game tree is comparable to the case with 5 cards and a fixed sequence of cards. The results depicted in Fig. 13 show a similar behavior of the sampling algorithms as observed in the previous case. OOS converges the fastest, followed by RM, and Exp3. The main difference is in the convergence of UCT, however, this is mostly due to the fact that a pure NE exists in Goofspiel with 4 cards; hence, UCT can better identify the best action to play and converges faster to a less exploitable strategy than in the case with 5 cards. Surprisingly, the convergence rates of the algorithms do not change that dramatically with the introduction of point utilities as in the previous case. The main reason is that the range of the utility values is smaller compared to the previous case (there is one card less in the present setting and the missing cards has the highest value). For comparison, we again use the vertical lines to denote times of exact algorithms. BI$\alpha\beta$ and DO$\alpha\beta$ are almost equally fast, with DO$\alpha\beta$ being slightly faster, followed by DO and BI.

6.4.2. Pursuit–evasion games

The results on pursuit–evasion games show more significant improvement when comparing DO$\alpha\beta$ and BI$\alpha\beta$ (see Fig. 14). In all settings, DO$\alpha\beta$ is significantly the fastest. When we compare the performance on a 5×5 graph with depth set to 6, BI evaluates more than 4.9×10^7 nodes taking more than 13 hours. On the other hand, BI$\alpha\beta$ evaluates on average 42,001 nodes taking almost 10 minutes (584 seconds). Interestingly, the benefits of a pure integration with alpha-beta search is not that helpful in this game. This is apparent from the results of DO algorithm that evaluates less than 2×10^6 nodes but it takes slightly over 9 minutes on average (547 seconds). Finally, DO$\alpha\beta$ evaluates only 6,692 nodes and it takes the algorithm less than 3 minutes.

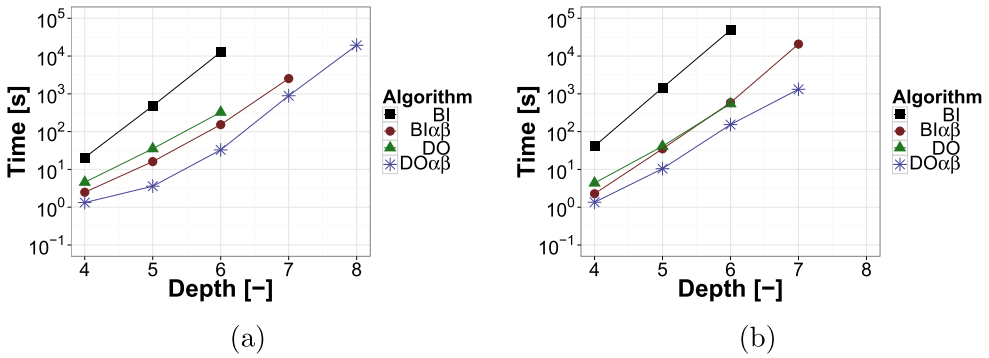


Fig. 14. Running times of exact algorithms on pursuit–evasion games with an increasing number of moves: subfigure (a) depicts the results on 4×4 grid graph, (b) depicts results for 5×5 grid.

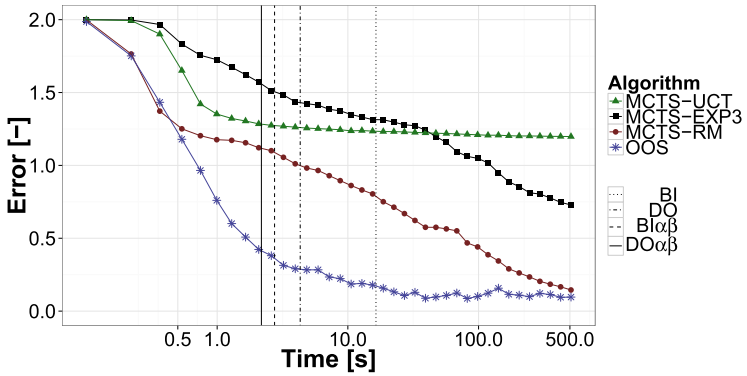


Fig. 15. Convergence of the sampling algorithms on a pursuit–evasion game, on a 4×4 graph, with depth set to 4. The vertical lines correspond to the computation times for the exact algorithms.

Large parts of these pursuit–evasion games can be solved by the serialized alpha-beta algorithms. These parts typically correspond to clearly winning, or clearly losing positions for a player; hence, the serialized alpha-beta algorithms are able to prune a substantial portion of the space. However, since there are only two pursuit units, it is still necessary to use mixed strategies for a final coordination (capturing the evader close to edge of the graph), and thus mixing strategy occurs near the end of the game tree. Therefore, serialized alpha-beta is not able to solve all subgames, while double-oracle provides additional pruning since many of the actions in the subgames are leading to the same outcome and not all of them required finding equilibrium strategies. This leads to additional reductions in the computation time for DO$\alpha\beta$ compared to BI$\alpha\beta$ and all the other algorithms.

We now turn to the convergence of the sampling algorithms. In terms of the number of iterations per second, again RM was the fastest and OOS the second fastest with similar performance as in Goofspiel. UCT achieved slightly less (1.7×10^5 iterations per second), and Exp3 only 2.6×10^4 iterations. The results are depicted in Fig. 15 for the smaller, 4×4 graph and 4 moves for each player (note again the logarithmic horizontal scale). The starting positions were selected such that there does not exist a pure NE strategy in the game. The results again show that OOS is overall the fastest out of all sampling algorithms. During the first iterations, RM preforms similarly, however, OOS is able to maintain its convergence rate, and RM starts converging more slowly. UCT again converges to an exploitable strategy with error 1.16 at best in the time limit of 500 seconds ($C = 2$). Finally, Exp3 is converging even more slowly than in Goofspiel. The main difference between the games is the size of the branching factor for the second player (the pursuer controls two simultaneously moving units), which can cause more difficulties for the sampling algorithms to estimate good strategies.

As before, the vertical lines represent the times for the exact algorithms. In a pursuit–evasion game of this setting, DO$\alpha\beta$ is slightly faster and finishes first in 2.77 seconds, following by BI$\alpha\beta$ (2.89 seconds), DO (5.48 seconds), and BI (12.5 seconds).

6.4.3. Oshi-Zumo

Many instances of the Oshi-Zumo game have Nash equilibria in pure strategies regardless of the type of the utility function. Although this does not hold for all the instances, the sizes of the subgames with pure NE are rather large and cause a dramatic computation speed-up for both algorithms using the serialized alpha-beta search. If the game does not have equilibria in pure strategies, the mixed strategies are still required only near the root node and large end-games are solved using alpha-beta search. Note that this is different than in the pursuit–evasion games, where mixed strategies were necessary close to the end of the game tree. Fig. 16 depicts the results with the parameter K set to 4 and for

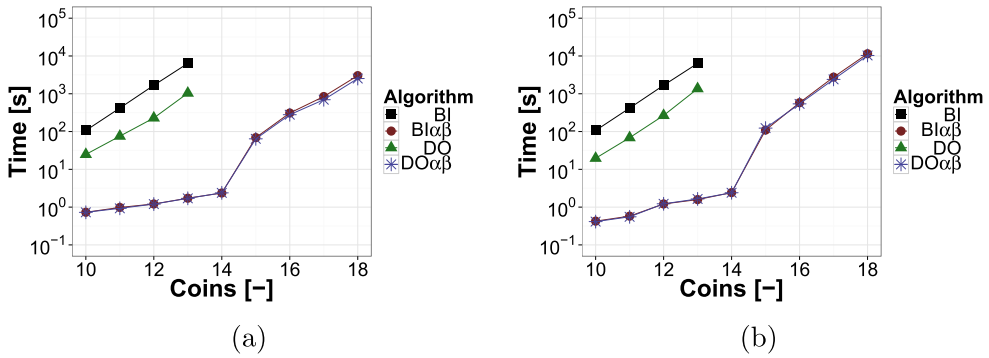


Fig. 16. Running times of the exact algorithms on Oshi-Zumo with K set to 4 and an increasing number of coins: subfigure (a) depicts the results for binary utilities, (b) depicts the results with point utilities.

two different settings of the utility function[7]; either win–tie–lose utilities (left subfigure) or point difference utilities (right subfigure). In both cases, the graphs show the breaking points when the game stops having an equilibrium in pure strategies (≥ 15 coins for each player). The advantage of BI$\alpha\beta$ and DO$\alpha\beta$ algorithms that exploit the serialized variants of alpha-beta algorithms is dramatic. We can see that both BI and DO scale rather badly. The algorithms were able to scale up to 13 coins in a reasonable time. For setting with $K = 4$ and 13 coins, it takes almost 2 hours for BI to solve the game (the algorithm evaluates 1.5×10^7 nodes) regardless of the utility values. DO improves the performance (the algorithm evaluates 2.8×10^6 nodes in 17 minutes for win–tie–lose utilities; the performance is slightly worse for point utilities: 5×10^6 nodes in 23 minutes). Both BI$\alpha\beta$ and DO$\alpha\beta$, however, solved a single alpha-beta search on each serialization finding a pure NE. Therefore, their performance is identical and it takes around 1.5 seconds to solve the game for both types of utilities. Although with an increasing number of coins the algorithms BI$\alpha\beta$ and DO$\alpha\beta$ need to find a mixed Nash equilibrium, their performance is very similar for both types of utilities. As expected, the case with point utilities is more challenging and the algorithms scale worse – for 18 coins both algorithms solve the game with win–tie–lose utilities in approximately 1 hour (BI$\alpha\beta$ in 50 minutes, DO$\alpha\beta$ in 64). It takes the algorithms around 3 hours to solve the case with point utilities (BI$\alpha\beta$ in 191 minutes, DO$\alpha\beta$ in 172 minutes).

Turning to the sampling algorithms reveals that the game is difficult to approximate even in the win–tie–lose setting. Fig. 17 depicts the results for the observed convergence rates of the sampling algorithms for the game with 10 coins, K set to 3 and the minimum bid set to 1. This is an easy game for DO$\alpha\beta$ and BI$\alpha\beta$ with a pure NE and both of these algorithms are able to solve the game in less than a second (0.73). However, due to a large branching factor for both players (10 actions at the root node for each player) all sampling algorithms converge extremely slowly. The performance of the algorithms in terms of iterations per second is similar to the previous games, however, OOS is slightly better in this case with 1.9×10^5 iterations per second compared to the RM with 1.6×10^5 iterations per second.

As before, OOS is the best converging algorithm, however, in a given time limit (500 seconds) the reached error was only slightly below 0.3 (0.29). On the other hand, all of the other sampling algorithms perform significantly worse – RM ends with error slightly over 1, UCT ($C = 2$) with 1.50, and Exp3 with 1.88. This confirms our findings from the previous experiment that increasing the branching factor slows down the convergence rate. Secondly, since there is a pure Nash equilibrium in this particular game configuration, the convergence of the algorithms is also slower since they essentially mix the strategy during the iterations in order to explore the unvisited parts of the game tree. Since none of the sampling algorithms can directly exploit this fact, their performance in offline solving of games like Oshi-Zumo is not compelling. On the other hand, the existence of pure NE explains the better performance of UCT compared to Exp3 that is forced to explore more broadly. Moreover, the convergence takes even more time in the point utility case, since the range of the utility values is larger. OOS is again the fastest and converges to error 0.45 within the time limit, RM to 1.41, UCT ($C = 4$) to 3.1, and Exp3 to 3.7.

6.4.4. Random games

In the first variant of the randomly generated games we used games with utility values randomly drawn from a uniform distribution on $[0, 1]$. Such games represent an extreme case, where neither alpha-beta nor the double-oracle algorithm can save much computation time, since each action can lead to arbitrarily good or bad terminal states. In these games, BI is typically the fastest. Even though both BI$\alpha\beta$ and DO$\alpha\beta$ evaluate marginally fewer nodes (less than 90%), the overhead of the algorithms (repeated computations of the serialized alpha-beta algorithm, repeatedly solving linear programs, etc.) causes a slower run time performance in this case.

However, completely random games are rarely instances that need to be solved in practice. The situation changes, when we use the intuition of good and bad moves and thus add correlation to the utility values. Fig. 18 depicts the results for two different branching factors 4 and 5 for each player and increasing depth. The results show that DO$\alpha\beta$ outperforms all

[7] We have also performed the same experiments with K set to 3, but the conclusions were the same as in case $K = 4$.

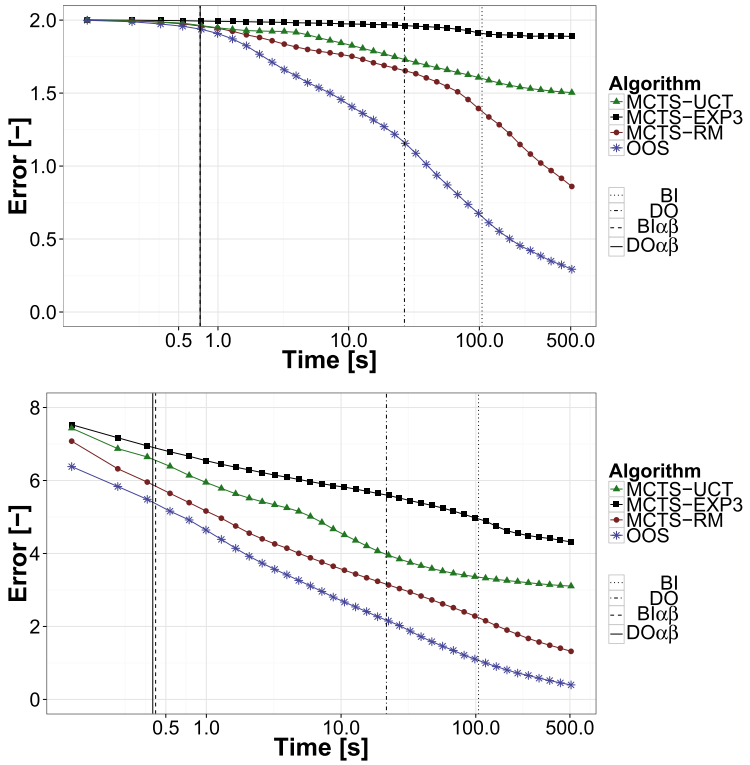


Fig. 17. Convergence of the sampling algorithms on Oshi-Zumo game with 10 coins, $K = 3$, and $M = 1$. The vertical lines correspond to the computation times for the exact algorithms. (Top) Oshi-Zumo with win–tie–lose utility values; (bottom) Oshi-Zumo with point utilities.

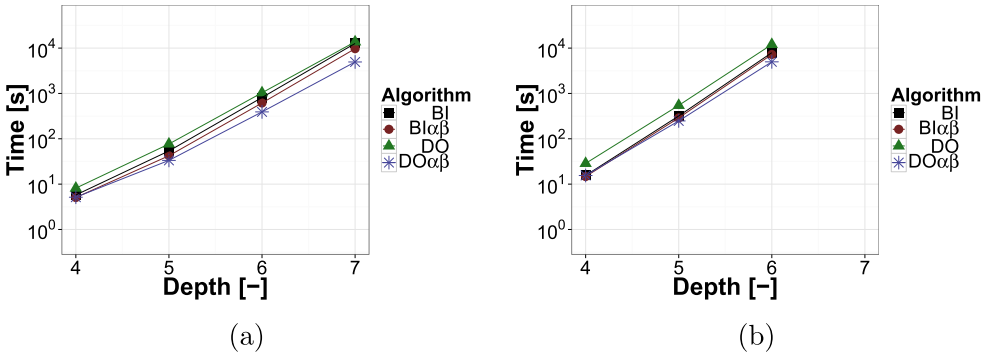


Fig. 18. Running times of the exact algorithms on randomly generated games with increasing depth: subfigure (a) depicts the results with branching factor set to 4 actions for each player, (b) depicts the results with branching factor 5.

remaining algorithms, although the difference is rather small (still statistically significant). On the other hand, DO without serialized alpha-beta is not able to outperform BI. This is most likely caused by a larger number of undominated actions that forces the double-oracle algorithm to enumerate most of the actions in each state. Moreover, this is also demonstrated by the performance of BI$\alpha\beta$ that is only slightly better compared to BI.

The fact that serialized alpha-beta is less successful in randomly generated games is noticeable also when comparing the number of evaluated nodes. For the case with branching factor set to 4 for both players and depth 7, BI evaluates almost 1.8×10^7 nodes in almost 3.5 hours, while BI$\alpha\beta$ evaluates more than 1×10^7 nodes in almost 3 hours. DO evaluates even more nodes compared to BI$\alpha\beta$ (1.2×10^7) and it is slower compared to both BI and BI$\alpha\beta$. Finally, DO$\alpha\beta$ evaluates 2×10^6 nodes on average and it takes the algorithm slightly over 80 minutes.

Fig. 19 depicts the results for convergence of the sampling algorithms for a random game with correlated utility values, branching factor set to 4 and depth 5. The number of iterations per second is similar to the situation in Goofspiel, with Exp3 being the exception able to achieve more than 6.5×10^4 iterations per second, which is still the lowest number of iterations. Interestingly, there is a much less difference between the performance of the sampling algorithms in these games. Since these games are generally more mixed (i.e., NE require to use mixed strategies in many states of the games), they are much more suitable for the sampling algorithms. OOS can be considered the winner in this setting, however, the performance of RM is very similar. Again, since the game is more mixed, Exp3 outperforms UCT in the longer run. The exploration constant for UCT was set to 12 due to a larger utility variance in this setting.

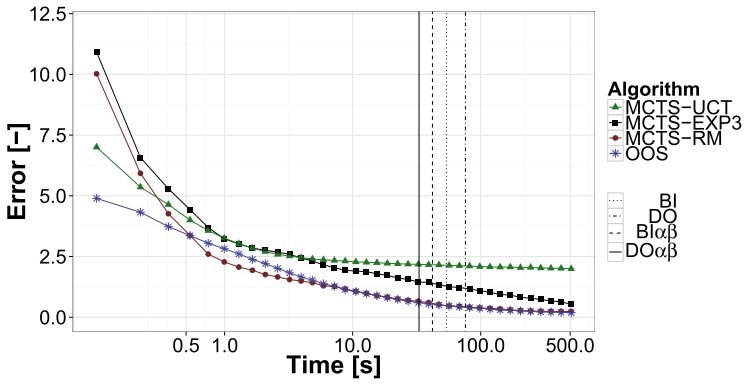


Fig. 19. Convergence of the sampling algorithms on a random game with branching factor 4 and depth 5. The vertical lines correspond to the computation times for the exact algorithms.

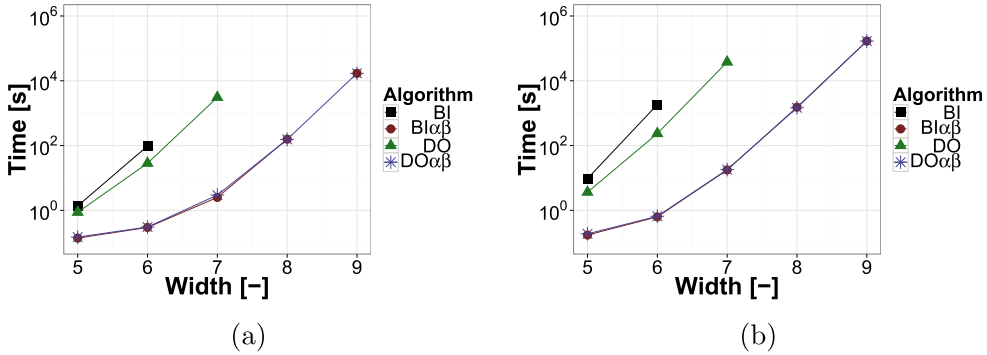


Fig. 20. Running times of the exact algorithms on Tron with increasing *width* of the grid graph: subfigure (a) depicts the results with *height* of the graph set to *width* − 1, (b) depicts the results with *height* = *width*.

6.4.5. Tron

Performance of the exact algorithms in Tron is affected by the fact that pure NE exist in all smaller instances (the results are depicted for two different ratios of dimensions of the grid in Fig. 20). Therefore, BI$\alpha\beta$ and DO$\alpha\beta$ are essentially the same since serialized alpha-beta is able to solve the game. Moreover, since the size of the game increases dramatically with the increasing size of the grid (the longest branch of the game tree has $(0.5 \cdot w \cdot l - 1)$ joint actions, where w and l are the dimensions of the grid), the performance of standard BI is very poor. While BI is able to solve the grid 5×6 in 96 seconds, it takes around 30 minutes to solve the 6×6 grid. By comparison, DO solves the 6×6 instance in 235 seconds, and both BI$\alpha\beta$ and DO$\alpha\beta$ in 0.6 seconds. BI$\alpha\beta$ and DO$\alpha\beta$ scale much better and the largest graph these algorithms solved had size 9×9 taking almost 2 days to solve.

The size of the game tree in Tron also causes a slow convergence for the sampling algorithms. This is apparent also in the number of iterations that is lower than before. OOS is the fastest performing 1.3×10^5 iterations per second, RM achieves 1.2×10^5, UCT only 8×10^4, and Exp3 is again the slowest with 7.8×10^4 iterations per second. Fig. 21 depicts the results for the grid 5×6. Consistently with the previous results, OOS performs the best and it is able to converge very close to an exact solution in 300 seconds. Similarly, both RM and Exp3 are again eventually able to converge to a very small error, however, it takes them more time and in the time limit they achieve error 0.05, or 0.02 respectively. Finally, UCT ($C = 5$) performs reasonably well during the first 10 seconds, where the exploitability is better than both RM and Exp3. This is most likely due to the existence of pure NE, however, the length of the game tree prohibits UCT from converging and the best error the algorithm was able to achieve in the time limit was equal to 0.68.

6.4.6. Summary of the offline equilibrium computation experiments

The offline comparison of the algorithms offer several conclusions. Among the exact algorithms, DO$\alpha\beta$ is clearly the best algorithm, since it typically outperforms all other algorithms (especially in pursuit–evasion games and random games). Although for smaller games (e.g., Goofspiel with 5 cards) BI$\alpha\beta$ can be slightly faster, this difference is not significant and DO$\alpha\beta$ is never significantly slower compared to BI$\alpha\beta$.

Among the sampling algorithms, OOS is the clear winner since it is often able to quickly converge to a very small error and significantly outperforms all variants of MCTS. On the other hand, comparing OOS and DO$\alpha\beta$, the exact DO$\alpha\beta$ algorithm is always faster and it is able to find an exact solution much faster compared to OOS. Moreover, DO$\alpha\beta$ has significantly lower memory requirements since it is a depth-first search algorithm and does not use any form of global cache, while OOS iteratively constructs the game tree in memory.

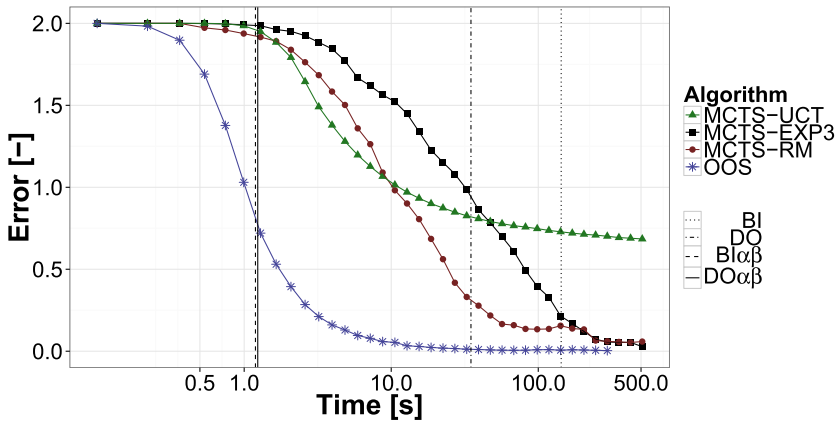


Fig. 21. Convergence comparison of different sampling algorithms on Tron on grid 5×6. The vertical lines correspond to the computation times for the exact algorithms.

6.5. Online search

We now compare the performance of the algorithms in head-to-head matches in the same games as in the offline equilibrium computation experiments, but we use much larger instances of these games. Each algorithm has a strictly limited computation time per move set to 1 second. After this time, the algorithm outputs an action to be played in the current game state, receives information about the action selected by its opponent, and the game proceeds to the next state. As described in Section 5, each algorithm keeps results of previous computations and does not start from scratch in the next state. We have also performed a large set of experiments with 5 seconds of computation time per move, however, the results are very similar to the results with 1 second per move. Therefore, we presents the results with 1 second in detail and only comment on the 5-second results where the additional time leads to an interesting difference.

We compare all of the approximative sampling algorithms and DO$\alpha\beta$ as a representative of backward induction algorithms, because it was clearly the fastest algorithm in all of the considered games. Finally, we also include a random player (denoted RAND) into the tournament to confirm that the algorithms choose much better strategies than the simple random game play. We report expected rewards and win rates of the algorithms, in which a tie counts as half of a win. The parameters of the algorithms are tuned for each domain separately. We first present the comparison of different algorithms and we discuss the influence of the parameters in Subsection 6.5.6.

In this subsection, we show cross tables of each algorithm (in each row) matched up against each competitor algorithm (in each column). Each entry represents a mean of at least 1000 matches with the half of the width of the 95% confidence interval shown in parentheses, e.g., 52.9(0.3) refers to $52.9\% \pm 0.3\%$. The result shown is the win rate for the row player, so as an example in the standard game of Goofspiel (top of Table 1) DO$\alpha\beta$ wins $67.2\% \pm 1.4\%$ of games against the random player. All evaluated games except the pursuit–evasion game are symmetric from the perspective of the first and the second player. We made even the random games symmetric by always playing matches on the same game instance in pairs with alternating players' positions. However, for easier comparison of the algorithms, we mirror the same results to both fields corresponding to a pair of players in the cross tables.

6.5.1. Goofspiel

In the head-to-head comparisons, our focus is primarily on the standard Goofspiel with 13 cards and chance nodes. Additionally, for the sake of consistency with the offline results, we also evaluate the variant with a fixed known sequence of cards. The full game has more than 2.4×10^{29} terminal states and the variant with a fixed sequence has still more than 3.8×10^{19} terminal states. The results are presented in Table 1, where the top table shows the win rates of the algorithms in the full game and the other two tables show the win rates and the expected number of points gained by the algorithms in the game with a fixed point card sequence. The results for the fixed card sequence are means over 10 fixed random sequences. For each table, the algorithms were set up to optimize the presented measure (i.e., win rate or points) and the exploration parameters were tuned to the values presented in the header of the table.

First, we can see that finding a good strategy in Goofspiel is difficult for all the algorithms. This is noticeable from the results of the RAND player, that performs reasonably well (RAND typically loses almost every match in all the remaining game domains). Next, we analyze the results of the DO$\alpha\beta$ algorithm compared to the sampling algorithms. The results show that even though DO$\alpha\beta$ uses a domain-specific heuristic evaluation function, it does not win significantly against any of the sampling algorithms that do not use any domain knowledge. The difference is always statistically significant with a large margin. When optimizing win percentage, DO$\alpha\beta$ loses the least against UCT while in optimizing the expected reward, UCT performs significantly best. The performance of the other sampling algorithms is very similar against DO$\alpha\beta$, with Exp3 winning the least in the reward optimization.

Table 1
Results of head-to-head matches in Goofspiel variants with exploration parameter settings indicated in the table headers.

Goofspiel: 13 cards, stochastic sequence of cards, win rate

	DO$\alpha\beta$	OOS(0.2)	UCT(0.6)	EXP3(0.3)	RM(0.1)	RAND
DO$\alpha\beta$	•	26.6(2.7)	36.0(2.9)	26.1(2.7)	25.9(2.7)	67.2(1.4)
OOS	73.4(2.7)	•	**51.2(2.1)**	52.5(2.2)	47.5(3.0)	81.4(1.7)
UCT	64.0(2.9)	48.8(2.1)	•	55.6(2.1)	**49.7(3.0)**	77.3(1.8)
EXP3	73.8(2.7)	47.5(2.2)	44.4(2.1)	•	41.1(3.0)	**86.1(1.5)**
RM	**74.1(2.7)**	**52.5(3.0)**	50.3(3.0)	**58.9(3.0)**	•	85.2(2.2)
RAND	32.8(1.4)	18.6(1.7)	22.7(1.8)	13.9(1.5)	14.8(2.2)	•

Goofspiel: 13 cards, known sequence of cards, win rate

	DO$\alpha\beta$	OOS(0.3)	UCT(0.8)	EXP3(0.2)	RM(0.1)	RAND
DO$\alpha\beta$	•	28.2(2.8)	35.0(2.9)	30.1(2.8)	31.5(2.8)	67.2(2.9)
OOS	**71.8(2.8)**	•	46.2(3.0)	51.8(3.0)	**49.6(3.0)**	83.8(2.3)
UCT	65.0(2.9)	**53.8(3.0)**	•	**57.1(2.9)**	48.6(2.9)	79.5(2.5)
EXP3	70.0(2.8)	48.2(3.0)	42.9(2.9)	•	46.5(3.0)	**85.8(2.1)**
RM	68.5(2.8)	50.4(3.0)	**51.4(2.9)**	53.5(3.0)	•	84.2(2.2)
RAND	32.8(2.9)	16.2(2.3)	20.5(2.5)	14.2(2.1)	15.8(2.2)	•

Goofspiel: 13 cards, known sequence of cards, point utilities

	DO$\alpha\beta$	OOS(0.3)	UCT(0.8)	EXP3(0.2)	RM(0.1)	RAND
DO$\alpha\beta$	•	−7.74(0.94)	−8.89(0.91)	−6.45(0.94)	−7.88(0.96)	6.67(0.99)
OOS	7.74(0.94)	•	1.19(0.78)	3.27(0.82)	**0.35(0.76)**	14.42(0.96)
UCT	**8.89(0.91)**	−1.19(0.78)	•	1.72(0.80)	−1.94(0.73)	13.30(1.00)
EXP3	6.45(0.94)	−3.27(0.82)	−1.72(0.80)	•	−5.02(0.79)	**14.79(0.97)**
RM	7.88(0.96)	**−0.35(0.76)**	**1.94(0.73)**	**5.02(0.79)**	•	14.20(0.98)
RAND	−6.67(0.99)	−14.42(0.96)	−13.30(1.00)	−14.79(0.97)	−14.20(0.98)	•

We compare the sampling algorithms in the game variants in the order of the presented tables. The differences in the performance of the sampling algorithms are relatively small between each other. They are more noticeable mainly against the weaker players, which are outperformed by all sampling algorithms. In the game with stochastic point card sequence, OOS, UCT and RM make approximately 10×10^3 iterations in the 1 second time limit in the root of the game. Exp3 is slightly slower with 8×10^3 iterations. The best algorithm in this game variant is RM, which wins against all other sampling algorithms and wins most often against DO$\alpha\beta$ and Exp3. The second best algorithm is OOS, which loses only against RM and Exp3 is the weakest algorithm losing against all other sampling algorithms.

The sampling algorithms in the second game variant (without chance) perform the same number of samples as in the first variant, with the exception of UCT, which performs 12×10^3 iterations per second. However, they each build a considerably deeper search tree, since the game tree is less wide. The exploration parameters were tuned to slightly larger numbers, which indicate that more exploration is beneficial in smaller games. The results are similar to the previous game variant. RM is still winning against all opponents, but it is not able to win more often against weaker players, which is consistent with playing close to a Nash equilibrium. UCT loses only against RM in this variant and it significantly outperforms OOS and Exp3. This indicates that UCT was able to better focus on the relevant part of the smaller game, which is supported also by a larger number of simulations, which can be caused by shorter random simulations after leaving the part of the search tree stored in memory.

When the players optimize the expected point difference, the differences between the algorithms are larger. We can see that RM and OOS perform significantly better than UCT and Exp3. OOS wins against all opponents and RM loses only against OOS. An important reason behind the decrease of the performance of Exp3 is that after normalizing the reward to unit interval, the important differences in values for reasonably good strategies become much smaller, which slows down the learning of the algorithm. UCT compensates the range of the rewards by the choice of the exploration parameter, but different nodes would benefit from different exploration parameters, which causes more inefficiencies with more variable rewards. An important advantage of OOS and RM is that their behavior is practically independent of the utility range.

In summary, RM is the only algorithm that did not lose significantly against any other sampling algorithm in any of the game variants and it often wins significantly. Exp3 is overall the weakest algorithm, losing to all other algorithms in all Goofspiel variants. Interestingly, Exp3 always performs the best against the random player, which indicates a slower convergence against more sophisticated strategies.

6.5.2. Oshi-Zumo

In Oshi-Zumo, we use the setting with 50 coins, $2 \cdot 3 + 1 = 7$ fields of the board (i.e., $K = 3$), and the minimal bet of 1. The size of the game is large with strictly more than 10^{15} terminal states and 50 actions for each player in the root.

The results are depicted in Table 2. As in the case of Goofspiel, we show the results for both the win rate as well as the point utilities. Moreover, our evaluation function in Oshi-Zumo is much more accurate than the one in Goofspiel and DO$\alpha\beta$

Table 2
Results of head-to-head matches in Oshi-Zumo variants with exploration parameter settings indicated in the header. In the first two tables only DO$\alpha\beta$ uses a heuristic evaluation function and in the third table all algorithms use the evaluation function.

Oshi-Zumo: 50 coins, $K = 3$, win rate

	DO$\alpha\beta$	OOS(0.2)	UCT(0.4)	EXP3(0.8)	RM(0.1)	RAND
DO$\alpha\beta$	•	**79.2(2.5)**	**77.6(2.6)**	**84.8(2.2)**	**84.0(2.3)**	98.8(0.5)
OOS	20.9(2.5)	•	27.7(2.6)	57.1(2.1)	51.2(2.1)	98.9(0.4)
UCT	**22.4(2.6)**	72.3(2.6)	•	83.0(2.0)	70.3(2.6)	**99.9(0.2)**
EXP3	15.2(2.2)	42.9(2.1)	17.0(2.0)	•	44.5(2.8)	98.5(0.5)
RM	16.0(2.3)	48.8(2.1)	29.6(2.6)	55.5(2.8)	•	99.0(0.4)
RAND	1.2(0.5)	1.1(0.4)	0.1(0.2)	1.5(0.5)	1.0(0.4)	•

Oshi-Zumo: 50 coins, $K = 3$, point utilities

	DO$\alpha\beta$	OOS(0.2)	UCT(0.4)	EXP3(0.8)	RM(0.1)	RAND
DO$\alpha\beta$	•	**2.33(0.19)**	**2.27(0.20)**	3.62(0.10)	**2.85(0.17)**	3.65(0.09)
OOS	−2.33(0.19)	•	−0.53(0.19)	3.46(0.10)	0.25(0.20)	3.87(0.05)
UCT	**−2.27(0.20)**	0.53(0.19)	•	**3.68(0.07)**	0.58(0.17)	**3.93(0.02)**
EXP3	−3.62(0.10)	−3.46(0.10)	−3.68(0.07)	•	−3.53(0.09)	1.31(0.17)
RM	−2.85(0.17)	−0.25(0.20)	−0.58(0.17)	3.53(0.09)	•	3.87(0.04)
RAND	−3.65(0.09)	−3.87(0.05)	−3.93(0.02)	−1.31(0.17)	−3.87(0.04)	•

Oshi-Zumo: 50 coins, $K = 3$, win rate, evaluation function

	DO$\alpha\beta$	OOS(0.3)	UCT(0.8)	EXP3(0.8)	RM(0.1)	RAND
DO$\alpha\beta$	•	63.0(2.1)	11.8(1.3)	52.9(2.2)	61.7(2.1)	98.6(0.5)
OOS	37.0(2.1)	•	24.8(1.9)	33.4(2.0)	43.6(2.1)	99.6(0.3)
UCT	**88.2(1.3)**	**75.2(1.9)**	•	**80.5(1.7)**	**71.1(2.0)**	**99.8(0.2)**
EXP3	47.1(2.2)	66.6(2.0)	19.5(1.7)	•	58.7(2.1)	98.7(0.5)
RM	38.3(2.1)	56.4(2.1)	**28.9(2.0)**	41.3(2.1)	•	99.6(0.3)
RAND	1.4(0.5)	0.4(0.3)	0.2(0.2)	1.3(0.5)	0.4(0.3)	•

is clearly outperforming all sampling algorithms when they do not use any domain specific knowledge. Therefore we also run experiments where the sampling algorithms also use an evaluation function instead of random rollout simulations.

In the offline experiment (Fig. 17), none of the sampling algorithms were able to converge anywhere close to the equilibrium in a short time. Moreover, the game used in the offline experiments was orders of magnitude smaller (there were only 10 coins for each player). In spite of the negative results in the offline experiments, all sampling algorithms are able to find a reasonably good strategy. UCT is clearly the strongest sampling algorithm in all variants. In the win rate setting, the strongest opponent of UCT among the sampling algorithms is RM (UCT wins 70.3% of games), followed by OOS performing only slightly worse (UCT wins 72.3% of games). Finally, Exp3 is clearly the weakest of all sampling algorithms. A possible reason may be that Exp3 manages to perform around 2.5×10^3 iterations per second in the root, while the other algorithms perform ten times more. This is caused by the quadratic dependence of its computational complexity on the number of actions, which is relatively high in this game. The situation remains similar when the algorithms optimize the point utilities.

We now turn to the experiments with the evaluation function, the results of which are presented in the third table of Table 2. The results show that the quality of play of all sampling algorithms is significantly improved. With this modification, UCT already significantly outperforms all algorithms including DO$\alpha\beta$. DO$\alpha\beta$ is the second best and still winning over the remaining sampling algorithms. Exp3 benefits from the evaluation function more than OOS and RM, which are relatively weaker with the evaluation function.

The reason why UCT performs well in this game is that the game mostly requires pure strategies, rather than precise mixing between multiple strategies (see Subsection 6.2). UCT is able to quickly disregard other actions, if a single action is optimal. So, the evaluation function generally helps every algorithm, but can make significant changes in ranking of the algorithms.

6.5.3. Random games

The next set of matches was played on 10 different random games with each player having 5 actions in each stage and depth 15. Hence, the game has more than 9.3×10^{20} terminal states. In order to compute the win-rates as in the other games, we use the sign of the utility value defined in Subsection 6.2. The results are presented in Table 3.

The clearly best performing algorithm in this domain is UCT that significantly outperforms the other sampling algorithms, and ties with DO$\alpha\beta$ that uses a rather strong evaluation function. This is true even though UCT performs around 11×10^3 iterations per second, which is the least form all sampling algorithms. DO$\alpha\beta$ wins over all other sampling algorithms. OOS has the weakest performance in spite of good convergence results in the offline settings (see Fig. 19). The reason is the quickly growing variance and decreasing number of samples in longer games, which we discuss in more details in Subsection 6.6. OOS performs 20×10^3 iterations per second and only around 3×10^3 of them actually update the regrets in the root. All the other iterations return with zero tail probability (x_i) in the root, which leads to no change in the regret values.

Table 3
Win-rate in head-to-head matches of random games with 5 actions for each player in each move and depth of 15 moves.

	DO$\alpha\beta$	OOS(0.1)	UCT(1.5)	EXP3(0.6)	RM(0.3)	RAND
DO$\alpha\beta$	•	57.4(2.9)	**49.6(2.8)**	53.4(2.8)	51.3(2.8)	88.8(1.8)
OOS	42.6(2.9)	•	33.5(2.5)	43.5(2.7)	42.5(2.8)	85.0(2.4)
UCT	**50.4(2.8)**	**66.5(2.5)**	•	**67.4(2.5)**	**55.7(2.6)**	95.9(1.2)
EXP3	46.6(2.8)	56.5(2.7)	32.6(2.5)	•	42.9(2.7)	**96.0(1.1)**
RM	48.7(2.8)	57.5(2.8)	44.3(2.6)	57.1(2.7)	•	93.1(1.5)
RAND	11.2(1.8)	15.0(2.4)	4.1(1.2)	4.0(1.1)	6.9(1.5)	•

Table 4
Win-rate in head-to-head matches of Tron with random simulations (top) and evaluation function in the sampling algorithms (bottom).

Tron: 13 × 13 grid, win rate

	DO$\alpha\beta$	OOS(0.1)	UCT(0.6)	EXP3(0.5)	RM(0.1)	RAND
DO$\alpha\beta$	•	**78.2(2.0)**	**53.8(2.0)**	**66.6(2.3)**	**65.0(2.2)**	**98.6(0.5)**
OOS	21.8(2.0)	•	29.4(2.2)	46.1(1.8)	38.0(2.2)	97.2(0.5)
UCT	**46.2(2.0)**	70.6(2.2)	•	64.8(2.2)	57.0(2.1)	98.0(0.6)
EXP3	33.4(2.3)	53.9(1.8)	35.1(2.2)	•	44.3(2.3)	97.7(0.5)
RM	35.0(2.2)	62.0(2.2)	43.0(2.1)	55.7(2.3)	•	97.6(0.9)
RAND	1.4(0.5)	2.9(0.5)	2.0(0.6)	2.3(0.5)	2.4(0.9)	•

Tron: 13 × 13 grid, win rate, evaluation function

	DO$\alpha\beta$	OOS(0.1)	UCT(2)	EXP3(0.1)	RM(0.2)	RAND
DO$\alpha\beta$	•	**50.2(1.3)**	42.7(1.5)	53.1(1.6)	46.3(1.6)	98.8(0.4)
OOS	49.8(1.3)	•	53.0(0.9)	**54.7(0.8)**	**52.2(0.8)**	**97.9(0.4)**
UCT	**57.3(1.5)**	47.0(0.9)	•	49.7(0.5)	46.7(0.6)	98.8(0.3)
EXP3	46.9(1.6)	45.3(0.8)	50.3(0.5)	•	45.8(0.6)	98.2(0.4)
RM	53.7(1.6)	47.8(0.8)	**53.3(0.6)**	54.2(0.6)	•	98.5(0.4)
RAND	1.2(0.4)	2.1(0.4)	1.2(0.3)	1.8(0.4)	1.5(0.4)	•

6.5.4. Tron

The large variant of Tron in our evaluation was played on an empty 13 × 13 board. The branching factor of this game is up to 4 for each player and its depth is up to 83 moves. This variant of Tron has more than 10^{21} terminal states.[8] The results are shown in Table 4.

The evaluation function in Tron approximates the situation in the game fairly well; hence, DO$\alpha\beta$ strongly outperforms all other algorithms when they do not use the evaluation function (top). Its win-rates are even higher with more time per move. UCT is the strongest opponent for DO$\alpha\beta$ – UCT loses 53.8% of matches and wins over all other sampling algorithms in mutual matches. This is again because of the low need for mixed strategies in this game (see Subsection 6.2). Again, OOS performs the worst despite its clearly fastest convergence on the smaller game variant in the offline setting due to the great depth of the game tree in this setting. It won only 21.8% matches against DO$\alpha\beta$ and 29.4% matches against UCT. In this game, the variance of the regret updates is likely not the key factor, since it is between 20 and 40. However, only 1×10^3 out of 12×10^3 iterations per second update regrets in the root.

The good performance of DO$\alpha\beta$ is consistent with the previous analysis in Tron where the best-performing algorithms, including the winner of the 2011 Google AI Challenge, were based on depth-limited minimax searches [57,84].

As in the case of Oshi-Zumo, we also run the matches with the evaluation function in place of the random rollout simulation in the sampling algorithms. We present the results in the second table of Table 4. Using the evaluation function improves the performance of all sampling algorithms against DO$\alpha\beta$ and it decreases the differences in performance between each algorithm. The difference is most notable for OOS, since using the evaluation function strongly reduces the length of the game. In this setting, both RM and UCT outperform DO$\alpha\beta$. Interestingly, while UCT performs quite well against DO$\alpha\beta$ and wins 57.3% of matches, it is not winning against any other sampling algorithm. Even Exp3 which loses against all other algorithms manages to slightly outperform it. OOS practically ties with DO$\alpha\beta$, but it wins significantly against all sampling algorithms. RM loses to OOS, but wins significantly against all other algorithms.

6.5.5. Pursuit–evasion game

Finally, we compare the algorithms on the pursuit–evasion game on an empty 10 × 10 grid with 15 moves time limit and 10 different randomly selected initial positions of the units. The branching factor is at most 12, causing the number of terminal states to be less than 10^{16}.

The results in Table 5 show that the game is strongly biased towards the first player, which is the evader. The self-play results on the diagonal show that DO$\alpha\beta$ won over 81.5% matches against itself as the evader. Adding more computational time typically improves the play of the pursuer in self-play. This is caused by a more complex optimal strategy of the

[8] The number only estimates the number of possible paths when both players stay on their half of the board.

Table 5
Win-rate in head-to-head matches of pursuit–evasion games with time limit of 15 moves and 10×10 grid board. The evader is the row player and pursuer is the column player.

	DO$\alpha\beta$	OOS(0.3)	UCT(0.8)	EXP3(0.5)	RM(0.1)	RAND
DO$\alpha\beta$	81.5(2.4)	89.1(1.9)	**61.8(3.0)**	**91.2(1.8)**	77.2(2.6)	**100.0(0.0)**
OOS(0.2)	77.5(2.6)	91.2(1.8)	57.8(3.1)	85.8(2.2)	79.3(2.5)	99.8(0.3)
UCT(1.5)	77.1(2.6)	**94.2(1.4)**	57.6(3.1)	88.9(1.9)	**82.2(2.4)**	**100.0(0.0)**
EXP3(0.2)	65.1(3.0)	92.1(1.7)	53.1(3.1)	83.9(2.3)	75.1(2.7)	99.8(0.3)
RM(0.1)	**81.8(2.4)**	92.7(1.6)	58.5(3.1)	86.7(2.1)	78.6(2.5)	99.8(0.3)
RAND	5.1(1.4)	28.8(2.8)	5.8(1.4)	1.7(0.8)	3.1(1.1)	71.1(2.8)

pursuer. This optimal strategy is more difficult to find due to a larger branching factor (recall that the pursuer controls two units) and the requirement for a more precise execution (a single move played incorrectly can cause an escape of the evader and can result in losing the game due to the time limit).

We first look at the differences in the performance of the algorithms on the side of the pursuer, which are more consistent. We need to compare the different columns, in which the pursuer tries to minimize the values. The clear winner is UCT that generally captures the evaders in approximately 40% of the matches. The second best pursuer is DO$\alpha\beta$ and the weakest is OOS that captures the non-random opponents in less than 10% of the cases.

The situation is less clear for the evader. Different algorithms performed best against different opponents. UCT was the best against OOS and RM, but DO$\alpha\beta$ was the best against UCT and Exp3. Exp3 is the weakest evader.

6.5.6. Parameter tuning

The exploration parameters can have a significant influence on the performance of the algorithm. We choose the parameters individually for each domain by running mutual matches with a pre-selected fixed pool of opponents. This pool includes DO$\alpha\beta$ and each of the sampling algorithms with one setting of the parameter selected based on the results of the offline experiments. These values are 0.6 for OOS, 2 for UCT, 0.2 for Exp3 and 0.1 for RM. For each domain, we created a table such as the two examples in Table 6. We then picked the parameter for the final cross tables presented above as the parameter with the best mean performance against all the fixed opponents.

In the presented variant of Goofspiel, the choice of the exploration parameter has a rather large influence on the performance against DO$\alpha\beta$. This is often the case for weaker players. The selection of the exploration parameter for OOS has little effect on the mean performance, with a noticeable drop in performance for 0.1. In UCT, less exploration is generally better, but the sudden drop of performance against Exp3 causes the optimum to be at 0.6. In Exp3, the optimal exploration parameter against DO$\alpha\beta$ would be even greater than 0.5, while the optimum against OOS would be 0.2. These kinds of inconsistencies are common with the Exp3 algorithm. In the mean over all opponents, the optimum is 0.3. With RM, the optimal exploration parameter against individual opponents stays around 0.1 and it is clearly the best value in the mean.

Parameter selection is generally more important when facing weaker players. The differences are more noticeable in matches against other algorithms, but since the optimal parameters vary depending on the different opponents, the mean performance presented in the last column does not vary much. OOS is consistently the least sensitive to different parameter settings, while the performance differences in the other algorithms from changing exploration strongly depends on the specific domains.

The differences between various parameter settings are larger in smaller games and mainly if an evaluation function is used. Consider the results for Oshi-Zumo in Table 6. For OOS, the exploration parameter of 0.3 is consistently the best against all opponents, with the exception of Exp3, which loses slightly more to OOS with exploration 0.4. However, the difference is far from significant even after 1000 matches. The differences in performance of UCT with different parameters are more often statistically significant. Overall, the best parameter is 0.8, even though the performance is significantly better against DO$\alpha\beta$ with smaller exploration and against UCT(2) with higher exploration. The best performance for Exp3 was surprisingly achieved with a very high exploration. The best of the tested values was 0.8, which means that 80% of the time, the next action to sample is selected randomly regardless of the collected statistics about move qualities. The higher values were consistently better for all opponents. RM seems to be quite sensitive to the parameter choice in this domain and the results for specific opponents are more inconclusive than for the other algorithms. When playing DO$\alpha\beta$, RM wins 7% more matches with parameter 0.3 than with the overall optimal 0.1. On the other hand, when playing OOS, an even smaller parameter value would be preferable.

The presented parameter tuning tables are representative of the behavior of the algorithms with different parameters. The choices of the optimal parameters generally depend much more on the domain than the selected opponent, but in some cases the optimal choice for one opponent is far from the optimum for another opponent. Especially with Exp3 and UCT, very different parameters are optimal for different domains. While in the presented results in Oshi-Zumo with evaluation function, 0.8 is best for Exp3, in Tron with evaluation function, the optimal parameter for Exp3 is 0.1. The range of optimal parameters is much smaller for OOS and RM, which were always between 0.1 and 0.3. This can be a notable advantage for playing previously unknown games without a sufficient time to tune the parameters for the specific domain.

Table 6
Sample parameter tuning tables for Goofspiel with stochastic point cards sequence and Oshi-Zumo.

Goofspiel: 13 cards, stochastic point card sequence

		DO$\alpha\beta$	OOS(0.6)	UCT(2)	EXP3(0.2)	RM(0.1)	Mean
OOS	0.5	73.8(2.7)	50.2(3.0)	54.4(4.2)	54.9(3.0)	49.4(3.0)	56.54
OOS	0.4	72.0(2.8)	50.5(3.0)	56.4(4.2)	54.1(3.0)	47.5(3.0)	56.1
OOS	0.3	73.0(2.7)	47.6(3.0)	58.4(4.2)	54.3(3.0)	48.0(3.1)	56.26
OOS	**0.2**	73.5(2.7)	50.2(3.0)	58.7(4.2)	54.3(3.0)	47.9(3.0)	**56.92**
OOS	0.1	70.2(2.8)	47.4(3.1)	53.4(4.3)	48.6(3.0)	43.9(3.0)	52.7
UCT	1.5	52.2(3.1)	45.4(3.0)	52.4(3.2)	53.9(3.9)	39.4(4.6)	48.66
UCT	1	52.5(3.0)	49.9(3.0)	58.3(3.2)	56.1(3.8)	43.1(4.6)	51.98
UCT	0.8	52.5(3.0)	51.1(3.0)	60.8(3.2)	59.7(3.8)	46.8(4.7)	54.18
UCT	**0.6**	54.2(3.0)	53.9(3.0)	61.2(3.1)	62.3(3.8)	46.6(4.7)	**55.64**
UCT	0.4	58.6(3.0)	54.9(3.0)	61.6(3.1)	58.6(3.8)	49.5(4.8)	55.04
EXP3	0.5	77.1(2.6)	42.6(3.0)	44.4(3.0)	47.4(3.0)	40.1(3.0)	50.32
EXP3	0.4	76.2(2.6)	44.8(3.0)	48.4(3.0)	49.5(3.0)	39.5(3.0)	51.68
EXP3	**0.3**	73.2(2.7)	44.5(3.0)	51.8(3.0)	51.1(3.0)	41.0(3.0)	**52.32**
EXP3	0.2	73.5(2.7)	47.2(3.0)	47.6(3.0)	50.0(3.0)	41.3(3.0)	51.92
EXP3	0.1	71.2(2.8)	44.9(3.0)	48.9(3.0)	51.2(3.0)	40.9(3.0)	51.42
RM	0.5	77.7(2.5)	44.9(3.0)	43.9(3.0)	46.9(3.0)	42.4(3.0)	51.16
RM	0.3	73.2(2.7)	49.3(3.0)	57.9(2.9)	53.9(3.0)	48.5(3.0)	56.56
RM	0.2	70.8(2.8)	50.7(3.0)	63.8(2.9)	57.8(3.0)	48.2(3.0)	58.26
RM	**0.1**	74.0(2.7)	54.1(3.0)	61.2(2.9)	58.1(3.0)	51.2(3.0)	**59.72**
RM	0.05	74.5(2.7)	51.6(3.0)	60.1(2.9)	59.0(3.0)	49.0(3.1)	58.84

Oshi-Zumo: 50 coins, $K = 3$, win rate, evaluation function

		DO$\alpha\beta$	OOS(0.6)	UCT(2)	EXP3(0.2)	RM(0.1)	Mean
OOS	0.5	35.3(2.9)	50.9(3.6)	28.5(3.3)	54.9(3.6)	43.7(3.5)	42.66
OOS	0.4	35.0(2.9)	56.0(3.6)	26.6(3.2)	56.1(3.6)	42.6(3.6)	43.26
OOS	**0.3**	36.5(3.0)	57.8(3.5)	27.7(3.2)	55.7(3.6)	44.8(3.6)	**44.5**
OOS	0.2	35.0(2.9)	53.1(3.6)	26.8(3.2)	54.1(3.6)	41.4(3.5)	42.08
OOS	0.1	34.6(2.9)	55.6(3.6)	24.1(3.1)	56.2(3.6)	43.0(3.6)	42.7
UCT	1.5	83.2(2.2)	74.0(3.8)	79.1(2.9)	87.4(2.9)	70.6(3.9)	78.86
UCT	1	83.8(2.1)	74.8(3.7)	81.4(2.7)	89.8(2.6)	68.8(4.0)	79.72
UCT	**0.8**	86.5(2.0)	77.9(3.6)	77.1(3.0)	89.2(2.7)	74.1(3.8)	**80.96**
UCT	0.6	89.4(1.8)	75.7(3.7)	54.9(3.9)	90.0(2.6)	74.1(3.7)	76.82
UCT	0.4	75.8(2.6)	75.0(3.7)	31.4(3.7)	89.8(2.6)	70.6(3.9)	68.52
EXP3	0.9	47.8(3.1)	68.2(2.8)	23.1(2.4)	67.2(2.8)	55.2(2.8)	52.3
EXP3	**0.8**	46.9(3.1)	68.4(3.6)	23.0(3.1)	74.2(3.4)	61.5(3.7)	**54.8**
EXP3	0.6	42.5(3.1)	67.6(3.7)	20.4(3.1)	65.4(3.7)	59.4(3.8)	51.06
EXP3	0.5	38.7(3.0)	60.9(3.8)	15.1(2.7)	64.7(3.7)	52.9(3.9)	46.46
EXP3	0.4	35.9(3.0)	57.5(3.9)	17.5(3.0)	64.1(3.8)	54.9(3.9)	45.98
RM	0.5	44.5(3.0)	41.1(3.5)	31.7(3.3)	49.4(3.6)	34.3(3.3)	40.2
RM	0.3	42.8(3.0)	52.1(3.5)	33.8(3.4)	61.2(3.5)	43.7(3.5)	46.72
RM	0.2	41.8(3.0)	55.7(3.6)	30.7(3.3)	59.2(3.5)	46.4(3.6)	46.76
RM	**0.1**	37.0(2.9)	58.1(3.5)	34.9(3.4)	57.6(3.6)	54.1(3.6)	**48.34**
RM	0.05	36.4(3.0)	59.6(3.5)	29.7(3.3)	59.3(3.5)	51.1(3.6)	47.22

6.5.7. *Summary of the online search experiments*

Several conclusions can be made from the head-to-head comparisons of the algorithms in larger games. First, the fast convergence and low exploitability of OOS in the smaller variants of the games is not a very good predictor of its performance in the online setting. OOS was often not the best algorithm in the online setting. In random games and Tron without the evaluation function, it was the worst performing algorithm. We discuss the possible reasons in detail in Subsection 6.6.

Second, DO$\alpha\beta$ with a good evaluation function often wins over the sampling algorithms without a domain specific knowledge. This is not the case with a weaker evaluation function, as we can see in Goofspiel. Moreover, when the sampling algorithms are allowed to use the evaluation functions, DO$\alpha\beta$ is outperformed by UCT in both domains tested with evaluation function and also by RM in Tron. Using a good evaluation function instead of random simulations helps all sampling algorithms, but the amount of improvement is different for individual algorithms in different domains.

Third, the novel RM and OOS algorithms have proven efficient in a wider range of domains. Besides Goofspiel used for evaluating earlier versions of the algorithms in [11], RM showed strong performance in random games and both RM and OOS were the best performing algorithms in Tron with the evaluation function. These algorithms did not exploit the weaker opponents the most but often won against all other competitors. A notable advantage of these algorithms is a lower sensitivity for the parameter tuning, since they perform well in a wide range of domains with similar exploration parameters.

Table 7
Measure of variances of estimated regret quantities in OOS and Regret Matching at the root of each game. T is the number of iterations each algorithm runs for, and Run marks the run number (instance).

Game	Run	T_{OOS}	$\widehat{\mathbf{Var}}_{\text{OOS}}$	T_{RM}	$\widehat{\mathbf{Var}}_{\text{RM}}$
Goofspiel(13)	1	12,582	32,939.94	11,939	283.03
Goofspiel(13)	2	13,888	26,737.95	7,160	359.96
Goofspiel(13)	3	13,906	27,283.47	7,897	552.24
OZ(50, 3, 1)	1	34,900	1,010.73	25,654	9.19
OZ(50, 3, 1)	2	40,876	1,225.89	26,719	7.93
OZ(50, 3, 1)	3	40,306	1,016.42	26,121	7.99
Tron(13, 13)	1	11,222	40.23	11,634	0.84
Tron(13, 13)	2	12,526	35.91	11,134	0.83
Tron(13, 13)	3	13,000	22.23	10,075	0.75

Fourth, when the algorithms have five times more time for finding a move to play, the differences between win rates of the sampling algorithms get smaller. Longer thinking time also has the same effect on parameter tuning and it also significantly improves the performance of the sampling algorithms against backward induction. This is expected, since the difference is too small for the DO$\alpha\beta$ algorithm to reach a greater depth, while it is sufficient for the sampling algorithms to execute five times more iterations improving their strategy.

Finally, the performance of Exp3 is the weakest in general. Its main problems are its larger computational complexity and problematic normalization for wider ranges of payoffs. Exp3 was significantly worse than other algorithms in both domains where we evaluated the point difference optimization and it performs an order of magnitude fewer iterations in Oshi-Zumo, compared to all other sampling algorithms.

6.6. Online Outcome Sampling versus Regret Matching

Given the similar nature of OOS and RM, one might wonder why RM typically performs better than OOS in online search, despite OOS being the only algorithm with provable convergence properties and the fastest converging algorithm in the offline setting. In this subsection, we investigate this phenomenon and present the results of additional experiments.

We need to look at the convergence properties of OOS, which is essentially an application of outcome sampling MCCFR. From the convergence bound of outcome sampling MCCFR presented in [86], after T iterations the strategy produced by the algorithm is an ϵ-Nash equilibrium with probability $1 - p$ and

$$\epsilon \leq O\left(\frac{\tilde{\Delta}_i|\mathcal{S}|}{\sqrt{T}} + \sqrt{\frac{\mathbf{Var}}{pT} + \frac{\mathbf{Cov}}{p}}\right),$$

where $\tilde{\Delta}_i$ is determined by the structure of the game, and **Var** and **Cov** are the maximal variance and covariance of the differences between the exact value of a regret of an action and its estimate computed based on the selected sample $(r^t(s,a) - \tilde{r}^t(s,a))$ over all states, actions, and time steps. Computing these quantities exactly is prohibitively expensive, and since the scale of the exact regrets is bounded by a relatively small range of utilities, we can estimate the variance of the difference by the variance of the sampled regrets, which has often a very large range due to the importance sampling correction (see Section 4.5). We measure the estimate $\widehat{\mathbf{Var}} = \text{Var}[\max_{a\in\mathcal{A}(s)} \tilde{r}^t(s,a)]$ in the root of the games, since they have the largest range of possible values of $\tilde{r}^t(s,a)$. Regret matching also computes a quantity similar to $\tilde{r}^t(s,a)$. The only difference is that they are not counterfactual, i.e., they take into account only the value of the current sample and not the expected value of the strategy used throughout the entire game. We show these variances for Goofspiel(13), OZ(50, 3, 1), and Tron(13, 13) in Table 7.

The results show that the variance of OOS is significantly higher than in case of RM. As such, even though RM may be introducing some kind of bias by bootstrapping value estimates from its own subgame, when there are so few samples this trade-off may be worthwhile to avoid the uncertainty introduced by the variance. This problem is not apparent in the smaller games, because the higher probability of sampling individual terminal histories causes smaller variance and OOS performs enough samples to make the regret estimates sufficiently close to the true values. For example, in Goofspiel(5) used for offline convergence experiments, OOS performs approximately 2×10^5 iterations per second and the variance is only around 350.

7. Conclusion and future research

In this paper, we provide an extensive analysis of algorithms for solving and playing zero-sum extensive-form games with perfect information and simultaneous moves. We describe a collection of exact algorithms based on backward induction as well as a collection of Monte Carlo tree search algorithms including our novel algorithms DO$\alpha\beta$, BI$\alpha\beta$, SM-OOS and SM-MCTS with regret matching selection function.

We empirically compare the performance of these algorithms on six substantially different games in two different settings. In the offline equilibrium computation setting, we show that our novel algorithm based on backward induction, DO$\alpha\beta$, is able to prune large parts of the search space. In most games, DO$\alpha\beta$ is several orders of magnitude faster than the classical backward induction and it is never significantly outperformed by any of its competitors. The only benefit of the sampling algorithms in the offline setting is a to get a rough approximation of the equilibrium solution in a short time. Their results are often inconsistent with short computation times. Given enough time, the results clearly show that SM-OOS achieves the fastest convergence to a Nash equilibrium. Finally, our offline experiments also explained different behavior reported in variants of SM-MCTS with UCT selection. We have shown that adding randomization to tie-breaking rules can significantly improve the performance of this algorithm.

The success in the offline equilibrium computation is, however, not a very good indicator of the game playing performance in the online setting of head-to-head matches. First of all, the sizes of the games used for online experiments are too large for exact algorithms to be applicable without a domain-specific evaluation function. Performance of the representative of the exact algorithms, DO$\alpha\beta$, depends heavily on the accuracy of the used evaluation function. Secondly, in spite of the fastest convergence of SM-OOS among the sampling algorithms, SM-OOS does not always perform well in the online game playing. This is mainly due to the large variance of the regret updates that increases significantly in these large games. Among the remaining sampling algorithms, SM-MCTS based on regret matching is often very good, but sometimes it was outperformed by SM-MCTS with UCT selection, especially in games that require less randomized strategies.

Our work opens several interesting directions for future research. After introducing a strong pruning algorithm, it is of interest to formally study the limitations of pruning for this class of games, similarly to the theory developed for games with sequential moves. Future work could show if these pruning techniques can be substantially improved or if they are in some sense optimal. The main prerequisite is, however, estimating the expected number of iterations of the double-oracle algorithms for single step matrix games, which still remains an open problem. Furthermore, running large head-to-head tournaments for evaluating the game playing performance is time consuming, sensitive to setting correct parameters, and provides only limited insights into the performance of the algorithms. Proximity to the Nash equilibrium is not always a good indicator of game playing performance; hence, it is interesting to study alternative measures of quality of the algorithms that would better predict their game-playing performance in large games.

Acknowledgements

This work is funded by the Czech Science Foundation (grant Nos. P202/12/2054 and 15-23235S) and the Netherlands Organisation for Scientific Research (NWO) in the framework of the project Go4Nature, grant number 612.000.938. Branislav Bošanský also acknowledges support from the Danish National Research Foundation and the National Science Foundation of China (under the grant 61361136003) for the Sino-Danish Center for the Theory of Interactive Computation, and the support from the Center for Research in Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council. The access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005) is highly appreciated.

References

[1] M. Campbell, A.J. Hoane, F. Hsu, Deep Blue, Artif. Intell. 134 (1–2) (2002) 57–83.
[2] J. Schaeffer, R. Lake, P. Lu, M. Bryant, Chinook: the world man–machine checkers champion, AI Mag. 17 (1) (1996) 21–29.
[3] G. Tesauro, Temporal difference learning and TD-Gammon, Commun. ACM 38 (3) (1995) 58–68.
[4] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[5] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, 3rd edition, Prentice Hall, 2009.
[6] A. Keuter, L. Nett, Ermes-auction in Germany. First simultaneous multiple-round auction in the European telecommunications market, Telecommun. Policy 21 (4) (1997) 297–307.
[7] D. Beard, P. Hingston, M. Masek, Using Monte Carlo tree search for replanning in a multistage simultaneous game, in: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), 2012.
[8] O. Teytaud, S. Flory, Upper confidence trees with short term partial information, in: Applications of Evolutionary Computation (EvoGames 2011), Part I, in: Lect. Notes Comput. Sci., vol. 6624, 2011, pp. 153–162.
[9] H. Gintis, Game Theory Evolving, 2nd edition, Princeton University Press, 2009.
[10] B. Bošanský, V. Lisý, J. Čermák, R. Vítek, M. Pěchouček, Using double-oracle method and serialized alpha-beta search for pruning in simultaneous moves games, in: Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI), 2013, pp. 48–54.
[11] M. Lanctot, V. Lisý, M.H.M. Winands, Monte Carlo tree search in simultaneous move games with applications to Goofspiel, in: Computer Games Workshop at IJCAI 2013, in: Commun. Comput. Inf. Sci., vol. 408, 2014, pp. 28–43.
[12] V. Lisý, V. Kovařík, M. Lanctot, B. Bošanský, Convergence of Monte Carlo tree search in simultaneous move games, in: Adv. Neural Inf. Process. Syst., vol. 26, 2013, pp. 2112–2120.
[13] V. Lisý, M. Lanctot, M. Bowling, Online Monte Carlo counterfactual regret minimization for search in imperfect information games, in: Proceedings of the 14th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2015, pp. 27–36.
[14] M. Shafiei, N. Sturtevant, J. Schaeffer, Comparing UCT versus CFR in simultaneous games, in: Proceeding of the IJCAI Workshop on General Game-Playing (GIGA), 2009, pp. 75–82.
[15] A. Saffidine, Solving games and all that, Ph.D. thesis, Université Paris-Dauphine, Paris, France, 2013.
[16] J.V. Neumann, Zur theorie der gesellschaftsspiele, Math. Ann. 100 (1928) 295–320.

[17] M.L. Littman, Markov games as a framework for multi-agent reinforcement learning, in: Proceedings of the 11th International Conference on Machine Learning (ICML), 1994, pp. 157–163.
[18] M.L. Littman, Value-function reinforcement learning in Markov games, Cogn. Syst. Res. 2 (1) (2001) 55–66.
[19] M.L. Littman, C. Szepesvári, A generalized reinforcement-learning model: convergence and applications, in: Proceedings of the 13th International Conference on Machine Learning (ICML), 1996, pp. 310–318.
[20] M.G. Lagoudakis, R. Parr, Value function approximation in zero-sum Markov games, in: Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI), 2002, pp. 283–292.
[21] U. Savagaonkar, R. Givan, E.K.P. Chong, Sampling techniques for zero-sum, discounted Markov games, in: Proceedings of the 40th Annual Allerton Conference on Communication, Control and Computing, 2002, pp. 285–294.
[22] J. Perolat, B. Scherrer, B. Piot, O. Pietquin, Approximate dynamic programming for two-player zero-sum Markov games, in: Proceedings of the 32nd International Conference on Machine Learning (ICML), 2015.
[23] S. Singh, M. Kearns, Y. Mansour, Nash convergence of gradient dynamics in general-sum games, in: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI), 2000, pp. 541–548.
[24] M. Bowling, M. Veloso, Convergence of gradient dynamics with a variable learning rate, in: Proceedings of the 18th International Conference on Machine Learning (ICML), 2001, pp. 27–34.
[25] M. Zinkevich, Online convex programming and generalized infinitesimal gradient ascent, in: Proceedings of 20th International Conference on Machine Learning (ICML), 2003, pp. 928–936.
[26] M. Bowling, Convergence and no-regret in multiagent learning, in: Adv. Neural Inf. Process. Syst., vol. 17, 2005, pp. 209–216.
[27] G. Gordon, No-regret algorithms for online convex programs, in: Proceedings of the 20th Annual Conference on Neural Information Processing Systems (NIPS), 2006, pp. 489–496.
[28] M. Zinkevich, M. Johanson, M. Bowling, C. Piccione, Regret minimization in games with incomplete information, in: Adv. Neural Inf. Process. Syst., 2008, pp. 1729–1736.
[29] M. Bowling, N. Burch, M. Johanson, O. Tammelin, Heads-up limit hold'em poker is solved, Science 347 (6218) (2015) 145–149.
[30] A. Nowé, P. Vrancx, Y.-M.D. Hauwere, Game theory and multi-agent reinforcement learning, in: Reinforcement Learning: State-of-the-Art, 2012, pp. 441–470 (Ch. 12).
[31] L. Buşoniu, R. Babuška, B.D. Schutter, A comprehensive survey of multi-agent reinforcement learning, IEEE Trans. Syst. Man Cybern., Part C, Appl. Rev. 38 (2) (2008) 156–172.
[32] D. Bloembergen, K. Tuyls, D. Hennes, M. Kaisers, Evolutionary dynamics of multi-agent learning: a survey, J. Artif. Intell. Res. 53 (2015) 659–697.
[33] Y. Shoham, K. Leyton-Brown, Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations, Cambridge University Press, 2009.
[34] R. Bellman, Dynamic Programming, Princeton University Press, 1957.
[35] S.M. Ross, Goofspiel — the game of pure strategy, J. Appl. Probab. 8 (3) (1971) 621–625.
[36] M. Buro, Solving the Oshi-Zumo game, in: Advances in Computer Games: Many Games, Many Challenges, in: IFIP Advances in Information and Communication Technology, vol. 135, 2003, pp. 361–366.
[37] G.C. Rhoads, L. Bartholdi, Computer solution to the game of pure strategy, Games 3 (4) (2012) 150–156.
[38] A. Saffidine, H. Finnsson, M. Buro, Alpha-beta pruning for games with simultaneous moves, in: Proceedings of the 32nd Conference on Artificial Intelligence (AAAI), 2012, pp. 556–562.
[39] H. McMahan, G. Gordon, A. Blum, Planning in the presence of cost functions controlled by an adversary, in: Proceedings of the 20th International Conference on Machine Learning (ICML), 2003, pp. 536–543.
[40] M. Zinkevich, M. Bowling, N. Burch, A new algorithm for generating equilibria in massive zero-sum games, in: Proceedings of the 27th Conference on Artificial Intelligence (AAAI), 2007, pp. 788–793.
[41] T.D. Hansen, P.B. Miltersen, T.B. Sørensen, On range of skill, in: Proceedings of the 28th Conference on Artificial Intelligence (AAAI), 2008, pp. 277–282.
[42] D. Koller, N. Megiddo, B. von Stengel, Efficient computation of equilibria for extensive two-person games, Games Econ. Behav. 14 (2) (1996) 247–259.
[43] T. Sandholm, The state of solving large incomplete-information games, and application to poker, AI Mag. 31 (4) (2010) 13–32.
[44] B. Bošanský, C. Kiekintveld, V. Lisý, M. Pěchouček, An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information, J. Artif. Intell. Res. 51 (2014) 829–866.
[45] R. Coulom, Efficient selectivity and backup operators in Monte-Carlo tree search, in: Proceedings of the 5th International Conference on Computers and Games (CG), in: Lect. Notes Comput. Sci., vol. 4630, 2006, pp. 72–83.
[46] L. Kocsis, C. Szepesvári, Bandit-based Monte Carlo planning, in: 15th European Conference on Machine Learning, in: Lect. Notes Comput. Sci., vol. 4212, 2006, pp. 282–293.
[47] S. Gelly, D. Silver, Monte-Carlo tree search and rapid action value estimation in computer Go, Artif. Intell. 175 (11) (2011) 1856–1875.
[48] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, O. Teytaud, The grand challenge of computer Go: Monte Carlo tree search and extensions, Commun. ACM 55 (3) (2012) 106–113.
[49] P. Ciancarini, G. Favini, Monte Carlo tree search in Kriegspiel, Artif. Intell. 174 (11) (2010) 670–684.
[50] P.I. Cowling, E.J. Powley, D. Whitehouse, Information set Monte Carlo tree search, IEEE Trans. Comput. Intell. AI Games 4 (2) (2012) 120–143.
[51] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed bandit problem, Mach. Learn. 47 (2–3) (2002) 235–256.
[52] M. Genesereth, N. Love, B. Pell, General game-playing: overview of the AAAI competition, AI Mag. 26 (2005) 73–84.
[53] H. Finnsson, Cadia-player: a general game playing agent, Master's thesis, Reykjavík University, 2007.
[54] H. Finnsson, Simulation-based general game playing, Ph.D. thesis, Reykjavík University, 2012.
[55] S. Samothrakis, D. Robles, S.M. Lucas, A UCT agent for Tron: initial investigations, in: Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games (CIG), 2010, pp. 365–371.
[56] P. Auer, N. Cesa-Bianchi, Y. Freund, R.E. Schapire, The nonstochastic multiarmed bandit problem, SIAM J. Comput. 32 (1) (2003) 48–77.
[57] P. Perick, D.L. St-Pierre, F. Maes, D. Ernst, Comparison of different selection strategies in Monte-Carlo tree search for the game of Tron, in: Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG), 2012, pp. 242–249.
[58] M. Lanctot, K. Waugh, M. Bowling, M. Zinkevich, Sampling for regret minimization in extensive games, in: Adv. Neural Inf. Process. Syst., 2009, pp. 1078–1086.
[59] V. Kovařík, V. Lisý, Analysis of Hannan consistent selection for Monte Carlo tree search in simultaneous move games, CoRR, arXiv:1509.00149.
[60] M. Lanctot, C. Wittlinger, M.H.M. Winands, N.G.P. Den Teuling, Monte Carlo tree search for simultaneous move games: a case study in the game of Tron, in: Proceedings of the 25th Benelux Conference on Artificial Intelligence (BNAIC), 2013, pp. 104–111.
[61] M.J.W. Tak, M.H.M. Winands, M. Lanctot, Monte Carlo tree search variants for simultaneous move games, in: Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG), 2014, pp. 232–239.
[62] T. Pepels, M.H. Winands, M. Lanctot, Real-time Monte Carlo tree search for Ms Pac-Man, IEEE Trans. Comput. Intell. AI Games 6 (3) (2014) 245–257.
[63] D. Perez, E.J. Powley, D. Whitehouse, P. Rohlfshagen, S. Samothrakis, P.I. Cowling, S.M. Lucas, Solving the physical traveling salesman problem: tree search and macro actions, IEEE Trans. Comput. Intell. AI Games 6 (1) (2014) 31–45.
[64] R.-K. Balla, A. Fern, UCT for tactical assault planning in real-time strategy games, in: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2009, pp. 40–45.

[65] P.I. Cowling, M. Buro, M. Bida, A. Botea, B. Bouzy, M.V. Butz, P. Hingston, H. Muñoz-Avila, D. Nau, M. Sipper, Search in real-time video games, in: Artificial and Computational Intelligence in Games, in: Dagstuhl Follow-Ups, vol. 6, 2013, pp. 1–19.
[66] M.G. Bellemare, Y. Naddaf, J. Veness, M. Bowling, The arcade learning environment: an evaluation platform for general agents, J. Artif. Intell. Res. 47 (2013) 253–279.
[67] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, M. Preuss, A survey of real-time strategy game AI research and competition in StarCraft, IEEE Trans. Comput. Intell. AI Games 5 (4) (2013) 293–311.
[68] A. Kovarsky, M. Buro, Heuristic search applied to abstract combat games, Adv. Artif. Intell. (2005) 55–77.
[69] F. Sailer, M. Buro, M. Lanctot, Adversarial planning through strategy simulation, in: IEEE Symposium on Computational Intelligence and Games (CIG), 2007, pp. 37–45.
[70] V. Lisý, B. Bošanský, M. Jakob, M. Pěchouček, Adversarial search with procedural knowledge heuristic, in: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2009, pp. 899–906.
[71] D. Churchill, A. Saffidine, M. Buro, Fast heuristic search for RTS game combat scenarios, in: 8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), 2012, pp. 112–117.
[72] A. Reinefeld, An improvement to the scout tree-search algorithm, ICCA J. 6 (4) (1983) 4–14.
[73] S. Hart, A. Mas-Colell, A simple adaptive procedure leading to correlated equilibrium, Econometrica 68 (5) (2000) 1127–1150.
[74] M. Lanctot, Monte Carlo sampling and regret minimization for equilibrium computation and decision-making in large extensive form games, Ph.D. thesis, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, June 2013.
[75] S. Gelly, D. Silver, Combining online and offline learning in UCT, in: Proceedings of the 24th International Conference on Machine Learning, 2007, pp. 273–280.
[76] R. Lorentz, Amazons discover Monte-Carlo, in: Proceedings of the 6th International Conference on Computers and Games (CG), in: Lect. Notes Comput. Sci., vol. 5131, 2008, pp. 13–24.
[77] M.H.M. Winands, Y. Björnsson, J.-T. Saito, Monte Carlo tree search in Lines of Action, IEEE Trans. Comput. Intell. AI Games 2 (4) (2010) 239–250.
[78] R. Lorentz, T. Horey, Programming Breakthrough, in: Proceedings of the 8th International Conference on Computers and Games (CG), in: Lect. Notes Comput. Sci., vol. 8427, 2013, pp. 49–59.
[79] M. Lanctot, M.H.M. Winands, T. Pepels, N.R. Sturtevant, Monte Carlo tree search with heuristic evaluations using implicit minimax backups, in: Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG), 2014, pp. 341–348.
[80] R. Ramanujan, B. Selman, Trade-offs in sampling-based adversarial planning, in: Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS), 2011, pp. 202–209.
[81] M. Lanctot, A. Saffidine, J. Veness, C. Archibald, M.H.M. Winands, Monte Carlo *-minimax search, in: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI), 2013, pp. 580–586.
[82] K.Q. Nguyen, R. Thawonmas, Monte Carlo tree search for collaboration control of ghosts in Ms. Pac-Man, IEEE Trans. Comput. Intell. AI Games 5 (1) (2013) 57–68.
[83] S. Smith, D. Nau, An analysis of forward pruning, in: Proceedings of the National Conference on Artificial Intelligence, 1995, p. 1386.
[84] N.G.P. Den Teuling, M.H.M. Winands, Monte-Carlo Tree Search for the simultaneous move game Tron, in: Proceedings of Computer Games Workshop (ECAI), 2012, pp. 126–141.
[85] M. Ponsen, S. de Jong, M. Lanctot, Computing approximate Nash equilibria and robust best-responses using sampling, J. Artif. Intell. Res. 42 (2011) 575–605.
[86] R. Gibson, M. Lanctot, N. Burch, D. Szafron, M. Bowling, Generalized sampling and variance in counterfactual regret minimization, in: Proceedings of the 26th Conference on Artificial Intelligence (AAAI), 2012, pp. 1355–1361.

Appendix C

Approximating maxmin strategies in imperfect recall games using A-loss recall property

International Journal of Approximate Reasoning 93 (2018) 290–326

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

Approximating maxmin strategies in imperfect recall games using A-loss recall property

Jiří Čermák *, Branislav Bošanský *, Karel Horák, Viliam Lisý, Michal Pěchouček

Department of Computer Science, Czech Technical University in Prague, Czechia

ARTICLE INFO

Article history:
Received 27 June 2017
Received in revised form 14 November 2017
Accepted 17 November 2017
Available online 23 November 2017

Keywords:
Imperfect recall
Abstraction
Maxmin strategy
A-loss recall

ABSTRACT

Extensive-form games with imperfect recall are an important model of dynamic games where the players are allowed to forget previously known information. Often, imperfect recall games result from an abstraction algorithm that simplifies a large game with perfect recall. Solving imperfect recall games is known to be a hard problem, and thus it is useful to search for a subclass of imperfect recall games which offers sufficient memory savings while being efficiently solvable. The abstraction process can then be guided to result in a game from this class. We focus on a subclass of imperfect recall games called A-loss recall games. First, we provide a complete picture of the complexity of solving imperfect recall and A-loss recall games. We show that the A-loss recall property allows us to compute a best response in polynomial time (computing a best response is NP-hard in imperfect recall games). This allows us to create a practical algorithm for approximating maxmin strategies in two-player games where the maximizing player has imperfect recall and the minimizing player has A-loss recall. This algorithm is capable of solving some games with up to $5 \cdot 10^9$ states in approximately 1 hour. Finally, we demonstrate that the use of imperfect recall abstraction can reduce the size of the strategy representation to as low as 0.03% of the size of the strategy representation in the original perfect recall game without sacrificing the quality of the maxmin strategy obtained by solving this abstraction.

1. Introduction

Extensive-form games (EFGs) are a model of dynamic games with a finite number of moves and are capable of describing scenarios with stochastic events and imperfect information. EFGs can model recreational games, such as poker, as well as real-world situations in physical security [1], auctions [2], or medicine [3]. EFGs are represented as game trees where nodes correspond to states of the game and edges to actions of players. Imperfect information is modeled by grouping indistinguishable states into information sets.

There are two approaches to making decisions in EFGs. First, there are online (or game-playing) algorithms which given the observations of the game state compute the action to be played. Second, there are offline algorithms which compute (approximate) the strategy in the whole game and play according to this strategy. The latter algorithms typically provide a better approximation of equilibrium strategies in large games compared to online algorithms [4]. One exception is the recently introduced continual resolving algorithm used in DeepStack [5], which provides less exploitable strategies than existing offline algorithms in heads-up no-limit Texas Hold'em poker. The main caveat of this algorithm is that it exploits

* Corresponding authors.
E-mail addresses: cermak@agents.fel.cvut.cz (J. Čermák), bosansky@agents.fel.cvut.cz (B. Bošanský), horak@agents.fel.cvut.cz (K. Horák), lisy@agents.fel.cvut.cz (V. Lisý), pechoucek@agents.fel.cvut.cz (M. Pěchouček).

https://doi.org/10.1016/j.ijar.2017.11.010
0888-613X/

the specific structure of poker where all actions of players are observable and its generalization to other games is not straightforward. Therefore, we focus on offline algorithms.

The offline algorithms are useful in real-world applications since the strategy (a probabilistic distribution over actions in each information set) is precomputed and can be simply stored on any device. It can then be accessed by deployed units such as park rangers (see, e.g., [6]) without the need of large computational resources or good internet connection necessary when using online algorithms. The main complication of offline algorithms is, however, the size of the strategy that needs to be stored. Most of the existing offline algorithms [7–9] require that players remember all the information gained during the game – a property denoted as a *perfect recall*. The main disadvantage of perfect recall is that the size of the strategy grows exponentially with the number of moves, as the perfect memory allows the player to condition his behavior on all his actions taken in the past. Therefore, a popular approach is to use *abstractions* [10] – create an abstracted game by merging certain information sets to reduce the size of the strategy representation, solve the abstracted game, and translate the resulting strategy back to the original game. The majority of existing algorithms (e.g., see [11–13]) create perfect recall abstractions, where the requirement of perfect memory severely limits possible reductions in the size of strategies, as it still grows exponentially with the increasing number of moves in the abstracted game. To achieve additional memory savings, the assumption of perfect recall may need to be violated in the abstracted game resulting in an *imperfect recall game*.

Solving imperfect recall games is known to be a difficult problem (see, e.g., [14–16]). Since we are interested mainly in solving imperfect recall games created by an abstraction algorithm, we focus on finding an efficiently solvable subclass of imperfect recall games. The abstraction algorithm can then be guided to result in a game from this class. Existing approaches create very specific abstracted games, so that perfect recall algorithms are still applicable: e.g., in *chance relaxed skew well-formed games* [17,18] or in *normal-form games with sequential strategies* [19,1]. The restrictions posed by these classes are unnecessarily strict, which can prevent us from fully exploiting the possibilities of abstractions and compact representation of dynamic games. We focus on a different subclass of imperfect recall games called *A-loss recall games* [20,21] where each loss of a memory of a player can be traced back to forgetting his own actions.

We provide the following contributions. We present a complete picture of the problem of solving imperfect recall games and show which computational tasks become easier when restricting to A-loss recall. Next, we use the properties of the A-loss recall to provide the first family of algorithms capable of approximating the strategies with the best worst-case expected outcome (maxmin strategies). In order to achieve this result, we require only the minimizing player to have A-loss recall, while the maximizing player is allowed to have imperfect recall. Finally, we experimentally demonstrate the effectiveness of the use of imperfect recall abstractions to reduce the size of strategies to be stored.

One of the most important theoretical properties discussed in this paper is the complexity of computing a best response since it is a subproblem of many algorithms solving EFGs. In general, it is NP-hard to find a best response in imperfect recall game. In games where the best responding player has A-loss recall, however, finding a best response can be done using the same algorithm as in the perfect recall case. Hence the problem is polynomially solvable [20,21]. We use this property to design the first family of algorithms for approximating maxmin strategies in imperfect recall games where the maximizing player has imperfect recall and the minimizing player has A-loss recall. Additionally, we provide novel necessary and sufficient (i.e., if and only if) condition for the existence of a Nash equilibrium (NE) in behavioral strategies in A-loss recall games, making A-loss recall games the only subclass of imperfect recall games for which such condition is known. Thus we show that A-loss recall forms an interesting subclass of imperfect recall. On the other hand, we extend the known hardness results of computing solution concepts in imperfect recall games due to Koller and Meggido [15] and Hansen [16] to A-loss recall games.[1]

From the computational perspective, we provide a novel approximate algorithm, denoted IRABnB (Imperfect Recall Abstraction Branch-and-Bound algorithm), for computing maxmin strategies in imperfect recall games where the maximizing player has imperfect recall and the minimizing player has A-loss recall.[2] We base the algorithm on the sequence-form linear program for computing maxmin strategies in perfect recall games [7,24] extended by bilinear constraints necessary for the correct representation of strategies of the maximizing player in imperfect recall games. We approximate the bilinear terms using recent Multiparametric Disaggregation Technique (MDT) [25] and provide a mixed-integer linear program (MILP) for approximating maxmin strategies. We propose a novel branch-and-bound algorithm that repeatedly solves the linear relaxation of the MILP. It simultaneously tightens the constraints that approximate bilinear terms and searches for the optimal assignment to the relaxed binary variables from the MILP. We prove that the algorithm has guaranteed convergence to maxmin strategy and we provide a bound on the number of steps needed.

We further extend the IRABnB algorithm by incremental strategy generation technique. The resulting algorithm is denoted DOIRABnB[3] (Double Oracle Imperfect Recall Abstraction Branch-and-Bound algorithm). While such techniques exist

[1] A part of this work appeared in [22]. Here we significantly improve the proofs and overall presentation. We further strengthen the relationship between A-loss recall games and chance relaxed skew well-formed games and provide more examples.

[2] A part of this work appeared in [23]. Here we mainly improve the description of the algorithm and proofs. We also provide a discussion of the heuristics used (section 5.2.1). Notice that in this paper we refer to the BnB algorithm from [23] as IRABnB to remove the naming clash with general branch-and-bound algorithm.

[3] A part of this work appeared in [26]. Here, we provide an improved version of the DOIRABnB algorithm which is more efficient. Furthermore, we significantly improve the description of the algorithm and the proof of correctness. Finally, we extend the experimental evaluation of the algorithm. Notice that in this paper we refer to the DOBnB algorithm from [26] as DOIRABnB to improve the clarity of presentation.

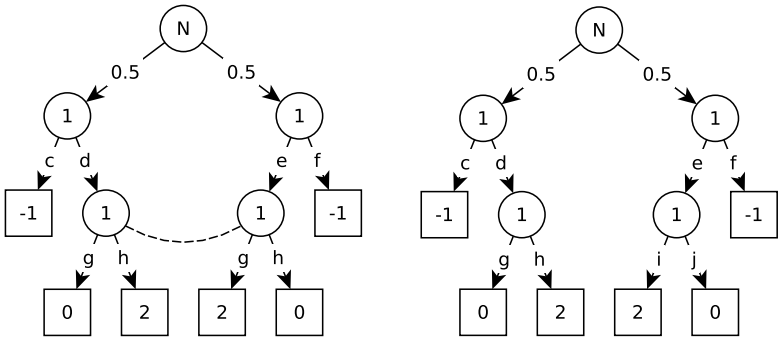


Fig. 1. (Left) An imperfect recall game. (Right) Its coarsest perfect recall refinement. Circle nodes represent the states of the game, numbers in the circles show which player acts in that node (player 1, player 2 or chance player N), dashed lines represent indistinguishable states and box nodes are the terminal states with utility value for player 1 (the game is zero-sum, hence player 1 maximizes the utility, player 2 minimizes it).

for perfect recall games [27], transferring the ideas to imperfect recall games presents a number of challenges that we address in this paper. First, we define the restricted EFG that is a subset of the original EFG. Second, we describe how the restricted game is solved via the IRABnB search and describe the details of the integration of these two iterative algorithms. Third, we describe how to expand the restricted EFG so that our algorithm preserves guarantees for approximating maxmin strategies. The experimental evaluation shows that DOIRABnB significantly improves the scalability of IRABnB. The algorithm is capable of solving some games with up to $5 \cdot 10^9$ states in approximately 1 hour.

Finally, we experimentally demonstrate the effectiveness of the use of imperfect recall abstractions to reduce the size of strategies to be stored. We show that employing simple abstractions which still allow us to compute the maxmin strategy of the original game can lead to strategies with the relative size as low as 0.03% of the size of the strategy in the original unabstracted game.

2. Extensive-form games

A two-player extensive-form game (EFG) is a tuple $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, u, \mathcal{C}, \mathcal{I})$, which is commonly visualized as a game tree (see Fig. 1). $\mathcal{N} = \{1, 2\}$ is a set of players, by i we refer to one of the players, and by $-i$ to his opponent. Additionally, the chance player (or nature) N represents the stochastic environment of the game. $\mathcal{A}$ denotes the set of all actions labeling the edges of the game tree. $\mathcal{H}$ is a finite set of *histories* of actions taken by all players and the chance player from the root of the game. Each history corresponds to a *node* in the game tree; hence, we use the terms history and node interchangeably. $\mathcal{Z} \subseteq \mathcal{H}$ is the set of all *terminal states* of the game corresponding to the leaves of the game tree. For each $z \in \mathcal{Z}$ and $i \in \mathcal{N}$ we define a *utility function* $u_i : \mathcal{Z} \to \mathbb{R}$. If $u_i(z) = -u_{-i}(z)$ for all $z \in \mathcal{Z}$, we say that the game is zero-sum. Chance player selects actions based on a fixed probability distribution known to all players. Function $\mathcal{C} : \mathcal{H} \to [0, 1]$ is the probability of reaching h obtained as the product of probabilities of actions of chance player preceding h. Imperfect observation of player i is modeled via *information sets* $\mathcal{I}_i$ that form a partition over $h \in \mathcal{H}$ where i takes action. Player i cannot distinguish between nodes in any $I_i \in \mathcal{I}_i$. We represent the information sets as nodes connected by dashed lines in the examples. $\mathcal{A}(I_i)$ denotes actions available in each $h \in I_i$. The action a uniquely identifies the information set where it is available, i.e., for all distinct $I, I' \in \mathcal{I}$ $\forall a \in \mathcal{A}(I)$ $\forall a' \in \mathcal{A}(I')$ $a \neq a'$. An ordered list of all actions of player i from the root to node h is referred to as a *sequence*, $\sigma_i = seq_i(h)$. Σ_i is a set of all sequences of player i. We use $seq_i(I_i)$ as a set of all sequences of player i leading to I_i. We use $inf_i(\sigma_i)$ as a set of all information sets to which sequence σ_i leads. A game has *perfect recall* iff $\forall i \in \mathcal{N}$ $\forall I_i \in \mathcal{I}_i$, for all $h, h' \in I_i$ holds that $seq_i(h) = seq_i(h')$. If there exists at least one information set where this does not hold (denoted as *imperfect recall information set*), the game has *imperfect recall*. We use $\mathcal{I}_i^{IR}$ as a set of all imperfect recall information sets of player i.

Finally, to be able to discuss the effect of imperfect recall in any given imperfect recall game G, we need to be able to construct a corresponding perfect recall game G' by adding the minimum amount of information for players to have perfect recall in G. We denote G' as the *coarsest perfect recall refinement* of G. To do this, we first define a partition $H(I_i)$ of states in every information set I_i of some imperfect recall game G to the largest possible subsets, not causing imperfect recall. More formally, let $H(I_i) = \{H_1, \ldots, H_n\}$ be a disjoint partition of all $h \in I_i$, where $\bigcup_{j=1}^{n} H_j = I_i$ and $\forall H_j \in H(I_i)$ $\forall h_k, h_l \in H_j : seq_i(h_k) = seq_i(h_l)$, additionally for all distinct $H_k, H_l \in H(I_i) : seq_i(H_k) \neq seq_i(H_l)$.

Definition 1. The coarsest perfect recall refinement G' of the imperfect recall game $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, u, \mathcal{C}, \mathcal{I})$ is a tuple $(\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}', u, \mathcal{C}, \mathcal{I}')$, where $\forall i \in \mathcal{N}$ $\forall I_i \in \mathcal{I}_i$, $H(I_i)$ defines the information set partition $\mathcal{I}'$. $\mathcal{A}'$ is a modification of $\mathcal{A}$, which guarantees that $\forall I \in \mathcal{I}'$ $\forall h_k, h_l \in I$ $\mathcal{A}'(h_k) = \mathcal{A}'(h_l)$, while for all distinct $I^k, I^l \in \mathcal{I}'$ $\forall a^k \in \mathcal{A}(I^k)$ $\forall a^l \in \mathcal{A}(I^l)$ $a^k \neq a^l$. We can limit the coarsest perfect recall refinement to player i and leave the information set structure of $-i$ unchanged.

In Fig. 1 we show an example of an imperfect recall game (left) and its coarsest perfect recall refinement (right).

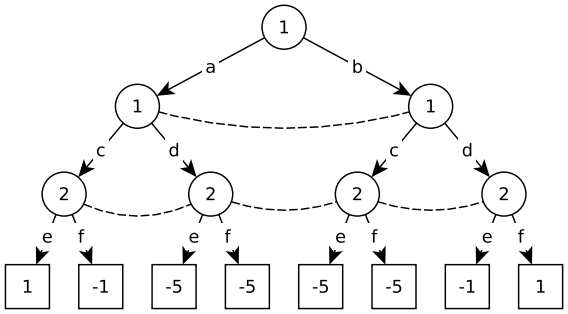


Fig. 2. An A-loss recall game where the Nash equilibrium in behavioral strategies does not exist (from [14]).

Notice, that in the Definition 1 we change the labeling of actions described by $\mathcal{A}$ to $\mathcal{A}'$, since we modify the structure of the imperfect information $\mathcal{I}$ to $\mathcal{I}'$ (e.g., actions g, h in Fig. 1 (left) being relabel to g, h and i, j due to the split of the information set in Fig. 1 (right)). Let E be the set of all edges of the game tree of G corresponding to actions of all players except the chance player. By $\Phi' : E \to \mathcal{A}'$ we denote a function which for an edge $e \in E$ returns its action label in G', similarly we define $\Phi : E \to \mathcal{A}$. When we talk about the equivalence of arbitrary strategy representation in G and G', we talk about the equivalence with respect to Φ and Φ'. Same goes for applying the strategy from G to G' and vice versa.

2.1. Strategies in imperfect recall games

There are several representations of strategies in EFGs. A *pure strategy* s_i for player i is a mapping assigning $\forall I_i \in \mathcal{I}_i$ an element of $\mathcal{A}(I_i)$. $\mathcal{S}_i$ is a set of all pure strategies for player i. A *mixed strategy* m_i is a probability distribution over $\mathcal{S}_i$, set of all mixed strategies of i is denoted as $\mathcal{M}_i$. *Behavioral strategy* b_i assigns a probability distribution over $\mathcal{A}(I_i)$ for each I_i. $\mathcal{B}_i$ is a set of all behavioral strategies for i, $\mathcal{B}_i^p \subseteq \mathcal{B}_i$ is the set of deterministic behavioral strategies for i. A *strategy profile* is a set of strategies, one strategy for each player.

Definition 2. A pair of strategies x_i, y_i of player i with arbitrary representation is *realization equivalent* if $\forall z \in \mathcal{Z} : \pi_i^{x_i}(z) = \pi_i^{y_i}(z)$, where $\pi_i^{x_i}(z)$ is a probability that z is reached due to strategy x_i of player i when the rest of the players play to reach z.

We overload the notation and use u_i as the expected utility of i when the players play according to pure (mixed, behavioral) strategies.

Behavioral strategies and mixed strategies have the same expressive power in perfect recall games, but it can differ in imperfect recall games [28].

Example 1. Consider the game depicted in Fig. 2. This game has 4 pure strategies for player 1 $\mathcal{S}_1 = \{(a, c), (a, d), (b, c), (b, d)\}$. A mixed strategy can condition the actions of players on information that the players should no longer have available. For example, a mixed strategy where (a, c) and (b, d) are played with a uniform probability 0.5 allows player 1 to condition playing c and d on the outcome of his stochastic choice in the root of the game, and thus randomize between the leftmost and the rightmost state in information set of player 2. Note that one cannot model the same behavior using a behavioral strategy that assigns a probability distribution over the actions available in every decision point without conditioning on any previous knowledge. Therefore no additional information can be disclosed to the player.

Moreover, the size of these representations differs significantly. Mixed strategies of player i state probability distribution over $\mathcal{S}_i$, where $|\mathcal{S}_i| \in \mathcal{O}(2^{|\mathcal{Z}|})$, behavioral strategies create probability distribution over the set of actions (note that its size is proportional to the number of information sets, which can be exponentially smaller than $|\mathcal{Z}|$). Hence when one wants to exploit the space savings caused by the reduced number of information sets in imperfect recall games, behavioral strategies need to be used.

Next, we define the maxmin strategy and Nash equilibrium in behavioral strategies.

Definition 3. We say that b_i^* is a *maxmin strategy* iff

$$b_i^* = \arg\max_{b_i \in \mathcal{B}_i} \min_{b_{-i} \in \mathcal{B}_{-i}} u_i(b_i, b_{-i}).$$

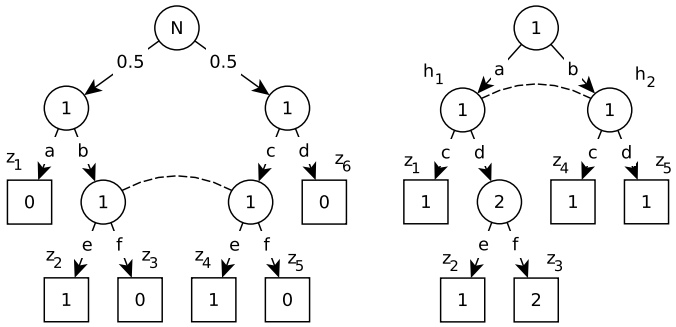


Fig. 3. (Left) A well-formed game which does not have A-loss recall. (Right) An A-loss recall game which is not chance relaxed skew well-formed.

Informally, maxmin strategy of player i maximizes the worst case expected outcome with respect to the behavior of $-i$.[4]

Definition 4. We say that strategy profile $b = \{b_i^*, b_{-i}^*\}$ is a *Nash equilibrium* (NE) in behavioral strategies iff $\forall i \in \mathcal{N}\ \forall b_i' \in \mathcal{B}_i$: $u_i(b_i^*, b_{-i}^*) \geq u_i(b_i', b_{-i}^*)$.

Informally, a strategy profile is a NE if no player wants to deviate to a different strategy.
Finally, we define the exploitability of a strategy.

Definition 5. We define the exploitability of a strategy b_i as

$$\min_{b_{-i} \in \mathcal{B}_{-i}} u_i(b_i, b_{-i}).$$

Informally, the exploitability of a strategy b_i is the worst case utility of i achievable when playing b_i.

2.2. A-loss recall games

In this section we formally define the subclass of imperfect recall games called A-loss recall games [20,21] and show the relationship of A-loss recall games to the only known subclasses of imperfect recall games being solved in the literature, namely well-formed games, skew well-formed games and chance relaxed skew well-formed games [17,18].

Definition 6. Player i has *A-loss recall* if and only if for every $I \in \mathcal{I}_i$ and nodes $h, h' \in I$ it holds either (1) $seq_i(h) = seq_i(h')$, or (2) $\exists I' \in \mathcal{I}_i$ and two distinct actions $a, a' \in \mathcal{A}_i(I'), a \neq a'$ such that $a \in seq_i(h) \wedge a' \in seq_i(h')$.

Condition (1) in the definition says that if player i has perfect recall then she also has A-loss recall. Condition (2) can be interpreted as requiring that each loss of memory of A-loss recall player can be traced back to some loss of memory of the player's own previous actions in one information set.

2.2.1. Well-formed games

The only known subclasses of imperfect recall games using the imperfect recall to reduce the memory requirements of strategy representation are chance relaxed skew well-formed games (CRSWFG) and their subsets well-formed games (WFG) and skew well-formed (SWFG) [17,18]. These classes of games restrict the structure of imperfect recall by requiring similarity of the states included in one imperfect recall information set both in the structure of past and future moves as well as the utilities reachable from the states. As a consequence, the perfect recall algorithms (namely Counterfactual regret minimization (CFR) [8]) are still guaranteed to converge to (ε-)NE in these games (see Appendix F for more details on applying CFR to imperfect recall games and CRSWFG).

Definition 7. We say that the imperfect recall game G is well-formed with respect to some perfect recall refinement G' of G if for all $i \in \mathcal{N}$, $I \in \mathcal{I}_i$, I', I'' (where I', I'' are information sets in G' which are unified to I) there exists a bijection $\alpha : \mathcal{Z}_{I'} \to \mathcal{Z}_{I''}$ and constants $k_{I',I''}, l_{I',I''} \in [0, \infty)$ such that for all $z \in \mathcal{Z}_{I'}$:

1. $u_i(z) = k_{I',I''} u_i(\alpha(z))$,
2. $\mathcal{C}(z) = l_{I',I''} \mathcal{C}(\alpha(z))$,

[4] Notice, that in maxmin the minimizing player takes into account only the utility of the maximizing player. Hence, we do not restrict the results concerning maxmin in the following sections to zero-sum games, as the utility of the minimizing player can be arbitrary.

3. in G, $seq_{-i}(z) = seq_{-i}(\alpha(z))$, and
4. in G, $seq_i(z[I'], z) = seq_i(\alpha(z)[I''], \alpha(z))$,

where $\mathcal{Z}_I$ stands for terminal states reachable from states in I, $z[I]$ denotes the state in I reached when moving from the root of the game to z and $seq_i(h, h')$ is a sequence of actions needed to reach h' from h. We say that G is well-formed game if it is well-formed with respect to some perfect recall refinement.

Both SWFG and CRSWFG relax only condition (1) and (2) in the well-formed game definition and still require condition (3) and (4) to hold (see [17,18] for more details).

While A-loss recall games restrict the structure of the game only above the imperfect recall information set (the requirement of being able to connect any loss of memory to forgetting player's own actions), the WFG, SWFG, and CRSWFG restrict the structure above, below and also the structure of the utilities. In the case of games with no chance, we can formally define the relationship between A-loss recall games and WFG, SWFG, and CRSWFG.

Lemma 1. *In games with no chance, the WFG, SWFG, and CRSWFG form a strict subset of A-loss recall games.*

Proof. We first prove the Lemma for WFG and then show that it extends to both SWFG and CRSWFG.

The only requirement in A-loss recall games is that players are able to connect any loss of memory to forgetting their own actions, hence the restriction in the information set I always concerns only the part of the tree above I. We, therefore, focus on condition (3) in the definition of WFG, since it is the only one restricting the upper part of the game tree. Condition (3) requires that for each $h' \in I'$ there must exist $h'' \in I''$ such that $seq_{-i}(h') = seq_{-i}(h'')$, which, combined with the assumption that there is no chance player, implies that there must exist difference in $seq_i(h')$ and $seq_i(h'')$. Furthermore, since $seq_{-i}(h') = seq_{-i}(h'')$ we are sure that there must exist $\bar{I} \in \mathcal{I}_i$ and distinct $a, a' \in \mathcal{A}(\bar{I})$ such that $a \in seq_i(h')$ and $a' \in seq_i(h'')$ which is exactly the condition (2) in the A-loss recall property. The rest of the requirements of the WFG restricts utilities and parts of the tree not restricted in A-loss recall games. Therefore the WFG form a subset of A-loss recall games.

Notice that the Lemma also holds for SWFG and CRSWFG, since they provide relaxations in conditions (1) and (2) only. □

In Fig. 3 (left) we show a WFG (with $\alpha : \{z_2 \to z_4, z_3 \to z_5\}$) with chance which does not have A-loss recall. The game does not have A-loss recall since player 1 forgets the information about the move of chance in the root of the game. Finally, in Fig. 3 (right) we present an A-loss recall game which is not CRSWFG. The game is not CRSWFG since the leaves z_2 and z_3 reachable from h_1 cannot be mapped to any leaf after h_2 without breaking condition 3 in Definition 7. Notice that when assuming rational player 2, however, it is safe to include h_1 and h_2 to one information set, since the expected utilities after every action available in h_1, h_2 are equal, and so any behavior optimal in h_1 is optimal in h_2. Hence CRSWFG are unnecessarily conservative in the restrictions posed on the game tree.

2.3. Best response computation

One of the main computational components in algorithmic game theory is the problem of computing a best response. Formally, a strategy of a player (e.g., pure, mixed, behavioral) is a best response to a given strategy of his opponent if its expected utility is maximal against this strategy compared to all other strategies from the particular class. To denote the best response regardless of the type of the strategy (i.e., regardless whether we consider mixed or behavioral strategies), we use the term *ex-ante best response*.

In perfect recall EFGs, it is sufficient to consider a pure best response. However, this is no longer true in imperfect recall EFGs. Consider a one-player game called the absentminded driver [29] depicted on the left in Fig. 4. We see that by playing any pure, or even a mixed strategy, the player cannot reach outcome higher than 1. Note that in a mixed strategy a player samples a pure strategy from the given distribution before the game begins; hence, there is no randomization when the information set is reached, and player 1 always follows the pure strategy sampled from the mixed strategy. When using a behavioral strategy, however, player samples the action from a given distribution independently every time an information set is reached. Hence, the ex-ante optimal strategy is a behavioral strategy $b(s) = \frac{2}{3}$ that reaches the expected value of $\frac{4}{3}$. In general, we need to consider randomized best responses in so-called *absent minded* games (games where there exists a path from the root of the game tree to some leaf such that at least one information set $I_i \in \mathcal{I}_i$ is visited more than once). In the following text, we will assume that the games are without absentmindedness, where it is sufficient to consider only pure strategies to find an ex-ante best response.

Lemma 2. *Let G be an imperfect recall game without absentmindedness and b_1 strategy of player 1. There exists an* ex-ante *pure best response of player 2.*

The proof is based on the fact that we search for an optimum of a multilinear function with independent variables over a convex polytope with vertices corresponding to pure strategies (see Appendix A for full proof).

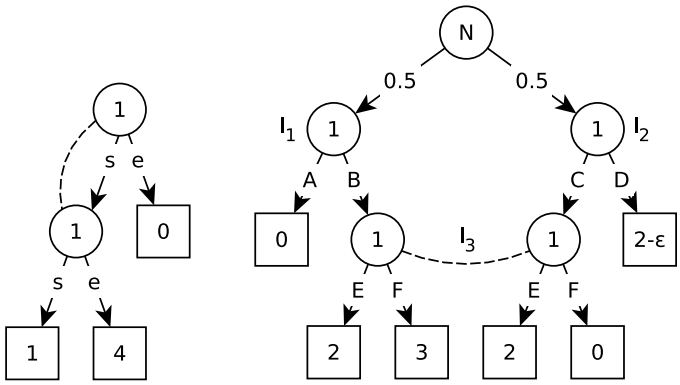


Fig. 4. (Left) Absentminded Driver. (Right) An EFG without A-loss recall where a time consistent best response (playing the best action in an information set) is not necessarily the ex-ante best response.

The problem of finding an ex-ante best response in games without absentmindedness is still NP-hard (follows from complexity results in [15]) while the problem is easy (polynomial) in perfect recall games. The main difference is that in imperfect recall games an ex-ante best response cannot be found by selecting an action with the highest expected utility to be played in each information set (called a *time consistent strategy* [21]). This is caused by the fact that the belief in an information set of a player is not perfectly determined by the strategy of the opponent and nature, but also by the strategy of the best-responding player.

Consider the game in Fig. 4 (right) between player 1 and chance. The ex-ante best response of player 1 in this game is to play B,D,F getting the utility of $\frac{5-\varepsilon}{2}$. Note, however, that since the belief of player 1 in his imperfect recall information sets depends on his behavior above the information set, one can reach a time consistent strategy playing B,C,E with the expected utility of 2. This strategy is time consistent since when checking every information set separately, there is no deviation of player 1, which could increase his expected value. Note that player 1 does not have A-loss recall in the game in Fig. 4 (right) since parents of the nodes in the information set I_3 are in two distinct information sets I_1, I_2 and their common predecessor is a chance node.

The equivalence between time consistent strategies and ex-ante best responses is shown to hold in A-loss recall games. Consequently, the computation of the best response is in P in A-loss recall games [20,21]. The following lemmas are a consequence of these facts.

Lemma 3. *Let G be an imperfect recall game where player i has A-loss recall. Let G' be the coarsest perfect recall refinement of G for player i. Every pure behavioral strategy b'_i of player i from G' has realization equivalent pure behavioral strategy b_i in G and vice versa.*

Lemma 4. *Let G be an imperfect recall game where player 2 has A-loss recall and b_1 a strategy of player 1. Let G' be the coarsest perfect recall refinement of G for player 2. Let b'_2 be a pure best response to b_1 in G' and let b_2 be a realization equivalent behavioral strategy to b'_2 in G, then b_2 is a pure best response to b_1 in G.*

The proofs of both lemmas can be found in Appendix A.

Lemma 4 allows us to formulate the concise mathematical program described in Section 4 which is used as the core of algorithms discussed in Section 5.

3. NE and maxmin strategies in A-loss recall games

To provide a complete picture of the complexity of solving A-loss recall games, we discuss the existence, numerical representation and computational complexity of maxmin and NE behavioral strategies in A-loss recall games.

3.1. Existence of NE in A-loss recall games

The guarantee of the existence of NE in finite games due to Nash [30] assumes mixed strategy representation only. Hence, this guarantee does not extend to NE in behavioral strategies in imperfect recall games because of the different descriptive power of mixed and behavioral strategies there [14]. Here, we discuss the necessary and sufficient condition for the existence of NE in behavioral strategies in A-loss recall games. First, we show an example of the imperfect recall game due to Wichardt [14] which does not have a NE in behavioral strategies and show that it has A-loss recall, which implies that A-loss recall games need not have NE in behavioral strategies. We then provide novel sufficient and necessary (i.e., if and only if) condition for the existence of NE in behavioral strategies in two-player A-loss recall games. Note, that thanks to this result, A-loss recall games are the only subclass of imperfect recall games, for which such condition is known (the only exception are well-formed games, where NE in behavioral strategies always exists, and so the condition is trivial).

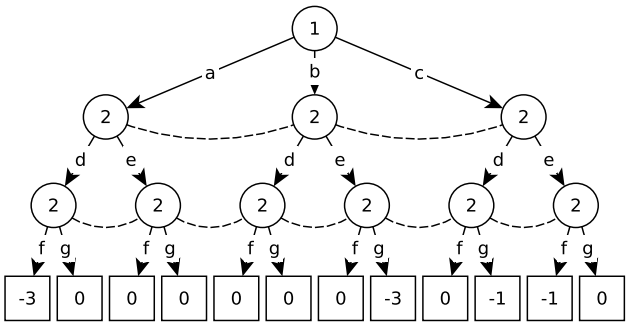


Fig. 5. An A-loss recall game where all maxmin strategies and NE require irrational numbers [15].

Informally, Theorem 1 states that A-loss recall game G has a NE in behavioral strategies if and only if there exists a behavioral NE in its coarsest perfect recall refinement G' which prescribes the same behavior in every information set which is connected to some imperfect recall information set of G.

Proposition 1. *The existence of NE in behavioral strategies is not guaranteed even in two-player zero-sum A-loss recall games.*

Proof. In Fig. 2 we present the imperfect recall game where there is no NE in behavioral strategies due to Wichardt [14]. This game has A-loss recall since only player 1 has imperfect recall and he forgets only his own choice in the root. This implies that the existence of NE is not guaranteed even in two-player zero-sum A-loss recall games. □

Theorem 1. *An A-loss recall game G has a NE in behavioral strategies if and only if there exists a NE strategy profile b in behavioral strategies of the coarsest perfect recall refinement G' of G, such that $\forall I \in \mathcal{I}$ of G, $\forall H_k, H_l \in H(I) : b(H_k) = b(H_l)$, where $b(H)$ stands for the behavioral strategy in the information set of G' formed by states in H.*

Proof. First, since b is a NE of G' we know that there exists no incentive for any player to deviate to any pure behavioral strategy in G'. From Lemma 3, it follows that there can exist no pure behavioral strategy in G to which any of the players want to deviate either. Additionally, from Lemma 2 it is sufficient to consider deviations to pure strategies in G since none of the players is absentminded. This, in combination with the fact, that b prescribes valid strategy in G implies that b is a NE in behavioral strategies of G.

Second, we prove that there exists no NE b' in behavioral strategies of G which is not a NE of G'. Let us assume that such b' exists. This would imply that there is no pure behavioral strategy in G to which players want to deviate when playing according to b', and therefore no pure behavioral strategy in G' either (Lemma 3), implying that b' is a NE in G'. This contradicts the assumption and completes the proof. □

Note that in general imperfect recall game this equivalence no longer holds. Consider the game in left subfigure of Fig. 1. Here the only NE in behavioral strategies is playing d and e deterministically and mixing uniformly between g, h. The only NE of its coarsest perfect recall refinement (shown in right subfigure of Fig. 1) is, however, playing d, e, h and i deterministically. For more details about the existence of NE in general imperfect recall games see Appendix B.

3.2. Representation of Nash equilibrium and maxmin strategies

In this section, we state negative results concerning strategy representation in A-loss recall games. In two-player perfect recall games with rational payoffs, there always exists a maxmin behavioral strategy which uses only rational probabilities [15]. In imperfect recall games, this no longer holds [15]. We present the example of the imperfect recall game provided in [15], where all maxmin strategies require irrational numbers and show that this game has A-loss recall. Moreover, since the maxmin strategies form a part of all the NE strategies of this game, we extend this result also to NE of A-loss recall games. It follows that computing exact maxmin and NE strategies requires exact representation of irrational numbers even in A-loss recall games.

Theorem 2. *All the maxmin strategies may require irrational numbers, even in two-player zero-sum A-loss recall game with rational payoffs.*

Proof. The example of the imperfect recall game used in [15] is depicted in Fig. 5. The maxmin strategy of player 2 is trying to maximize

$$\min\{3b_2(d)b_2(f), 3(1-b_2(d))(1-b_2(f)), b_2(d)(1-b_2(f))+(1-b_2(d))b_2(f)\}. \quad (1)$$

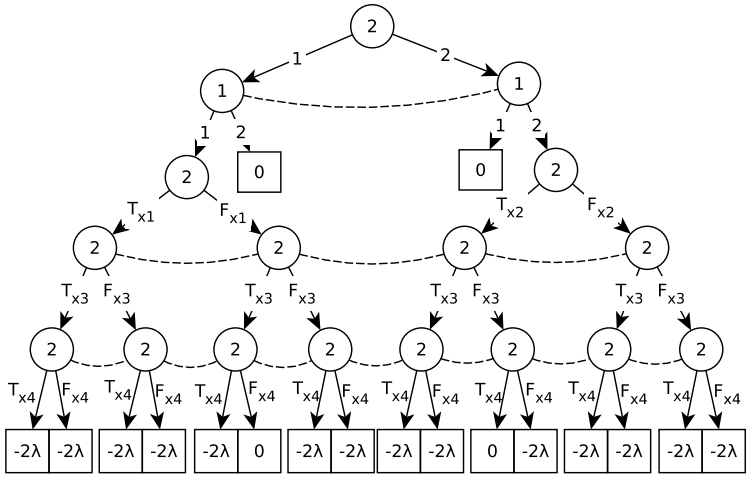


Fig. 6. An A-loss recall game reduction from Theorem 4 of 3-SAT problem $x_1 \vee \neg x_3 \vee x_4 \wedge \neg x_2 \vee x_3 \vee \neg x_4$.

This is maximized when

$$3b_2(d)b_2(f) = 3(1-b_2(d))(1-b_2(f)) = b_2(d)(1-b_2(f)) + (1-b_2(d))b_2(f), \tag{2}$$

which leads to $b_2(d) = 0.1(5 \pm \sqrt{5})$.

This game has A-loss recall since the only information player 2 forgets is his own choice in his first information set. □

Theorem 3. *All the Nash equilibrium strategies may require irrational numbers, even in two-player zero-sum A-loss recall game with rational payoffs.*

Proof. Strategy profiles $b_1(a) = b_1(b) = 0.2, b_2(d) = b_2(g) = 0.1(5 \pm \sqrt{5})$ form all the Nash equilibria of the game in Fig. 5. This holds since none of the players wants to deviate and the strategies for player 2 are the only solutions of eq. (2). Hence, it follows that any other strategy of player 2 has worse expected value against the best responding opponent, and therefore cannot be stable. □

3.3. Computational complexity in imperfect recall games

Now we turn to the computational complexity of solving imperfect recall games. Computing maxmin strategies is NP-hard in imperfect recall games [15] and it is NP-hard to decide whether there exists a NE in behavioral strategies in imperfect recall games [16] (both theorems and their proofs are presented in Appendix C for completeness). We show that both negative results directly translate to A-loss recall games. Notice that unlike ours, the reduction used in [16] requires a game with absentmindedness, hence we significantly extend the class of games for which it is known that deciding whether there exists a NE in behavioral strategies is NP-hard.

Theorem 4. *The problem of deciding whether player 2 having an A-loss recall can guarantee an expected payoff of at least λ is* NP*-hard even if player 1 has perfect recall, there are no chance moves, and the game is zero-sum.*

Proof. The proof is made by reduction from the 3-SAT problem. It is a modification of the original proof of Koller [15] for imperfect recall games. The example of the reduction is given in Fig. 6. Given n clauses $x_{j,1} \vee x_{j,2} \vee x_{j,3}$ we create a two person zero-sum game in the following way. In the root of the game player 2 chooses between n actions, each corresponding to one clause. Player 1 plays next with no information about the action chosen by player 2. He has again n actions, each corresponding to one clause. In every state of player 1, $n-1$ actions lead directly to a terminal state with utility 0 for player 1 and one action (corresponding to the same clause as the action of player 2 preceding this state) leads to a state of player 2. Every such state of player 2 corresponds to the variable $x_{j,1}$ where j is the index of the clause chosen in the root of the game. Every such state has actions $T_{x_{j,1}}, F_{x_{j,1}}$ available. These actions correspond to setting the variable $x_{j,1}$ to true or false respectively. After both $T_{x_{j,1}}, F_{x_{j,1}}$ in $x_{j,1}$ we reach the state representing the assignment to $x_{j,2}$ with the same setup (state representing the assignment to $x_{j,3}$ is reached after that). After the assignment to $x_{j,3}$ we reach the terminal state with utility $-n\lambda$ for player 1 if the assignment to $x_{j,1}$, $x_{j,2}$ and $x_{j,3}$ satisfies the clause $x_{j,1} \vee x_{j,2} \vee x_{j,3}$, 0 otherwise. The information sets of player 2 group together all the states corresponding to the assignment to one variable in the original 3-SAT problem (note that we assume that the order of variables in every clause follows some complete ordering on the whole set of variables in the 3-SAT problem).

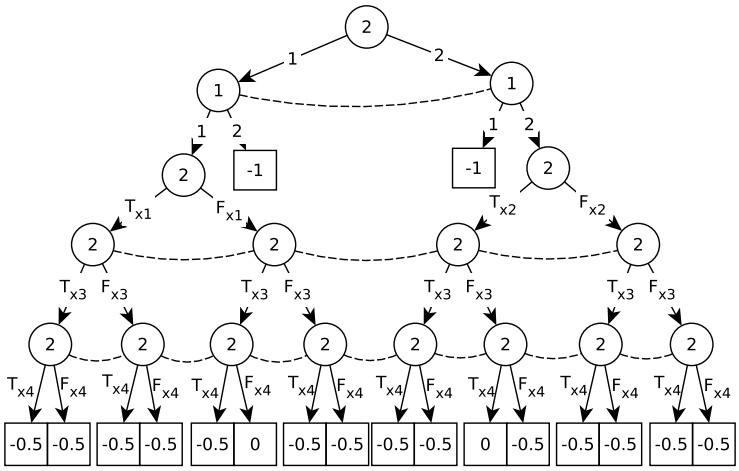


Fig. 7. An A-loss recall game reduction from Theorem 5 of 3-SAT problem $x_1 \vee \neg x_3 \vee x_4 \wedge \neg x_2 \vee x_3 \vee \neg x_4$.

We will show that player 2 can guarantee the worst case expected value λ if and only if the original 3-SAT problem is satisfiable. First, we show that if the original 3-SAT problem is satisfiable player 2 can guarantee the worst case expected value λ. The worst case expected value λ is achieved when player 2 mixes uniformly in the root of the game and plays according to the assignment which satisfies the original 3-SAT problem in the rest of the tree.

Next, we show that if player 2 can guarantee the worst case expected value λ, the original 3-SAT problem has to be satisfiable. There are two cases we need to discuss.

Case 1: Player 2 plays a non-uniform strategy b_2 in the root. In this case player 1 will play action $a \in \mathcal{A}_1$ corresponding to the same clause as the action $a_{min} \in \arg\min_{a \in \mathcal{A}_2(root)} b_2(a)$. Since b_2 is non-uniform in the root we know that $b_2(a_{min}) < \frac{1}{n}$ and hence the expected value of player 2 must be lower than $\frac{n\lambda}{n}$ no matter what happens in the rest of the game.

Case 2: The last chance to guarantee expected value λ is when player 2 plays a uniform strategy b_2 in the root. Here we show that λ can be guaranteed only when the corresponding 3-SAT problem is satisfiable. If the 3-SAT problem is not satisfiable, that means that there always exists a state h of player 2 after the action of player 1, where $u_2^h(b_2) < n\lambda$, where $u_2^h(b_2)$ stands for the expected value in h when player 2 plays according to b_2. By playing action leading to this state, player 1 guarantees that the expected value for player 2 is lower than $\frac{n\lambda}{n}$. If the 3-SAT is satisfiable on the other hand, the uniform b_2 in the root and playing according to the assignment satisfying the 3-SAT guarantees the expected value λ.

The reduction is polynomial, since the game has $n(n-1) + 2^3 n$ leaves.

The last thing which remains to be shown is that player 2 has A-loss recall. This is satisfied since any loss of information about actions of player 1 can be tracked back to forgetting his own action taken in the root or to setting some of the SAT variables to true or false. □

We leave the question whether the problem stated in Theorem 4 belongs to NP as an open problem. Even though Theorem 2 states that the solution to this problem might require irrational numbers, it is not a sufficient argument for showing that this problem does not belong to NP. From this perspective, the problem from Theorem 4 is similar to *square-root sum* problem, since the square-root sum problem also requires operations with irrational numbers. However, deciding whether square root sum problem belongs to NP is a major open problem [31] and there are known connections of square-root sum problem to other problems in game theory, e.g., computing Nash equilibrium in 3-player games [32].

Theorem 5. *It is* NP*-hard to check whether there exists a Nash equilibrium in behavioral strategies in two-player A-loss recall games even if player 1 has perfect recall, there are no chance moves, and the game is zero-sum.*

Proof. The proof is made by reduction from the 3-SAT problem. The reduction results in a two-player zero-sum game similar to the one in proof of Theorem 4. The only change in the game is the substitution of the utility in the leaves directly following actions of player 1 by -1 for player 1 and in the leaves corresponding to satisfying the given clause by -0.5 for player 1. The example of the reduction is shown in Fig. 7.

We will show that a NE in behavioral strategies exists if and only if the corresponding 3-SAT problem is satisfiable. First, if the 3-SAT problem is satisfiable, this game has a NE where both players mix uniformly in first two levels of the game, and the player 2 plays according to the assignment of variables satisfying this problem.

Next, we show that if the NE exists, the corresponding 3-SAT problem has to be satisfiable. There are two cases we need to discuss.

Case 1: Player 1 plays a non-uniform strategy b_1. In this case, player 2 will always prefer to play action $a \in \mathcal{A}_2(root)$ which corresponds to the same clause as $a_{min} \in \arg\min_{a \in \mathcal{A}_1} b_1(a)$ deterministically in the root and make the clause corresponding to a_{min} satisfiable. This way player 2 maximizes the probability that the immediate worst possible outcome -1

for player 1 will be reached and minimizes the value player 1 gets when the following state of player 2 is reached. Hence b_1 is not stable against a best responding opponent.

Case 2: The last chance for NE to exist is when player 1 plays a uniform strategy b_1. Here we show that in this case, the NE exists only when the corresponding 3-SAT problem is satisfiable. If the original 3-SAT problem is not satisfiable player 2 will prefer to play a subset of actions $\mathcal{A}'_2 \subset \mathcal{A}_2(root)$ in the root corresponding to a subset of clauses that can be satisfied at the same time, while playing the assignment satisfying these clauses in the rest of the tree. In this case, however, player 1 wants to deviate to playing any distribution over his actions corresponding to the clauses of actions in $\mathcal{A}'_2$. If the 3-SAT is satisfiable on the other hand, the uniform b_1 forms a part of NE when player 2 plays a uniform strategy in the root and according to the assignment satisfying the 3-SAT in the rest of the tree. □

4. The mathematical program for approximating maxmin strategies in imperfect recall games

In this section, we present a mathematical program approximating maxmin strategies for two player games without absentmindedness where the maximizing player has imperfect recall, first when assuming that the minimizing player has A-loss recall, followed by its generalization where there are no restrictions for the minimizing player. Recall that computing exact maxmin strategy in this class of games requires exact representation of irrational numbers (Theorem 2) and so approximating maxmin strategies is the only alternative. The main idea behind this formulation is to add bilinear constraints into the sequence form LP [7] to restrict to imperfect recall strategies of the maximizing player. First, we present the exact bilinear program, followed by explanation of Multiparametric Disaggregation Technique (MDT) [25] which will be used for approximating bilinear terms. Next, we provide the mixed integer linear program (MILP) resulting from the application of the MDT to the bilinear reformulation of the sequence form LP. Finally, we discuss how to use the result of this MILP to construct a strategy with a bounded difference of its expected worst case utility from the maxmin value.

4.1. Exact bilinear sequence form against A-loss recall opponent

$$\max_{x,r,v} v(root, \emptyset) \tag{3a}$$

$$s.t. \quad r(\emptyset) = 1 \tag{3b}$$

$$0 \le r(\sigma_1) \le 1 \qquad \forall \sigma_1 \in \Sigma_1 \tag{3c}$$

$$\sum_{a \in \mathcal{A}(I)} r(\sigma_1 a) = r(\sigma_1) \qquad \forall \sigma_1 \in \Sigma_1, \forall I \in inf_1(\sigma_1) \tag{3d}$$

$$\sum_{a \in \mathcal{A}(I)} x(a) = 1 \qquad \forall I \in \mathcal{I}_1^{IR} \tag{3e}$$

$$0 \le x(a) \le 1 \qquad \forall I \in \mathcal{I}_1^{IR}, \forall a \in \mathcal{A}(I) \tag{3f}$$

$$r(\sigma_1) \cdot x(a) = r(\sigma_1 a) \qquad \forall I \in \mathcal{I}_1^{IR}, \forall a \in \mathcal{A}(I), \forall \sigma_1 \in seq_1(I) \tag{3g}$$

$$\sum_{\sigma_1 \in \Sigma_1} g(\sigma_1, \sigma_2 a) r(\sigma_1) + \sum_{I' \in inf_2(\sigma_2 a)} v(I', \sigma_2 a) \ge v(I, \sigma_2) \qquad \forall I \in \mathcal{I}_2, \forall a \in \mathcal{A}(I), \forall \sigma_2 \in seq_2(I) \tag{3h}$$

The mathematical program (3) is a bilinear reformulation of the sequence-form LP [7] applied to the information set structure of a game G where the player 1 has imperfect recall and the player 2 has A-loss recall. The objective of player 1 is to find a strategy that maximizes the expected utility against the best responding opponent in G. The strategy of the maximizing player is represented as a realization plan (variables r) that assigns the probability to a sequence: $r(\sigma_1)$ is the probability that $\sigma_1 \in \Sigma_1$ will be played assuming that information sets in which actions of the sequence σ_1 are applicable are reached due to player 2. The realization plan r must satisfy the network flow Constraints (3b)–(3d). Finally, a strategy of player 1 is constrained by the best-responding opponent that selects an action minimizing the expected value of player 1 in each $I \in \mathcal{I}_2$ and for each $\sigma_2 \in seq_2(I)$ that was used to reach I (Constraint (3h)). These constraints ensure that the opponent plays the best response in the coarsest perfect recall refinement of G and thus also in G by Lemma 4. The expected utility for each action in Constraint (3h) is a sum of the expected utility values from immediately reachable information sets I' and from immediately reachable leaves. For the latter we use generalized utility function $g : \Sigma_1 \times \Sigma_2 \to \mathbb{R}$ defined as $g(\sigma_1, \sigma_2) = \sum_{z \in \mathcal{Z} | seq_1(z) = \sigma_1 \wedge seq_2(z) = \sigma_2} u_1(z) \mathcal{C}(z)$. In imperfect recall games, multiple σ_i can lead to some imperfect recall information set $I_i \in \mathcal{I}_i^{IR}$; hence, realization plans over these sequences do not have to induce the same behavioral strategy for I_i. Therefore, for each $I_1 \in \mathcal{I}_1^{IR}$ and each $a \in \mathcal{A}(I_1)$ we define behavioral strategy $x(a)$ (Constraints (3e)–(3f)). To ensure that the realization probabilities induce the same behavioral strategy in I_1, we add bilinear constraint $r(\sigma_1 a) = x(a) \cdot r(\sigma_1)$ for every $\sigma_1 \in seq_1(I_1)$ (Constraint (3g)).

Lemma 5. *Let G be a two-player game where the maximizing player has imperfect recall and the minimizing player has A-loss recall. The assignment to r variables satisfies constraints (3b)–(3g) if and only if there exists a behavioral strategy $b_1 \in \mathcal{B}_1$ in G realization equivalent to r.*

Proof. First, we show that for every $b_1 \in \mathcal{B}_1$ there exists an assignment to r variables that satisfies constraints (3b)–(3g). For every sequence $\sigma_1 \in \Sigma_1$ we can compute such $r(\sigma_1)$ as

$$r(\sigma_1) = \begin{cases} 1, \text{if } \sigma_1 = \emptyset \\ \prod_{a \in \sigma_1} b_1(a), \text{otherwise.} \end{cases} \tag{4}$$

The network flow constraints (3b)–(3d) are satisfied from the construction of r in (4). Constraints (3e)–(3g) are satisfied since b_1 is a probability distribution over actions in information sets of G, and $\forall I \in \mathcal{I}_1^{IR}\ \forall a \in \mathcal{A}(I)\ b_1(a) = x(a)$.

Second, we show that for every assignment to r satisfying constraints (3b)–(3g) there exists realization equivalent $b_1 \in \mathcal{B}_1$. Such b_1 can be constructed from r in the following way. In each $I \in \mathcal{I}_1 \setminus \mathcal{I}_1^{IR}$, and for each $a \in \mathcal{A}(I)$, $b(a) = r(seq_1(I)a)/r(seq_1(I))$. In case of $I \in \mathcal{I}_1^{IR}$, however, $seq_1(I)$ is no longer a singleton. But from constraints (3e)–(3g) we know that

$$\forall \sigma_1, \sigma_1' \in seq_1(I) \forall a \in \mathcal{A}(I)\ r(\sigma_1 a)/r(\sigma_1) = r(\sigma_1' a)/r(\sigma_1') = x(a),$$

hence we can use $b_1(a) = x(a)$ for any $I \in \mathcal{I}_1^{IR}$ and $a \in \mathcal{A}(I)$. Finally, from constraints (3b)–(3d) follows that such b_1 is a probability distribution over actions in information sets of G. □

Lemma 6. *Let G be a two-player game where the maximizing player has imperfect recall and the minimizing player has A-loss recall. Assume that we fix r variables to arbitrary values in the mathematical program (3) applied to G, such that r satisfies constraints (3b)–(3g). The optimal objective value of such program corresponds to the worst case expected value of player 1, when playing according to r.*

Proof. We need to show that the objective value of the mathematical program (3) applied to G with the fixed r corresponds to the expected value of the player 1 playing r against the best responding player 2 minimizing the expected value of player 1.

Let G' be the coarsest perfect recall refinement of G for the minimizing player 2. From Lemma 4 we know that when searching for a best response to any b_1 in G, it is sufficient to find a pure best response to b_1 in G'. The sequence-form LP applied to G' ensures that the minimizing player plays a best response using constraints

$$\sum_{\sigma_1 \in \Sigma_1} g(\sigma_1, \sigma_2 a) r(\sigma_1) + \sum_{I' \in inf_2(\sigma_2 a)} v(I') \geq v(I) \quad \forall I \in \mathcal{I}_2', \forall a \in \mathcal{A}(I), \tag{5}$$

where $\sigma_2 = seq_2(I)$. The fact that player 2 plays a best response in the solution of the mathematical program (3) for G is ensured by constraints (3h) which are identical to constraints (5) applied to G'. This holds since all the pairs $\{(I, \sigma_2) |\ I \in \mathcal{I}_2 \wedge \sigma_2 \in seq_2(I)\}$ for player 2 in G exactly correspond to information sets of player 2 in G' (definition of A-loss recall). Hence, variables v and the quantifiers in (3h) for G exactly correspond to variables v and the quantifiers in (5) for G'. Finally, also the left sides of (3h) for G and (5) for G' are equal, since the extended utility function is the same in G and G' and the pairs $(I', \sigma_2 a)$ in the second sum of (3h) correspond to I' in the second sum of (5). Hence, the objective value corresponds to the expected utility of player 1 playing r against the best responding player 2 minimizing the expected value of player 1 in G' and therefore also in G. □

Theorem 6. *Let G be a two-player game where the maximizing player has imperfect recall and the minimizing player has A-loss recall. Realization plan r is a part of some optimal solution[5] of the mathematical program (3) if and only if r is a maxmin strategy for the maximizing player in G.*

Proof. Let b_1 be a behavioral strategy realization equivalent to r. We need to show that r is a part of some optimal solution of the mathematical program (3) if and only if for b_1 holds that

$$b_1 \in \arg\max_{b_1' \in \mathcal{B}_1} \min_{b_2 \in \mathcal{B}_2} u_1(b_1', b_2). \tag{6}$$

First, we prove that if r is a part of an optimal solution of the mathematical program (3) then the eq. (6) holds for behavioral strategy b_1 realization equivalent to r (such strategy always exists from Lemma 5). This follows from the fact

[5] r is only a part of the optimal solution since the r variables form a strict subset of all the variables present in (3).

that the mathematical program maximizes the worst case expected value of player 1 (Lemma 6) over all possible strategies of player 1 (Lemma 5). Hence, b_1 is a maxmin strategy of player 1 in G.

Finally, we prove that if the eq. (6) holds for a behavioral strategy b_1, then the realization plan r, realization equivalent to b_1, is a part of the optimal solution of the mathematical program (3). Let's assume that such r is not a part of any optimal solution of the mathematical program (3). Since r satisfies constraints (3b)–(3g) (Lemma 5), it is a part of a valid solution of (3). This would imply that the mathematical program (3) found r' which guarantees higher worst case expected value (Lemma 6), and hence b_1 would not be a maxmin strategy for player 1. □

4.2. Player 2 without A-loss recall

If player 2 does not have A-loss recall, the mathematical program must use each possible pure best response of player 2 (hence in the worst case each $s_2 \in \mathcal{S}_2$) as a constraint since the time consistent strategy of player 2, ensured by constraint (3h) in the previous case, need not be his ex ante best response (as discussed in Section 2.3). This results in the following bilinear program with size exponential in the size of the solved game.

$$\max_{x,r,v} \; v(root) \tag{7a}$$

$$\text{Constraints (3b)–(3g)}$$

$$\sum_{z \in \mathcal{Z} \mid \pi_2^{s_2}(z)=1} u_1(z)\mathcal{C}(z)r(seq_1(z)) \geq v(root) \qquad \forall s_2 \in \mathcal{S}_2, \tag{7b}$$

where $\pi_2^{s_2}(z)$ is 1 if s_2 prescribes all actions in $seq_2(z)$, 0 otherwise. Since the mathematical program (7) does not change the parts of the program related to the approximation of strategies of player 1, all the following approximation methods, theorems, and the branch-and-bound algorithm can also be applied to (7). However, the scalability would be significantly worse for the mathematical program (7), since even the subproblem of computing the best response of the minimizing player is NP-hard (follows from complexity results in [15]), and hence it requires exponential number of constraints (7b). The algorithm presented in Section 5.2 iteratively solves the mathematical program, hence when using the formulation (7) every iteration of the algorithm would require solving exponentially larger mathematical program compared to the case where the minimizing player has A-loss recall.

4.3. Approximating bilinear terms

The final technical tool that we use to formulate the mathematical program is the approximation of bilinear terms by Multiparametric Disaggregation Technique (MDT) [25]. The main idea of the approximation is to use a digit-wise discretization of one of the variables from a bilinear term. The main advantage of this approximation is a low number of newly introduced integer variables and an experimentally confirmed speed-up over the standard technique of piecewise McCormick envelopes [25].

Let $a = bc$ be a bilinear term. MDT discretizes the variable b and introduces new binary variables $w_{k,\ell}$ that indicate whether the digit k is on ℓ-th position.

$$\sum_{k=0}^{9} w_{k,\ell} = 1 \qquad \ell \in \mathbb{Z} \tag{8a}$$

$$w_{k,\ell} \in \{0,1\} \tag{8b}$$

$$\sum_{\ell \in \mathbb{Z}} \sum_{k=0}^{9} 10^{\ell} \cdot k \cdot w_{k,\ell} = b \tag{8c}$$

$$c^L \cdot w_{k,\ell} \leq \hat{c}_{k,\ell} \leq c^U \cdot w_{k,\ell} \qquad \forall \ell \in \mathbb{Z}, \forall k \in \{0..9\} \tag{8d}$$

$$\sum_{k=0}^{9} \hat{c}_{k,\ell} = c \qquad \forall \ell \in \mathbb{Z} \tag{8e}$$

$$\sum_{\ell \in \mathbb{Z}} \sum_{k=0}^{9} 10^{\ell} \cdot k \cdot \hat{c}_{k,\ell} = a \tag{8f}$$

Constraint (8a) ensures that for each position ℓ there is exactly one digit chosen. All digits used according to $w_{k,l}$ variables must sum to b (Constraint (8c)). Next, we introduce variables $\hat{c}_{k,\ell}$ that are equal to c for such k and ℓ where $w_{k,l} = 1$, and $\hat{c}_{k,\ell} = 0$ otherwise (eqs. (8d), (8e)). c^L and c^U are bounds on the value of variable c. The value of a is given by Constraint (8f).

This is an exact formulation that requires infinite sums and an infinite number of constraints. By restricting the set of all possible positions ℓ to a finite set $\{P_L, \ldots, P_U\}$ we get a lower bound approximation. Following the approach in [25] we can extend the lower bound formulation to compute an upper bound:

Constraints (8a), (8b), (8d), (8e)

$$\sum_{\ell\in\{P_L,\dots,P_U\}}\sum_{k=0}^{9}10^{\ell}\cdot k\cdot w_{k,\ell}+\Delta b=b \tag{9a}$$

$$0\le\Delta b\le 10^{P_L} \tag{9b}$$

$$\sum_{\ell\in\{P_L,\dots,P_U\}}\sum_{k=0}^{9}10^{\ell}\cdot k\cdot \hat{c}_{k,\ell}+\Delta a=a \tag{9c}$$

$$c^L\cdot\Delta b\le\Delta a\le c^U\cdot\Delta b \tag{9d}$$

$$\left(c-c^U\right)\cdot 10^{P_L}+c^U\cdot\Delta b\le\Delta a \tag{9e}$$

$$\left(c-c^L\right)\cdot 10^{P_L}+c^L\cdot\Delta b\ge\Delta a \tag{9f}$$

Here, Δb is assigned to every discretized variable b allowing it to take up the value between the discretization points (Constraints (9a)–(9b)). Similarly, we allow the product variable a to be increased with variable $\Delta a=\Delta b\cdot c$. To approximate the product of the delta variables, we use the McCormick envelope defined by Constraints (9c)–(9f).

4.4. Upper bound MILP approximation

We are now ready to state the main MILP for computing the upper bound on the optimal value of the bilinear program (3) and hence also on the maxmin value of the solved game. The MILP formulation follows the MDT example and uses ideas from Section 4.3 to approximate the bilinear term $r(\sigma_1)x(a)$ in Constraint (3g). In accord with the MDT, we represent every variable $x(a)$ using a finite number of digits of precision. Since $x(a)$ is a probability, we use $dig(\ell)$ as the function which for every precision $\ell\in\{-P..0\}$ returns the set of digits used to represent $x(a)$, i.e.,

$$dig(\ell)=\begin{cases}\{0,1\}, \text{if } \ell=0\\ \{0,\dots,9\} \text{ otherwise.}\end{cases}$$

Binary variables $w^{I_1,a}_{k,\ell}$ correspond to $w_{k,\ell}$ variables from (9) and are used for the digit-wise discretization of $x(a)$. Finally, $\hat{r}(\sigma_1)^a_{k,\ell}$ variables correspond to $\hat{c}_{k,\ell}$ variables from (9). To allow variable $x(a)$ to attain an arbitrary value from $[0,1]$ interval using a finite number of digits of precision, we add an additional real variable $0\le\Delta x(a)\le 10^{-P}$ that can span the gap between two adjacent discretization points. Constraints (10d) and (10e) describe this loosening. Variables $\Delta x(a)$ also have to be propagated to bilinear terms $r(\sigma_1)\cdot x(a)$ involving $x(a)$. We cannot represent the product $\Delta r(\sigma_1 a)=r(\sigma_1)\cdot\Delta x(a)$ exactly and therefore we give bounds based on the McCormick envelope (Constraints (10i)–(10j)).

$$\max_{x,r,v} v(root,\emptyset) \tag{10a}$$

s.t. Constraints (3b)–(3f), (3h)

$$w^{I,a}_{k,\ell}\in\{0,1\} \qquad \forall I\in\mathcal{I}^{IR}_1,\forall a\in\mathcal{A}(I),\ \forall\ell\in\{-P..-1\},\forall k\in dig(\ell) \tag{10b}$$

$$\sum_{k\in dig(\ell)} w^{I,a}_{k,\ell}=1 \qquad \forall I\in\mathcal{I}^{IR}_1,\forall a\in\mathcal{A}(I),\ \forall\ell\in\{-P..0\} \tag{10c}$$

$$\sum_{\ell=-P}^{0}\sum_{k\in dig(\ell)} 10^{\ell}\cdot k\cdot w^{I,a}_{k,\ell}+\Delta x(a)=x(a) \qquad \forall I\in\mathcal{I}^{IR}_1,\forall a\in\mathcal{A}(I) \tag{10d}$$

$$0\le\Delta x(a)\le 10^{-P} \qquad \forall I\in\mathcal{I}^{IR}_1,\forall a\in\mathcal{A}(I) \tag{10e}$$

$$0\le\hat{r}(\sigma)^a_{k,\ell}\le w^{I,a}_{k,\ell} \qquad \forall I\in\mathcal{I}^{IR}_1,\forall a\in\mathcal{A}(I),\ \forall\sigma\in seq_1(I),\forall\ell\in\{-P..0\},\forall k\in dig(\ell) \tag{10f}$$

$$\sum_{k\in dig(\ell)}\hat{r}(\sigma)^a_{k,\ell}=r(\sigma) \qquad \forall I\in\mathcal{I}^{IR}_1,\forall\sigma\in seq_1(I),\ \forall\ell\in\{-P..0\} \tag{10g}$$

$$\sum_{\ell=-P}^{0}\sum_{k\in dig(\ell)} 10^{\ell}\cdot k\cdot\hat{r}(\sigma)^a_{k,\ell}+\Delta r(\sigma a)=r(\sigma a) \qquad \forall I\in\mathcal{I}^{IR}_1,\forall a\in\mathcal{A}(I),\ \forall\sigma\in seq_1(I) \tag{10h}$$

$$(r(\sigma)-1)\cdot 10^{-P}+\Delta x(a)\leq \Delta r(\sigma a)\leq 10^{-P}\cdot r(\sigma) \qquad \forall I\in\mathcal{I}_1^{IR},\forall a\in\mathcal{A}(I),\ \forall\sigma\in seq_1(I) \tag{10i}$$

$$0\leq \Delta r(\sigma a)\leq \Delta x(a) \qquad \forall I\in\mathcal{I}_1^{IR},\forall\sigma\in seq_1(I),\ \forall a\in\mathcal{A}(I) \tag{10j}$$

Note that the MILP has both the number of variables and the number of constraints bounded by $O(|\mathcal{I}|\cdot|\Sigma|\cdot P)$, where $|\Sigma|$ is the number of sequences of both players. The number of binary variables is equal to $10\cdot|\mathcal{I}_1^{IR}|\cdot\mathcal{A}_1^{max}\cdot P$, where $\mathcal{A}_1^{max}=\max_{I\in\mathcal{I}_1^{IR}}|\mathcal{A}_1(I)|$.

4.5. Theoretical analysis of the Upper Bound MILP

Here we show how to use the result of the Upper Bound MILP to construct a strategy with a bounded difference of its expected worst case utility from the maxmin value of the solved game.

The variables $\Delta x(a)$ and $\Delta r(\sigma)$ ensure that the optimal value of the MILP is an upper bound on the value of the bilinear program (3) and therefore also on the maxmin value. The drawback of using $\Delta x(a)$ and $\Delta r(\sigma)$ is that the realization probabilities do not have to induce a valid strategy in the imperfect recall game G, i.e., if σ^1,σ^2 are two sequences leading to an imperfect recall information set $I_1\in\mathcal{I}_1^{IR}$ where action $a\in\mathcal{A}(I_1)$ can be played, $r(\sigma^1 a)/r(\sigma^1)$ need not equal $r(\sigma^2 a)/r(\sigma^2)$. In the following text we will show how to create a valid corrected strategy in G from r which decreases the expected value against a best responding opponent by at most ε compared to the value of the Upper Bound MILP (10), while deriving bound on this ε.

Let $b_1^1(I_1),\dots,b_1^k(I_1)$ be behavioral strategies in the imperfect recall information set $I_1\in\mathcal{I}_1^{IR}$ corresponding to realization probabilities of continuations of sequences $\sigma^1,\dots,\sigma^k\in seq_1(I_1)$ leading to I_1. These behavioral strategies can be obtained from the realization plan as $b_1^j(I_1,a)=r(\sigma^j a)/r(\sigma^j)$ for all $\sigma^j\in seq_1(I_1)$ and $a\in\mathcal{A}(I_1)$. We will omit the information set and use $b_1(a)$ whenever it is clear from the context. Since the imperfect recall is violated in I_1, $b_1^j(a)$ may not be equal to $b_1^l(a)$ for some j, l and action $a\in\mathcal{A}(I_1)$.

Proposition 2. *Using any of the $b_1^1(I_1),\dots,b_1^k(I_1)$ as the corrected strategy $b_1(I_1)$ in every $I_1\in\mathcal{I}_1$ ensures that $\|b_1(I_1)-b_1^j(I_1)\|_1\leq|\mathcal{A}(I_1)|\cdot 10^{-P}$ for every $b_1^j(I_1)\in\{b_1^1(I_1),\dots,b_1^k(I_1)\}$, where P is the number of digits used to approximate the bilinear terms.*[6]

Proof. Let us first show that probabilities of playing action a in $b_1^1,\dots,b_1^k$ can differ by at most 10^{-P}, i.e. $|b_1^j(a)-b_1^l(a)|\leq 10^{-P}$ for every j,l and action $a\in\mathcal{A}(I_1)$. This is based on the MDT we used to discretize the bilinear program.

Let us denote

$$\underline{r}(\sigma_1 a)=\sum_{\ell=-P}^{0}\sum_{k\in dig(\ell)}10^{\ell}\cdot k\cdot\hat{r}(\sigma_1)_{k,\ell}^{a} \tag{11}$$

$$\underline{x}(I_1,a)=\sum_{\ell=-P}^{0}\sum_{k\in dig(\ell)}10^{\ell}\cdot k\cdot w_{k,\ell}^{I_1,a}, \tag{12}$$

as the part of the strategy representation without the Δr and Δx variables. Notice that constraints (10f) and (10g) ensure that $\underline{r}(\sigma_1 a)=r(\sigma_1)\cdot\underline{x}(I_1,a)$. Hence, the difference in $b^1(I_1),\dots,b^k(I_1)$ is caused solely by the $\Delta r(\sigma_1 a)$ variables. Furthermore, we know that $\Delta r(\sigma_1 a)\leq 10^{-P}\cdot r(\sigma_1)$ (Constraint (10i)) which ensures that the maximum difference in $b_1^1(a),\dots,b_1^k(a)$ for any a is at most 10^{-P}. Taking any of the behavioral strategies $b_1^1,\dots,b_1^k$ as the corrected behavioral strategy $b_1(I_1)$, therefore satisfies

$$\|b_1(I_1)-b_1^j(I_1)\|_1\leq\sum_{a\in\mathcal{A}(I_1)}10^{-P}=|\mathcal{A}(I_1)|\cdot 10^{-P}. \qquad \square$$

We now connect the distance of the corrected strategy $b_1(I_1)$ from the set of behavioral strategies $b_1^1(I_1),\dots,b_1^k(I_1)$ in $I_1\in\mathcal{I}_1^{IR}$ to the maximum possible distance in worst case expected values. First, we show this on the level of a single history. Finally, we extend this result to the distance of the worst case expected value of the corrected strategy b_1 from the maxmin value.

[6] The L1 norm is taken as $\|x_1-x_2\|_1=\sum_{a\in\mathcal{A}(I_1)}|x_1(a)-x_2(a)|$.

Lemma 7. *Let $h \in I_1$ be a history and b_1^1, b_1^2 be behavioral strategies prescribing different behavior in I_1 but prescribing the same behavior in all subsequent states $h \sqsubset h'$. Let $v_{max}(h)$ and $v_{min}(h)$ be maximal and minimal utility of player 1 in the subtree of h. Then the following holds:*

$$\max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(b_1^1, b_2^1) - u_1^h(b_1^2, b_2^2)| \leq \frac{v_{max}(h) - v_{min}(h)}{2} \cdot \|b_1^1(I_1) - b_1^2(I_1)\|_1,$$

where $u_1^h(b_1, b_2)$ is the expected utility of player 1, when starting in h and playing according to b_1, b_2.

The proof can be found in Appendix D.

Now we are ready to bound the distance of the worst case expected value of the corrected strategy b_1 from the maxmin value.

Theorem 7. *The distance of the worst case expected value of the corrected strategy b_1 from the maxmin value is bounded by*

$$\varepsilon = 10^{-P} \cdot d \cdot \mathcal{A}_1^{max} \cdot \frac{v_{max}(\emptyset) - v_{min}(\emptyset)}{2},$$

where d is the maximum number of player 1's imperfect recall information sets on any path from the root to a terminal node, $\mathcal{A}_1^{max} = \max_{I_1 \in \mathcal{I}_1^{IR}} |\mathcal{A}(I_1)|$ is the maximal branching factor and $v_{min}(\emptyset)$, $v_{max}(\emptyset)$ are the lowest and highest utilities for player 1 in the whole game, respectively.

Proof. We show an inductive way to compute the upper bound on the distance of the worst case expected value of the corrected strategy b_1 from the maxmin value. Throughout the derivation we assume that all players play to maximize the bound to guarantee that we obtain a valid upper bound. We proceed in a bottom-up fashion over the nodes in the game tree, computing the bound $L(h)$ on the maximum loss player 1 could have accumulated by correcting his behavioral strategy in the subtree of h. The ε is obtained as the value of this bound in the root of the game. The bound $L(h)$ in every $h \in \mathcal{H}$ is guaranteed to be higher or equal to

$$\max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(y_1, b_2^1) - u_1^h(b_1, b_2^2)|, \tag{13}$$

where y_1 is created by joining all $b_1^1(I_1), \ldots, b_1^k(I_1)$ from the solution of the Upper Bound MILP for all $I_1 \in \mathcal{I}_1$ (y_1 prescribes behavior only on a level of sequences since $b_1^1(I_1), \ldots, b_1^k(I_1)$ can specify different behavior for every sequence leading to I_1, by $y(\sigma_1, a)$ we denote the probability that a will be played after sequence σ_1), b_1 is the strategy created by correcting y_1 in the whole tree.

The description of the computation of $L(h)$ follows in a case to case manner.

(1) In leaves, $L(h) = 0$ as there is no correction made.

(2) In node h where player 2 or nature acts,

$$L(h) = \max_{a \in \mathcal{A}(h)} L(h \cdot a),$$

since there can be no loss accumulated and in the worst case the direct successor with the highest loss is chosen.

(3) In player 1's node h, which is not a part of an imperfect recall information set, no corrective steps need to be taken. The expected bound at node h is therefore $\sum_{a \in \mathcal{A}(h)} y_1(seq_1(h), a) L(h \cdot a)$. In the worst case player 1's behavioral strategy $y_1(seq_1(h))$ selects deterministically the direct successor with the highest bound, therefore again we use the bound $L(h) = \max_{a \in \mathcal{A}(h)} L(h \cdot a)$.

(4) In player 1's node h, which is a part of an imperfect recall information set, the correction step may have to be taken. Let y_1^{-h} be the strategy created from y_1 by taking corrective steps in all successors of h and let us construct a strategy y_1^h from y_1^{-h} by correcting it also in h. We know that the loss caused by changing y_1 to y_1^{-h} is at most $\max_{a \in \mathcal{A}(h)} L(h \cdot a)$, hence

$$\max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(y_1, b_2^1) - u_1^h(y_1^{-h}, b_2^2)| \leq \max_{a \in \mathcal{A}(h)} L(h \cdot a).$$

Now we have to take the corrective step in the node h and construct strategy y_1^h. When using the corrected strategy from Proposition 2, we get the following bound (Lemma 7):

$$\begin{aligned} \max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(y_1^{-h}, b_2^1) - u_1^h(y_1^h, b_2^2)| &\leq \frac{v_{diff}(h)}{2} \cdot 10^{-P} |\mathcal{A}_1(I_1)| \\ &\leq \frac{v_{diff}(\emptyset)}{2} \cdot 10^{-P} \mathcal{A}_1^{max}. \end{aligned}$$

It follows that

$$\begin{aligned}
&\max_{b_2^1,b_2^2\in\mathcal{B}_2} |u_1^h(y_1,b_2^1)-u_1^h(y_1^h,b_2^2)| \\
&\qquad\le \max_{b_2^1,b_2^2\in\mathcal{B}_2} |u_1^h(y_1^{-h},b_2^1)-u_1^h(y_1^h,b_2^2)| + \max_{b_2^1,b_2^2\in\mathcal{B}_2} |u_1^h(y_1,b_2^1)-u_1^h(y_1^{-h},b_2^2)| \\
&\qquad\le \frac{v_{diff}(\emptyset)}{2}\cdot 10^{-P}\mathcal{A}_1^{max} + \max_{a\in\mathcal{A}(h)} L(h\cdot a),
\end{aligned}$$

hence we use

$$L(h) = \frac{v_{diff}(\emptyset)}{2}\cdot 10^{-P}\mathcal{A}_1^{max} + \max_{a\in\mathcal{A}(h)} L(h\cdot a).$$

Finally, we provide a bound on the loss in the root node

$$L(\emptyset) \ge \max_{b_2^1,b_2^2\in\mathcal{B}_2} |u_1(y_1,b_2^1)-u_1(b_1,b_2^2)|. \tag{14}$$

We have shown that in order to prove the worst case bound it suffices to consider deterministic choice of action at every node h – this means that a single path in the game tree is pursued during the propagation of the bound. The bound is increased exclusively in nodes which are a part of some imperfect recall information set. We can encounter at most d such nodes on any path from the root. The increase of the bound in each such node is bounded by

$$\frac{v_{max}(\emptyset)-v_{min}(\emptyset)}{2}\cdot 10^{-P}\mathcal{A}_1^{max},$$

therefore the bound in the root is

$$\varepsilon = L(\emptyset) = \frac{v_{max}(\emptyset)-v_{min}(\emptyset)}{2}\cdot d\cdot 10^{-P}\mathcal{A}_1^{max}$$

From eq. (14) follows that ε is guaranteed to be higher or equal to the actual difference of worst case expected values of y_1 and b_1, since it forms an upper bound even in the case where we maximize the difference of the expected values over all pairs of player 2's strategies, while in case of the worst case expected value player 2 is restricted to playing a best response. It follows that the worst case expected value of the strategy we have found lies within the interval $[v^*-\varepsilon, v^*]$, where v^* is the worst case expected value of y_1, and therefore the optimal value of the Upper Bound MILP. As v^* is an upper bound on the solution of the original bilinear program and therefore also on the maxmin value, no strategy can have a better worst case expected value than v^*. Hence the strategy b_1 guarantees the ε distance from the maxmin value. □

5. Algorithms for approximating maxmin strategies in imperfect recall games

Algorithms for solving perfect recall games are either not applicable to imperfect recall games and A-loss recall games or they do not provide any guarantees on the quality of the obtained solutions (see Appendix F for more details). Hence, new algorithms for solving this class of games are required. In this section we describe a family of algorithms that use the Upper Bound MILP formulation (10) introduced in the previous section to approximate the maxmin strategy in two-player games where the maximizing player has imperfect recall and the minimizing player has A-loss recall. First, we describe a simple approach (denoted as Base). Base starts with some initial precision of the representation of bilinear terms and iteratively increases the precision until the distance of the corrected strategy obtained from the solution of the Upper Bound MILP with the current precision from the maxmin value is below a given threshold. Next, to reduce the number of binary variables and hence to improve the scalability of Base we present a branch-and-bound based algorithm (denoted as IRABnB). IRABnB works on a linear relaxation of the Upper Bound MILP and simultaneously searches the possible precision improvements of bilinear terms and the assignment to the relaxed binary variables until the error in the worst case expected value is below a given threshold. Finally, to reduce the size of the mathematical program that needs to be solved, we extend IRABnB with incremental strategy generation technique (the algorithm is denoted as DOIRABnB).

Notice, that the restriction to A-loss recall minimizing player leads to a following properties in all the algorithms.

Proposition 3. *Let G be a two player game where the maximizing player has imperfect recall and the minimizing player has A-loss recall. Let G' be the coarsest perfect recall refinement of G for the minimizing player with no modifications to the information set structure of the maximizing player. Computing the maxmin strategy in G reduces to computing the maxmin strategy in G'.*

Proof. Follows directly from Lemma 4. □

Corollary 1. *Let G be a two-player game where the maximizing player has imperfect recall and the minimizing player has A-loss recall. Let's assume that G is created as an imperfect recall abstraction of some perfect recall game G', such that G' is the coarsest perfect recall refinement of G for the minimizing player. When computing maxmin strategy in G we effectively compute the least exploitable strategy in any game with more refined information set structure of the maximizing player (hence also in G') that can be represented in G.*

Consequently, the maxmin value computed in G gives us the exploitability of the resulting strategy directly in G', hence the value can be used to evaluate the quality of the abstraction (the further the maxmin value of G is from the maxmin value of G', the more exploitable strategies resulting from solving G are).

5.1. Iterative precision refining MILP

Here we describe the Base algorithm, using the Upper Bound MILP formulation (10) to approximate the maxmin strategies in two-player games where the maximizing player has imperfect recall and the minimizing player has A-loss recall.

The distance of the worst case expected value of the corrected strategy b_1 from the maxmin value is a function of P, which is the precision of all approximations of bilinear terms. We design the following algorithm (denoted as Base): (1) start with the precision set to 0 for all bilinear terms, (2) for each approximation of a bilinear term calculate the current error contribution (the difference between $\Delta r(\sigma_1 a)$ and $r(\sigma_1)\Delta x(a)$ multiplied by the expected utility). Choose from the terms which do not yet have maximal allowed precision the term that contributes to the overall error the most and increase the precision of its representation by 1. The algorithm terminates when none of the terms which do not yet have maximal allowed precision contributes to the error.

5.2. Branch-and-bound algorithm

We now introduce a branch-and-bound search (denoted as IRABnB, Imperfect Recall Abstraction Branch-and-Bound algorithm) for approximating maxmin strategies of two-player games where the maximizing player has imperfect recall and the minimizing player has A-loss recall. We follow the standard practice in solving MILPs and apply the branch-and-bound search to the linear relaxation of the Upper Bound MILP. Recall, that we linearize the $w_{k,\ell}^{I_1,a}$ variables that control digit-wise discretization of $x(a)$. Furthermore, we exploit the following observation in the IRABnB.

Observation 1. Even if the current assignment to variables $w_{k,\ell}^{I_1,a}$ is not feasible (they are not set to binary values), we can correct the resulting strategy as described in Section 5.2.1 and use it to estimate the lower bound on the maxmin value of player 1 without a complete assignment of all $w_{k,\ell}^{I_1,a}$ variables to either 0 or 1. The lower bound is computed as the expected value of the corrected strategy against a best response to it.

The IRABnB algorithm starts with the linear relaxation of the Upper Bound MILP with bilinear terms approximated using 0 digits of precision. It builds and searches a branch and bound tree. In every node n of the branch and bound tree, the algorithm solves the LP corresponding to n, heuristically selects the information set I and action a contributing to the current approximation error the most, and creates successors of n by restricting the probability $b_1(I, a)$ that a is played in I. The successors are created by adding new constraints to the LP corresponding to n depending on the value of $b_1(I, a)$ by constraining (and/or introducing new) relaxed binary variables $w_{k,l}^{I_1,a}$. This way, the algorithm simultaneously searches for the optimal approximation of bilinear terms as well as the assignment to binary variables. The algorithm terminates when ε-optimal maxmin strategy is found (using the difference of the global upper bound computed by solving the LP relaxation and the lower bound computed as described in Observation 1).

Algorithm 1 depicts the complete IRABnB algorithm. The algorithm creates and traverses nodes of the branch and bound tree. Every node n has associated LP with the strategy of player 1 restricted to a certain degree of precision. Additionally, n keeps the lower bound on the overall maxmin value of player 1 and the upper bound on the values of the LPs achievable in the subtree of n. The algorithm starts in the root of the branch and bound tree, where the maxmin strategy is approximated using 0 digits of precision after the decimal point (i.e., precision $P(I_1, a) = 0$ for every variable $x(a)$). The algorithm maintains a set of active branch-and-bound nodes (fringe) and a node opt with the highest guaranteed expected value of player 1 against the best responding opponent that corresponds to the global lower bound on the worst-case guaranteed expected value. In each iteration, the algorithm selects the node with the highest upper bound from fringe (lines 4–5). If there is no potential for improvement in the unexplored parts of the branch and bound tree (i.e., all the nodes in the fringe have upper bound lower than the lower bound in opt), the current best solution is returned (line 7) (upper bounds of the nodes added to the fringe in the future will never be higher than the current upper bound). Next, the algorithm checks whether the current solution has better lower bound than opt, if yes, the opt is replaced by the current node (line 9). Since the algorithm always selects the most promising node with respect to the upper bound, we are sure that if the lower bound and upper bound have distance at most ε, the algorithm found an ε-optimal solution and it can terminate (line 11) (upper bounds of the nodes added to the fringe in the future will never be higher than the current upper bound). Otherwise, the algorithm heuristically selects an action having the highest effect on the gap between the upper and lower bound in the selected

Algorithm 1: IRABNB algorithm.

input : Initial LP relaxation LP_0 of Upper Bound MILP using a $P = 0$ discretization
output : ε-optimal strategy for a player having imperfect recall
parameters: Bound on maximum error ε, precision bounds for $x(a)$ variables $P_{max}(I_1, a)$

1 fringe $\leftarrow$ {`CreateNode(`LP_0`)`}
2 opt $\leftarrow$ (nil, $-\infty$, ∞)
3 **while** fringe $\neq \varnothing$ **do**
4 (LP, lb, ub) $\leftarrow \arg\max_{n \in \text{fringe}} n.\text{ub}$
5 fringe $\leftarrow$ fringe $\setminus$ (LP, lb, ub)
6 **if** opt.lb $\geq$ ub **then**
7 **return** `ReconstructStrategy(`opt`)`
8 **if** opt.lb $<$ lb **then**
9 opt $\leftarrow$ (LP, lb, ub)
10 **if** ub $-$ lb $\leq \varepsilon$ **then**
11 **return** `ReconstructStrategy(`opt`)`
12 **else**
13 $(I_1, a) \leftarrow$ `SelectAction(`LP`)`
14 $P \leftarrow$ number of digits of precision representing $x(a)$ in LP
15 fringe $\leftarrow$ fringe $\cup$ {`CreateNode(`LP $\cup$ $\{\sum_{k=0}^{\lfloor \frac{a_{ub}+a_{lb}}{2} \rfloor_P} w_{k,P}^{I_1,a} = 1\}$`)`}
16 fringe $\leftarrow$ fringe $\cup$ {`CreateNode(`LP $\cup$ $\{\sum_{k=\lfloor \frac{a_{ub}+a_{lb}}{2} \rfloor_P}^{9} w_{k,P}^{I_1,a} = 1\}$`)`}
17 **if** $P < P_{max}(I_1, a)$ **then**
18 fringe $\leftarrow$ fringe $\cup$ {`CreateNode(`LP $\cup$ $\{w_{LP.x(a)_{-P},P}^{I_1,a} = 1$, *introduce vars* $w_{0,P+1}^{I_1,a}, \ldots, w_{9,P+1}^{I_1,a}$ *and corresponding constraints from MDT* $\}$`)`}
19 **return** `ReconstructStrategy(`opt`)`
20 **function** `CreateNode(`LP`)`
21 ub $\leftarrow$ `Solve(`LP`)`
22 $b_1 \leftarrow$ `ReconstructStrategy(`LP`)`
23 lb $\leftarrow u_1(b_1, \texttt{BestResponse}(b_1))$
24 **return** (LP, lb, ub)

node n (line 13, as described in Section 5.2.1). Next, it retrieves the precision used to represent behavioral probability of this action. By default, two successors of the current branch-and-bound node n are added, each with one of the following constraints. $x(a) \leq \lfloor \frac{a_{ub}+a_{lb}}{2} \rfloor_P$ (line 15) and $x(a) \geq \lfloor \frac{a_{ub}+a_{lb}}{2} \rfloor_P$ (line 16), where $\lfloor \cdot \rfloor_p$ is flooring of a number towards p digits of precision and a_{ub} and a_{lb} are the lowest and highest allowed probabilities of playing $x(a)$. This step performs binary halving restricting allowed values of $x(a)$ in the current precision. Additionally, if the current precision is lower than the maximal allowed precision $P_{max}(I_1, a)$ the gap between bounds may be caused by the lack of discretization points; hence, the algorithm adds one more successor with constraint $\lfloor v \rfloor_P \leq x(a) \leq \lceil v \rceil_P$, where v is the current probability of playing a, while increasing the precision used for representing $x(a)$ (line 18) (all the restriction to $x(a)$ in all 3 cases are done via $w_{k,l}^{I_1,a}$ variables).

The function `CreateNode` computes the upper bound on the values achievable in the subtree of the current node by solving the given LP (line 21) and the lower bound on the overall maxmin value of player 1 as described in Observation 1, by using the heuristic construction of a valid strategy b_1 from the solution of the given LP (line 22, as described in Section 5.2.1) and computing the expected value of b_1 against a best response to it.

5.2.1. LP for strategy reconstruction and action selection

We provide a linear program that is used as a heuristic to compute a corrected behavioral strategy in a given $I_1 \in \mathcal{I}_1$ and to estimate the contribution of the actions to the overall approximation error. It takes into account the realization probabilities $r(\sigma_1^j)$ of sequences $\sigma_1^j \in seq_1(I_1)$ leading to I_1 as well as errors that can be accumulated in the subtrees of individual histories $h \in I_1$. Let us denote by $\{1, \ldots, k\}$ the set of indices of all sequences in $seq_1(I_1)$. By b_1^j, for each $j \in \{1, \ldots, k\}$ we denote the behavioral strategy corresponding to the realization probability of sequence σ_1^j and its continuations. b_1 is the final corrected behavioral strategy.

$$\min_{b,L,\alpha} \sum_{\sigma_1^j \in seq_1(I_1)} r(\sigma_1^j) \cdot L(\sigma_1^j) \tag{15a}$$

$$\text{s.t.} \quad L(\sigma_1^j, a) \geq [b_1^j(a) - b_1(a)] \cdot v_{max}(\sigma_1^j \cdot a) \qquad \forall j \in \{1, \ldots, k\}, \forall a \in \mathcal{A}(I_1) \tag{15b}$$

$$L(\sigma_1^j, a) \geq [b_1(a) - b_1^j(a)] \cdot (-v_{min}(\sigma_1^j \cdot a)) \qquad \forall j \in \{1, \ldots, k\}, \forall a \in \mathcal{A}(I_1) \tag{15c}$$

$$L(\sigma_1^j) = \sum_{a \in \mathcal{A}(I_1)} L(\sigma_1^j, a) \qquad \forall \sigma_1^j \in seq_1(I_1) \tag{15d}$$

$$b_1(a) = \sum_{j \in \{1,\dots,k\}} \alpha(\sigma_1^j) \cdot b_1^j(a) \qquad \forall a \in \mathcal{A}(I_1) \tag{15e}$$

$$0 \leq \alpha(\sigma_1^j) \leq 1 \qquad \forall \sigma_1^j \in seq_1(I_1) \tag{15f}$$

$$\sum_{\sigma_1^j \in seq_1(I_1)} \alpha(\sigma_1^j) = 1 \tag{15g}$$

The LP finds a strategy minimizing the estimated error in the following way. Constraints (15b), (15c) compute the maximum cost $L(\sigma_1^j, a)$ of changing the probability that action a is played after σ_1^j, assuming that the worst possible outcome in the subtree following playing $\sigma_1^j a$ is reached. Constraint (15d) computes the estimated errors $L(\sigma_1^j)$ for every σ_1^j by summing all the $L(\sigma_1^j, a)$ for all relevant a. The sum of $L(\sigma_1^j)$ weighted by the realization probability of corresponding sequences is minimized in the objective. Constraints (15e) to (15g) make sure that the result will be a convex combination of all the b_1^j strategies, with the α variables being the coefficients of the convex combination.

The bound from Theorem 7 on the error of a strategy constructed in this way holds, since we have shown that the L1 distance of any pair of behavioral strategies b_1^i, b_1^j obtained from realization plans in I_1 is at most $10^{-P}|\mathcal{A}_1(I_1)|$ — the distance to their convex combination b_1 cannot be larger. Hence, the algorithm uses this LP to construct a valid strategy b_1 in every imperfect recall information set where the results prescribe inconsistent behavior.

Finally, we use

$$\sum_{\sigma_1^k \in seq_1(I_1)} L(\sigma_1^k, a)$$

as the heuristic estimate of the contribution of action a to the overall approximation error. The function SelectAction returns the action with the highest such estimate over all $I_1 \in \mathcal{I}_1^{IR}$.

5.2.2. Theoretical properties of the IRABNB algorithm

The IRABNB algorithm takes the error bound ε as the input. First, we provide a method for setting the $P_{max}(I_1, a)$ parameters to guarantee that IRABNB returns ε-maxmin strategy for player 1. Finally, we provide a bound on the number of iterations the algorithm needs to terminate. Notice that the NP-hardness result from Theorem 11 applies to both settings where IRABNB is applicable.

Theorem 8. *Let $P_{max}(I_1)$ be the maximum number of digits of precision used for representing variables $x(a), \forall a \in \mathcal{A}(I_1)$ set as*

$$P_{max}(I_1) = \left\lceil \max_{h \in I_1} \log_{10} \frac{|\mathcal{A}(I_1)| \cdot d \cdot v_{max}(h) - v_{min}(h)}{2\varepsilon} \right\rceil,$$

where $v_{min}(\emptyset)$, $v_{max}(\emptyset)$ are the lowest and highest utilities for player 1 in the whole game, respectively. With this setting IRABNB *is guaranteed to return an ε-optimal maxmin strategy for player 1.*

Proof. Let us first show that the limits on the number of refinements $P_{max}(I_1)$ are sufficient to allow representation of ε-maxmin strategy of player 1. The proof is conducted in the same case by case manner as the proof of Theorem 7. Here we focus only on the case 4 from the proof of Theorem 7, which handles histories from some imperfect recall information set. In the rest of the cases, we again assume that players play such that the action leading to the child with maximal bound on loss is chosen.

Let $I_1 \in \mathcal{I}_1^{IR}$ and $h \in I_1$. We know that when using $P_{max}(I_1)$ of digits to represent the strategy in I_1, the L1 distance between behavioral strategies in I_1 is at most $10^{-P_{max}(I_1)} \cdot |\mathcal{A}(I_1)|$ (Proposition 2). This means that the bound in h from case 4 in the proof of Theorem 7 is modified to:

$$\begin{aligned}
&\max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(y_1, b_2^1) - u_1^h(y_1^h, b_2^2)| \\
&\quad \leq \max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(y_1^{-h}, b_2^1) - u_1^h(y_1^h, b_2^2)| + \max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(y_1, b_2^1) - u_1^h(y_1^{-h}, b_2^2)| \\
&\quad \leq \frac{v_{diff}(h)}{2} \cdot 10^{-P_{max}(I_1)} \cdot |\mathcal{A}(I_1)| + \max_{a \in \mathcal{A}(h)} L(h \cdot a) \\
&\quad \leq \frac{v_{diff}(h)}{2} \cdot \frac{|\mathcal{A}(I_1)| \cdot 2\varepsilon}{|\mathcal{A}(I_1)| \cdot d \cdot v_{diff}(h)} + \max_{a \in \mathcal{A}(h)} L(h \cdot a) \\
&\quad = \frac{\varepsilon}{d} + \max_{a \in \mathcal{A}(h)} L(h \cdot a).
\end{aligned}$$

Similarly to the proof of Theorem 7, it suffices to assume that players choose actions deterministically in every node to obtain the upper bound on the error. The path induced by these choices contains at most d imperfect recall nodes, thus $L(\emptyset) = d \cdot \varepsilon / d = \varepsilon$.

Finally, we show that IRABnB is guaranteed to reach the precision guarantees which result in ε-optimal maxmin strategy. This holds since (1) the upper and lower bound on the best worst case value of player 1's strategy with a given precision restrictions are correct (follows directly from their computation), hence the branch-and-bound search never prunes away the branch with the optimal solution, (2) the IRABnB always retrieves the node with the highest upper bound from the fringe and (3) the algorithm terminates only when an ε-maxmin strategy for player 1 is found. □

Theorem 9. *When using $P_{max}(I_1)$ from Theorem 8 for all $I_1 \in \mathcal{I}_1^{IR}$ and all $a \in \mathcal{A}(I_1)$, the number of iterations of the* IRABnB *algorithm needed to find an ε-optimal solution is in*

$$\mathcal{O}\left(\left(\frac{3^{4\log_{10}(S_1)+4}}{\varepsilon^2}\right)^{S_1}\right)$$

where $S_1 = |\mathcal{I}_1^{IR}|\mathcal{A}_1^{max}$.

The proof is based on bounding the number of nodes in the branch and bound tree as a function of $P_{max}(I_1)$ and computing the final bound by substituting the $P_{max}(I_1)$ from Theorem 8 (see Appendix E for detailed proof).

The main disadvantage of IRABnB is that the size of the LP solved in every iteration is linear in the size of the game and the algorithm can refine the precision of bilinear term approximation in parts of the game that may not be relevant for the final solution. To reduce the size of the solved LP and to focus the refinements of the precision of bilinear term approximation to relevant parts of the game, an incremental strategy-generation technique modified for imperfect recall EFGs can be employed.

5.3. Double oracle for perfect recall EFGs

The double oracle algorithm for solving perfect recall EFGs (DOEFG, [27]) is an adaptation of column/constraint generation techniques for EFGs. The main idea of DOEFG is to create a restricted game where only a subset of actions is allowed to be played by players and then incrementally expand this restricted game by allowing new actions. The restricted game is solved as a standard zero-sum extensive-form game using the sequence-form linear program [24,7]. Afterward, best response algorithms search the original unrestricted game to find new sequences to add to the restricted game for each player. The algorithm terminates when the best response calculated on the unrestricted game provides no improvement to the solution of the restricted game for either of the players.

DOEFG uses two main ideas: (1) the algorithm assumes that players play some pure default strategy outside the restricted game (e.g., playing the first action in each information set given some ordering), (2) temporary utility values are assigned to leaves in the restricted game that correspond to inner nodes in the original unrestricted game (so-called temporary leaves), which form an upper bound on the expected utility.

5.4. Double oracle IRABnB for imperfect recall EFGs

In this section, we introduce the DOIRABnB (Double Oracle Imperfect Recall Abstraction Branch-and-Bound) algorithm combining ideas of IRABnB and DOEFG. Adapting the ideas of DOEFG for games with imperfect recall poses several challenges that we need to address. To solve the restricted game means to compute the maxmin strategy for player 1. However, solving the restricted game requires calling IRABnB search that iteratively refines the approximation of bilinear terms instead of solving a single (or a pair of) LPs in DOEFG for perfect recall games. DOIRABnB thus makes an integration of two iterative methods and decides when to expand the restricted game and when to refine the approximation of bilinear terms already in the restricted game.

We first provide the pseudocode of the algorithm with its description, followed by formal definitions of all the necessary components of the algorithm.

In Algorithm 2 we present the DOIRABnB algorithm. Similarly to IRABnB, the algorithm performs a branch and bound search. Every branch and bound node n stores the LP with corresponding precision adjustments to the bilinear term approximation, lower bound on the maxmin value of player 1 and an upper bound on the value achievable in the whole game in the subtree of n. The list of currently active nodes is stored in the fringe. The node with the highest lower bound encountered is stored in opt. There are two differences from IRABnB: (1) through the run, DOIRABnB incrementally builds the restricted game $\bar{G}$, and when solving the LP for any branch and bound node, the LP is always built to solve the current $\bar{G}$. (2) DOIRABnB uses function `Add` to add any node to the fringe. The function `Add` (lines 20 to 29) repeatedly uses the maximizing player oracle (line 24, Section 5.4.2) to make sure that before adding the node to the fringe we first update the restricted game so that solving the LP for $\bar{G}$ and current precision restriction gives an upper bound on the value of the

LP applied to the original game with the same precision restrictions (see Section 5.4.3 for more details). This is required to guarantee the convergence of the algorithm to ε-maxmin strategy for player 1.

Note that DOIRABnB does not simply use the double oracle approach to solve LP in every single node to optimality, instead it applies the oracles of the maximizing and minimizing player separately to avoid increasing the size of the restricted game unnecessarily, while making sure that the algorithm works with valid upper bound on the value in the original game.

Algorithm 2: DOIRABnB algorithm.

input : Initial LP relaxation LP_0 of Upper Bound MILP, Initial restricted game $\bar{\mathsf{G}}$
output : ε-optimal strategy for the maximizing player
parameters: Bound on maximum error ε, bound P_{max} for bilinear term precision approximation

```
1  fringe ← {(LP_0, −∞, ∞)}
2  opt ← (LP_0, −∞, ∞)
3  while fringe ≠ ∅ do
4      (LP, lb, ub) ← arg max_{n∈fringe} n.ub
5      fringe ← fringe \ (LP, lb, ub)
6      if opt.lb ≥ ub then
7          return ReconstructStrategy(opt)
8      if opt.lb < lb then
9          opt ← (LP, lb, ub)
10     if ub − lb ≤ ε then
11         return ReconstructStrategy(opt)
12     if FromSmallerG(Ḡ, LP) then
13         Ḡ ←Add(Ḡ, LP)
14     else if ExpandableByMinPlayerOracle(Ḡ, LP) then
15         (Ḡ, LP) ← ExpandByMinPlayerOracle(Ḡ, LP)
16         Ḡ ←Add(Ḡ, LP)
17     else
18         (I_1, a) ← SelectAction(LP)
19         AddSuccessors(LP, I_1, a, P_max, Ḡ)
20 function Add(Ḡ, LP)
21     LP_0 ← LP, t ← 1
22     (lb, ub, B̂_2^1) ← Resolve(Ḡ, LP_0)
23     while t = 1 || LP_{t−2} ≠ LP_{t−1} do
24         (Ḡ, LP_t) ← ExpandByMaxPlayerOracle(Ḡ, LP_{t−1}, B̂_2^t)
25         (Ḡ, LP_t) ← UpdateUtilities(Ḡ, LP_t, B̂_2^t)
26         (lb, ub, B_2^{LP_t}) ← Resolve(Ḡ, LP_t)
27         B̂_2^{t+1} ← B̂_2^t ∪ B_2^{LP_t}, t ← t + 1
28     fringe ← fringe ∪ (LP_{t−1}, lb, ub)
29     return Ḡ
```

The algorithm starts with an empty restricted game $\bar{G}$. Lines 1 to 11 are the same as in the IRABnB algorithm. Additionally, in every iteration, DOIRABnB checks whether the bounds in the current node were computed in some smaller restricted game than the current $\bar{G}$ (line 12). If yes, DOIRABnB recomputes the bounds on the current restricted game, returns the node to the fringe (line 13) and continues with the next iteration. This is done to make sure that DOIRABnB does not make unnecessary precision adjustments due to imprecise bounds. Else, if bounds come from the same game as the current restricted game $\bar{G}$, the algorithm checks whether $\bar{G}$ can be expanded by the minimizing player oracle (line 14, see Section 5.4.2). If $\bar{G}$ can be expanded, we expand it, resolve with the current precision restrictions and return the node to the fringe (lines 15, 16). Note that we do not use the maximizing player oracle at this point, because the expansion by maximizing player oracle is used when adding the node to the fringe in function `Add` (as described in Section 5.4.3). Otherwise, if $\bar{G}$ cannot be expanded, the algorithm continues in the same way as IRABnB. It heuristically selects bilinear terms corresponding to action a from the current restricted game $\bar{G}$ (line 18, as described in Section 5.2.1). The algorithm then creates new nodes and adds new variables and constraints into the LPs in these nodes that further restrict possible values of $x(a)$. Next, if the maximal allowed precision permits, DOIRABnB creates an additional node with increased precision of representation of $x(a)$. Finally, it adds the new nodes to the fringe (line 19) using the `Add` function.

5.4.1. The restricted game

This section formally defines the restricted game $\bar{G} = (\mathcal{N}, \bar{\mathcal{H}}, \bar{\mathcal{Z}}, \bar{\mathcal{A}}, \bar{u}, C, \bar{\mathcal{I}})$ of the original unrestricted game $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, u, C, \mathcal{I})$. The restricted game $\bar{G}$ is built to guarantee that solving the LP with a given precision restrictions for $\bar{G}$ gives an upper bound on the solution of the LP for the original game with the same precision restrictions, when the

player 2 plays some strategy from a set $\hat{\mathcal{B}}_2$. By $\hat{\mathcal{B}}_2$ we denote a subset of all strategies of player 2 from the restricted game $\bar{G}$ extended by the default strategy (playing first action available in each decision point). In Section 5.4.2 we present the oracles used to construct the restricted game to guarantee such bound. In Section 5.4.3 we explain how to iteratively build $\hat{\mathcal{B}}_2$ which combined with the use of the oracles guarantees that the solution of the LP gives an upper bound, this time when the player 2 can play any strategy from the original game G.

The restricted game is limited by a set of allowed sequences $\bar{\Phi} \subseteq \Sigma$, that are returned by the oracles. An allowed sequence $\sigma_i \in \bar{\Phi}$ might not be playable to the full length due to missing compatible sequences of the opponent. Therefore, the restricted game is defined using the maximal compatible set of sequences $\bar{\Sigma} \subseteq \bar{\Phi}$. Formally

$$\bar{\Sigma}_i = \{\sigma_i \in \bar{\Phi}_i | \exists \sigma_{-i} \in \bar{\Phi}_{-i}\ \exists h \in \mathcal{H} : seq_i(h) = \sigma_i \wedge seq_{-i}(h) = \sigma_{-i}\}, \forall i \in \mathcal{N}. \tag{16}$$

The sets $\bar{\mathcal{H}}$, $\bar{\mathcal{A}}$ are the subsets of $\mathcal{H}$, $\mathcal{A}$ reachable when playing sequences from $\bar{\Sigma}$. $\bar{\mathcal{I}}$ defines the same partition as $\mathcal{I}$ on the reduced set $\bar{\mathcal{H}}$, i.e., for all $h, h' \in \bar{\mathcal{H}}$, holds that $h, h' \in I$ for some $I \in \bar{\mathcal{I}}$ in the restricted game if and only if $h, h' \in I$ for some $I \in \mathcal{I}$ in the original game. The set of leaves in $\bar{G}$ is a union of leaf nodes of G present in $\bar{G}$ and inner nodes from G that do not have a valid continuation in $\bar{\Sigma}$

$$\bar{\mathcal{Z}} = (\mathcal{Z} \cap \bar{\mathcal{H}}) \cup \{h \in \bar{\mathcal{H}} \setminus \mathcal{Z} | \bar{\mathcal{A}}(h) = \emptyset\}. \tag{17}$$

We refer to the members of the set $\bar{\mathcal{Z}} \setminus \mathcal{Z}$ as *temporary leaves*. Note that if not stated otherwise, when we operate with a strategy from the restricted game in the whole unrestricted game, we automatically assume that it is extended by a default strategy playing the first action available as in DOEFG.

We define the *temporary utility value* in every $z \in \bar{\mathcal{Z}}$ as $\bar{u}_1(z, \hat{\mathcal{B}}_2)$ so that $\bar{u}_1(z, \hat{\mathcal{B}}_2)$ is an upper bound on the value the player 1 can guarantee in the original game G in z, when the minimizing player plays any strategy from the set $\hat{\mathcal{B}}_2$. Formally, we use

$$\bar{u}_1(z, \hat{\mathcal{B}}_2) = \max_{b_2 \in \hat{\mathcal{B}}_2} \hat{u}_1^z(b_2), \forall z \in \bar{\mathcal{Z}}, \tag{18}$$

where $\hat{u}_1^z(b_2)$ stands for the expected value of player 1 in the original game G when starting in z and playing a strategy from the coarsest perfect recall refinement of G maximizing the expected value in z against b_2 (remember that b_2 is extended by the default strategy). We define $\hat{u}_1^z(b_2)$ in such way since the best response of player 1 against b_2 in the coarsest perfect recall refinement is easy to compute as shown in Lemma 4 (player 1 has imperfect recall in G hence computing the best response there is NP-hard). Furthermore, since we give more information to player 1 in the coarsest perfect recall refinement, $\hat{u}_1^z(b_2)$ is guaranteed to be an upper bound on the maximal expected value in z achievable by player 1 against $b_2 \in \hat{\mathcal{B}}_2$ in G. The set $\hat{\mathcal{B}}_2$ is built in function `Add` using best responses of player 2 taken from the solution of the LP by finding actions corresponding to active Constraint (3h) (see Section 5.4.2 for details). Notice that the $\bar{u}_1$ might differ in every iteration of the algorithm, since $\hat{\mathcal{B}}_2$ can change.

5.4.2. Updating the restricted game

In this section we discuss the oracles used in DOIRABnB and the way their results are used to expand the restricted game (lines 15 and 24 in Algorithm 2). Note that the oracle of the maximizing player is given $\hat{\mathcal{B}}_2$ (see Section 5.4.3 for details on construction of $\hat{\mathcal{B}}_2$) and expands the restricted game with respect to the strategies in $\hat{\mathcal{B}}_2$.

Minimizing player oracle. The minimizing player plays a best response in the final maxmin solution of the game, hence, similarly to DOEFG we use the best response computation as the oracle of player 2. In every iteration we compute $b_2^{BR} \in BR_2(b_1)$ in the original game G, where b_1 is the strategy of player 1 computed by DOIRABnB in the current node and current restricted game extended by the default strategy. The algorithm extends $\bar{\Phi}_2$ by all the valid continuations of $\sigma_2 \in \bar{\Phi}_2$ by actions in b_2^{BR} and update $\bar{\Sigma}$ accordingly.

Maximizing player oracle. The best response is not a sufficient oracle of the maximizing player 1 since his maxmin strategy does not have to consist of best responses to pure strategies of the minimizing player.

Example 2. Consider the game in Fig. 8. The maxmin strategy for player 1 is playing b and e deterministically, guaranteeing the maxmin value -1. Notice, however, that playing b and e is not a best response to any pure strategy of player 2. Since during the run the DOIRABnB only computes pure best responses of player 2, the best response oracle for the maximizing player 1 would never add states h_2 and h_3 and so the DOIRABnB would never find the correct solution.

To fix this, the algorithm keeps track of possible extensions of the restricted game by taking actions in states of the maximizing player 1. To do that, the algorithm uses a set of pending states

$$\mathcal{H}_p = \{h \in \mathcal{H} \setminus \bar{\mathcal{H}} | \exists h' \in \bar{\mathcal{H}}_1 \exists a \in A(h') : h'a = h\}, \tag{19}$$

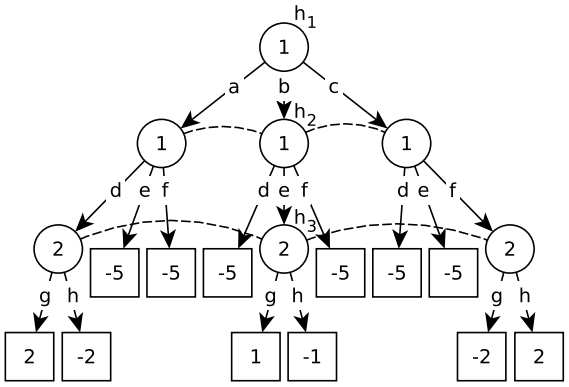


Fig. 8. An A-loss recall game where the maxmin strategy for player 1 is not a best response to any of the pure best responses of player 2.

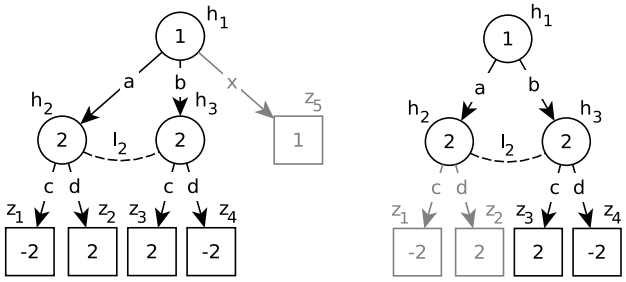


Fig. 9. Games for demonstration of the necessity of the `Add` function.

which contains all the states h not in the restricted game, whose parent h' is in the restricted game and player 1 makes decision in h'. Since we build $\bar{G}$ to find strategy of player 1, which is optimal against the strategies from $\hat{\mathcal{B}}_2$ there is no point in adding $h \in \mathcal{H}_p$ which are not reachable by any $b_2 \in \hat{\mathcal{B}}_2$. Hence, we take a subset $\mathcal{H}'_p \subseteq \mathcal{H}_p$ such that all $h \in \mathcal{H}'_p$ are reachable by some $b_2 \in \hat{\mathcal{B}}_2$. Furthermore, we can exclude pending states which cannot improve the expected value of the player 1 against any $b_2 \in \hat{\mathcal{B}}_2$. Formally, by $\mathcal{H}^*_p$ we denote a subset of $\mathcal{H}'_p$, where for all $h \in \mathcal{H}^*_p$ holds that

$$\bar{u}_1(h, \hat{\mathcal{B}}_2) \geq \min_{b_2 \in \hat{\mathcal{B}}_2} u_1^{h'}(b_1, b_2),$$

where h' is the parent of h, b_1 is the strategy of player 1 from the current LP extended by the default strategy and $u_1^h(b_1, b_2)$ stands for the expected value in state h when players play according to b_1, b_2. When expanding the restricted game, we add to $\bar{\Phi}$ all the sequences leading to all $h \in \mathcal{H}^*_p$.

Example 2 *(continued).* When using the pending states as an oracle of player 1 in the game in Fig. 8, we always add the states h_2 and h_3. For example when h_2 is not a part of the restricted game, player 1 cannot play action b. h_2 is then added by the maximizing player oracle since all the strategies of player 1 not playing action b can guarantee the expected value of at most -2 in the h_1 against the worst case opponent. On the other hand, $\bar{u}_1(h_2, \hat{\mathcal{B}}_2) \geq -1, \forall \hat{\mathcal{B}}_2$ and so h_2 is added to the restricted game.

Finally, let us explain the functions used in Algorithm 2. `ExpandableByMinPlayerOracle` checks whether the oracle of the minimizing player suggests any sequence to be added to the restricted game. `ExpandByMinPlayerOracle` and `ExpandableByMaxPlayerOracle` add to the restricted game all the sequences suggested by the minimizing player oracle and the maximizing player oracle respectively.

5.4.3. *Adding nodes to the* fringe

There are two requirements $\bar{G}$ needs to fulfill before adding any given node to the fringe in function `Add` (lines 20 to 29) in order to guarantee that solving the LP for $\bar{G}$ and given precision restrictions gives an upper bound on the value of the LP for the original game with the same precision restrictions. (1) The function `Add` needs to make sure that the restricted game is built so that player 1 has no deviation outside of the restricted game, which could increase his expected value. Let us demonstrate this in the following example.

Example 3. Consider the game in Fig. 9 (left). Assume that the restricted game $\bar{G}$ consists of states $\{h_1, h_2, h_3, z_1, z_2, z_3, z_4\}$ and there are no precision restrictions. If we solve the restricted game we obtain the value 0 for player 1, however, maxmin strategy for player 1 is to play x guaranteeing the expected value 1.

(2) The utility $\bar{u}_1$ in all $z \in \bar{\mathcal{Z}}$, which is going to be used to construct the LP must be an upper bound on the worst case utility of player 1 against all the possible best responses $b_2 \in \mathcal{B}_2^{LP}$ corresponding to active Constraints (3h) after solving this LP. Hence the utility must be set according to the possible behavior of the minimizing player, which depends on the utility in question. This is required since strategies $b_2 \in \mathcal{B}_2^{LP}$ can define behavior also outside of the restricted game due to information sets of player 2 that can be only partially present in the restricted game. It is, therefore, insufficient to assume that player 2 plays using only default strategy in every information set outside of the restricted game. Since the algorithm sets the utility in the $\bar{\mathcal{Z}} \setminus \mathcal{Z}$ to a fixed value (therefore not reflecting the changing behavior of player 2 in the LP), it needs to make sure that the value is an upper bound against all possible strategies player 1 can face to obtain the required upper bound by solving the LP. Let us demonstrate this in the following example.

Example 4. Consider the game in Fig. 9 (right). Assume that the restricted game $\bar{G}$ consists of states $\{h_1, h_2, h_3, z_3, z_4\}$ and there are no precision restrictions. h_2 is a temporary leaf in $\bar{G}$, hence we need to compute a temporary utility value for it. Let us first discuss what would happen if we do not consider the behavior in the restricted game and use the value from the leaf reachable after the default strategy (playing the first action in every state). The default strategy leads to the terminal state z_1 with utility -2. Solving the restricted game using -2 as the temporary utility value for h_2 leads to strategy with the worst case expected value -2. However, the maxmin value of player 1 in the original game is 0 achievable by playing uniformly in h_1.

In the function `Add` we iteratively update the restricted game until we are guaranteed to obtain a correct upper bound. To do that, the function `Add` builds in every iteration T a set $\hat{\mathcal{B}}_2^T$ as a union of all the $\mathcal{B}_2^{LP_t}$ obtained by solving the LP_t in every iteration $t \in \{0, \dots, T-1\}$ (lines 26, 27) in the current invocation of the function `Add`. In every iteration T the function `Add` expands $\bar{G}$ using the oracle of the maximizing player for the current set $\hat{\mathcal{B}}_2^T$ (line 24) and updates the utilities in $\bar{G}$ again using the current set $\hat{\mathcal{B}}_2^T$ (line 25). The algorithm iterates in function `Add` until the LPs from last two iterations are equal, and only then is the given node added to the fringe.

Example 3 *(continued).* Consider again the game in Fig. 9 (left). The function `Add` will ensure that z_5 is added to $\bar{G}$ since $z_5 \in \mathcal{H}_p^*$ and so it will be added by the maximizing player oracle.

Example 4 *(continued).* Consider again the game in Fig. 9 (right). We did not receive a correct upper bound by solving the restricted game, since setting the temporary utility value to -2 in h_2 is incorrect. Player 2 plays action d in the solution of the LP_0 to force the resulting value to be -2 but the -2 in h_2 was obtained assuming that player 2 will play c as a part of his default strategy (playing c and d is mutually exclusive). To solve this, the function `Add` performs another iteration, where it sets the utility in h_2 to $\bar{u}_1(h_2, \hat{\mathcal{B}}_2^1)$, where $\hat{\mathcal{B}}_2^1 = \mathcal{B}_2^{LP_0}$ is the singleton containing the strategy playing d obtained as the best response from the solution of the LP_0. In this iteration the algorithm correctly sets $\bar{u}_1(h_2, \hat{\mathcal{B}}_2) = 2$. After solving $\bar{G}$ we get value 2, which is the desired upper bound on the maxmin value in the original game with no precision restrictions.

5.4.4. Theoretical properties

Here we demonstrate that if IRABnB is guaranteed to find ε-optimal maxmin strategy for some precision parameters P_{max}, DOIRABnB is also guaranteed to find ε-optimal maxmin strategy for the same P_{max}.

Lemma 8. *Every node $n \in$ fringe in every iteration of* DOIRABnB *has a valid lower bound on the maxmin value of player 1 in the original game G.*

Proof. The lower bound is valid, since it is computed as $u_1(b_1, b_2)$, where b_1 is the current solution of the LP corresponding to n applied to the current restricted game $\bar{G}$, extended by the default strategy and $b_2 \in BR_2(b_1)$ in the original game G. Since the maxmin strategy b_1^* of player 1 maximizes its expected value assuming the worst case opponent, $u_1(b_1, b_2)$ must be lower or equal to the maxmin value of the game. □

Lemma 9. *The upper bound in every node $n \in$ fringe for the corresponding precision restrictions in every iteration of* DOIRABnB *forms an upper bound on the value of the LP with the same precision restrictions applied to the original game G, hence the upper bound in the node n is higher or equal to the upper bound in the node used in* IRABnB *applied to G with the same precision restrictions.*

Proof. We add all nodes to the fringe using function `Add`. Since we iterate in function `Add` until the LP_T in the current iteration T is equal to the LP_{T-1}, we are sure that there is no deviation of the player 1 outside of $\bar{G}$ which can increase his worst case expected value against any $b_2 \in \mathcal{B}_2^{LP_{T-1}}$, since adding any pending state would cause LP_T to be different from LP_{T-1}. Additionally, since the LP_T and LP_{T-1} are equal, we are sure that none of the $b_2 \in \mathcal{B}_2^{LP_{T-1}}$ change the utility structure created using $\hat{\mathcal{B}}_2^{T-1}$, hence

$$\bar{u}_1(z, \hat{\mathcal{B}}_2^{T-1}) \geq \bar{u}_1(z, \mathcal{B}_2^{LP_{T-1}}), \forall z \in \bar{\mathcal{Z}}. \tag{20}$$

Player 1 cannot increase his expected value by playing outside of the restricted game against any $b_2 \in \mathcal{B}_2^{LP_{T-1}}$. Furthermore, every $z \in \bar{\mathcal{Z}}$ has assigned an upper bound on the expected value player 1 can guarantee in z against any $b_2 \in \mathcal{B}_2^{LP_{T-1}}$ in the original game. Hence, the value of the LP_{T-1} is an upper bound on the value of the LP with the same precision restrictions in G. Finally, since LP_T is equal to LP_{T-1} the same holds for the value of LP_T. □

Theorem 10. *If the* IRABnB *algorithm is guaranteed to return ε-optimal maxmin strategy for precision parameters P_{max}, the* DOIRABnB *returns ε-optimal maxmin strategy for the same precision parameters P_{max}.*

Proof. When DOIRABnB reaches a node where the upper and lower bound are at most ε distant, we are sure that we have found an ε-optimal solution of the original game. This holds since (1) the node has correct bounds on the value achievable in the original game with given precision restrictions (Lemmas 8, 9), (2) the DOIRABnB always retrieves the node with the highest upper bound from the fringe (line 4). Additionally, DOIRABnB is guaranteed to reach this node, since it never prunes away the branch containing the optimal solution in the space of precision restrictions (again from the correctness of the upper and lower bound from Lemmas 8, 9).

When DOIRABnB reaches the node with the precision restrictions guaranteeing an ε-optimal solution in the original game, the bounds might be more than ε distant due to the insufficiently built restricted game. This is caused by the temporary leaves $z \in \bar{\mathcal{Z}} \setminus \mathcal{Z}$. The DOIRABnB assigns to every $z \in \bar{\mathcal{Z}} \setminus \mathcal{Z}$ a temporary utility which is an upper bound on the actual utility which player 1 can guarantee against worst-case opponent in z in the original game. Hence the strategy b_1 computed for the current restricted game can prefer some temporary leaf $z \in \bar{\mathcal{Z}} \setminus \mathcal{Z}$ based on this possibly overestimated utility value. However, when computing the $b_2^{BR} \in BR(b_1)$ where b_1 is extended by the default strategy in the original game, the value $u_1^z(b_1, b_2^{BR})$ can be significantly smaller than the temporary utility value in z. The oracle of the minimizing player, however, expands $\bar{G}$ by actions in b_2^{BR}. Since b_2^{BR} exploits the difference in the temporary utility in z and the actual expected value in z obtained when playing according to b_1, b_2^{BR}, z has to be reached when playing according to these strategies. The restricted game is, therefore, expanded by the action $a \in \mathcal{A}(z)$ played in b_2^{BR} and z is no longer a temporary leaf after the expansion. The expansion of the temporary leaves continues until there is no temporary leaf where player 2 can exploit the overestimated value of the temporary utility (lines 14, 24 in Algorithm 2). Hence the reason for the difference in the bounds directly implies that the expansion of the restricted game on line 14 will occur. The DOIRABnB terminates when the restricted game is built sufficiently to allow the distance of bounds to decrease to at most ε. This must happen after a finite number of steps since in the worst case the algorithm builds the entire original game.

Finally, the fact that there is ε-optimal solution when using given precision parameters P_{max} is guaranteed by the assumption that IRABnB is guaranteed to find the ε-optimal solution for the same parameters in the original game. □

6. Experiments

In this section, we provide an experimental evaluation of the performance of DOIRABnB, IRABnB and the Base. Furthermore, we demonstrate the possible space savings in the size of the strategy representation when employing imperfect recall abstractions and discuss the quality of strategies resulting from solving these abstractions. Since there is no standardized collection of benchmark EFGs, the experiments are conducted on a set of Random games, an imperfect recall search game and an imperfect recall variant of OshiZumo. All algorithms were implemented in Java, each algorithm uses a single thread, 8 GB memory limit and we use IBM ILOG CPLEX 12.6.2 to solve all LPs/MILPs.

Random games. We use randomly generated games to obtain statistically significant results. We randomly generate a perfect recall game with varying branching factor and fixed depth of 6. To control the information set structure, we use observations assigned to every action – for player i, nodes h with the same observations generated by all actions in history belong to the same information set. To obtain imperfect recall games with a non-trivial information set structure, we run a random abstraction algorithm which merges information sets for all $i \in \mathcal{N}$ according to parameter p in the following way. Let $\{\mathcal{I}_i^1, \ldots, \mathcal{I}_i^n\}$ be the largest possible disjoint subsets of $\mathcal{I}_i$ of the perfect recall game such that

$$\forall \mathcal{I}_i^k \in \{\mathcal{I}_i^1, \ldots, \mathcal{I}_i^n\} \forall I_i, I_i' \in \mathcal{I}_i^k | seq_i(I_i)| = |seq_i(I_i')| \wedge |A(I_i)| = |A(I_i')|,$$

and $\bigcup_{\mathcal{I}_i^k \in \{\mathcal{I}_i^1, \ldots, \mathcal{I}_i^n\}} \mathcal{I}_i^k = \mathcal{I}_i$. Each $\mathcal{I}_i^k \in \{\mathcal{I}_i^1, \ldots, \mathcal{I}_i^n\}$ contains candidates for merging. Let $\mathcal{J} \subseteq \mathcal{I}_i^k$ be a set that initially contains a random element $I \in \mathcal{I}_i^k$. We iterate over all $I' \in \mathcal{I}_i^k \setminus I$, and add I' to $\mathcal{J}$ with probability p. To create the abstraction we iteratively choose the subset $\mathcal{J}$ of $\mathcal{I}_i^k$, create abstracted set containing all elements of $\mathcal{J}$, and remove $\mathcal{J}$ from $\mathcal{I}_i^k$. This procedure repeats until $\mathcal{I}_i^k$ is empty and is performed for all $\mathcal{I}_i^k \in \{\mathcal{I}_i^1, \ldots, \mathcal{I}_i^n\}$. We further update the abstraction by splitting the information sets of the minimizing player to make sure that he has A-loss recall. We generate a set of experimental instances by varying the branching factor and the parameter p. Our random games are rather difficult to solve since (1) information sets can span multiple levels of the game tree (i.e., the nodes in an information set often have histories with differing sizes) and (2) actions can easily lead to leaves with very different utility values.

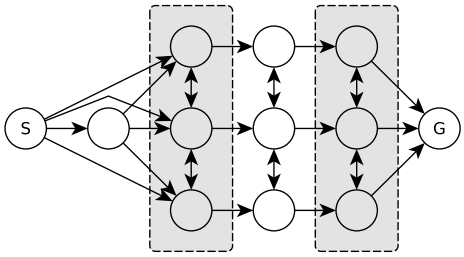


Fig. 10. Graph for the Search game. The attacker starts in the node S and tries to reach the node G. The defender operates two units, each moving in one of the shaded areas.

OshiZumo. We use a modification of OshiZumo described, e.g., in [33]. The game is played by two players; both start with a given number of coins. At the beginning of a game, a sumo wrestler is positioned at the center of a one-dimensional playing field which consists of 7 positions. In every turn of the game, each player uses some amount of his coins to place a bid. The highest bidder pushes the wrestler one location towards the opponent's side. If the bids are equal, the wrestler does not move. Either way, both players lose all the coins they used to make the bid and the game proceeds until the money runs out or the wrestler is pushed off the field. The players observe only their bid and whether they won or not. The bid of the opponent is never revealed. It is a zero-sum game, where the final position of the wrestler determines the winner: if he is located at the center, the game result is a draw. Otherwise, the player in whose half the wrestler is located gets a negative utility equal to the number of positions between the wrestler and the center of the playing field. In this paper, we create different instances of OshiZumo by changing the number of coins available to players.

To create the imperfect recall abstraction of the OshiZumo, we give the maximizing player only the information about the number of coins he has left, and whether he has won in each of the previous rounds, hence he does not remember the exact bids he had made. The minimizing player remembers both his bids and whether he won in each round. Notice that we do not modify the information set structure of the minimizing player, and so the original game is the coarsest perfect recall refinement of this abstraction for the minimizing player.

Search game. Our third domain is an instance of search (or pursuit-evasion) game, used, e.g., in [27]. Search games are commonly used for evaluating incremental algorithms [34]. The game is played on a directed graph shown in Fig. 10 between attacker and defender. The attacker tries to cross from the starting node S to his destination G. The attacker can either move every turn, leaving tracks in each node he visits, or he can move every other turn without leaving any tracks. The defender operates two units, each moving in one of the shaded areas, trying to intercept the attacker by capturing him in a node. The defender observes only the tracks left by the attacker and only in case one of his units steps on the node with the track. The attacker does not have any information about the defender units. The players move simultaneously. It is a zero-sum game, where the attacker obtains utility 1 for reaching his destination and defender obtains utility 1 for intercepting the attacker. If a given number of moves is depleted without either of the events happening, the game is considered a draw and both obtain utility 0. We assume the defender to be the maximizing player.

To create an imperfect recall abstraction of the Search game, we give the defender only information about the tracks he currently observes and the position of both of his units without remembering the history of moves leading there. The attacker knows only the sequence of his actions in the past. Notice that we do not modify the information set structure of the minimizing player, and hence the original game is the coarsest perfect recall refinement of this abstraction for the minimizing player.

6.1. Results

The main experiments are divided into 2 parts. (1) We compare the Base, IRABnB and DOIRABnB algorithms to show how the different components used in the algorithms influence the scalability. The results show that DOIRABnB outperforms IRABnB and Base on smaller domains while providing significantly better scalability than the rest thanks to the fact that the incremental strategy generation keeps the LP being solved small while focusing the precision adjustments to relevant parts of the game tree. (2) We demonstrate the immense space savings in the strategy representation achievable by employing the simple imperfect recall abstractions described above. Additionally, we show that solving these abstractions results in finding the maxmin strategy of the original unabstracted game.

The ε in all the experiments was set to $10^{-4} \cdot u_{max}$, where u_{max} is the maximal utility of the solved game.

6.1.1. Runtime comparison

Random games. In Fig. 11 we present the runtime results in seconds obtained on random games. Every plot depicts the cumulative relative number of instances (y-axis) solved under a given time limit (logarithmic x-axis). There were 100 instances of random games solved for every setting. The rows contain results for random games with varying p, the first column for branching factor 3, second for branching factor 4. The runtime of the algorithms was limited to 2 hours on every instance,

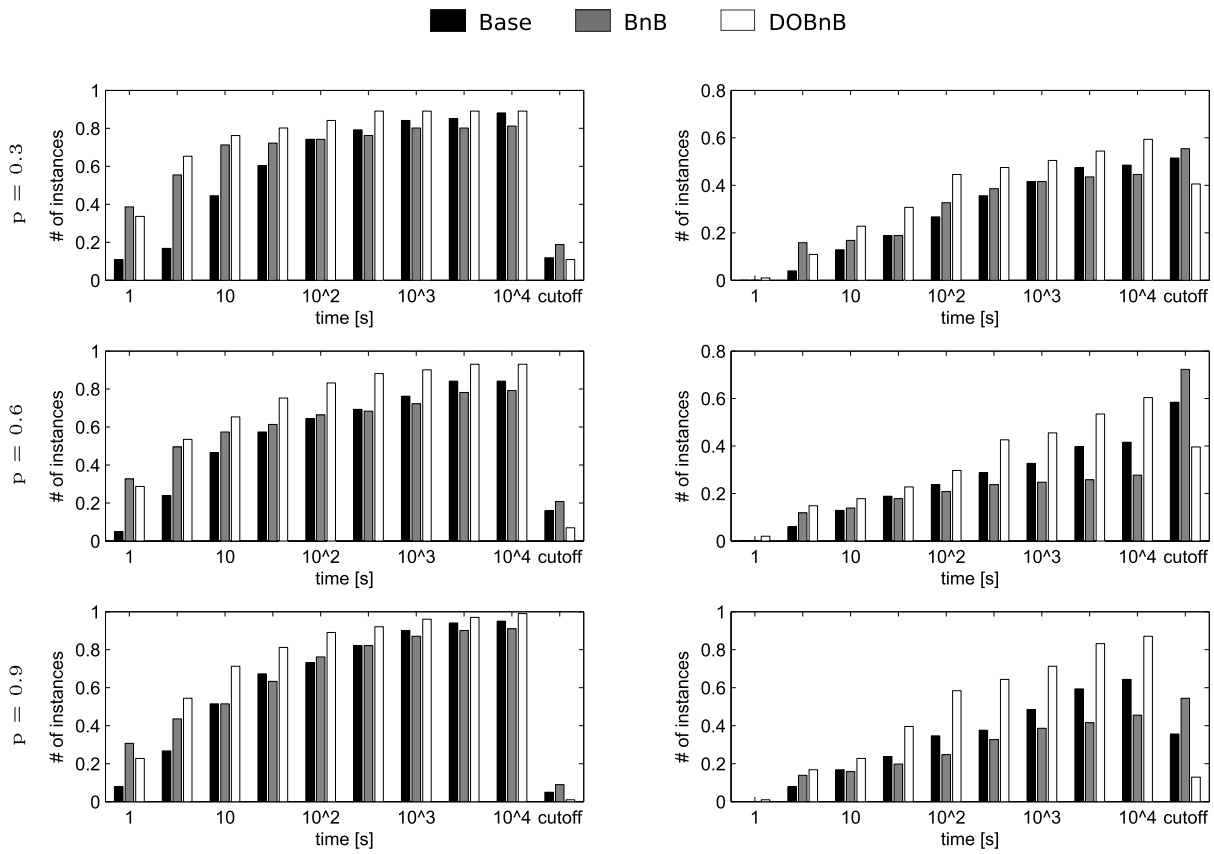


Fig. 11. Results for random games showing the relative cumulative number of instances (y-axis) solved under a given time limit (x-axis) and the relative amount of instances terminated due to the exceeded runtime in bars labeled *cutoff*. Rows contain results for $p = 0.3$, $p = 0.6$, $p = 0.9$, columns show results for branching factor 3 and 4.

Table 1
Average relative amount of sequences for maximizing and minimizing player respectively, added to the restricted game by DOIRABnB in random games.

p \ b.	3	4
0.3	$46.1\% \pm 2.9\%$, $22.2\% \pm 1.8\%$	$62.5\% \pm 3.1\%$, $17.5\% \pm 2.2\%$
0.6	$58.9\% \pm 2.8\%$, $23.3\% \pm 2.0\%$	$71.7\% \pm 2.8\%$, $17.1\% \pm 2.1\%$
0.9	$68.2\% \pm 2.5\%$, $24.0\% \pm 1.8\%$	$76.6\% \pm 2.9\%$, $18.5\% \pm 1.1\%$

the relative number of instances terminated after this limit is reported in the bars labeled *cutoff*. The IRABnB usually dominates the other two algorithms in the number of instances solved in the first time interval. As the runtime grows, however, the performance of IRABnB decreases. This is because IRABnB tends to spend a lot of time making adjustments in the irrelevant parts of the game tree. On the other hand, the DOIRABnB outperforms the other two algorithms across all the settings, and we can see a significant decrease in the number of instances not solved in a given 2-hour limit, compared to Base and IRABnB. This is because the DOIRABnB focuses adjustments to approximation precision to the relevant parts of the game tree present in the restricted game while keeping the underlying LP smaller. Note that the random games form an unfavorable scenario for all the presented algorithms since the construction of the abstraction is completely random, which makes conflicting behavior in merged information sets common. As we can see, however, even in these scenarios the DOIRABnB is capable of solving the majority of instances with branching factor 4 which have $\sim$ 3000 nodes in under 2 hours.

In Table 1 we present the average relative amount of sequences for each player needed by DOIRABnB to solve the random games for each setting along with the standard error. The relative amount of sequences needed by the minimizing player is consistently smaller because the restricted game is built to compute maximizing player's robust strategy, while the minimizing player only plays best responses during the computation. Even though the size of the restricted game remains similar across all values of p, we observe an increase in the relative size, since the number of sequences decreases as p increases.

OshiZumo. The DOIRABnB solved the game with 11 coins in 44 minutes using 0.9% sequences for the maximizing player and 0.2% sequences for the minimizing player. The game has $3.5 \cdot 10^6$ states, $2.8 \cdot 10^5$ and $1.4 \cdot 10^6$ sequences

Table 2
The relative amount of information sets of the maximizing player in the imperfect recall abstractions with different number of remembered moves w.r.t. the number of information sets of the maximizing player in the original game for the OshiZumo with 10 and 11 coins (left) and the Search game with depth 6 and 7 (right).

c. \ r. m.	0	1	2
10	3.27%	5.54%	9.30%
11	1.90%	3.27%	5.64%

d. \ r. m.	0	1	2
6	0.10%	0.46%	1.81%
7	0.03%	0.11%	0.42%

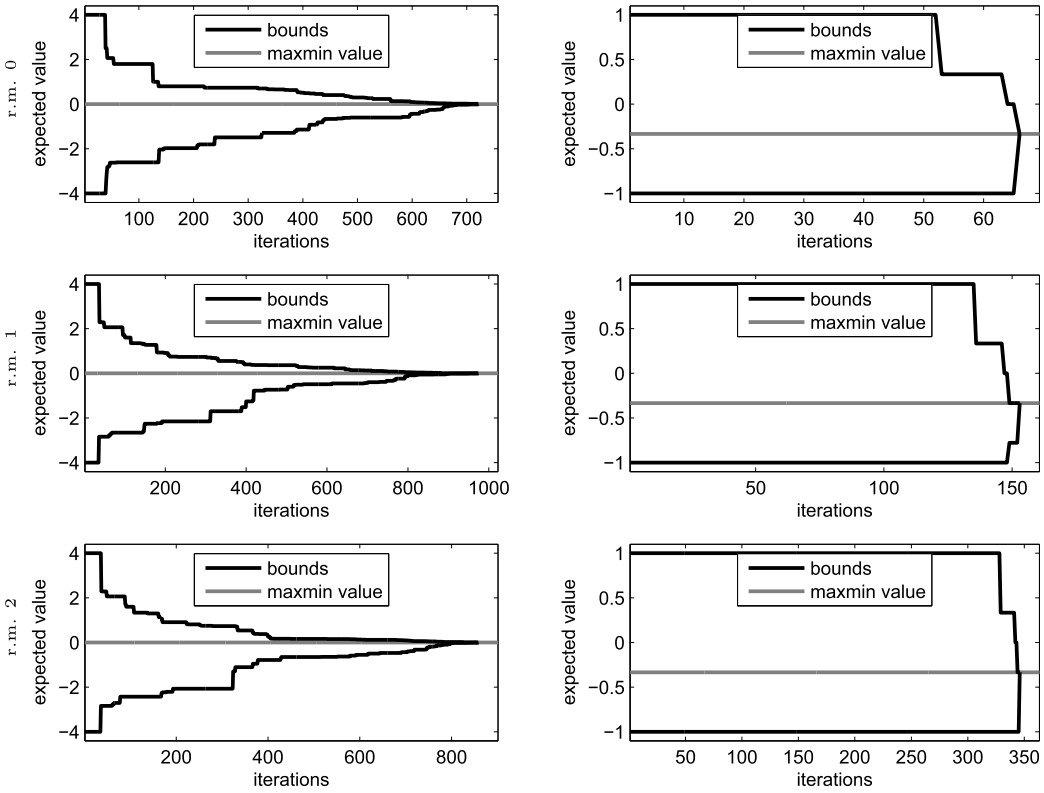


Fig. 12. The first column shows the convergence of the lower and upper bound in DOIRABnB on OshiZumo with 11 coins as a function of iterations, each row for a different number of remembered moves. Second column shows the convergence of the lower and upper bound in DOIRABnB on Search game with depth 7 as a function of iterations, each row for a different number of remembered moves.

for the maximizing and minimizing player respectively. IRABnB and Base were able to solve the game with 9 coins in 20 seconds and 2 hours respectively, however, on the game with 10 coins none of the two algorithms finished in 10 hours.

Search game. In case of Search game, the DOIRABnB was able to solve a game with 10 moves allowed for each player (with $\sim 5 \cdot 10^9$ states, $\sim 2 \cdot 10^4$ sequences for the attacker and $\sim 4 \cdot 10^7$ sequences for the defender) using 0.002% of sequences for the defender and 0.2% sequences for the attacker in 1.2 hours. IRABnB and Base were able to solve the game with 5 moves for every player in 12 seconds and 40 minutes respectively, however, on the game with 6 moves none of the two algorithms finished in 10 hours.

The presented results show that DOIRABnB provides scalability which cannot be achieved by IRABnB and Base because of their requirement to build the entire game. Furthermore, the results on random games show that even on small games, where IRABnB and Base can be used, DOIRABnB provides the best performance and hence it is the most efficient algorithm.

6.1.2. Size of strategy representation and quality of resulting strategies

Here, we discuss the size of strategy representation needed in imperfect recall abstractions compared to their perfect recall counterparts. Note that we use the number of information sets of the maximizing player present in a given game for this purpose since in the worst case a strategy needs to define behavior in each of them. Additionally, we provide results showing the quality of the strategy resulting from solving these abstractions in the original unabstracted game.

In Table 2 we present the relative amount of information sets of the maximizing player in a specific abstraction compared to the unabstracted game for OshiZumo (left) and Search game (right). In both domains, we use the perfect recall game and its imperfect recall abstraction with rules described earlier in this section. Moreover, in both domains, we experiment with refining the abstraction by giving the maximizing player information about the last k moves he has made in the past (k is specified in the first row of every column). In the case of OshiZumo, each row of the table represents the setting with the

number of coins specified in the first column. In the case of Search game, every row corresponds to a different number of moves allowed for every player specified in the first column. As you can see the number of information sets is dramatically smaller in all the presented settings, showing that the use of imperfect recall abstractions can lead to significant space savings. Additionally, the results suggest that the relative size will further decrease with the increase in the size of the original unabstracted games.

Finally, we provide results showing the actual bounds on the maxmin value computed during the run of DOIRABnB and the quality of the resulting strategies.

In the first column of Fig. 12 we present more detailed results of DOIRABnB in OshiZumo with 11 coins. The plots depict the bounds on the maxmin value in every iteration of the DOIRABnB algorithm, each row for an instance with different number of remembered moves. As we can see DOIRABnB in all the abstractions converges to a strategy with exploitability 0. Since the maxmin value of the original game is also 0 and all the assumptions in Corollary 1 are satisfied, it follows that the maxmin strategy obtained by solving the abstraction is the maxmin strategy of the original game. In the second column of Fig. 12 we show similar results for Search game with 7 moves for every player. The DOIRABnB in all the abstractions converges to a strategy with exploitability $-\frac{1}{3}$ which is again the maxmin value of the original perfect recall game. Since all the assumptions in Corollary 1 are satisfied, it follows that the maxmin strategy obtained by solving the abstraction is the maxmin strategy of the original game.

7. Conclusion

In this paper, we are interested in exploring the possible limitations and the space savings achievable by the use of imperfect recall abstractions in the size of strategy representation. We focus on A-loss recall games, a subclass of imperfect recall games where each loss of memory of a player can be tracked to loosing information about his actions [20,21]. We provide a complete picture of solving imperfect recall and A-loss recall games. We show that most of the hardness results known for imperfect recall games still hold in A-loss recall games. On the other hand, we provide sufficient and necessary (i.e., if and only if) condition for the existence of Nash equilibrium in A-loss recall games. This result makes the A-loss recall games the only subset of imperfect recall games, where such conditions are known. Additionally, we show that A-loss recall property allows us to compute a best response in polynomial time (computing best response is NP-hard in imperfect recall games), which in turn allows us to create the first set of algorithm for approximating maxmin strategy of a player having imperfect recall, when the minimizing player has A-loss recall. More specifically, we introduce a bilinear program with size linear to the size of the game for approximating the maxmin strategy, and we approximate this bilinear program using Multiparametric Disaggregation Technique (MDT) [25]. MDT uses a digitwise approximation of the bilinear terms using a specified number of digits, which results in a mixed integer linear program (MILP). We first devise an algorithm, denoted as IRABnB, which employs branch-and-bound search on the linear relaxation of this MILP. IRABnB simultaneously searches for the assignment to binary variables and improves the precision of the bilinear term approximation until the desired approximation of the maxmin strategy is reached. To increase the scalability, we further extend IRABnB with incremental strategy generation (the resulting algorithm is denoted DOIRABnB). To create DOIRABnB, we provide a non-trivial combination of two iterative procedures (the incremental strategy generation and the IRABnB) and provide guarantees of convergence of this algorithm to ε-maxmin strategy. We show that DOIRABnB is capable of solving games with up to $5 \cdot 10^9$ states in approximately 1 hour.

Finally, we experimentally demonstrate that employing simple imperfect recall abstractions which still allow us to compute the maxmin strategy of the original game can lead to strategies with the relative size as low as 0.03% of the size of the strategy in the original unabstracted game.

Acknowledgements

This research was supported by the Czech Science Foundation (grant no. 15-23235S), and by the Grant Agency of the Czech Technical University in Prague, grant No. SGS16/235/OHK3/3T/13. Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures".

Appendix A. Supplementary material for Section 2.3

Lemma 2. *Let G be an imperfect recall game without absentmindedness and b_1 strategy of player 1. There exists an* ex ante *pure best response of player 2.*

Proof. If the strategy of player 1 is fixed, finding the best response for player 2 corresponds to finding an optimum of a function over a closed convex polytope of all possible behavioral strategies of player 2 created by constraining the strategy by

$$\sum_{a \in A(I)} b_2(a) = 1, \forall I \in \mathcal{I}_2.$$

Vertices of this polytope are formed by pure behavioral strategies. Since each action can be chosen at most once in a game without absentmindedness, the objective function is multilinear in a form

$$\sum_{z\in\mathcal{Z}} C(z)b_1(z)u_2(z)\prod_{a\in seq_2(z)} b_2(a).$$

Notice that, thanks to the assumption of no absentmindedness, the variables $b_2(a)$ in every product always describe behavior at most once for one information set and so the variables are independent. Hence, an optimum must be in one of the vertices of this polytope – that is a pure behavioral strategy. □

Lemma 3. *Let G be an imperfect recall game where player i has A-loss recall. Let G' be the coarsest perfect recall refinement of G for player i. Every pure behavioral strategy b'_i of player i from G' has realization equivalent pure behavioral strategy b_i in G and vice versa.*

Proof. First we show how b_i is constructed from b'_i. Consider an information set I of player i in G and corresponding information sets $I^1,\dots,I^j$ created according to $H(I)$ in the coarsest perfect refinement G' of G.

Let us first show that at most one of information sets $I^1,\dots,I^j$ can be reached when players play according to strategy profile (b'_i, b_{-i}), for every $b_{-i}\in\mathcal{B}_{-i}$ in G'. Due to A-loss recall of player i, for every pair of nodes h_k, h_l from two different information sets I^k, $I^l\in\{I^1,\dots,I^j\}$, there exists an information set I' of player i and two distinct actions $a,a'\in\mathcal{A}_i(I'), a\neq a'$ such that $a\in seq_i(h_k)\wedge a'\in seq_i(h_l)$. However, since b'_i is a pure strategy, only one action among the pair of actions a,a' can be played with a non-zero probability and consequently, only one information set in every pair $I^k, I^l\in\{I^1,\dots,I^j\}$ can be reached.

We use this property to construct b_i. For information set I from G that is divided into information sets $I^1,\dots,I^j$ in G' we define $b_i(I,a)=1$ for action $a\in\mathcal{A}(I^k)$, where $b'_i(I^k,a)=1$ and I^k is the only reachable set from $I^1,\dots,I^j$ as shown above. If no information set from $I^1,\dots,I^j$ is reachable, we set b_i in I arbitrarily. For all information sets $I'\in\mathcal{I}'$ that are not split in the coarsest perfect recall refinement (and therefore are the same as in G), we set $b_i(I')=b'_i(I')$. Due to the construction, the realization equivalence between b'_i and b_i follows immediately.

The same construction yields realization equivalent strategy also in the opposite direction. □

Lemma 4. *Let G be an imperfect recall game where player 2 has A-loss recall and b_1 a strategy of player 1. Let G' be the coarsest perfect recall refinement of G for player 2. Let b'_2 be a pure best response to b_1 in G' and let b_2 be a realization equivalent behavioral strategy to b'_2 in G, then b_2 is a pure best response to b_1 in G.*

Proof. Note that b_1 is a valid strategy in both games since the information set structure for player 1 in G and G' is identical. Since player 2 is not absentminded in G, it is enough to consider pure behavioral strategies (Lemma 2) as a best response to b_1 in G. Furthermore from Lemma 3 we know that every pure behavioral strategy $\hat{b}'_2$ from G' has realization equivalent pure behavioral strategy $\hat{b}_2$ in G, hence also the expected utility $u_1(b_1,\hat{b}'_2)$ in G' is equal to $u_1(b_1,\hat{b}_2)$ in G. Since b'_2 is a best response to b_1 in G', it holds that for every pure behavioral strategy $\hat{b}'_2$ in G' and its realization equivalent counterpart $\hat{b}_2$ in G,

$$u_2(b_1,\hat{b}_2)=u_2(b_1,\hat{b}'_2)\leq u_2(b_1,b'_2)=u_2(b_1,b_2).$$

Finally, since also every pure behavioral strategy $\hat{b}_2$ in G has realization equivalent pure behavioral strategy in G' (Lemma 3), there can be no $\hat{b}_2$ for which $u_2(b_1,\hat{b}_2)>u_2(b_1,b_2)$. □

Appendix B. Supplementary material for Section 3.1

In this section, we provide a sufficient condition for the existence of NE in general imperfect recall games.

Corollary 2. *An imperfect recall game G (not restricted to A-loss recall) has a Nash equilibrium in behavioral strategies if there exists a Nash equilibrium strategy profile b in behavioral strategies of the coarsest perfect recall refinement G' of G, such that $\forall I\in\mathcal{I}^G\ \forall H_k, H_l\in H(I): b(H_k)=b(H_l)$, where $b(H)$ stands for the behavioral strategy in the information set of G' formed by states in H, the opposite does not hold.*

Proof. The proof follows from the fact that every pure behavioral strategy of G has realization equivalent pure behavioral strategy in G' since we only merge information sets in G' to obtain G. The merging of information sets eliminates pure behavioral strategies with mutually exclusive behavior in these sets. Since b is a Nash Equilibrium of G', we know that there exists no incentive for any player to deviate to any pure behavioral strategy in G' and therefore also no incentive to deviate to any pure behavioral strategy in G. This, in combination with the fact, that b prescribes valid strategy in G implies that b is a Nash Equilibrium in behavioral strategies of G.

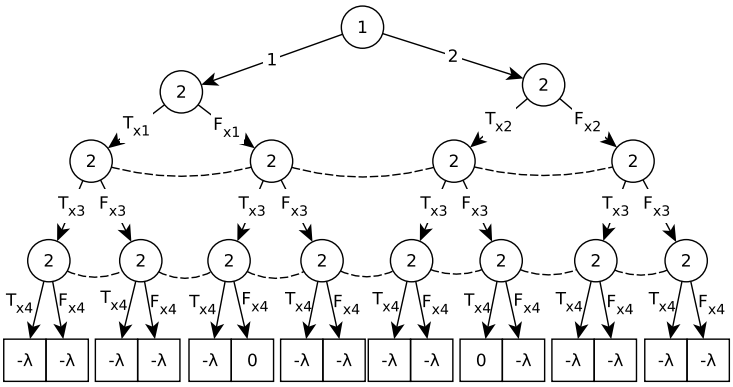


Fig. C.13. An imperfect recall game reduction from Theorem 11 of 3-SAT problem $x_1 \vee \neg x_3 \vee x_4 \wedge \neg x_2 \vee x_3 \vee \neg x_4$.

Finally, we provide a counter-example showing that the opposite direction of the implication does not hold. Consider the game in left subfigure of Fig. 1. Here the only Nash equilibrium in behavioral strategies is playing d and e deterministically and mixing uniformly between g, h. The only Nash equilibrium of its coarsest perfect recall refinement (shown in right subfigure of Fig. 1) is, however, playing d, e, h and i deterministically. □

Appendix C. Supplementary material for Section 3.3

Theorem 11. *The problem of deciding whether player 2 having an imperfect recall can guarantee an expected payoff of at least λ is* NP*-hard even if player 1 has perfect recall, there are no chance moves and the game is zero-sum [15].*

Proof. The proof is made by reduction from 3-SAT problem. The example of the reduction is shown in Fig. C.13. Given n clauses $x_{j,1} \vee x_{j,2} \vee x_{j,3}$ we create a two-player zero-sum game in a following way. In the root of the game player 1 chooses between n actions, each corresponding to one clause. Each action of player 1 leads to a state of player 2. Every such state of player 2 corresponds to the variable $x_{j,1}$ where j is the index of the clause chosen in the root of the game. Every such state has actions $T_{x_{j,1}}, F_{x_{j,1}}$ available, these actions correspond to setting the variable $x_{j,1}$ to true or false respectively. After both $T_{x_{j,1}}, F_{x_{j,1}}$ in $x_{j,1}$ we reach the state representing the assignment to $x_{j,2}$ with the same setup (state representing the assignment to $x_{j,3}$ is reached after that). After the assignment to $x_{j,3}$ we reach the terminal state with utility $-\lambda$ for player 1 if the assignment to $x_{j,1}$, $x_{j,2}$ and $x_{j,3}$ satisfies the clause $x_{j,1} \vee x_{j,2} \vee x_{j,3}$, 0 otherwise. The information sets of player 2 group together all the states corresponding to the assignment to one variable in the original 3-SAT problem (note that we assume that the order of variables in every clause follows some complete ordering on the whole set of variables in the 3-SAT problem).

We will show that player 2 can guarantee the worst case expected value λ if and only if the original 3-SAT problem is satisfiable. First, we show that if the original 3-SAT problem is satisfiable player 2 can guarantee the worst case expected value λ. The worst case expected value λ is achieved when player 2 plays according to the assignment which satisfies the original 3-SAT problem. Next, we show that if player 2 can guarantee the worst case expected value λ, the original 3-SAT problem has to be satisfiable. This holds since if there would be at least one clause not satisfied, player 1 will always choose the action corresponding to this clause, causing the expected value smaller than λ.

The reduction is polynomial, since the game has $2^3 n$ leaves. □

Theorem 12. *It is* NP*-hard to check whether there exists a Nash equilibrium in behavioral strategies in two player imperfect recall games even if the game is zero-sum and there are no chance moves [16].*

Proof. The proof is by reduction from 3-SAT. Given a 3-CNF formula F with n clauses we construct a zero-sum two-player game G as follows. Player 1 (the max-player) starts the game by making two actions, each time choosing one of n clauses of F. We put all corresponding $n+1$ nodes (the root plus n nodes in the next layer) in one information set. If he fails to choose the same clause twice, he receives a payoff of $-n^3$ and the game stops. Otherwise, the game continues in the same way as in the proof of Theorem 11. If the choices of player 2 satisfy the clause, player 1 receives payoff 0. If none of them do, player 1 receives payoff 1. An example of the reduction is shown in Fig. C.14.

The proof is now concluded by the following claim: G has an equilibrium in behavior strategies if and only if F is satisfiable. Assume first that F is satisfiable. G then has the following equilibrium (which happens to be pure): player 2 plays according to a satisfying assignment while player 1 uses an arbitrary pure strategy. The payoff is 0 for both players, and no player can modify their behavior to improve this, so we have an equilibrium. Next, assume that G has an equilibrium. We shall argue that F has a satisfying assignment. We first observe that player 1 in equilibrium must have expected payoff at least 0. If not, he could switch to an arbitrary pure strategy and would be guaranteed a payoff of at least 0. Now, look at the two actions (i.e., clauses) that player 1 is most likely to choose. Let clause i be the most likely and let clause j be

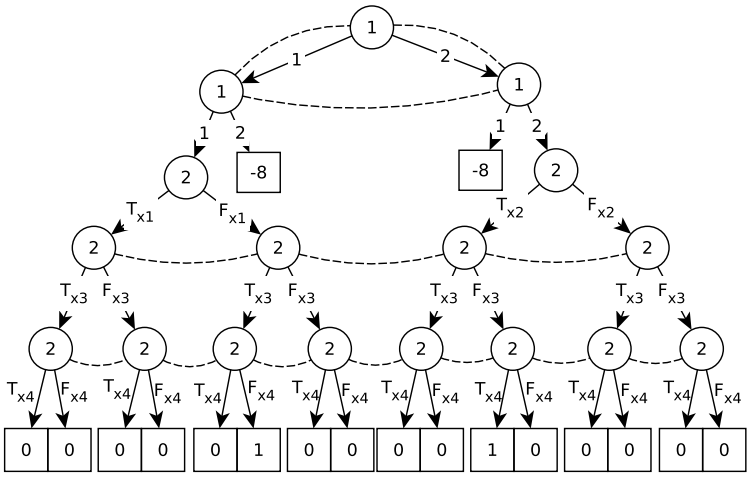


Fig. C.14. An A-loss recall game reduction from Theorem 12 of 3-SAT problem $x_1 \vee \neg x_3 \vee x_4 \wedge \neg x_2 \vee x_3 \vee \neg x_4$.

the second-most likely. If player 1 chooses i and then j, he gets a payoff of $-n^3$. His maximum possible payoff is 1, and his expected payoff is at least 0. Hence, we must have that $-n^3 p_i p_j + 1 \geq 0$. Since $p_i \geq \frac{1}{n}$, we have that $p_j \leq \frac{1}{n^2}$. Since clause j was the second most likely choice, we in fact have that $p_i \geq 1 - (n-1)(\frac{1}{n^2}) > 1 - \frac{1}{n}$. Thus, there is one clause that player 1 plays with probability above $1 - \frac{1}{n}$. Player 2 could then guarantee an expected payoff of less than $\frac{1}{n}$ for player 1 by playing any assignment satisfying this clause. Since we are playing an equilibrium, this would not decrease the payoff of player 1 so player 1 currently has an expected payoff less than $\frac{1}{n}$. Now, look at the assignment defined by the most likely choices of player 2 (i.e., the choices he makes with probability at least 0.5, breaking ties in an arbitrary way). We claim that this assignment satisfies F. Suppose not. Then there is some clause not satisfied by F. If player 1 changes his current strategy to the pure strategy choosing this clause, he obtains an expected payoff of at least $(\frac{1}{2})^3 \geq \frac{1}{n}$ (supposing, wlog, that $n \geq 8$). This contradicts the equilibrium property, and we conclude that the assignment, in fact, does satisfy F. □

Appendix D. Supplementary material for Section 4.5

Lemma 7. *Let $h \in I_1$ be a history and b_1^1, b_1^2 be behavioral strategies prescribing different behavior in I_1 but prescribing the same behavior in all subsequent states $h \sqsubset h'$. Let $v_{max}(h)$ and $v_{min}(h)$ be maximal and minimal utility of player 1 in the subtree of h. Then the following holds:*

$$\max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(b_1^1, b_2^1) - u_1^h(b_1^2, b_2^2)| \leq \frac{v_{max}(h) - v_{min}(h)}{2} \cdot \|b_1^1(I_1) - b_1^2(I_1)\|_1,$$

where $u_1^h(b_1, b_2)$ is the expected utility of player 1, when starting in h and playing according to b_1, b_2.

Proof. When comparing b_1^1 and b_1^2, we can identify two subsets of $\mathcal{A}(I_1)$ – a set of actions A^+ where the probability of playing the action in b_1^2 is higher than in b_1^1 and A^- where the probability in b_1^2 is lower than in b_1^1. Let us denote

$$C^+ = \sum_{a \in A^+} |b_1^1(I_1, a) - b_1^2(I_1, a)| \tag{D.1}$$

$$C^- = \sum_{a \in A^-} |b_1^1(I_1, a) - b_1^2(I_1, a)| \tag{D.2}$$

We know that $C^+ = C^-$, moreover $\|b_1^1(I_1) - b_1^2(I_1)\|_1 = C^+ + C^-$. In the worst case, decreasing the probability of playing action $a \in A^-$ results in the decrease of the expected value

$$v_{max}(h) \cdot \sum_{a \in A^-} |b_1^1(I_1, a) - b_1^2(I_1, a)| = v_{max}(h) \cdot C^-.$$

Similarly the increase of the probabilities of actions in A^+ can add in the worst case $v_{min}(h) \cdot C^+$ to the expected value of the strategy. Hence,

$$\begin{aligned} \max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(b_1^1, b_2^1) - u_1^h(b_1^2, b_2^2)| &\leq v_{max}(h) \cdot C^- - v_{min}(h) \cdot C^+ \\ &= [v_{max}(h) - v_{min}(h)] \cdot C^+ \end{aligned}$$

$$= \frac{v_{max}(h) - v_{min}(h)}{2} \cdot 2C^{+}$$
$$= \frac{v_{max}(h) - v_{min}(h)}{2} \cdot \|b_1^1(I_1) - b_1^2(I_1)\|_1. \qquad \square$$

Appendix E. Supplementary material for Section 5.2.2

Theorem 9. *When using $P_{max}(I_1)$ from Theorem 8 for all $I_1 \in \mathcal{I}_1^{IR}$ and all $a \in \mathcal{A}(I_1)$, the number of iterations of the* IRABNB *algorithm needed to find an ε-optimal solution is in*

$$\mathcal{O}\left(\left(\frac{3^{4\log_{10}(S_1)+4}}{\varepsilon^2}\right)^{S_1}\right)$$

where $S_1 = |\mathcal{I}_1^{IR}|\mathcal{A}_1^{max}$.

Proof. We start by proving that there is $n \in \mathcal{O}(3^{4|\mathcal{I}_1|\mathcal{A}_1^{max}P_{max}})$ nodes in the BnB tree, where $\mathcal{A}_1^{max} = \max_{I\in\mathcal{I}^{IR}{}_1} |\mathcal{A}(I)|$ and $P_{max} = \max_{I\in\mathcal{I}^{IR}{}_1} P_{max}(I)$. This holds since in the worst case we branch for every action in every information set, hence $|\mathcal{I}^{IR}{}_1|\mathcal{A}_1^{max}$ branchings. We can bound the number of branchings for a fixed action by $4 \cdot P_{max}$, since there are 10 digits on which we use binary halving and at most P_{max} number of digits of precision are required. $4|\mathcal{I}_1|\mathcal{A}_1^{max}P_{max}$ is, therefore, the maximum depth of the branch-and-bound tree. Finally, the branching factor of the branch-and-bound tree is at most 3 (we add at most 3 successors in every iteration of Algorithm 1).

By substituting

$$\max_{I_1\in\mathcal{I}_1}\left\lceil \max_{h\in I_1} \log_{10}\frac{|\mathcal{A}(I_1)|\cdot d\cdot v_{diff}(h)}{2\varepsilon}\right\rceil$$

for P_{max} in the above bound (Theorem 8), we obtain

$$n \in \mathcal{O}\left(3^{4S_1 \max_{I_1\in\mathcal{I}_1}\left\lceil \max_{h\in I_1}\log_{10}\frac{|\mathcal{A}(I_1)|\cdot d\cdot v_{diff}(h)}{2\varepsilon}\right\rceil}\right), \text{ where } S_1 = |\mathcal{I}_1|\mathcal{A}_1^{max}$$
$$\in \mathcal{O}\left(3^{4S_1 \max_{I_1\in\mathcal{I}_1}\left\lceil \log_{10}\frac{|\mathcal{A}(I_1)|\cdot d\cdot v_{diff}(\emptyset)}{2\varepsilon}\right\rceil}\right)$$
$$\in \mathcal{O}\left(3^{4S_1 \max_{I_1\in\mathcal{I}_1}\left\lceil \log_{10}\frac{S_1 v_{diff}(\emptyset)}{2\varepsilon}\right\rceil}\right)$$
$$\in \mathcal{O}\left(3^{4S_1\left\lceil \log_{10}\frac{S_1\cdot v_{diff}(\emptyset)}{2\varepsilon}\right\rceil}\right)$$
$$\in \mathcal{O}\left(3^{4S_1\left(\log_{10}\frac{S_1\cdot v_{diff}(\emptyset)}{2\varepsilon}+1\right)}\right)$$
$$\in \mathcal{O}\left(3^{4S_1\left(\log_{10}(S_1\cdot v_{diff}(\emptyset))-\log_{10}(2\varepsilon)+1\right)}\right)$$
$$\in \mathcal{O}\left(3^{4S_1\left(\log_{10}(S_1\cdot v_{diff}(\emptyset))+1\right)}3^{-4S_1\log_{10}(2\varepsilon)}\right)$$
$$\in \mathcal{O}\left(3^{4S_1\left(\log_{10}(S_1\cdot v_{diff}(\emptyset))+1\right)}3^{-4S_1\frac{\log_3(2\varepsilon)}{\log_3(10)}}\right)$$
$$\in \mathcal{O}\left(3^{4S_1\left(\log_{10}(S_1\cdot v_{diff}(\emptyset))+1\right)}(2\varepsilon)^{\frac{-4S_1}{\log_3(10)}}\right)$$
$$\in \mathcal{O}\left(3^{4S_1\left(\log_{10}(S_1\cdot v_{diff}(\emptyset))+1\right)}(2\varepsilon)^{-2S_1}\right)$$
$$\in \mathcal{O}\left(3^{4S_1\left(\log_{10}(S_1\cdot v_{diff}(\emptyset))+1\right)}2^{-2S_1}\varepsilon^{-2S_1}\right)$$
$$\in \mathcal{O}\left(3^{4S_1\log_{10}(S_1\cdot v_{diff}(\emptyset))}3^{4S_1}2^{-2S_1}\varepsilon^{-2S_1}\right)$$

$$\in \mathcal{O}\left(3^{4S_1 \log_{10}(S_1 \cdot v_{diff}(\emptyset))} 21^{S_1} \varepsilon^{-2S_1}\right)$$
$$\in \mathcal{O}\left(\left(\frac{3^{4\log_{10}(S_1 \cdot v_{diff}(\emptyset))+4}}{\varepsilon^2}\right)^{S_1}\right)$$

Finally, we can substitute $v_{diff}(\emptyset)$ by 1, since we can modify the utility structure of the game to have utilities in $[0, 1]$ interval. □

Appendix F. CFR in imperfect recall games

In this section we discuss the problems which prevent the convergence of Counterfactual regret minimization algorithm [8] (CFR) in imperfect recall games. First, we briefly describe the ideas behind external regret and CFR. Finally, we discuss why CFR does not have to converge to no-regret strategies in imperfect recall games and show an example where CFR can converge to a strategy profile with arbitrarily worse expected value than the maxmin value of the game.

F.1. External regret

Given a sequence of behavioral strategy profiles $b^1, \ldots, b^T$, the external regret for player i,

$$R_i^T = \max_{b_i' \in \mathcal{B}_i} \sum_{t=1}^{T} (u_i(b_i', b_{-i}^t) - u_i(b_i^t, b_{-i}^t)), \tag{F.1}$$

is the amount of utility player i could have gained if he played the best possible strategy across all time steps $t \in \{1, \ldots, T\}$. An algorithm is a no-regret algorithm for player i, if the average positive regret approaches zero; i.e.

$$\lim_{T \to \infty} \frac{R_i^{T,+}}{T} = 0,$$

where $R_i^{T,+} = \max(R_i^T, 0)$.

F.2. Counterfactual regret minimization

Counterfactual regret is defined for each iteration t, player i, information set $I \in \mathcal{I}_i$ and action $a \in A(I)$ as $r_i^t(I, a) = v_i(b_{I \to a}^t, I) - v_i(b^t, I)$, where $b_{I \to a}^t$ is the strategy profile b^t except for I, where a is played and

$$v(b, I) = \sum_{z \in Z_I} u_i(z) \pi_{-i,c}^b(z[I]) \pi^b(z[I], z),$$

where $\pi^b(h)$ is the probability that h will be reached when players play according to the strategy profile b, with π_i^b being the contribution of player i and $\pi_{-i,c}^b$ the contribution of $-i$ and chance. $z[I]$ is state h which is visited in I in order to reach leaf z. Finally, $\pi^b(h, h')$ stands for the probability that h' will be reached from h when players play according to b. The immediate counterfactual regret is defined as

$$R_{i,imm}^T(I) = \frac{1}{T} \max_{a \in A(I)} \sum_{t=1}^{T} r_i^t(I, a).$$

In games having perfect recall, minimizing the immediate counterfactual regrets at every information set minimizes the average external regret. This holds because perfect recall implies that the external regret is bounded by the sum of positive parts of immediate counterfactual regrets [8],

$$R_i^T \leq \sum_{I \in \mathcal{I}_i} \max_{a \in A(I)} R_i^{T,+}(I, a), \tag{F.2}$$

and thus

$$\frac{R_i^T}{T} \leq \frac{\Delta_i |\mathcal{I}_i| \sqrt{|A_i|}}{T}, \tag{F.3}$$

where $|A_i| = \max_{I \in \mathcal{I}_i} |\mathcal{A}(I)|$. In imperfect recall games, however, equations (F.2) and (F.3) are not guaranteed to hold.

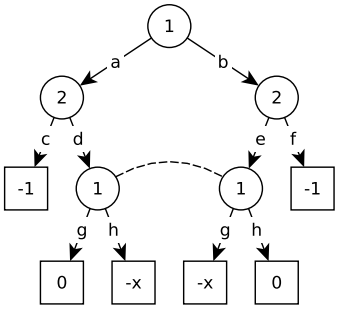


Fig. F.15. An A-loss recall game where CFR finds a strategy with the expected utility arbitrarily worse than the maxmin value.

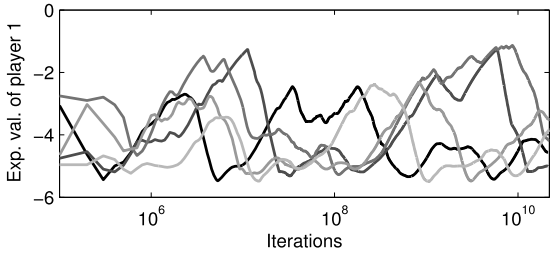


Fig. F.16. The exploitability of the average strategy computed by the CFR with outcome sampling (y-axis) with increasing number of iterations (logarithmic x-axis) for 5 different seeds.

The no-regret learning cannot work in general in imperfect recall games since the loss function

$$l^t(b_i) = u_i(b_i^t, b_{-i}^t) - u_i(b_i, b_{-i}^t) \tag{F.4}$$

used in equation (F.1) can be non-convex over the probability simplex of behavioral strategies (the loss function must be convex for no-regret learning to have convergence guarantees [35]).

Example 5. Assume we are in the step T of a no-regret learning algorithm solving the game from Fig. F.15, and we evaluate the loss from eq. (F.4) of some strategy b_1 in step $t < T$. Let $b_1^t(a) = b_1^t(g) = 0.5$ and $b_2^t(d) = b_2^t(e) = 1$. Let $b_1(a) = b_1(g) = 1$, $b_1'(b) = b_1'(h) = 1$, and $b_1''(a) = b_1''(g) = 0.5$. The losses of these strategies are $l^t(b_1) = -x$, $l^t(b_1') = -x$, $l^t(b_1'') = 0$. Since b_1'' is a convex combination of b_1 and b_1' with uniform weights, it follows that the loss function is non-convex, hence the convergence guarantees used in CFR due to Gordon [35] no longer apply. By increasing $x > 2$ in the game from Fig. F.15, the CFR can find a strategy with its exploitability arbitrarily worse than the maxmin value -1 (we show the exploitability of strategies resulting from CFR in the next section), since mixing between actions a and b can yield the expected value strictly worse than the expected value reached by deterministic samples containing a and b if player 2 plays d and e with positive probability. The game has A-loss recall and has 2 NE, playing (a, g) or (b, h) deterministically for player 1 and (c, f) for player 2 (no mix between these two NE strategies for player 1 is a NE).

The non-convexity of the loss function shown in Example 5 will never appear in case of perfect recall games since the behavior of i after any $a, a' \in A(I_i)$ is independent $\forall I \in \mathcal{I}_i$. Furthermore, the guarantee of convergence of CFR to (ε-)optimal strategies in chance relaxed skew well-formed games [17] is based on bounding the non-convexity of the loss function.

F.3. Experimental evaluation of strategies computed by CFR

Here, we empirically demonstrate the performance of the outcome-sampling version of CFR [36] on the example game from Fig. F.15. Fig. F.16 depicts the exploitability of the average strategy computed by the CFR (logarithmic x-axis shows the number of iterations, the y-axis shows the exploitability for the average strategy of player 1, every line represents one run for a given seed). The algorithm does not converge to any fixed strategy, moreover, the exploitability differs significantly from the maxmin value of -1 for player 1. Note that vanilla CFR (see, e.g., [36], page 22) does not work either, since for example when initialized to uniform strategy, player 1 will never change his strategy since the expected values after his actions are always equal.

References

[1] V. Lisý, T. Davis, M. Bowling, Counterfactual regret minimization in sequential security games, in: Proceedings of AAAI Conference on Artificial Intelligence, 2016.
[2] G. Christodoulou, A. Kovács, M. Schapira, Bayesian combinatorial auctions, Autom. Lang. Program. (2008) 820–832.

[3] T. Sandholm, Steering evolution strategically: computational game theory and opponent exploitation for treatment planning, drug design, and synthetic biology, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI Press, 2015, pp. 4057–4061.
[4] B. Bošanský, V. Lisý, M. Lanctot, J. Čermák, M.H. Winands, Algorithms for computing strategies in two-player simultaneous move games, Artif. Intell. 237 (2016) 1–40.
[5] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, M. Bowling, DeepStack: expert-level artificial intelligence in heads-up no-limit poker, Science 356 (6337) (2017) 508–513, https://doi.org/10.1126/science.aam6960.
[6] F. Fang, T.H. Nguyen, R. Pickles, W.Y. Lam, G.R. Clements, B. An, A. Singh, B.C. Schwedock, M. Tambe, A. Lemieux, PAWS—a deployed game-theoretic application to combat poaching, AI Mag. 38 (1) (2017) 23–36, https://doi.org/10.1609/aimag.v38i1.2710.
[7] B. von Stengel, Efficient computation of behavior strategies, Games Econ. Behav. 14 (1996) 220–246.
[8] M. Zinkevich, M. Johanson, M.H. Bowling, C. Piccione, Regret minimization in games with incomplete information, in: Advances in Neural Information Processing Systems, 2007, pp. 1729–1736.
[9] S. Hoda, A. Gilpin, J. Peña, T. Sandholm, Smoothing techniques for computing Nash equilibria of sequential games, Math. Oper. Res. 35 (2) (2010) 494–512.
[10] A. Gilpin, T. Sandholm, T.B. Sørensen, Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker, in: Proceedings of the National Conference on Artificial Intelligence, vol. 22, 2007, p. 50.
[11] A. Gilpin, T. Sandholm, Lossless abstraction of imperfect information games, J. ACM 54 (5) (2007) 25.
[12] C. Kroer, T. Sandholm, Extensive-form game abstraction with bounds, in: Proceedings of the Fifteenth ACM Conference on Economics and Computation, ACM, 2014, pp. 621–638.
[13] N. Brown, T.W. Sandholm, Simultaneous abstraction and equilibrium finding in games, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.
[14] P.C. Wichardt, Existence of Nash equilibria in finite extensive form games with imperfect recall: a counterexample, Games Econ. Behav. 63 (1) (2008) 366–369.
[15] D. Koller, N. Megiddo, The complexity of two-person zero-sum games in extensive form, Games Econ. Behav. 4 (1992) 528–552.
[16] K.A. Hansen, P.B. Miltersen, T.B. Sørensen, Finding equilibria in games of no chance, in: Computing and Combinatorics, Springer, 2007, pp. 274–284.
[17] C. Kroer, T. Sandholm, Imperfect-recall abstractions with bounds in games, in: Proceedings of the Seventeenth ACM Conference on Economics and Computation, ACM, 2016, pp. 459–476.
[18] M. Lanctot, N. Burch, M. Zinkevich, M. Bowling, R.G. Gibson, No-regret learning in extensive-form games with imperfect recall, in: Proceedings of the 29th International Conference on Machine Learning, ICML-12, 2012, pp. 65–72.
[19] B. Bošanský, A.X. Jiang, M. Tambe, C. Kiekintveld, Combining compact representation and incremental generation in large games with sequential strategies, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015, pp. 812–818.
[20] M. Kaneko, J.J. Kline, Behavior strategies, mixed strategies and perfect recall, Int. J. Game Theory 24 (1995) 127–145.
[21] J.J. Kline, Minimum memory for equivalence between ex ante optimality and time-consistency, Games Econ. Behav. 38 (2002) 278–305.
[22] J. Čermák, B. Bošanský, Towards solving imperfect recall games, in: Proceedings of AAAI Computer Poker Workshop, 2017.
[23] B. Bosansky, J. Cermak, K. Horak, M. Pechoucek, Computing maxmin strategies in extensive-form zero-sum games with imperfect recall, in: ICAART, 2017.
[24] D. Koller, N. Megiddo, B. Von Stengel, Efficient computation of equilibria for extensive two-person games, Games Econ. Behav. 14 (2) (1996) 247–259.
[25] S. Kolodziej, P.M. Castro, I.E. Grossmann, Global optimization of bilinear programs with a multiparametric disaggregation technique, J. Glob. Optim. 57 (4) (2013) 1039–1063.
[26] J. Čermák, B. Bošanský, M. Pěchouček, Combining incremental strategy generation and branch and bound search for computing maxmin strategies in imperfect recall games, in: Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems, AAMAS, 2017.
[27] B. Bosansky, C. Kiekintveld, V. Lisy, M. Pechoucek, An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information, J. Artif. Intell. Res. (2014) 829–866.
[28] H. Kuhn, Extensive games and the problem of information, in: H. Kuhn, A. Tucker (Eds.), Contributions to the Theory of Games, 2016, pp. 193–216.
[29] M. Piccione, A. Rubinstein, On the interpretation of decision problems with imperfect recall, Games Econ. Behav. 20 (1) (1997) 3–24.
[30] J.F. Nash, et al., Equilibrium points in n-person games, Proc. Natl. Acad. Sci. 36 (1) (1950) 48–49.
[31] M.R. Garey, R.L. Graham, D.S. Johnson, Some NP-complete geometric problems, in: Proceedings of the Eighth Annual ACM Symposium on Theory of Computing, ACM, 1976, pp. 10–22.
[32] K. Etessami, M. Yannakakis, On the complexity of nash equilibria and other fixed points, SIAM J. Comput. 39 (6) (2010) 2531–2597.
[33] B. Bošanský, V. Lisý, M. Lanctot, J. Čermák, M.H. Winands, Algorithms for computing strategies in two-player simultaneous move games, Artif. Intell. 237 (2016) 1–40.
[34] H.B. McMahan, G.J. Gordon, A. Blum, Planning in the presence of cost functions controlled by an adversary, in: Proceedings of the International Conference on Machine Learning, 2003, pp. 536–543.
[35] G.J. Gordon, No-regret algorithms for online convex programs, in: Advances in Neural Information Processing Systems, 2006, pp. 489–496.
[36] M. Lanctot, Monte Carlo, Sampling and Regret Minimization for Equilibrium Computation and Decision-Making in Large Extensive Form Games, University of Alberta, 2013.

Appendix D

Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games

Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games

Karel Horák and **Branislav Bošanský** and **Michal Pěchouček**
Department of Computer Science, Faculty of Electrical Engineering
Czech Technical University in Prague
{horak,bosansky,pechoucek}@agents.fel.cvut.cz

Abstract

Security problems can be modeled as two-player partially observable stochastic games with one-sided partial observability and infinite horizon (one-sided POSGs). We seek for optimal strategies of player 1 that correspond to robust strategies against the worst-case opponent (player 2) that is assumed to have a perfect information about the game. We present a novel algorithm for approximately solving one-sided POSGs based on the heuristic search value iteration (HSVI) for POMDPs. Our results include (1) theoretical properties of one-sided POSGs and their value functions, (2) guarantees showing the convergence of our algorithm to optimal strategies, and (3) practical demonstration of applicability and scalability of our algorithm on three different domains: pursuit-evasion, patrolling, and search games.

Introduction

Game theory is widely used in security problems and strategies from game-theoretic models are applied to protect critical infrastructures (Pita et al. 2008; Kiekintveld et al. 2009; Shieh et al. 2012), computer networks (Vanek et al. 2012) or wildlife (Fang, Stone, and Tambe 2015; Fang et al. 2016). Many real-world situations, however, contain a dynamic strategic interaction between the players that has to be addressed in the models. Players can observe (possibly imperfectly) information about actions of their opponent and react to these observations. Examples include patrolling games (Basilico, Gatti, and Amigoni 2009; Vorobeychik et al. 2014; Basilico, Nittis, and Gatti 2016), where a defender protects a set of targets against an attacker, pursuit-evasion (Chung, Hollinger, and Isler 2011), or search games, where a defender is trying to find and capture an attacker.

Finding optimal strategies in such dynamic games with imperfect information is often computationally challenging. If the horizon of the interaction is restricted, we can use the *extensive-form games* formulation. Typically, the size of this representation grows exponentially with the horizon and prohibits us from solving large games. If the horizon is infinite (or indefinite), we can use *partially observable stochastic games (POSGs)*. In POSGs, however, many problems are undecidable (Madani, Hanks, and Condon 1999) even when we use a discount factor to restrict future gains.

However, real-world security scenarios naturally require partial observability and no strictly defined horizon. The goal is to find best robust strategies that provide guarantees on the expected outcome for one player (the defender) against any opponent (the attacker). Therefore, we focus on *discounted two-player zero-sum POSGs with concurrent moves and one-sided partial observability* where it is assumed that the attacker has full information about the game – the attacker knows the state of the game as well as the history of actions played. One-sided partial observability has been used in specific domains such as patrolling games, e.g. (Vorobeychik et al. 2014), or pursuit-evasion games, e.g. (Horak and Bosansky 2016). We generalize this concept to a broad class of POSGs.

Our main contribution is the first domain-independent algorithm that has guarantees to approximate optimal strategies in one-sided POSGs. Our algorithm is a generalization of the heuristic search value iteration algorithm (HSVI) for Partially Observable Markov Decision Processes (POMDPs). Similarly to POMDPs, one-sided POSGs allow us to compactly represent strategies and value functions representing values of the game based on the belief the first player has about the state of the game. Contrary to POMDPs, the presence of the opponent player causes significant technical challenges that we address in this paper. First, we show that the assumption of the one-sided partial observability guarantees that the value functions are convex. Second, we define a value backup operator and show that an iterative application of this operator converges to the optimal values. Third, we generalize the ideas behind HSVI towards one-sided POSGs, and show that our algorithm approximates optimal strategies. Finally, we demonstrate the applicability and scalability of our algorithm on three different domains – patrolling games (including the variant with alarms), pursuit-evasion games, and search games. The results show that our algorithm can closely approximate solutions of large games with more than 4000 states.

Related Work

There are only a few relevant algorithms for computing strategies in POSGs. An algorithm for computing strategies in POSGs where all players have imperfect information was proposed in (Hansen, Bernstein, and Zilberstein 2004). The algorithm approximates an infinite horizon game by increas-

ing the horizon in a finite-horizon game and uses dynamic programming to incrementally construct a set of relevant pure strategies by eliminating dominated strategies. The set of such strategies is then used to form a normal-form (or matrix) representation of the POSG. However, the exponential transformation to the normal form prevents this algorithm from scaling up. One-sided partial observability allows us to avoid such enumeration of pure strategies.

The closest works related to the algorithm presented in this paper are two works on a specific subclass of one-sided POSGs – pursuit-evasion games (PEGs). First, a class of *one-sided partially observable PEGs* was presented and theoretical results on the shape of the value functions and the definition of the value backup operator were provided in (Horak and Bosansky 2017). Second, an HSVI-based algorithm was introduced in (Horak and Bosansky 2016).

Our algorithm can be seen as a significant generalization of this approach to a broader class of one-sided POSGs. First, the set of observations is very limited in PEGs – player 1 is able to observe his own actions only and the only direct information about the position of the opponent is given when player 2 is captured. Considering general observations presents additional challenges for the model and the algorithm which we address in this paper. Secondly, the previous work relied on a uniform sampling of belief points to guarantee the convergence, our algorithm approximates the solution in a deterministic manner.

Two-Player One-Sided POSGs

A *one-sided partially observable stochastic game* G is a tuple $G = \langle \mathcal{S}, \mathcal{A}_1, \mathcal{A}_2, \mathcal{O}, \mathcal{T}, \mathcal{R} \rangle$. The game is played for an infinite number of *stages*. At each stage, the game is in one of the states $s \in \mathcal{S}$ and players choose their actions $a \in \mathcal{A}_1$ and $a' \in \mathcal{A}_2$ simultaneously. An initial state of the game is drawn from a probability distribution $b^0 \in \Delta(\mathcal{S})$, which we treat as a parameter of the game and term the *initial belief*.

The choice of actions determines the outcome of the current stage: Player 1 gets an *observation* $o \in \mathcal{O}$ and the game moves to a state $s' \in \mathcal{S}$ with probability $\mathcal{T}_{s,a,a'}(o, s')$, where s is the current state. Furthermore he gets a reward $\mathcal{R}(s, a, a')$ for this transition. We assume the zero-sum case, hence player 2 receives $-\mathcal{R}(s, a, a')$, and we assume that the rewards are discounted over time with discount factor $\gamma < 1$. Players do not observe their rewards during the game.

We assume perfect recall, hence both players remember their respective histories. A history of the first player is formed by actions he played and observations he received, i.e. $(\mathcal{A}_1 \times \mathcal{O})^t$. The second player has complete observation, hence $\mathcal{S} \times (\mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{O} \times \mathcal{S})^t$ is a set of her histories. The strategies σ_1, σ_2 of the players map each of their histories to a distribution over their actions.

Value of a Strategy and Value of the Game

In this section, we show that the value of a strategy (the expected reward of the first player playing σ_1 when the opponent plays her best response) has a linear dependence on the belief.

The value of the game G is the value of the best strategy available for each of the initial beliefs $b^0 \in \Delta(\mathcal{S})$. We represent the value of a game (based on the initial belief) as a *value function*. This function is a pointwise maximum taken over values of all strategies of the first player, which, since the value of every strategy is linear, forms a convex function.

In the convergence proof of our algorithm, we exploit that the rate of change in the value function is bounded in terms of minimum and maximum rewards of G, i.e. the value function is Lipschitz continuous.

Definition 1 (Value functions)**.** *The value of a strategy σ_1 of the first player is a function $v_{\sigma_1} : \Delta(\mathcal{S}) \to \mathbb{R}$ which assigns the expected utility $v_{\sigma_1}(b^0)$ of the player 1 in the game with initial belief b^0 when the first player follows σ_1 and the second player best-responds. The value function of the game G is a function $v^* : \Delta(\mathcal{S}) \to \mathbb{R}$ that assigns the value $v^*(b^0)$ of the best strategy of the first player for each of the beliefs, i.e. $v^*(b^0) = \sup_{\sigma_1} v_{\sigma_1}(b^0)$.*

Lemma 1. *The value v_{σ_1} of a fixed strategy σ_1 of the first player is linear in the initial belief.*

The proof relies on the fact that the player 2 knows the initial state of the game; hence, the initial belief forms a convex combination of values of best responses for individual states. Due to the space constraints, full proofs of all lemmas can be found in the full version of the paper.

We say that a function f is K-Lipschitz if it satisfies $|f(x) - f(y)| \leq K \cdot \|x - y\|_2$. The key observation to derive the Lipschitz continuity is that the value of the game lies in a bounded interval $[L, U]$ where

$$L = \min_{(s,a,a')} \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s, a, a') \,, \quad U = \max_{(s,a,a')} \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s, a, a') \,.$$

The proof of the following lemma then relies on defining the value of the strategy by assigning these extreme values to the vertices of the belief simplex and identifying the configuration with the largest rate of change.

Lemma 2. *Value function v_{σ_1} of a fixed strategy σ_1 of player 1 is $(U-L)$-Lipschitz.*

Theorem 1. *Value function v^* of the game G is convex in the initial belief and $(U-L)$-Lipschitz.*

Proof. The value function v^* is the supremum taken over a set of $(U - L)$-Lipschitz functions corresponding to values of strategies available to player 1 (Def. 1, Lemma 2). Supremum taken over a family of bounded $(U-L)$-Lipschitz continuous functions is $(U-L)$-Lipschitz continuous. Moreover since these functions are linear (Lemma 1), the resulting value function is convex. □

Value Backup

Now we present a value iteration algorithm for solving one-sided POSGs. The algorithm approximates the value function v^* of the infinite horizon game G by considering value

functions of the game with a restricted horizon. Each iteration of the algorithm improves the approximation by increasing the horizon by one step using the *value backup operator* (denoted H). Applying this operator means that players choose their Nash equilibrium strategies in the current step while assuming that the value of the subsequent stage is represented by the value function from the previous iteration.

The algorithm constructs a sequence $\{v^t\}_{t=0}^{\infty}$, starting with a value function v^0 of a game where only immediate rewards are considered. First, we discuss application of the operator in a single stage. Afterward we show the convergence when the operator is applied repeatedly.

Value Backup Operator

The value backup operator H evaluated at belief b — $[Hv](b)$— corresponds to solving a *stage game* where players choose their Nash equilibrium strategies for one stage of the game (in latter text we use $[Hv](b)$ to refer to this game as well). We denote strategies for one stage $\pi_1 \in \Delta(\mathcal{A}_1)$ for the first player and $\pi_2 : \mathcal{S} \to \Delta(\mathcal{A}_2)$ for the player 2. The utilities in $[Hv](b)$ depend both on the immediate rewards $\mathcal{R}$ and the discounted value of the subsequent game represented by value function v. The immediate rewards part depends solely on the actions played by the players:

$$R^{\text{imm}}_{\pi_1,\pi_2} = \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}_1} \sum_{a'\in\mathcal{A}_2} b(s)\cdot\pi_1(a)\cdot\pi_2(s,a')\cdot\mathcal{R}(s,a,a') \quad (1)$$

The first player both knows the action a he played and observes observation o. He can use this information to derive his belief for the subsequent game:

$$b^{a,o}_{\pi_2}(s') = \frac{1}{\Pr[o|a,\pi_2]} \sum_{s\in\mathcal{S}} \sum_{a'\in\mathcal{A}_2} \mathcal{T}_{s,a,a'}(o,s')\cdot b(s)\cdot\pi_2(s,a') \quad (2)$$

The value of the subsequent game is then the expectation taken over individual action-observation pairs (a, o) of the first player from the values of a game starting in belief $b^{a,o}_{\pi_2}$:

$$R^{\text{subs}}_{\pi_1,\pi_2}(v) = \sum_{a\in\mathcal{A}_1} \sum_{o\in\mathcal{O}} \pi_1(a)\cdot\Pr[o|a,\pi_2]\cdot v(b^{a,o}_{\pi_2}) \,. \quad (3)$$

Since the value function is convex, utility of playing strategy profile (π_1, π_2) is convex when π_1 is fixed and linear when we fix π_2. The minimax theorem (von Neumann 1928; Nikaido 1954) applies and the Nash equilibrium strategy is solved by maximin/minimax:

$$[Hv](b) = \min_{\pi_2} \max_{\pi_1} \left(R^{\text{imm}}_{\pi_1,\pi_2} + \gamma \cdot R^{\text{subs}}_{\pi_1,\pi_2}(v) \right) . \quad (4)$$

Computation of Value Backup Operator

Finally, we present the way of computing $[Hv](b)$. When the value function v is piecewise linear and convex (PWLC), it can be represented by a set Γ of α-vectors and the value backup $[Hv](b)$ can be evaluated by means of linear programming. Each α-vector $\alpha \in \Gamma$ is an $|\mathcal{S}|$-tuple representing the affine value function v_{σ_1} of a fixed strategy σ_1 by specifying its values in each of the pure beliefs ($\alpha(s)$ for each $s \in \mathcal{S}$). We focus on the problem of solving the problem from the perspective of the second player first, who has to choose her strategy π_2 such that the utility V of the best responding player 1 (who chooses his pure best response $a \in \mathcal{A}_1$) is minimized.

The value of playing strategy π_2 against action $a \in \mathcal{A}_1$ equals $R^{\text{imm}}_{a,\pi_2} + \gamma R^{\text{succ}}_{a,\pi_2}(v)$, which allows us to construct a set of best-response constraints (one for each action a)

$$V \geq \sum_{s\in\mathcal{S}} \sum_{a'\in\mathcal{A}_2} b(s)\cdot\pi_2(s,a')\cdot\mathcal{R}(s,a,a') + \\ + \gamma \sum_{o\in\mathcal{O}} \Pr[o|a,\pi_2]\cdot v(b^{a,o}_{\pi_2}) \,. \quad (5)$$

Assuming that the value function v is represented by a set Γ of α-vectors, such that $v(b) = \max_{\alpha\in\Gamma}\langle\alpha, b\rangle$ ($\langle\cdot,\cdot\rangle$ denotes an inner product), its value can be rewritten by a set of inequalities

$$v(b^{a,o}_{\pi_2}) \geq \sum_{s'\in\mathcal{S}} \alpha(s')\cdot b^{a,o}_{\pi_2}(s') \qquad \forall\alpha\in\Gamma \quad (6)$$

where $b^{a,o}_{\pi_2}(s')$ is represented by linear constraints corresponding to Eq. (2). The term $\Pr[o|a,\pi_2]$ occuring in Eqs. (2) and (5) cancels out to form the resulting linear program.

Strategy of the First Player One way to approximate the value function by a PWLC function is to use a finite subset of strategies of the first player. Value functions of these strategies are linear (Lemma 1), and the pointwise maximum from these linear functions gives us the desired PWLC approximation. In such a case, each of the vectors in Γ corresponds to the value function of one of the strategies. The dual linear program is used to find the optimal control strategy of the first player, when duals of Eq. (5) correspond to the strategy to play in the first stage (when the history of the first player is empty) and duals of Eq. (6) prescribes what strategy to follow when (a, o) was observed in the first stage.

Convergence of the Value Backup Operator

In this section we show that a repetitive application of the value backup operator H converges to the *same* value function v^* of the infinite horizon game regardless of what value function it is applied on. We show this by demonstrating that the operator H is a contraction mapping with a factor $\gamma < 1$.

Lemma 3. *Let v, v' be value functions, $b \in \Delta(\mathcal{S})$ be a belief and π_1, π_2 (resp. π'_1, π'_2) be equilibrial strategies in $[Hv](b)$ (resp. $[Hv'](b)$). Assume that for every action-observation pair (a, o) of the first player, $|v(b^{a,o}_{\pi_2}) - v'(b^{a,o}_{\pi_2})| \leq \mu$. Then $|[Hv](b) - [Hv'](b)| \leq \gamma\mu$.*

The lemma is proven by modyfing Nash equilibrium strategy profiles in games $[Hv](b)$ and $[Hv'](b)$ and bounding the difference by the difference of their expected utilities.

Theorem 2. *The operator H is a contraction mapping under the norm $\|v - v'\| = \max_{b\in\Delta(\mathcal{S})} |v(b) - v'(b)|$. It thus has a unique fixpoint – the value function of the infinite horizon game.*

Data: Game $\langle \mathcal{S}, \mathcal{A}_1, \mathcal{A}_2, \mathcal{O}, \mathcal{T}, \mathcal{R} \rangle$, initial belief b^0, discount factor γ, desired precision $\epsilon > 0$, neighborhood parameter R
Result: Approximate value function $\hat{v}$
1 Initialize $\hat{v}$
2 **while** $\mathrm{gap}(\hat{v}(b^0)) > \epsilon$ **do**
3 | `Explore` $(b^0, \epsilon, R, 0)$
4 **return** $\hat{v}$
5 **procedure** `Explore` (b, ϵ, R, t)
6 | $\pi_2 \leftarrow$ optimal strategy of player 2 in $[H\underline{v}](b)$
7 | $(a, o) \leftarrow$ select according to forward exploration heuristic
8 | **if** $\mathrm{excess}(\hat{v}(b^{a,o}_{\pi_2}), t+1) > 0$ **then**
9 | | `Explore` $(b^{a,o}_{\pi_2}, \epsilon, R, t+1)$
10 | $\Gamma \leftarrow \Gamma \cup \{L\Gamma(b)\}$
11 | $\Upsilon \leftarrow \Upsilon \cup \{U\Upsilon(b)\}$ and make $\overline{v}$ $(U-L)$-Lipschitz
Algorithm 1: HSVI algorithm for one-sided POSGs

Proof. Let $\|v - v'\| \leq \mu$. Then for every $b^{a,o}_{\pi_2}$ from Lemma 3 $|v(b^{a,o}_{\pi_2}) - v'(b^{a,o}_{\pi_2})| \leq \mu$ and for every belief b, $|[Hv](b) - [Hv'](b)| \leq \gamma\mu$. The uniqueness of the fixpoint and the convergence properties follow from the Banach's fixed point theorem (Ciesielski 2007). □

HSVI Algorithm for POSGs

Similarly to POMDPs, the value iteration algorithm cannot scale for practical problems. We thus present a point-based algorithm (Algorithm 1) that by sampling the belief space bounds and approximates the true value function v^* of the game by a pair of PWLC functions $\underline{v}$ (*lower bound*), represented by a set of α-vectors Γ, and $\overline{v}$ (*upper bound*) represented as a lower envelope of a set of points Υ. We refer to these functions jointly as $\hat{v}$. The goal of the algorithm is to ensure that the *gap* in the initial belief b^0 of the game induced by the approximation defined as $\mathrm{gap}(\hat{v}(b)) = \overline{v}(b) - \underline{v}(b)$ is no higher than the required precision. Functions $\hat{v}$ are refined by adding new elements to their sets. These new elements result from *point-based updates* of operator H at a single belief point b.

The algorithm is initialized with $\underline{v}$ (and Γ) corresponding to the value of a uniform strategy of the first player and the upper bound $\overline{v}$ (and Υ) results from solving a perfect information refinement of the game. In every iteration, a finite set of beliefs is updated by *forward exploration* (lines 6-9). Beliefs selected by this process contribute to the fact that the gap at b^0 is not sufficiently small, and hence the approximation in these beliefs needs to be improved by applying point-based updates (lines 10 and 11). We now describe how the updates are performed, followed by the description of the forward exploration search.

Point-Based Updates

A point-based update at belief point b updates the lower and upper bound functions $\underline{v}$ and $\overline{v}$ using the optimal strategies in games $[H\underline{v}](b)$ and $[H\overline{v}](b)$. In order to prove the convergence, we require that the functions $\underline{v}$ and $\overline{v}$ are $(U-L)$-Lipschitz; hence, the update has to preserve this property.

The update of $\underline{v}$ adds an α-vector corresponding to the value of a Nash equilibrium strategy of the first player in $[H\underline{v}](b)$ (denoted $L\Gamma(b)$) computed from duals of the linear program (Eqs. (5)-(6)). The value of such strategy is linear and $(U-L)$-Lipschitz (Lemma 2), hence the expansion of Γ by $L\Gamma(b)$ preserves $(U-L)$-Lipschitz continuity of $\underline{v}$.

The upper bound function $\overline{v}$ is represented by a set of points Υ. Update of upper bound adds one point, $U\Upsilon(b) = b \to [H\overline{v}](b)$, that corresponds to the evaluation of the value backup at belief b. We cannot use the linear program outlined in Eqs. (5)-(6) to compute $[H\overline{v}](b)$ directly since the function $\overline{v}$ is not represented using α-vectors. We, therefore, use a transformation presented in (Horak and Bosansky 2016) which performs projections of beliefs to the lower envelope of $\overline{v}$ while preserving linearity of the constraints.

Adding a point to Υ can break the $(U-L)$-Lipschitz continuity of $\overline{v}$. We can fix this by constructing a piecewise linear approximation of a lower $(U-L)$-Lipschitz envelope:

$$\overline{v}(b) \coloneqq \inf_{b' \in \Upsilon} \left\{ \overline{v}(b') + (U-L) \cdot \|b - b'\|_2 \right\} . \tag{7}$$

The resulting function is $c(U-L)$-Lipschitz when c depends on the accuracy of the approximation and can be arbitrarily close to 1.

Forward Exploration

The value backup operator H expresses the value in belief b in terms of values of subsequent beliefs $b^{a,o}_{\pi_2}$. When applied to value functions $\hat{v}$, it also propagates the approximation error. In order to minimize the gap in the initial belief b^0, we need to achieve sufficient accuracy also in beliefs encountered at a later *time*.

The forward exploration simulates a play between the players while assuming that the second player follows a strategy obtained from the application of H on the lower bound $\underline{v}$ (i.e. she is overly optimistic with her strategy). When a belief b is encountered at time t (we term such a pair (b, t) a *timed belief*) and its approximation $\hat{v}(b)$ is not sufficiently accurate, we say that it has positive *excess gap*.

Definition 2 (Excess gap)**.** *Let ϵ be the desired precision and $R > 0$ be a neighborhood parameter. Let*

$$\rho(t) = \epsilon\gamma^{-t} - \sum_{i=1}^{t} 2R(U-L)\gamma^{-i} . \tag{8}$$

We define the excess gap of a timed belief (b, t) as

$$\mathrm{excess}(b, t) = \mathrm{gap}(\hat{v}(b)) - \rho(t) . \tag{9}$$

Later we show that if all subsequent timed beliefs $(b^{a,o}_{\pi_2}, t+1)$ have negative excess gap, a point-based update at (b, t) makes the excess gap $\mathrm{excess}(b, t)$ negative as well (in fact, $\mathrm{excess}(b, t) \leq -2R(U-L)$; we then term (b, t) as *closed*). If this does not hold for the belief (b, t) currently explored, the forward exploration process selects one of the subsequent beliefs $(b^{a,o}_{\pi_2}, t+1)$ with a positive excess gap for further exploration and the process is repeated with the timed belief $(b^{a,o}_{\pi_2}, t+1)$. If all subsequent beliefs have a negative excess gap, the forward exploration process terminates.

The termination is guaranteed if the neighborhood parameter R is chosen so that the sequence $\rho(t)$ is monotonically increasing in t and unbounded.

Forward Exploration Heuristic A positive excess gap of a belief contributes to the approximation error in the initial belief. If there are multiple subsequent timed beliefs $(b^{a,o}_{\pi_2}, t+1)$ with a positive excess gap, we select the one with the highest *weighted* excess gap which is similar to the weighted excess heuristic used in (Smith and Simmons 2004). The excess gap is weighted by both the observation probability *and* the probability that the first player plays a given action when using the strategy obtained from the upper bound value function $\overline{v}$ (i.e. according to the strategy π_1 from the game $[H\overline{v}](b)$). The action observation pair (a, o) selected in timed belief (b, t) for the further exploration maximizes $\pi_1(a) \cdot \Pr[o|a, \pi_2] \cdot \mathrm{excess}(b^{a,o}_{\pi_2}, t+1)$.

Convergence of the Algorithm

The goal of the HSVI algorithm is to make the excess gap negative in all reachable timed beliefs and thus sufficiently decrease the gap in the initial belief. Contrary to POMDPs, reachable beliefs in POSGs are influenced by the strategy of the second player – she can change her strategy to reach a belief (b', t) with a positive excess gap instead of a closed belief (b, t), while b' stays arbitrarily close to b.

We avoid this by ensuring that if (b', t) with a positive excess gap is reached by the forward exploration, it lies sufficiently far from all previously closed beliefs at time t – the minimum distance between the beliefs being controlled by the neighborhood parameter $R > 0$ from the definition of the excess gap. Unlike in POMDPs, our modified definition of the excess gap ensures that not only a closed belief itself gets a negative excess gap: all beliefs within its R-neighborhood get a negative excess gap as well (Lemma 4). The convergence of the algorithm follows since there is only a finite number of such R-separated belief points.

Lemma 4. *Let (b, t) be a timed belief and π_2 be the optimal strategy of the second player in $[H\underline{v}](b)$. If $\mathrm{excess}(b^{a,o}_{\pi_2}, t+1) \leq 0$ for all action-observation pairs (a, o) of the first player, then after performing a point-based update at b it holds that (i) $\mathrm{excess}(b, t) \leq -2R(U-L)$ and (ii) all belief points b' in the R-neighborhood of b (i.e. $\|b - b'\|_2 \leq R$) have a negative excess gap $\mathrm{excess}(b', t)$.*

The first part of the lemma follows from Lemma 3, the latter follows from $2(U-L)$-Lipschitz continuity of difference of $(U-L)$-Lipschitz functions $\underline{v}$ and $\overline{v}$.

Definition 3. *Let t be time. The set of all beliefs with negative excess gap at time t is denoted Ψ_t;*

$$\Psi_t = \{b \in \Delta(S) \mid \mathrm{gap}(\hat{v}(b)) \leq \rho(t)\} \ . \tag{10}$$

Theorem 3. *HSVI algorithm converges to the precision ϵ.*

Proof. In each iteration, the algorithm performs a forward exploration until it encounters a timed belief (b, t) such that all subsequent timed beliefs $(b^{a,o}_{\pi_f}, t+1)$ have a negative excess gap. Since $\mathrm{gap}(b^{a,o}_{\pi_f})$ is bounded by $U-L$, this happens after at most $t_{\max}$ steps, where

$$t_{\max} = \left\lceil \log_{1/\gamma} \left(\frac{U-L}{\epsilon} \cdot \left[1 + 2R \frac{1-\gamma^t}{\gamma^t(1-\gamma)} \right] \right) \right\rceil \ . \tag{11}$$

When the terminal timed belief (b, t) is reached, then $b \notin \Psi_t$ and all subsequent timed beliefs have negative excess gap. After performing the point-based update at (b, t), the excess gap of (b, t), as well as of all timed beliefs in the R-neighborhood of (b, t), is negative (Lemma 4) and Ψ_t is expanded. We show that the expansion of the sets $\Psi_{t'}$ guarantees that eventually $\Psi_{t'} = \Delta(\mathcal{S})$ for all times $t' \leq t_{\max}$, unless the desired precision ϵ is achieved beforehand.

The distance of b from the nearest belief b' in Ψ_t previously closed by the algorithm is at least R, since all points in the R-neighborhood of b' have a negative excess gap and thus are in Ψ_t. In each iteration, Ψ_t is expanded by at least one belief and (at least) its R-neighborhood.

The number of such expansions of timed beliefs is finite. In fact, the problem of finding maximum set of R-separated beliefs can be seen as a *hypersphere packing* (a higher dimensional version of the sphere packing (Hales 2011)) filling the belief simplex using non-overlapping hyperspheres of radius $R/2$, since the hyperspheres do not overlap exactly when the distance between their centers is at least R. When no hypersphere can be further inserted, it means that we cannot find any belief with a positive excess gap, hence we reached the desired precision in the whole belief space. □

Experiments

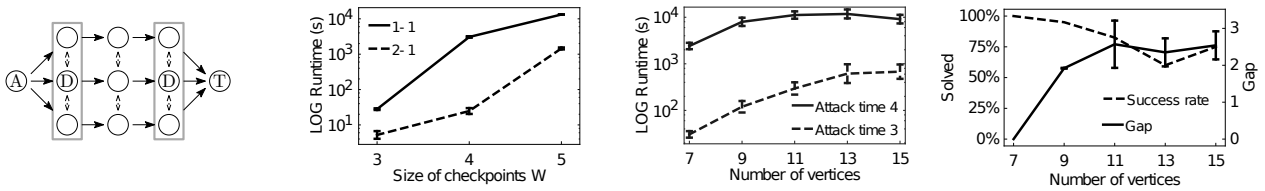
We demonstrate application possibilities and scalability of our algorithm on three types of games: pursuit-evasion games (e.g., evaluated in (Horak and Bosansky 2016)), intrusion search games (e.g., see (Bosansky et al. 2014)), and patrolling games with a discount factor (e.g., see (Vorobeychik et al. 2014)). Each player is assigned a team of units (either one or multiple units) located in vertices of a graph and he or she controls their movement on the graph. A move consists of moving the units simultaneously to vertices adjacent to their current positions, or they can wait.

The utilities are scaled so that the values of the games lies in the interval $[0, 100]$ (or $[-100, 0]$, respectively). Unless stated otherwise the discount factor is $\gamma = 0.95$ and we ran the algorithm until $\mathrm{gap}(\hat{v}(b^0)) \leq 1$.

Algorithm Settings

We initialize the value functions by solving the perfect-information refinement of the game (for $\overline{v}$) and as a best response to a uniform strategy of player 1 (for $\underline{v}$). We use standard value iteration for stochastic games, or MDPs, respectively, and terminate the initialization when either change in valuations between iterations is lower than 0.025, or $20s$ time limit has expired. The initialization time is included in the computation times of the algorithms.

Similarly to (Smith and Simmons 2004), we adjust ϵ in each iteration using formula $\epsilon = 0.25 + \eta(\mathrm{gap}(\hat{v}(b^0)) - 0.25)$ with $\eta = 0.9$. We set the neighborhood parameter R to the largest value satisfying $\rho(t) \geq 0.25\gamma^{-t}$ for all $t \leq t_{\max}$ from the proof of Theorem 3.



(a) (b) (c) (d)

Figure 1: (a) Intrusion-search game: $W = 3$, configuration 1-1: A denotes initial position of the attacker, D initial positions of defender's units, T is attacker's target (b) Intrusion-search games with 2 zones, each with W vertices: Time to reach $\mathrm{gap}(\hat{v}(b^0)) \leq 1$ (c) Patrolling games played on graphs generated from $ER(0.25)$: Time to reach $\mathrm{gap}(\hat{v}(b^0)) \leq 1$ (only successfully solved instances within 10 hours) (d) Patrolling games played on graphs generated from $ER(0.25)$: Percentage of successfully solved instances with $t_\times = 4$ and the gap on failed instances after 10 hours

Finally, we remove dominated points and vectors from sets Γ and Υ whenever their size grows by 20% to reduce the size of the linear programs. Again, this is similar to POMDPs (Smith and Simmons 2004).

Pursuit-Evasion Games (PEGs)

A team of centrally controlled pursuers aims to locate and capture the evader, and receive the utility of +100; the evader aims for the opposite. We consider $3 \times N$ grid graphs (we vary the number of columns N), two pursuing units start in top left positions, the evader starts in bottom right corner. Our algorithm achieves similar scalability as the existing algorithm designed specifically for one-sided PEGs (Horak and Bosansky 2016) and displays exponential dependence of the runtime on the width of the grid N. The game with $N = 3$ was solved in 9s on average, the game with $N = 6$ took 3.5 hours to be solved to the gap 1. A graph depicting the dependence of the runtime on N can be found in the full version of the paper. Sizes of the games range from 143 states and 2671 transitions to 1299 states and 34807 transitions.

Search Games

In search games that model intrusion, the defender patrols checkpoint zones (see Figure 1a, the zones are marked with box). The attacker aims to cross the graph, while not being captured by the defender. If the attacker crosses the graph unharmed, the defender receives a utility of -100. Whenever the attacker enters a node, she leaves a trace and the defender can later detect it. She can either wait for one move to conceal her presence (and clean up the trace), or move further.

We consider games with 2 checkpoint zones with varying sizes W (i.e. width of the graph) and 2 configurations of the defending forces – with one defender in each of the checkpoint zones (denoted 1-1), and 2 defenders in the first zone while just 1 defender being in the second one (denoted 2-1). The results are shown in Figure 1b (with 5 runs for each parameterization, the confidence intervals mark the standard error in our graphs). The largest game ($W = 5$ and 2 defenders in the first zone) has 4656 states and 121239 transitions and can be solved within 27 minutes. This case highlights that our algorithm can solve even large games. However, a much smaller game with the configuration 1-1 (964 states and 9633 transitions) is more challenging, since the coordination problem with just 1 defender in the first zone is harder, and is solved within 3.5 hours.

Patrolling Games

In patrolling games (Basilico, Gatti, and Amigoni 2009; Vorobeychik et al. 2014) the *patroller* patrols vertices of a graph by moving over the graph. The attacker decides the vertex she will attack and the time she will do so. The patroller does not know if an attack has started, however, he has a limited time (termed *attack time*, denoted $t_\times$) to reach the vertex under the attack. Otherwise, the vertex is successfully attacked and the patroller receives a negative reward associated to that vertex.

Following the setting in (Vorobeychik et al. 2014), we focus on graphs generated from Erdos-Renyi model (Newman 2010) with parameter $p = 0.25$ (denoted $ER(0.25)$) with attack times 3 and 4 and number of vertices $|\mathcal{V}|$ ranging from 7 to 15. Each instance with attack time $t_\times = 3$ was solved by our algorithm in less than 12 minutes (see Figure 1c). This result generally outperforms the computation times reported for tailored algorithm for solving discounted patrolling games (Vorobeychik et al. 2014). For attack time $t_\times = 4$, however, some number of instances failed to reach the precision $\mathrm{gap}(\hat{v}(b^0)) \leq 1$ within the time limit of 10 hours. For the most difficult setting, $|\mathcal{V}| = 13$, the algorithm reached desired precision in 60% of instances (see Figure 1d). For unsolved instances, mean $\mathrm{gap}(\hat{v}(b^0))$ after the cutoff after 10 hours is however reasonably small (also depicted in Figure 1d, see the solid line and right y-axes). The results include games with up to 856 states and 6409 transitions.

Since our algorithm is domain-independent, it can also solve variants of patrolling games with alarms (Basilico, Nittis, and Gatti 2016), including all types of imprecise signals (false positives, false negatives). The results for this setting can be found in the full version of the paper.

Conclusions

We focus on two-player zero-sum partially observable stochastic games (POSGs) with discounted rewards and one-sided observability where the second player has perfect information about the game. We propose the first approximate

algorithm that generalizes the ideas behind point-based algorithms designed for Partially Observable Markov Decision Processes (POMDPs) and transfers these techniques to POSGs. We provide theoretical guarantees as well as an experimental evaluation of our algorithm on three fundamentally different games.

Our work opens a completely new direction in research of POSGs and sequential decision making and allows to design new scalable algorithm for one-sided POSGs that can be applied in many real-world scenarios. While the current scalability of our algorithm is limited, it is the first step in a new direction of research. Many heuristics proven useful for POMDPs can be translated and evaluated in this new setting, and can further improve the scalability and applicability of our results.

Acknowledgments

This research was supported by the Czech Science Foundation (grant no. 15-23235S) and by the Grant Agency of the Czech Technical University in Prague, grant No. SGS16/235/OHK3/3T/13.

References

Basilico, N.; Gatti, N.; and Amigoni, F. 2009. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 57–64.

Basilico, N.; Nittis, G. D.; and Gatti, N. 2016. A Security Game Combining Patrolling and Alarm-Triggered Responses Under Spatial and Detection Uncertainties. In *Proceedings of the 30th Conference on Artificial Intelligence (AAAI)*, 397–403.

Bosansky, B.; Kiekintveld, C.; Lisy, V.; and Pechoucek, M. 2014. An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information. *Journal of Artificial Intelligence Research* 51:829–866.

Chung, T. H.; Hollinger, G. A.; and Isler, V. 2011. Search and pursuit-evasion in mobile robotics. *Autonomous robots* 31(4):299–316.

Ciesielski, K. 2007. On Stefan Banach and some of his results. *Banach Journal of Mathematical Analysis* 1(1):1–10.

Fang, F.; Nguyen, T. H.; Pickles, R.; Lam, W. Y.; Clements, G. R.; An, B.; Singh, A.; Tambe, M.; and Lemieux, A. 2016. Deploying PAWS: Field optimization of the protection assistant for wildlife security. In *Proceedings of the 30th Conference on Artificial Intelligence (AAAI)*, 3966–3973.

Fang, F.; Stone, P.; and Tambe, M. 2015. When Security Games Go Green: Designing Defender Strategies to Prevent Poaching and Illegal Fishing. In *In Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2589–2595.

Hales, T. C. 2011. Historical overview of the Kepler conjecture. In *The Kepler Conjecture*. Springer. 65–82.

Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004. Dynamic Programming for Partially Observable Stochastic Games. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI)*, 709–715.

Horak, K., and Bosansky, B. 2016. A Point-Based Approximate Algorithm for One-Sided Partially Observable Pursuit-Evasion Games. In *Proceedings of the Conference on Decision and Game Theory for Security*, 435–454.

Horak, K., and Bosansky, B. 2017. Dynamic Programming for One-Sided Partially Observable Pursuit-Evasion Games. In *Proceeding of the International Conference on Agents and Artificial Intelligence (ICAART) — to appear*.

Kiekintveld, C.; Jain, M.; Tsai, J.; Pita, J.; Ordóñez, F.; and Tambe, M. 2009. Computing optimal randomized resource allocations for massive security games. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 689–696.

Madani, O.; Hanks, S.; and Condon, A. 1999. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *Proceedings of the 16th National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, 541–548.

Newman, M. 2010. *Networks: an introduction*. Oxford university press.

Nikaido, H. 1954. On von Neumann's minimax theorem. *Pacific J. Math.* 4(1):65–72.

Pita, J.; Jain, M.; Marecki, J.; Ordóñez, F.; Portway, C.; Tambe, M.; Western, C.; Paruchuri, P.; and Kraus, S. 2008. Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles International Airport. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 125–132.

Shieh, E.; An, B.; Yang, R.; Tambe, M.; Baldwin, C.; Direnzo, J.; Meyer, G.; Baldwin, C. W.; Maule, B. J.; and Meyer, G. R. 2012. PROTECT : A Deployed Game Theoretic System to Protect the Ports of the United States. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 13–20.

Smith, T., and Simmons, R. 2004. Heuristic search value iteration for POMDPs. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, 520–527. AUAI Press.

Vanek, O.; Yin, Z.; Jain, M.; Bosansky, B.; Tambe, M.; and Pechoucek, M. 2012. Game-theoretic Resource Allocation for Malicious Packet Detection in Computer Networks. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 902–915.

von Neumann, J. 1928. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen* 100(1):295–320.

Vorobeychik, Y.; An, B.; Tambe, M.; and Singh, S. P. 2014. Computing Solutions in Infinite-Horizon Discounted Adversarial Patrolling Games. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 314–322.

Appendix E

Solving Partially Observable Stochastic Games with Public Observations

Solving Partially Observable Stochastic Games with Public Observations

Karel Horák and **Branislav Bošanský**
Department of Computer Science, Faculty of Electrical Engineering
Czech Technical University in Prague
{horak,bosansky}@agents.fel.cvut.cz

Abstract

In many real-world problems, there is a dynamic interaction between competitive agents. Partially observable stochastic games (POSGs) are among the most general formal models that capture such dynamic scenarios. The model captures stochastic events, partial information of players about the environment, and the scenario does not have a fixed horizon. Solving POSGs in the most general setting is intractable. Therefore, the research has been focused on subclasses of POSGs that have a value of the game and admit designing (approximate) optimal algorithms. We propose such a subclass for two-player zero-sum games with discounted-sum objective function—POSGs with *public observations* (PO-POSGs)—where each player is able to reconstruct beliefs of the other player over the unobserved states. Our results include: (1) theoretical analysis of PO-POSGs and their value functions showing convexity (concavity) in beliefs of maximizing (minimizing) player, (2) a novel algorithm for approximating the value of the game, and (3) a practical demonstration of scalability of our algorithm. Experimental results show that our algorithm can closely approximate the value of non-trivial games with hundreds of states.

Introduction

Game theory describes the optimal behavior of rational agents and is recently widely applied to solving security problems. Game-theoretic strategies are used to protect critical infrastructures (Pita et al. 2008; Kiekintveld et al. 2009; Shieh et al. 2012), secure computer networks (Vanek et al. 2012; Nguyen, Wellman, and Singh 2017; Durkota et al. 2017) or wildlife (Fang, Stone, and Tambe 2015; Fang et al. 2016). In many real-world situations, there is a dynamic strategic interaction between the players, and the players do not have perfect information about the environment. Moreover, a pre-defined horizon (number of moves in the scenario) is only rarely given in practice and thus these games belong to the class of partially observable stochastic games (POSGs). Examples include patrolling games (Basilico, Gatti, and Amigoni 2009; Vorobeychik et al. 2014; Basilico, Nittis, and Gatti 2016; Brazdil, Kucera, and Rehak 2018), where a defender protects a set of targets against an attacker, pursuit-evasion (Chung, Hollinger, and Isler 2011), or search games, where a defender is trying to find and capture an attacker.

We focus on two-player zero-sum POSGs, and even with this restriction it is intractable to compute optimal strategies in the most general case. Since the players do not perfectly observe the environment, each player has a belief over possible states of the environment. However, the reward the player receives for choosing some action(s) also depends on the action of the other player who decides based on their belief. Therefore, player 1 has to consider also the belief of player 2 and belief that player 2 has about player 1, and so on. This reasoning is called *nested beliefs* (e.g., in (MacDermed 2013)) and it causes a doubly-exponential number of histories to consider for each agent.

However, real-world security scenarios require partial observability without a strictly defined horizon. Therefore, one can restrict to subclasses of POSGs where games *have a value* (i.e., the value of the game exists) and (approximate) optimal algorithms can be designed. Examples of such works are stochastic games in which both the players' actions and observations are public (Ghosh, McDonald, and Sinha 2004), games in which the support of private/public observations does not depend on states and actions (Cole and Kocherlakota 2001), or games where only one player has imperfect information (also called One-Sided) (Chatterjee and Doyen 2014; Basu and Stettner 2015; Horak, Bosansky, and Pechoucek 2017). The practical motivation for such subclasses is to compute robust strategies for the defender assuming the worst case scenario where the attacker has additional information (Vorobeychik et al. 2014; Horak and Bosansky 2016).

In this paper, we propose a new subclass of POSGs in which we avoid the problem of nested beliefs, called *POSGs with public observations* (PO-POSGs), that generalizes previous subclasses. In this model, each player is able to exactly reconstruct the belief of the opponent. The key characteristics are: (1) the state space is factored – each player observes his private state, but the state of the other player is not observed; (2) each observation that modifies belief about the state of the other player is public (both players are aware of this observation); (3) the true state of the player is observed privately by that player. We restrict to two-players zero-sum games with discounted future rewards and give the following contributions: (1) We

show that games in this class have a value; (2) We show that the value function of PO-POSGs is convex in the belief of the maximizing player and concave in the belief of the minimizing player; (3) We introduce a novel algorithm based on Heuristic Value Iteration Search (HSVI) for One-Sided POSGs (Horak, Bosansky, and Pechoucek 2017; Smith and Simmons 2004) and show that this algorithm converges to the (approximate) optimal values.

We demonstrate our algorithm on two different domains – a patrolling game, where the attacker has imprecise information about the position of the defender (Basilico et al. 2009), and a lasertag game based on a single-player variant (Pineau, Gordon, and Thrun 2003). The results show that, for the first time, there is a practical domain-independent algorithm able to closely approximate optimal values of non-trivial infinite-horizon POSGs with hundreds of states where both players have partial information about the environment.

Related Work

The notion of public actions and observations is common in dynamic games. For finite horizon games, the concept of public states and publicly observed actions creates separated subgames that allow designing limited-lookahead algorithms for imperfect information games (Moravcik et al. 2017; Brown, Sandholm, and Amos 2018).

In games with an infinite horizon, the problem with nested beliefs prevents one from designing an (approximate) optimal algorithm for fully general settings. Nested beliefs can be tackled directly with histories – one of few such approaches is a bottom-up dynamic programming for constructing relevant finite-horizon policy trees for individual players while pruning-out dominated strategies (Hansen, Bernstein, and Zilberstein 2004; Kumar and Zilberstein 2009). However, due to the explicit dependence on the histories, the scalability in the horizon is very limited.

A more common approach is to focus on a subclass of POSGs. In (Ghosh, McDonald, and Sinha 2004), zero-sum POSGs with public actions and observations are considered. The authors show that the value of the game exists and present an algorithm that exploits the transformation of such a model into a game with complete information. In our approach, we assume only public observations (i.e., actions are private to the players). Moreover, we factor the state space according to the players (i.e., each player has his own state that is perfectly observable to this player, and the state of the opponent is unknown). Similar factorization of the state space is used also in (Cole and Kocherlakota 2001), however, in this work the authors assume that the support of observations cannot change due to states or actions of the players. We remove this assumption and actions and observations can be generated in states arbitrarily. Alternatively, some works assume that only one player has partial information (Chatterjee and Doyen 2014; Basu and Stettner 2015; Horak, Bosansky, and Pechoucek 2017). Again, we remove this assumption and allow both players to have partial information about the states of the other player. While our algorithm is based on the algorithm for the one-sided case, we provide significant generalizations of the previous work, especially in the representation of value function, definition and algorithms for computing value-backup operator.

Finally, (MacDermed 2013) gives a transformation of POSGs to Markov Games of Incomplete Information (MaGIIs) as a more efficient representation if observations have Markov property. While the examples of games that we consider satisfy this property, the author demonstrates the benefits of this representation for the common-payoff case of Dec-POMDPs only. We solve zero-sum games, which is a more complex problem and since there is no apparent way to exploit MaGIIs, we use a more common formalism.

POSGs with Public Observations

Definition 1. A partially observable stochastic game with public observations (PO-POSG) is a two-player zero-sum game (played by players $i \in \{1, 2\}$) represented by a tuple $\langle S_i, A_i, O_i, Z_i, T_i, R, b_i^{(0)}, \gamma \rangle$, where

- S_i is a finite set of (private) states of player i
- A_i is a finite set of actions available to player i
- O_i is a finite set of observations for player i
- $Z_i(o_i | s_{-i} a_{-i})$ is the probability to generate observation o_i for player i, given that his opponent[1] $-i$ played an action a_{-i} in state s_{-i}
- $T_i(s_i' | s_i a_i o_i o_{-i})$ is the probability to transition from s_i to s_{-i} when player i played a_i and observations o_i and o_{-i} have been generated
- $R(s_1 s_2 a_1 a_2)$ is the reward of player 1 when actions (a_1, a_2) have been jointly played in the joint state (s_1, s_2)
- $b_i^{(0)} \in \Delta(S_{-i})$ is the initial belief of player i over states S_{-i} of his opponent
- $\gamma \in [0, 1)$ is the discount factor.

A play in a PO-POSG proceeds as follows. First, the initial joint state $(s_1^{(1)}, s_2^{(1)})$ is drawn with probability $b_2^{(0)}(s_1^{(1)}) \cdot b_1^{(0)}(s_2^{(1)})$. Then, in each round t, players observe their current private state (player i observes $s_i^{(t)}$, but not $s_{-i}^{(t)}$ of his opponent). Based on this information (and history), each player i chooses an action $a_i^{(t)} \in A_i$ independently of the decision of his opponent $-i$. As a consequence of this choice, player 1 receives reward $r^{(t)} = R(s_1^{(t)} s_2^{(t)} a_1^{(t)} a_2^{(t)})$ and player 2 receives negated reward $-R(s_1^{(t)} s_2^{(t)} a_1^{(t)} a_2^{(t)})$. Furthermore, an observation $o_i^{(t)}$ for each player is generated and made publicly known to both players with probability $Z_i(o_i^{(t)} | s_{-i}^{(t)} a_{-i}^{(t)})$ and a new private state $s_i^{(t+1)}$ of each player is drawn from $T_i(\cdot | s_i^{(t)} a_i^{(t)} o_i^{(t)} o_{-i}^{(t)})$. We consider discounted setting and the utility of player 1 is thus $\sum_{t=1}^{\infty} \gamma^{t-1} r^{(t)}$ (and negative value for the opponent as the game is zero-sum).

Definition 2. The history of player i up to time T is a sequence $\{s_i^{(t)} a_i^{(t)} o_i^{(t)} o_{-i}^{(t)}\}_{t=1}^{T} s_i^{T+1}$.

Definition 3. The (history-dependent) strategy of player i is a mapping $\sigma_i : (S_i A_i O_i O_{-i})^* S_i \to \Delta(A_i)$ from histories of player i to randomized decisions.

[1] As it is commonly used, $-i$ denotes opponent of player i.

Observe that in PO-POSGs, the player i updates his belief solely on the information about the public observations (o_i, o_{-i}) and the knowledge of the strategy used by the adversary for the *current stage* only—we denote such one-stage strategy by π_{-i} as opposed to the full strategy σ_{-i}. Assuming that the adversary $-i$ chooses an action a_{-i} in a state s_{-i} with probability $\pi_{-i}(a_{-i}|s_{-i})$ in the current stage of the game (given the information available to him) and that observations (o_i, o_{-i}) have been generated, player i can update his belief $b_i \in \Delta(S_{-i})$ to a belief $\tau_{\pi_{-i}}(b_i o_i o_{-i})$ where the updated probability of being in a state s'_{-i} is

$$\tau_{\pi_{-i}}(b_i o_i o_{-i})(s'_{-i}) = \frac{1}{\Pr_{\pi_{-i}}[o_i]} \sum_{s_{-i}, a_{-i}} b_i(s_{-i}) \cdot \tag{1}$$
$$\cdot\, \pi_{-i}(a_{-i}|s_{-i}) \cdot Z(o_i|s_{-i}o_{-i}) \cdot T(s'_{-i}|s_{-i}a_{-i}o_{-i}o_i)\ .$$

Since both the strategy π_{-i} and the public observations (o_i, o_{-i}) are known to player $-i$ as well, she can reconstruct $\tau_{\pi_{-i}}(b_i o_i o_{-i})$, and the belief update is essentially public.

Value of PO-POSGs

We now establish the value function V^* to capture the utility of playing optimal strategies in a PO-POSG (i.e., the value of the game) based on the beliefs the players have.

Definition 4. The optimal *value function* of a PO-POSG is a function $V^* : \Delta(S_2) \times \Delta(S_1) \to \mathbb{R}$ mapping each possible initial belief (b_1, b_2) of the game to the expected utility of player 1 in the equilibrium (i.e., the value of the game).

Since any finite-horizon approximation of a PO-POSGs has a value (von Neumann 1928) and discounted-sum utilities are considered, the value of a PO-POSG is well defined.

Theorem 1. *The value of the game exists in PO-POSGs.*

Proof (sketch). Denote v_T the value of a finite approximation with horizon $T \in \mathbb{N}$. The approximation considers all rewards from the first T steps. The equilibrial strategies in v_T can thus only be inferior in the full, infinite-horizon game, when rewards after T steps are considered. Hence

$$v_T + \textstyle\sum_{t=T+1}^{\infty} \gamma^{t-1} \min R(\cdot) \le V^*[b_1^{(0)}, b_2^{(0)}] \le \\ \le v_T + \textstyle\sum_{t=T+1}^{\infty} \gamma^{t-1} \max R(\cdot)\ . \tag{2}$$

As $T \to \infty$, the bounds converge to $V^*[b_1^{(0)}, b_2^{(0)}]$. □

Contrary to previous works, the optimal value function V^* is neither convex nor concave. We show, however, that due to the factorization of the state space, V^* is convex in the belief b_1 of the maximizing player 1 and concave in the belief b_2 of the minimizing player 2.

Lemma 1. *Let σ_i be a strategy of player i, and b_{-i} be the belief of the adversary. Then the expected utility $V^{\sigma_i|b_{-i}} : \Delta(S_{-i}) \to \mathbb{R}$ of playing σ_i against the best-responding opponent $-i$ parametrized by the belief of player i is linear and $(U-L)/\sqrt{2}$-Lipschitz continuous.*

Proof (sketch). Player $-i$ knows σ_i as well as his true state s_{-i}, and his only uncertainty is about the state s_i (the probability of which is $b_{-i}(s_i)$). It is thus possible to focus on the best response for each state s_{-i} separately. Let us denote the expected utility of playing the best response against σ_i starting from s_{-i} (when $s_i \sim b_{-i}$) by $\xi(s_{-i})$. Since the strategy σ_i is fixed (and thus does not depend on b_i), the expected utility of playing σ_i against the best response of the adversary is the expectation over the values $\xi(s_{-i})$, $V^{\sigma_i|b_{-i}}(b_i) = \sum_{s_{-i}} b_i(s_{-i}) \cdot \xi(s_{-i})$, and thus the value $V^{\sigma_i|b_{-i}}$ is linear in b_i. Moreover, observe that

$$L = \frac{\min R(\cdot)}{1-\gamma} \le V^{\sigma_i|b_{-i}}(b_i) \le \frac{\max R(\cdot)}{1-\gamma} = U \tag{3}$$

which makes V^{σ_i} be $(U-L)/\sqrt{2}$-Lipschitz continuous. □

Theorem 2. *The value function V^* is convex in b_1 and concave in b_2. Moreover, it is $(U-L)$-Lipschitz continuous.*

Proof. For a fixed b_2, player 1 chooses a strategy that maximizes the utility, hence

$$V^*[b_1, b_2] = \max_{\sigma_1} V^{\sigma_1|b_2}(b_1)\ . \tag{4}$$

As all $V^{\sigma_1|b_2}$ are linear, V^* is convex in b_1. Vice versa, for given fixed b_1, player 2 chooses a minimizing strategy,

$$V^*[b_1, b_2] = \min_{\sigma_2} V^{\sigma_2|b_1}(b_2)\ , \tag{5}$$

and V^* is concave in b_2. Since V^* is a pointwise maximum/minimum (Equations (4) and (5)) from $(U-L)/\sqrt{2}$-Lipschitz continuous functions $V^{\sigma_i|b_{-i}}$, V^* is $(U-L)/\sqrt{2}$-Lipschitz continuous in the dimension of b_1, as well as b_2. Combining the Lipschitz constants in these two dimensions results in $\sqrt{2} \cdot (U-L)/\sqrt{2}$-Lipschitz continuity of V^*. □

Properties of Nash Equilibrium of PO-POSGs

Consider a Nash equilibrium strategy profile (σ_1, σ_2) and let $\pi_i(\cdot|s_i) = \sigma_i(s_i)$. If observations (o_1, o_2) are generated, the probability of transitioning to the joint state (s'_1, s'_2) is $\tau_{\pi_1}(b_2 o_2 o_1)(s'_1) \cdot \tau_{\pi_2}(b_1 o_1 o_2)(s'_2)$. Since the dynamics of the game is Markovian, the equilibrial strategies aim to optimize the payoff in the subgame after (o_1, o_2) is seen by the players—i.e., the expected discounted sum of the rewards starting from the joint belief $(\tau_{\pi_2}(b_1 o_1 o_2)(s'_2), \tau_{\pi_1}(b_2 o_2 o_1)(s'_1))$. Since in the equilibrium both players know this distribution, this expectation is equal to $V[\tau_{\pi_2}(b_1 o_1 o_2), \tau_{\pi_1}(b_2 o_2 o_1)]$ (since both players have strategies that guarantee this expected long-term reward when starting from the given joint belief).

This fact makes it possible to express the value of the equilibrial strategy profile (σ_1, σ_2) in terms of the immediate reward (direct consequences of the decisions in the first stage of the game) $R_{\pi_1\pi_2}$,

$$R_{\pi_1\pi_2} = \sum_{s_1, s_2, a_1, a_2} b_2(s_1) b_1(s_2) \pi_1(a_1|s_1) \pi_2(a_2|s_2) R(s_1 s_2 a_1 a_2) \tag{6}$$

and the values $V[\tau_{\pi_2}(b_1 o_1 o_2), \tau_{\pi_1}(b_2 o_2 o_1)]$ of the subgames:

$$R_{\pi_1\pi_2} + \tag{7}$$
$$+ \gamma \sum_{o_1, o_2} \Pr_{\pi_1\pi_2}[o_1 o_2] \cdot V[\tau_{\pi_2}(b_1 o_1 o_2), \tau_{\pi_1}(b_2 o_2 o_1)]\ .$$

Relaxing the assumption of known equilibrial strategies and performing the maximin optimization over Equation (7) gives us the value of the game starting in the joint belief (b_1, b_2) as a fixpoint equation over value functions

$$V^*[b_1,b_2] = HV^*[b_1,b_2] = \max_{\pi_1}\min_{\pi_2}\Big[R_{\pi_1\pi_2} + \tag{8}$$
$$+\gamma\sum_{o_1,o_2}\Pr_{\pi_1\pi_2}[o_1o_2]\cdot V[\tau_{\pi_2}(b_1o_1o_2),\tau_{\pi_1}(b_2o_2o_1)]\Big]\,.$$

Moreover, since $\gamma < 1$, the operator H defined over value functions $V : \Delta(S_2) \times \Delta(S_1) \to \mathbb{R}$ is a contraction. The Equation (8) can thus be used to approximate V^* iteratively.

Algorithm

Evaluating the dynamic programming operator H directly (as defined in Equation (8)) is impossible since the set of all joint beliefs is infinite. To design a practical algorithm, we need to establish an approximation scheme for V^* that we describe in this section first. Then we provide mathematical programs for computing HV when this approximation scheme is used. Finally, we state our algorithm to obtain ϵ-approximation of V^* in PO-POSGs.

Approximating V^*

In POMDPs (or one-sided POSGs), the value function V^* is commonly represented either as a point-wise maximum over a set of linear functions (termed α-vectors) or by considering a convex hull of a set of points. Both of these approaches leverage that the value function V^* is convex, which is not the case for PO-POSGs. In this section, we present a way to form a lower bound approximation $\underline{V}$ of a convex-concave function V^* inspired by both of the approaches mentioned above (the construction of the upper bound $\overline{V}$ is analogous).

To represent the value of $\underline{V}$ in the dimension of S_2, we use an extended notion of α-vectors, termed $\alpha\beta$-vectors. In PO-POSGs, the linear value $V^{\sigma_1|b_2}$ of a strategy depends on the belief b_2 of the adversary (see Lemma 1). Hence, also our $\alpha\beta$-vectors depend on the belief of the adversary denoted β. An $\alpha\beta$-vector consists of two components (see the thick line in Figure 1). First, there is a linear function $\alpha : \Delta(S_2) \to \mathbb{R}$ representing the value of the $\alpha\beta$-vector in the $\Delta(S_2)$ dimension. Second, there is a belief of the adversary $\beta \in \Delta(S_2)$ which informally positions the $\alpha\beta$-vector in the $\Delta(S_1)$ dimension. As a simplification, an $\alpha\beta$-vector can be seen as a value $V^{\sigma_1|b_2}$ of a strategy σ_1 in belief b_2, where $\alpha = V^{\sigma_1|b_2}$ and $\beta = b_2$. However, an $\alpha\beta$-vector of player 1 is an arbitrary function that lower bounds V^* in general.

Definition 5. An $\alpha\beta$-vector of player i is a tuple consisting of a linear function $\alpha : \Delta(S_{-i}) \to \mathbb{R}$ and the belief of the adversary $\beta \in \Delta(S_i)$ satisfying

$$\alpha(b_1) \le V^*[b_1,\beta] \qquad \text{, or} \qquad \alpha(b_2) \ge V^*[\beta,b_2] \tag{9}$$

for player 1 or player 2, respectively. The set of all $\alpha\beta$-vectors of player 1 (player 2) used to construct the approximating function $\underline{V}$ ($\overline{V}$) is denoted Γ_1 (Γ_2, respectively).

Since V^* is concave in the belief of player 2 (i.e., $\Delta(S_1)$), every convex combination of $\alpha\beta$-vectors in Γ_1 forms a lower bound on V^*, and for every coefficients of a convex combination $\lambda(\alpha\beta) \ge 0$ satisfying $\sum_{\alpha\beta\in\Gamma_1}\lambda(\alpha\beta) = 1$,

$$\alpha'(b_1) = \sum_{\alpha\beta\in\Gamma_1}\lambda(\alpha\beta)\alpha(b_1), \beta' = \sum_{\alpha\beta\in\Gamma_1}\lambda(\alpha\beta)\beta \tag{10}$$

$\alpha'\beta'$ is also an (implicit) $\alpha\beta$-vector. The implicit $\alpha\beta$-vectors form facets in Figure 1.

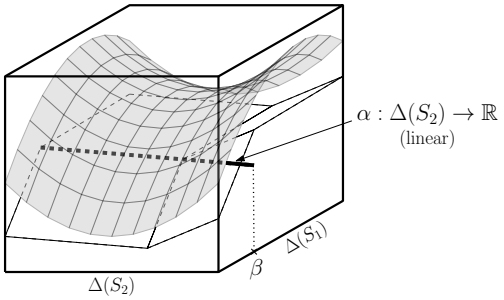


Figure 1: Lower bound on V^* using $\alpha\beta$-vectors of player 1.

Now, we leverage the α-vector representation of value functions as commonly used in POMDPs (or one-sided POSGs). To obtain the value $\underline{V}[b_1,b_2]$, a point-wise maximum over all (implicit) $\alpha\beta$-vectors with $\beta = b_2$ is taken.

$$\underline{V}[b_1,b_2] = \max_{\lambda(\cdot)\ge 0}\Big\{\sum_{\alpha\beta\in\Gamma_1}\lambda(\alpha\beta)\alpha(b_1)\;\Big|\;\sum_{\alpha\beta\in\Gamma_1}\lambda(\alpha\beta)\beta = b_2\Big\} \tag{11}$$

The upper-bounding value function $\overline{V}$ is constructed by considering $\alpha\beta$-vectors Γ_2 of player 2 and using a point-wise minimum instead of maximum. Lower and upper-bound approximations define the approximation error.

Definition 6. Let $\underline{V}$ and $\overline{V}$ be the current approximations of V^*. The approximation error (*gap*) in joint belief (b_1, b_2) is

$$\hat{V}[b_1,b_2] = \overline{V}[b_1,b_2] - \underline{V}[b_1,b_2]\,. \tag{12}$$

Computing $HV[b_1,b_2]$ via linear programming

When considering approximate value functions $\underline{V}$ and $\overline{V}$ described in the previous section, the point-based value backup $H\underline{V}[b_1,b_2]$ (or $H\overline{V}[b_1,b_2]$) can be evaluated using linear programming. We again focus on the construction of a linear program for computing lower bound $H\underline{V}[b_1,b_2]$ (denoted LP($H\underline{V}[b_1,b_2]$)), the case for LP($H\overline{V}[b_1,b_2]$) is analogous.

We start by rewriting the optimization problem $H\underline{V}[b_1,b_2]$ (as defined in Equation (7)) by evaluating $\underline{V}$ according to the Equation (11).

$$\max_{\pi_1,\lambda}\;\min_{\pi_2}\Big[R_{\pi_1\pi_2} + \gamma\sum_{o_1,o_2}\Pr_{\pi_2}[o_1]\cdot\Pr_{\pi_1}[o_2]\cdot \tag{13a}$$
$$\cdot\sum_{s_2'}\tau_{\pi_2}(b_1o_1o_2)(s_2')\sum_{\alpha\beta\in\Gamma_1}\lambda^{o_1o_2}(\alpha\beta)\cdot\alpha(s_2')\Big]$$
$$\text{s.t.}\;\sum_{\alpha\beta\in\Gamma_1}\lambda^{o_1o_2}(\alpha\beta)\cdot\beta(s_1') = \tau_{\pi_1}(b_2o_2o_1)(s_1') \tag{13b}$$
$$\forall(o_1,o_2)\in O_1\times O_2\;\;\forall s_1'\in S_1$$
$$\lambda(\cdot)\ge 0 \tag{13c}$$

Variables $\lambda(\cdot)$ from Equation (11) have been replaced with $\lambda^{o_1 o_2}(\cdot)$ for each observation pair. The term $1/Pr_{\pi_2}[o_1]$ in $\tau_{\pi_2}(b_1 o_1 o_2)$ cancels out, hence the objective becomes

$$\max_{\pi_1,\lambda} \min_{\pi_2} \Big[R_{\pi_1\pi_2} + \gamma \sum_{o_1,o_2} \Pr_{\pi_1}[o_2] \sum_{s_2,a_2,s_2'} b_1(s_2)\pi_2(a_2|s_2) \cdot \cdot Z(o_1|s_2 a_2) T(s_2'|s_2 a_2 o_2 o_1) \sum_{\alpha\beta\in\Gamma_1} \lambda^{o_1 o_2}(\alpha\beta)\alpha(s_2') \Big] . \quad (14)$$

Similarly, it is possible to cancel out $\Pr_{\pi_1}[o_2]$ in $\tau_{\pi_1}(b_2 o_2 o_1)$ by substituting $\hat{\lambda}^{o_1 o_2}(\cdot) = \Pr_{\pi_1}[o_2] \cdot \lambda^{o_1 o_2}(\cdot)$.

$$\max_{\pi_1,\hat{\lambda}} \min_{\pi_2} \Big[R_{\pi_1\pi_2} + \gamma \sum_{s_2,a_2,o_1,o_2,s_2'} b_1(s_2)\pi_2(a_2|s_2) Z[o_1|s_2 a_2] \cdot \cdot T[s_2'|s_2 a_2 o_2 o_1] \sum_{\alpha\beta\in\Gamma_1} \hat{\lambda}^{o_1 o_2}(\alpha\beta) \cdot \alpha(s_2') \Big] \quad (15a)$$

$$\text{s.t.} \sum_{\alpha\beta\in\Gamma_1} \hat{\lambda}^{o_1 o_2}(\alpha\beta) \cdot \beta(s_1') = \sum_{s_1,a_1} b_2(s_1)\pi_1(a_1|s_1) \cdot \cdot Z(o_2|s_1 a_1) T(s_1'|s_1 a_1 o_1 o_2) \quad \forall (o_1,o_2) \forall s_1' \quad (15b)$$

$$\hat{\lambda}(\cdot) \geq 0 \quad (15c)$$

When π_1 and $\hat{\lambda}$ variables are fixed, the value is linear in π_2. Hence the optimum will be in a pure strategy π_2. The minimization over a finite number of pure strategies can be rewritten using a set of linear inequality constraints. Moreover, we leverage the fact that the adversary (player 2) knows his current state. Therefore, it is possible to compute $\pi_2(\cdot|s_2)$ for each state s_2 of player 2 separately and compute the expectation over values of individual states. The resulting linear program follows.

$$\max_{\pi_1,\hat{\lambda}} \sum_{s_2} b_1(s_2) \cdot V(s_2)$$

$$\text{s.t. } V(s_2) \leq \sum_{s_1,a_1} b_2(s_1)\pi_1(a_1|s_1) R(s_1 s_2 a_1 a_2) + \quad (16a)$$
$$+ \gamma \sum_{o_1,o_2,s_2'} Z(o_1|s_2 a_2) T(s_2'|s_2 a_2 o_2 o_1) \cdot \cdot \sum_{\alpha\beta\in\Gamma_1} \hat{\lambda}^{o_1 o_2}(\alpha\beta) \cdot \alpha(s_2') \qquad \forall s_2, a_2$$

$$\sum_{\alpha\beta\in\Gamma_1} \hat{\lambda}^{o_1 o_2}(\alpha\beta) \cdot \beta(s_1') = \sum_{s_1,a_1} b_2(s_1)\pi_1(a_1|s_1) \cdot \quad (16b)$$
$$\cdot Z(o_2|s_1 a_1) T(s_1'|s_1 a_1 o_1 o_2) \qquad \forall (o_1,o_2) \forall s_1'$$

$$\hat{\lambda}(\cdot) \geq 0 \quad (16c)$$

Note that the variables $V(s_2)$ correspond to the values of playing a strategy represented by values of variables π_1 and $\hat{\lambda}$ in the unobserved state s_2 of the opponent. Such strategy prescribes player 1 to play according to strategy π_1 in the first stage of the game and then, after observing (o_1, o_2), follow a strategy the value of which is greater than the convex combination of $\alpha\beta$-vectors with coefficients $\lambda(\alpha\beta)$,

$$\lambda(\alpha\beta) = \hat{\lambda}^{o_1 o_2}(\alpha\beta) / \sum_{\alpha\beta\in\Gamma_1} \hat{\lambda}^{o_1 o_2}(\alpha\beta) . \quad (17)$$

Hence, we can use values of the variables $V(s_2)$ to form a new $\alpha\beta$-vector ($\beta = b_2$) such that $\alpha(b_1) = \sum_{s_2} b_1(s_2) \cdot V(s_2)$. For states s_2 with $b_1(s_2) = 0$, the value $V(s_2)$ may, however underestimate, due to the lack of pressure on $V(s_2)$. In these cases, we compute the minimum represented by constraints (16a) separately.

The algorithm

We are now ready to state our algorithm to compute an ϵ-approximation of V^* in the joint belief $(b_1^{(0)}, b_2^{(0)})$ and to prove its correctness. The algorithm (Algorithm 1) follows the ideas of the HSVI algorithm for POMDPs (Smith and Simmons 2004) and one-sided POSGs (Horak, Bosansky, and Pechoucek 2017) while replacing the point-based update step with the computation of optimal $\alpha\beta$-vectors to add using the linear program from Equations (16).

1 Initialize $\underline{V}$ and $\overline{V}$
2 **while** $\hat{V}[b_1^{(0)}, b_2^{(0)}] > \epsilon$ **do** `explore` $(b_1^{(0)}, b_2^{(0)}, 0)$
3 **procedure** `explore` (b_1, b_2, t)
4 **if** $\hat{V}[b_1, b_2] \leq \rho(t)$ **then return**
5 Extract $\overline{\pi}_1$ from LP($H\overline{V}[b_1, b_2]$) and $\underline{\pi}_2$ from LP($H\underline{V}[b_1, b_2]$)
6 $(o_1^*, o_2^*) \leftarrow \arg\max_{o_1,o_2} Pr_{\overline{\pi}_1\underline{\pi}_2}[o_1 o_2] \cdot \text{excess}^{t+1}(\tau_{\underline{\pi}_2}(b_1 o_1 o_2), \tau_{\overline{\pi}_1}(b_2 o_2 o_1))$
7 `explore` $(\tau_{\underline{\pi}_2}(b_1 o_1 o_2), \tau_{\overline{\pi}_1}(b_2 o_2 o_1), t+1)$
8 Extract α_1 from LP($H\underline{V}[b_1, b_2]$) ($V(s_2)$ variables)
9 Extract α_2 from LP($H\overline{V}[b_1, b_2]$) ($V(s_1)$ variables)
10 $\Gamma_1 \leftarrow \Gamma_1 \cup \{\alpha_1 b_2\}$; $\Gamma_2 \leftarrow \Gamma_2 \cup \{\alpha_2 b_1\}$

Algorithm 1: HSVI algorithm for PO-POSGs.

Since we want to focus on the key characteristics of the algorithm, we initialize $\underline{V}$ and $\overline{V}$ using the minimum and maximum possible utilities of player 1,

$$L = \min_{s_1,s_2,a_1,a_2} R(s_1 s_2 a_1 a_2)/(1-\gamma) \quad (18)$$

$$U = \max_{s_1,s_2,a_1,a_2} R(s_1 s_2 a_1 a_2)/(1-\gamma) . \quad (19)$$

In practice, we can obtain tighter bounds (and consequently faster convergence) by either leveraging domain knowledge, or solving a simplified version of the game.

To obtain an ϵ-approximation of $V^*[b_1^{(0)}, b_2^{(0)}]$, it is sufficient that beliefs (b_1, b_2) reached at depth t (the value of which is therefore multiplied by γ^t) satisfy $\hat{V}[b_1, b_2] \leq \rho(t)$, where $\rho(t)$ is an increasing and unbounded sequence (for sufficiently small $R > 0$),

$$\rho(t) = \epsilon\gamma^{-t} - \sum_{i=1}^{t} 2R(U-L)\gamma^{-i} . \quad (20)$$

If $\hat{V}[b_1, b_2] > \rho(t)$, we say that (b_1, b_2) has a positive excess gap $\text{excess}^t(b_1, b_2) = \hat{V}[b_1, b_2] - \rho(t)$.

Once Algorithm 1 terminates, an ϵ-approximation of $V^*[b_1^{(0)}, b_2^{(0)}]$ has been found (see line 2). Moreover, since

the sequence $\rho(t)$ is increasing and unbounded while the maximum gap is bounded by $U - L$, the condition on line 4 is always eventually met and every call to `explore` therefore terminates in a bounded number of recursion levels (denote this bound $T_{\max}$). It is therefore sufficient to show that the number of calls to `explore` is finite.

Denote $\{(b_1^{(t)}, b_2^{(t)})\}_{t=0}^{T}$ the beliefs that have been visited during a trial of length T. Observe that $\hat{V}[b_1^{(T-1)}, b_2^{(T-1)}] > \rho(T-1)$ (otherwise the trial would have terminated at depth $(T-1)$). On the contrary, when considering belief $(b_1^{(T-1)}, b_2^{(T-1)})$ and the corresponding strategy profile $(\overline{\pi}_1, \underline{\pi}_2)$ from line 5, the reachable beliefs satisfy $\hat{V}[\tau_{\underline{\pi}_2}(b_1^{(T-1)} o_1 o_2), \tau_{\overline{\pi}_1}(b_2^{(T-1)} o_2 o_1)] \leq \rho(T)$ for every (o_1, o_2) seen with positive probability.

Lemma 2. *Consider a trial $\{(b_1^{(t)}, b_2^{(t)})\}_{t=0}^{T}$ of length T and consider that point-based updates on lines 8–10 of Algorithm 1 have been performed. Then*
(1) $\hat{V}[b_1^{(T-1)}, b_2^{(T-1)}] \leq \rho(T-1) - 2R(U-L)$, and
(2) For every (b_1, b_2) satisfying $\|(b_1, b_2) - (b_1^{(T-1)}, b_2^{(T-1)})\|_2 \leq R$, it holds $\hat{V}[b_1, b_2] \leq \rho(T-1)$.

Proof (sketch). Observe that from the definition of the sequence $\rho(t)$ in Equation (20) it follows that

$$\gamma\rho(T) = \rho(T-1) - 2R(U-L)\,. \tag{21}$$

Moreover, the trial terminated at depth T. Therefore, all beliefs that can be reached from $(b_1^{(T-1)}, b_2^{(T-1)})$ when following $(\overline{\pi}_1, \underline{\pi}_1)$ from line 5 must satisfy

$$\hat{V}[\tau_{\underline{\pi}_2}(b_1^{(T-1)} o_1 o_2), \tau_{\overline{\pi}_1}(b_2^{(T-1)} o_2 o_1)] \leq \rho(T)\,. \tag{22}$$

Let $(\underline{\pi}_1, \underline{\pi}_2)$ (and $(\overline{\pi}_1, \overline{\pi}_2)$) be equilibrial strategy profiles in $H\underline{V}[b_1^{(T-1)}, b_2^{(T-1)}]$ (and $H\overline{V}[b_1^{(T-1)}, b_2^{(T-1)}]$, respectively) and denote $u^V(\pi_1, \pi_2)$ the utility of playing strategies (π_1, π_2) in $HV[b_1^{(T-1)}, b_2^{(T-1)}]$. By deviating from the equilibrium, the players can only worsen their utility. Hence,

$$\begin{aligned} u^{\underline{V}}(\overline{\pi}_1, \underline{\pi}_2) \leq u^{\underline{V}}(\underline{\pi}_1, \underline{\pi}_2) = H\underline{V}[b_1^{(T-1)}, b_2^{(T-1)}] \leq \\ \leq H\overline{V}[b_1^{(T-1)}, b_2^{(T-1)}] = u^{\overline{V}}(\overline{\pi}_1, \overline{\pi}_2) \leq u^{\overline{V}}(\overline{\pi}_1, \underline{\pi}_2)\,. \end{aligned} \tag{23}$$

Since the same strategy profile $(\overline{\pi}_1, \underline{\pi}_2)$ is considered in both $u^{\underline{V}}(\overline{\pi}_1, \underline{\pi}_2)$ and $u^{\overline{V}}(\overline{\pi}_1, \underline{\pi}_2)$, the difference satisfies

$$\begin{aligned} u^{\overline{V}}(\overline{\pi}_1, \underline{\pi}_2) - u^{\underline{V}}(\overline{\pi}_1, \underline{\pi}_2) = \gamma \textstyle\sum_{o_1 o_2} \Pr_{\overline{\pi}_1 \underline{\pi}_2}[o_1 o_2] \cdot \\ \cdot \hat{V}[\tau_{\underline{\pi}_2}(b_1^{(T-1)} o_1 o_2), \tau_{\overline{\pi}_1}(b_2^{(T-1)} o_2 o_1)]\,. \end{aligned} \tag{24}$$

The gap $\hat{V}[\tau_{\underline{\pi}_2}(b_1 o_1 o_2), \tau_{\overline{\pi}_1}(b_2 o_2 o_1)]$ of all beliefs reachable using $(\overline{\pi}_1, \underline{\pi}_2)$ is smaller than $\rho(T)$ and hence $u^{\overline{V}}(\overline{\pi}_1, \underline{\pi}_2) - u^{\underline{V}}(\overline{\pi}_1, \underline{\pi}_2) \leq \gamma\rho(T)$. The point-based update in $(b_1^{(T-1)}, b_2^{(T-1)})$ renders $\hat{V}[b_1^{(T-1)}, b_2^{(T-1)}] \leq \rho(T-1) - 2R(U-L)$ which concludes the proof of (1).

Now, since V^* is $(U-L)$-Lipschitz continuous (Theorem 2), it is possible to consider $(U-L)$-Lipschitz continuous approximations $\underline{V}$ and $\overline{V}$. Function $\hat{V} = \overline{V} - \underline{V}$ is then $2(U-L)$-Lipschitz continuous and therefore the value of any belief within the R-neighborhood of $(b_1^{(T-1)}, b_2^{(T-1)})$ cannot be higher than $\rho(T-1)$ which proves (2). □

We are now ready to prove the correctness of the algorithm by showing that it can only perform a finite number of trials of given length.

Theorem 3. *Algorithm 1 terminates with an ϵ-approximation of $V^*[b_1^{(0)}, b_2^{(0)}]$.*

Proof. Assume for the sake of contradiction that the algorithm does not terminate and generates an infinite number of `explore` trials. Since the length of a trial is bounded by a finite number $T_{\max}$, the number of trials of length T (for some $0 \leq T \leq T_{\max}$) must be infinite. It is impossible to fit an infinite number of belief points (b_1, b_2) satisfying $\|(b_1, b_2) - (b_1', b_2')\|_2 > R$ within $\Delta(S_1) \times \Delta(S_2)$. Hence there must be two trials of length T, $\{(b_{11}^{(t)}, b_{21}^{(t)})\}_{t=0}^{T}$ and $\{(b_{12}^{(t)}, b_{22}^{(t)})\}_{t=0}^{T}$, such that $\|(b_{11}^{(T-1)}, b_{21}^{(T-1)}) - (b_{12}^{(T-1)}, b_{22}^{(T-1)})\|_2 \leq R$. Without loss of generality, assume that $(b_{11}^{(T-1)}, b_{21}^{(T-1)})$ was visited the first. According to Lemma 2, the point-based update in $(b_{11}^{(T-1)}, b_{21}^{(T-1)})$ resulted in $\hat{V}[(b_{12}^{(T-1)}, b_{22}^{(T-1)})] \leq \rho(T-1)$—which contradicts that the condition on line 4 of Algorithm 1 has not been satisfied for $(b_{12}^{(T-1)}, b_{22}^{(T-1)})$ (and hence that $\{(b_{12}^{(t)}, b_{22}^{(t)})\}_{t=0}^{T}$ was a trial of length T). □

Implementation details

In this section, we provide some of the details on our practical implementation of the HSVI algorithm for PO-POSGs.

Pruning The number of $\alpha\beta$-vectors grows in the course of the algorithm, however, not all of the vectors are needed to represent $\underline{V}$ (or $\overline{V}$) accurately. To counteract this growth, we run a pruning procedure every time the size of Γ_i gets $1.5\times$ larger than after the pruning was last performed. An $\alpha\beta$-vector is pruned if there exists a convex combination of vectors in Γ_i that dominates it.

Lipschitz continuity The theoretical proof of the correctness relies on the fact that the approximating functions are $(U-L)$-Lipschitz continuous. While the extracted $\alpha\beta$-vectors from the linear programs are $(U-L)$-Lipschitz continuous, the implicitly computed convex/concave hull need not satisfy this property (and thus may potentially render the Lipschitz constant of $\underline{V}$ or $\overline{V}$ impossible to bound). While this issue can be fixed by computing a $(U-L)$-Lipschitz envelope of $\underline{V}$ or $\overline{V}$ by adding additional $\alpha\beta$-vectors to Γ_i, we omit this step in our implementation. The computation of the envelope significantly increases the number of $\alpha\beta$-vectors (and thus the number of expensive pruning steps) and our experimental results show that the algorithm converges in practice even when the assumption of boundedly Lipschitz-continuous approximations is relaxed.

Other We use the idea of modifying ϵ between iterations similarly to (Horak, Bosansky, and Pechoucek 2017). The ϵ_{imm} for the current iteration is obtained as $\epsilon_{\text{imm}} = \epsilon + 0.5(\hat{V}[b_1^{(0)}, b_2^{(0)}] - \epsilon)$. This allows the algorithm to perform shorter trials in the initial phases of the search (when the bounds do not provide accurate information about what parts of the belief space to target).

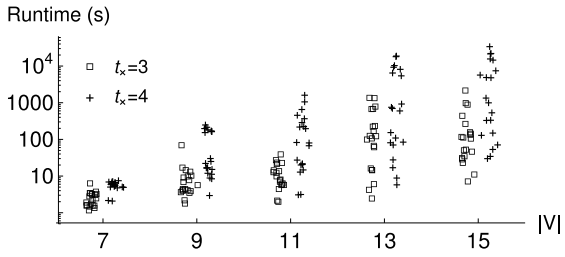


Figure 2: Experimental results on the Patrolling domain for different sizes of graph $|V|$. Time to reach $\hat{V}[b_1^{(0)}, b_2^{(0)}] \leq 1$.

We construct a compact version of the linear programs LP($HV[b_1, b_2]$). Namely, we consider only states, actions and observation pairs that can be played/observed in the current joint belief (b_1, b_2). Furthermore, we adopt a column generation approach to incrementally add variables $\lambda^{o_i o_j}(\cdot)$. Initially, we start with one $\alpha\beta$-vector (and its $\lambda^{o_i o_j}(\alpha\beta)$) for each pure belief of the opponent and we add additional $\alpha\beta$-vectors once they are necessary to accurately represent $V[\tau(b_1 o_1 o_2), \tau(b_2 o_2 o_1)]$.

Experiments

We demonstrate the scalability of our algorithm on two fundamentally different domains—partially observable patrolling inspired by (Basilico et al. 2009) and a lasertag game inspired by *Tag* from (Pineau, Gordon, and Thrun 2003). All experiments use discount factor $\gamma = 0.95$ and were run on Intel i7-8700K (solving 6 instances in parallel).

Patrolling The game is played by two players—the *patroller* and the *intruder*. The patroller moves between vertices V of a graph $G = (V, E)$ and attempts to locate an intruder before the intruder succeeds in causing damage. The intruder starts initially outside of the graph and observes the position of the patroller whenever he steps on one of the observable vertices $O \subseteq V$ (otherwise the position of the patroller remains hidden). The intruder may decide to attack any target vertex $v \in T$, $T \subseteq O$. Once the intruder decides to attack, he has to stay undetected in the chosen vertex v for $t_\times$ time steps to complete his attack and get a reward $c(v)$.

In our experimental evaluation, we consider $t_\times = 3$ and $t_\times = 4$ and generate random graphs from the Dorogovtsev-Mendes model such that the shortest cycle covering all targets is longer than $t_\times$ (i.e., the patroller cannot cover the targets perfectly). There are $|T| = \lceil V/4 \rceil$ targets and $|O| = \lceil 2V/3 \rceil$ observable nodes. The costs $c(v)$ of targets are generated uniformly from the $[70, 100]$ interval. Figure 2 summarizes the runtime of our algorithm on 200 randomly generated instances of Patrolling (time to reach precision 1, i.e., 1% of the maximum cost, is reported). All instances have been solved within 10 hours, while 97 instances with $t_\times = 3$ out of 100 and 82 instances with $t_\times = 4$ out of 100 have been solved in less than 20 minutes.

Lasertag The game is played by two players—the *tagger* and the *evader*—on a grid. In each time step, the players can decide to move to an adjacent square (free of an obstacle), or, the tagger can additionally shoot a laser beam either horizontally or vertically (which is effective until hitting the first obstacle). If the beam tags the evader, the tagger receives a reward $+10$ and the game ends, otherwise his reward is -10 and the game continues. Unless the tagger decides to use the laser beam, his reward is -1 in each step. Hence, the tagger attempts to terminate the game by tagging the adversary as quickly as possible. Neither player knows the position of each other until the tagger decides to shoot when the evader can observe the light ray (and thus deduce possible positions of the tagger).

We consider lasertag games played on a 4×4 grid with 3 obstacles where the tagger starts in the top-left corner, while the evader starts at position $(3, 4)$ next to the opposite corner. The obstacles are placed randomly while guaranteeing the existence of a path between the players (we discard symmetrical instances). We ran the algorithm with $\epsilon = 0.05$ for 5 hours. While the algorithm did not terminate within this limit on 16 out of 20 instances, the average excess gap in the initial belief relative to the value of the lower bound was 10%±2.6% (where the confidence interval marks standard error). For grid size 3×3, all non-symmetric instances with players starting in opposite corners have been solved in less than 8 seconds.

Analysis We provide a detailed analysis of the performance of the algorithm for two instances of patrolling, an 11-vertex instance with $t_\times = 4$ solved in 307s and a 13-vertex instance with $t_\times = 4$ solved in 11004s. On both of the instances, 85% of the runtime corresponds to the operations with the approximating functions (especially computing values for a joint belief), while the construction and solving LP($HV[b_1, b_2]$) took only 10% of the runtime. The remaining 5% of the runtime corresponds to the pruning step, initiated 95 times on the larger instance within the 1556 iterations. The pruning eliminated 22126 $\alpha\beta$-vectors out of 50404 generated on the larger instance. Unlike in the patrolling domain, on a lasertag instance solved in 9337s the pruning was much more frequent (approximately one pruning per 6 iterations) and considerably more demanding (took 22% of runtime).

Conclusions

We present a subclass of partially observable stochastic games (POSGs) where the observations are publicly observable by the players. We provide the formal definition of such games and a novel, practical algorithm with proven convergence to approximately solve games in this class. Our algorithm is, to the best of our knowledge, the first practical general algorithm to solve a broad subclass of infinite-horizon games where both players lack information about the game state.

There is a large volume of possible future work. One direction is to adopt recent advancements from single-player POMDPs and apply them to the class of POSGs to improve scalability. Next, one can fine-tune representation of value functions and their initialization for specific games in security domains (e.g., in cybersecurity) to solve much larger

instances that correspond to real-world problems. Another way is to generalize the approach presented in this paper and relax some of the assumptions made: (1) adapt the approach for objective functions different from the discounted sum of rewards or (2) relax the factorization over the states.

Acknowledgments

This research was sponsored by the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics" and the Army Research Laboratory, and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

Basilico, N.; Gatti, N.; Rossi, T.; Ceppi, S.; and Amigoni, F. 2009. Extending Algorithms for Mobile Robot Patrolling in the Presence of Adversaries to More Realistic Settings. In *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*.

Basilico, N.; Gatti, N.; and Amigoni, F. 2009. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *AAMAS*.

Basilico, N.; Nittis, G. D.; and Gatti, N. 2016. A Security Game Combining Patrolling and Alarm–Triggered Responses Under Spatial and Detection Uncertainties. In *AAAI*.

Basu, A., and Stettner, L. 2015. Finite- and infinite-horizon shapley games with nonsymmetric partial observation. *SIAM Journal on Control and Optimization* 53(6):3584–3619.

Brazdil, T.; Kucera, A.; and Rehak, V. 2018. Solving Patrolling Problems in the Internet Environment. In *IJCAI*.

Brown, N.; Sandholm, T.; and Amos, B. 2018. Depth-Limited Solving for Imperfect-Information Games. In *NIPS*.

Chatterjee, K., and Doyen, L. 2014. Partial-observation stochastic games: How to win when belief fails. *ACM Transactions on Computational Logic* 15(2):16.

Chung, T. H.; Hollinger, G. A.; and Isler, V. 2011. Search and pursuit-evasion in mobile robotics. *Autonomous robots* 31(4):299–316.

Cole, H. L., and Kocherlakota, N. 2001. Dynamic games with hidden actions and hidden states. *Journal of Economic Theory* 98(1):114–126.

Durkota, K.; Lisý, V.; Kiekintveld, C.; Horák, K.; Bošanský, B.; and Pevný, T. 2017. Optimal Strategies for Detecting Data Exfiltration by Internal and External Attackers. In *GameSec*.

Fang, F.; Nguyen, T. H.; Pickles, R.; Lam, W. Y.; Clements, G. R.; An, B.; Singh, A.; Tambe, M.; and Lemieux, A. 2016. Deploying PAWS: Field optimization of the protection assistant for wildlife security. In *AAAI*.

Fang, F.; Stone, P.; and Tambe, M. 2015. When Security Games Go Green: Designing Defender Strategies to Prevent Poaching and Illegal Fishing. In *IJCAI*.

Ghosh, M. K.; McDonald, D.; and Sinha, S. 2004. Zero-Sum Stochastic Games with Partial Information. *Journal of Optimization Theory and Applications* 121(1):99–118.

Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004. Dynamic Programming for Partially Observable Stochastic Games. In *AAAI*.

Horak, K., and Bosansky, B. 2016. A Point-Based Approximate Algorithm for One-Sided Partially Observable Pursuit-Evasion Games. In *GameSec*.

Horak, K.; Bosansky, B.; and Pechoucek, M. 2017. Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games. In *AAAI*.

Kiekintveld, C.; Jain, M.; Tsai, J.; Pita, J.; Ordóñez, F.; and Tambe, M. 2009. Computing optimal randomized resource allocations for massive security games. In *AAMAS*.

Kumar, A., and Zilberstein, S. 2009. Dynamic programming approximations for partially observable stochastic games. In *FLAIRS*.

MacDermed, L. C. 2013. *Value Methods for Efficiently Solving Stochastic Games of Complete and Incomplete Information*. Ph.D. Dissertation, Georgia Institute of Technology.

Moravcik, M.; Schmid, M.; Burch, N.; Lisý, V.; Morrill, D.; Bard, N.; Davis, T.; Waugh, K.; Johanson, M.; and Bowling, M. 2017. DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker. *Science* 356(6337).

Nguyen, T. H.; Wellman, M. P.; and Singh, S. 2017. A Stackelberg Game Model for Botnet Data Exfiltration. In *GameSec*.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*.

Pita, J.; Jain, M.; Marecki, J.; Ordóñez, F.; Portway, C.; Tambe, M.; Western, C.; Paruchuri, P.; and Kraus, S. 2008. Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles International Airport. In *AAMAS*.

Shieh, E.; An, B.; Yang, R.; Tambe, M.; Baldwin, C.; Direnzo, J.; Meyer, G.; Baldwin, C. W.; Maule, B. J.; and Meyer, G. R. 2012. PROTECT : A Deployed Game Theoretic System to Protect the Ports of the United States. In *AAMAS*.

Smith, T., and Simmons, R. 2004. Heuristic search value iteration for POMDPs. In *UAI*.

Vanek, O.; Yin, Z.; Jain, M.; Bosansky, B.; Tambe, M.; and Pechoucek, M. 2012. Game-theoretic Resource Allocation for Malicious Packet Detection in Computer Networks. In *AAMAS*.

von Neumann, J. 1928. Zur theorie der gesellschaftsspiele. *Mathematische Annalen* 100(1):295–320.

Vorobeychik, Y.; An, B.; Tambe, M.; and Singh, S. P. 2014. Computing Solutions in Infinite-Horizon Discounted Adversarial Patrolling Games. In *ICAPS*.

Appendix F

Sequence-Form Algorithm for Computing Stackelberg Equilibria in Extensive-Form Games

Sequence-Form Algorithm for Computing Stackelberg Equilibria in Extensive-Form Games

Branislav Bošanský[1,2], **Jiří Čermák**[1]
[1] Agent Technology Center, Faculty of Electrical Engineering, Czech Technical University in Prague
[2] Computer Science Department, Aarhus University
bosansky@cs.au.dk, jiri.cermak@agents.fel.cvut.cz

Abstract

Stackelberg equilibrium is a solution concept prescribing for a player an optimal strategy to commit to, assuming the opponent knows this commitment and plays the best response. Although this solution concept is a cornerstone of many security applications, the existing works typically do not consider situations where the players can observe and react to the actions of the opponent during the course of the game. We extend the existing algorithmic work to extensive-form games and introduce novel algorithm for computing Stackelberg equilibria that exploits the compact sequence-form representation of strategies. Our algorithm reduces the size of the linear programs from exponential in the baseline approach to linear in the size of the game tree. Experimental evaluation on randomly generated games and a security-inspired search game demonstrates significant improvement in the scalability compared to the baseline approach.

Introduction

Solving games and computing game-theoretic solutions is one of the key topics of artificial intelligence. The best known solution concepts in game theory is Nash equilibrium that prescribes optimal *strategy profile* (one strategy for each player) such that no player can gain anything by unilaterally changing their strategy. However, a line of research that analyzes two-player games where the first player (termed *the leader*) is able to commit to a publicly known strategy before the other player (termed *the follower*) moves has received a significant focus in recent years. This solution concept is known as Stackelberg (or leader-follower) equilibrium (von Stackelberg 1934; Leitmann 1978; von Stengel and Zamir 2004) and it is a cornerstone of many security applications of game theory. The examples include airport security (Pita et al. 2008), assigning Air Marshals to flights (Tsai et al. 2009), or protecting the coast (Shieh et al. 2012).

Most of the existing applications of Stackelberg equilibria (SE) typically focus on single-step normal-form or Bayesian games where the players do not learn anything during the course of the game – the players are not able to observe the actions performed by the opponent and cannot react to this information. Such an assumption can be too strong in practice, since conditioning the strategies of the players by the current development in the game can be desirable (e.g., security procedures based on a suspicious behavior). Game theory provides representation for modeling finite sequential interactions between the players known as *extensive-form games* (EFGs). This model is sufficiently generic to represent uncertainty in observation (i.e., players are not able to perfectly observe the current state of the game), or stochastic environment.

The problem of computing SE in EFGs has been studied from the computational perspective and it was shown that computing SE in extensive-form games with imperfect information and stochastic events is NP-hard (Letchford and Conitzer 2010). However, a tailored algorithm for computing an exact SE in EFGs is currently missing. A baseline approach is thus to transform an EFG to a single step normal-form game and use one of the existing approaches based on computing multiple linear programs (LPs) (Conitzer and Sandholm 2006), or formulating the problem as a mixed-integer linear program (MILP) (Paruchuri et al. 2008). This transformation is, however, exponential and we show that the scalability of the baseline approach is very limited.

This paper aims to address this issue and provides the first specific algorithm for computing SE in two player EFGs. Our approach exploits the compact representation of strategies known as *the sequence form* (Koller, Megiddo, and von Stengel 1996; von Stengel 1996). We generalize two existing algorithms for computing SE in single-step games and introduce two variants of our novel algorithm: (1) based on solving multiple LPs, and (2) based on reformulating the problem as a MILP. Moreover, the size of our MILP is linear in the size of the game, and it contains only binary integer variables; hence, it provides an upper bound on the computational complexity of the problem of computing SE and shows that it is an NP-complete problem (Karp 1971).

We first give necessary technical background on EFGs, Nash and Stackelberg solution concepts, following by the description of the existing algorithms for computing each of these concepts. Next, we describe our novel algorithm that exploits the sequence form for computing SE in EFGs. Finally, we experimentally demonstrate the scalability of our algorithm on randomly generated games and a security-inspired search game. The results show that our MILP algorithm is significantly faster and can solve games several magnitudes larger compared to the baseline approach.

Technical Background

Extensive-form games (EFGs) model sequential interactions between the players and can be visually represented as game trees. Nodes in the game tree represent the states of the game; each state of the game corresponds to a sequence of moves executed by all players in the game. Each node is assigned to a player that acts in the game state associated with this node. An edge from a node corresponds to an action that can be performed by the player who acts in this node. EFGs model limited observations of the players by grouping certain nodes into *information sets*; a player cannot distinguish between nodes that belong to the same information set. The model also represents uncertainty about the environment and stochastic events by using a special *Nature player*.

Formally, a two-player EFG is defined as a tuple $G = (N, H, Z, A, \rho, u, \mathcal{C}, \mathcal{I})$: $N = \{1, 2\}$ is a set of two players; we use i to refer to one of the two players (either 1 or 2), and $-i$ to refer to the opponent of i. H denotes a finite set of *nodes* in the game tree. Each node corresponds to a unique *history* of actions taken by all players and Nature from the root of the game; hence, we use the terms history and node interchangeably. We denote by $Z \subseteq H$ the set of all *terminal nodes* of the game. A denotes the set of all actions and we overload the notation and use $A(h) \subseteq A$ to represent the set of actions available to the player acting in node $h \in H$. We specify $ha = h' \in H$ to be node h' reached from node h by performing action $a \in A(h)$. For each terminal node $z \in Z$ we define a *utility function* for each player i ($u_i : Z \to \mathbb{R}$).

The function $\rho : H \to N \cup \{c\}$ assigns each node to a player who takes an action in the node, where c means that the Nature player selects an action in the node based on a fixed probability distribution known to all players. We use function $\mathcal{C} : H \to [0, 1]$ to denote the probability of reaching node h due to Nature (i.e., assuming that both players play all required actions to reach node h). The value of $\mathcal{C}(h)$ is defined to be the product of the probabilities assigned to all actions taken by the Nature player in history h.

Imperfect observation of player i is modeled via *information sets* $\mathcal{I}_i$ that form a partition over the nodes assigned to player i $\{h \in H : \rho(h) = i\}$. Every information set contains at least one node and each node belongs to exactly one information set. Nodes in an information set of a player are not distinguishable to the player. All nodes h in a single information set $I_i \in \mathcal{I}_i$ have the same set of possible actions $A(h)$; hence, an action a from $A(h)$ uniquely identifies information set I_i and there cannot exist any other node $h' \in H$ that does not belong to information set I_i and for which a is allowed to be played (i.e., $a \in A(h')$). We overload the notation and use $A(I_i)$ to denote the set of actions defined for each node h in this information set. We assume *perfect recall*, which means that players perfectly remember their own actions and all information gained during the course of the game. As a consequence, all nodes in any information set I_i have the same history of actions for player i.

Strategies and Solution Concepts in EFGs

Solving a game implies finding a strategy profile (i.e., one strategy for each player) that satisfies conditions given by a specific solution concept. *Pure strategies* (denoted as Π_i) correspond to assignments of exactly one action for each information set. A mixed strategy is a probability distribution over the set of all pure strategies of a player. We denote by Δ_i the set of all mixed strategies of player i. For any pair of strategies $\delta \in \Delta = (\Delta_1, \Delta_2)$ we use $u_i(\delta) = u_i(\delta_i, \delta_{-i})$ for the expected outcome of the game for player i when players follow strategies δ. A *best response* of player i to the opponent's strategy δ_{-i} is a strategy $\delta_i^{BR} = BR_i(\delta_{-i})$, for which $u_i(\delta_i^{BR}, \delta_{-i}) \geq u_i(\delta_i', \delta_{-i})$ for all strategies $\delta_i' \in \Delta_i$.

Nash equilibrium (NE) is the best known solution concept in game theory and it describes the behavior of agents under certain assumptions about their rationality. In a Nash equilibrium, every player plays a best response to the strategies of the other players. Formally, a strategy profile $\delta = (\delta_1, \delta_2)$ is a NE if and only if for each player i it holds that δ_i is a best response to δ_{-i} (i.e., $\forall i \in N\ \delta_i = BR_i(\delta_{-i})$).

In the *Stackelberg* setting, the first player commits to a certain strategy δ_1 that is observed by the second player that plays a pure best response π_2 to this strategy ($\pi_2 = BR_2(\delta_1)$). Two strategies (δ_1, π_2) are in a *Stackelberg equilibrium* (SE) if $\pi_2 = BR_2(\delta_1)$ and the expected utility of the leader is maximal (i.e., $\forall \delta_1' \in \Delta_1, \forall \pi_2' \in \Pi_2$ such that $\pi_2' = BR_2(\delta_1')$ it holds $u_1(\delta_1, \pi_2) \geq u_1(\delta_1', \pi_2')$). In case player 2 has multiple possible best response strategies, we assume the ties are broken in favor of player 1. This assumption is common in the literature and the equilibrium is called *Strong Stackelberg Equilibrium* (SSE) (Leitmann 1978; Conitzer and Sandholm 2006; Paruchuri et al. 2008; Yin et al. 2010).

Sequence-Form and Computing NE in EFGs

Strategies in EFGs with perfect recall can be compactly represented by using the notion of sequences (Koller, Megiddo, and von Stengel 1996; von Stengel 1996). A *sequence* σ_i is an ordered list of actions taken by a single player i in history h. The number of actions (i.e., the length of sequence σ_i) is denoted as $|\sigma_i|$, the empty sequence (i.e., a sequence with no actions) is denoted as $\emptyset$. The set of all possible sequences for player i is denoted by Σ_i. A sequence $\sigma_i \in \Sigma_i$ can be extended by a single action a taken by player i, denoted by $\sigma_i a = \sigma_i'$. In games with perfect recall, all nodes in an information set I_i share the same sequence of actions for player i and we use $\mathsf{seq}_i(I_i)$ to denote this sequence. We overload the notation and use $\mathsf{seq}_i(h)$ to denote the sequence of actions of player i leading to node h. Since action a uniquely identifies information set I_i and all nodes in an information set share the same history, each sequence uniquely identifies an information set. We use the function $\mathsf{inf}_i(\sigma_i')$ to denote the information set in which the last action of the sequence σ_i' is taken. For an empty sequence, function $\mathsf{inf}_i(\emptyset)$ denotes the information set of the root node. A mixed strategy of a player can now be represented as a probability distribution over the sequences and it is called a *realization plan* (denoted $r_i : \Sigma_i \to \mathbb{R}$). A realization plan for a sequence σ_i is the probability that player i will play this sequence of actions under the assumption that the opponent plays in a way which allows the actions specified in σ_i to be played.

When the strategies are represented as realization plans, we can compute a Nash equilibrium of a two-player general-sum extensive-form game using a linear complimentary program (LCP) of a polynomial size in the size of the game tree using the sequence form (Koller, Megiddo, and von Stengel 1996; von Stengel 1996). To describe the program we need to define payoff function $g_i : \Sigma \rightarrow \mathbb{R}$ that extends the utility function to all nodes in the game tree. The payoff function g_i represents the expected utility of all nodes reachable by sequentially executing the actions specified in a pair of sequences σ:

$$g_i(\sigma_i, \sigma_{-i}) = \sum_{h \in Z \,:\, \forall j \in N \;\; \sigma_j = \mathsf{seq}_j(h)} u_i(h) \cdot \mathcal{C}(h) \qquad (1)$$

The value of the payoff function is defined to be 0 if no leaf is reachable by sequentially executing all of the actions in the sequences σ – i.e., either all actions from the pair of sequences σ are executed and an inner node $h \in H \setminus Z$ is reached, or there is no further action that can be executed from a node that is reached during the execution of the sequences. We say that a pair of sequences σ is *compatible* if there exists a node $h \in H$ such that sequence σ_i of every player i equals to $\mathsf{seq}_i(h)$. Now, the equilibrium realization plans can be computed using the following feasibility LCP (e.g., see (Shoham and Leyton-Brown 2009) p. 135):

$$v_{\mathsf{inf}_i(\sigma_i)} = s_{\sigma_i} + \sum_{I'_i \in \mathcal{I}_i : \mathsf{seq}_i(I'_i) = \sigma_i} v_{I'_i} + \sum_{\sigma_{-i} \in \Sigma_{-i}} g_i(\sigma_i, \sigma_{-i}) \cdot r_{-i}(\sigma_{-i}) \qquad \forall i \in N \; \forall \sigma_i \in \Sigma_i \qquad (2)$$

$$r_i(\emptyset) = 1 \qquad \forall i \in N \qquad (3)$$

$$r_i(\sigma_i) = \sum_{a \in A(I_i)} r_i(\sigma_i a) \qquad \forall i \in N \; \forall I_i \in \mathcal{I}_i, \; \sigma_i = \mathsf{seq}_i(I_i) \qquad (4)$$

$$0 = r_i(\sigma_i) \cdot s_{\sigma_i} \qquad \forall i \in N \; \forall \sigma_i \in \Sigma_i \qquad (5)$$

$$0 \leq r_i(\sigma_i) \; ; \qquad 0 \leq s_{\sigma_i} \qquad \forall i \in N \; \forall \sigma_i \in \Sigma_i \qquad (6)$$

Constraints (2) ensure that the expected value v_{I_i} in each information set of player i equals to the value of the best response in this information set for player i against the strategy of the opponent $-i$ (i.e., for each action applicable in information set I_i, v_{I_i} is greater than sum of all expected values of information sets and leafs reachable after playing an action in this information set). Note that in our formulation we represent the inequality as an equality with slack variables s_{σ_i}; there is one positive slack variable for each sequence of each player σ_i. We use this representation as it would be convenient for our novel algorithms to operate with these constraints by using the slack variables.

Constraints (3-4) ensure that the realization plans of both players satisfy the network-flow constraints: the probability of reaching information set I_i using sequence $\sigma_i = \mathsf{seq}_i(I_i)$ must be equal to the sum of probabilities of sequences extended by actions a defined in I_i ($a \in A(I_i)$).

Finally, there are complementary slackness conditions (constraints (5)) stating that a sequence σ_i is either never played (i.e., $r_i(\sigma_i) = 0$) or slack variable $s_{\sigma_i} = 0$, which ensures that sequence σ_i is part of the best response for player i ensured by the constraint (2).

Existing Algorithms for Computing SSE

We now focus on the Stackelberg setting and fix the roles of the two players. Player 1 is the leader that commits to a strategy. Player 2 is the follower that plays the best response to the strategy of the leader. The baseline algorithm for computing SSE in single-step normal-form games (NFGs), introduced by (Conitzer and Sandholm 2006), is based on solving multiple linear programs – for every pure strategy of the follower $\pi_2 \in \Pi_2$ we can compute a mixed strategy for the leader $\delta_1^{\pi_2}$ such that (1) playing π_2 is the best response of the follower against $\delta_1^{\pi_2}$ (i.e., $BR_2(\delta_1^{\pi_2}) = \pi_2$) and (2) $\delta_1^{\pi_2}$ is such that it maximizes expected utility of the leader. Such a mixed strategy $\delta_1^{\pi_2}$ can be found using linear program; hence, the baseline algorithm calculates $|\Pi_2|$ linear programs, and for each pure strategy of the leader the algorithm computes $\delta_1^{\pi_2}$ and the expected utility of the leader $u_1(\delta_1^{\pi_2}, \pi_2)$. Finally the algorithm selects such strategy profile $(\delta_1^{\pi_2^*}, \pi_2^*)$ for which the expected utility u_1 is maximal. Often, certain pure strategies of the follower can never be best responses (e.g., strictly dominated strategies), in which case the LPs for these pure strategies are not feasible.

Follow-up works primarily focused on the Bayesian setting, where the leader is playing against one of possible followers with different preferences. Although the work (Conitzer and Sandholm 2006) showed that finding SSE in Bayesian games is NP-hard, new algorithms were able to scale to real-world scenarios. First successful algorithm was based on mixed-integer linear programming (MILP) introduced in (Paruchuri et al. 2008). The main advantage of the MILP formulation is in avoiding the exponential Harsanyi transformation of a Bayesian game into a normal-form game. New algorithms that followed were inspired by column/constraint generation techniques that restrict the number of linear programs to be solved in multiple LPs approach exploiting hierarchical consideration of the types of the follower (Jain, Tambe, and Kiekintveld 2011), and more tight calculation of bounds by using convex hull relaxation and Bender's decomposition in (Yin and Tambe 2012). Large volume of works focused on more specific game models including security games (Kiekintveld et al. 2009; Tambe 2011), or patrolling games (Basilico, Gatti, and Amigoni 2009; Vorobeychik, An, and Tambe 2012).

Described baseline algorithms could be, in principle, applied also for solving EFGs – we can transform any EFG into a NFG using the concept of pure and mixed strategies. However, since the number of pure strategies is exponential, the baseline algorithm would have to solve exponentially many LPs (one LP for each pure strategy of the follower). Moreover, to ensure that the currently fixed strategy of the follower is the best response, each LP would be of an exponential size (exponentially many constraints, one constraint for each pure strategy of the follower). Therefore, the scalability of such approach is very limited.

In the next section we therefore introduce a new algorithm that builds on the existing algorithms for computing SSE, however, that directly exploits the compact representation of strategies using the sequence form.

Sequence-Form for SSE in EFGs

We first describe the modification of the baseline algorithm for computing Strong Stackelberg Equilibrium (SSE) based on solving multiple LPs. We introduce a novel LP that for a fixed pure strategy of the follower π_2 computes a mixed strategy δ_1 satisfying 2 conditions required for the SSE: (1) π_2 is the best response to δ_1 and (2) δ_1 is such that the expected utility for the leader is maximal. Afterwards we extend this LP and formulate a MILP for computing SSE in EFGs. For each algorithm, we first describe the key ideas leading to the formulation, following by the complete listing of all the constraints of the mathematical programs.

Multiple Sequence-Form LPs for SSE

Our algorithm exploits the compact sequence form and represents the strategy of both players as realization plans. First, the realization plan of the leader r_1 needs to satisfy the network-flow constraints (3-4). Next, let us denote Σ_2^{BR} sequences of the follower that correspond to the pure realization plan of the follower (i.e., the sequences that are played with probability 1 in the fixed pure realization plan of the follower π_2). The algorithm needs to ensure that the realization plan of the leader is such that the sequences in Σ_2^{BR} are the best response to r_1. To do this, the algorithm exploits the constraint (2); there will be one such constraint for each sequence of the follower $\sigma_2 \in \Sigma_2$. To ensure that sequences from Σ_2^{BR} form the best response, we strictly set slack variables s_{σ_2} to be equal to zero for sequences in Σ_2^{BR}. Such tightening of the slack variables causes the expected utility value for a certain information set to be equal to the expected utility gained by playing actions corresponding to sequences in Σ_2^{BR}. More precisely, consider a sequence $\sigma_2 \in \Sigma_2^{BR}$, where the last action of the sequence, a, is applicable in information set $I = \mathsf{inf}_2(\sigma_2)$ (i.e., $\exists \sigma_2' \in \Sigma_2^{BR}$ such that $\sigma_2 = \sigma_2' a$). Now, if a slack variable s_{σ_2} is forced to be zero, then it holds

$$v_I = \sum_{I_2' \in \mathcal{I}_2 : \mathsf{seq}_2(I_2') = \sigma_2} v_{I_2'} + \sum_{\sigma_1 \in \Sigma_1} g_2(\sigma_2, \sigma_1) \cdot r_1(\sigma_1) \qquad (7)$$

For any other action applicable in I (i.e., $\forall b \in A(I)\ a \neq b$) the constraints for sequences $\sigma_2' b$ have a positive slack variable. Therefore, the realization plan of the leader r_1 must be such that the expected value of the right side in this constraint gained by actions b must be less or equal to v_I because the expected value of the right side of the constraint can be only increased with a positive slack variable. This restriction, however, corresponds to the fact that action a is best to be played in information set I. Since this holds for all sequences in Σ_2^{BR} and these sequences correspond to a pure realization plan for the follower, the instantiations of constraints (2) restrict the strategy of the leader r_1 such that Σ_2^{BR} is the best response of the follower.

Finally, we need to specify the objective function. Since we have variables representing the realization plan of the leader r_1 and fixed set of sequences of the opponent Σ_2^{BR}, we can simply calculate an expected utility for the leader using function g_1 only for the sequences in Σ_2^{BR}. We thus arrive to the final formulation of the linear program for computing the desired realization plan for the leader:

$$\max_{r_1, v_I, s_\sigma} \sum_{\sigma_1 \in \Sigma_1} \sum_{\sigma_2 \in \Sigma_2^{BR}} r_1(\sigma_1) g_1(\sigma_1, \sigma_2) \qquad (8)$$

$$v_{\mathsf{inf}_2(\sigma_2)} = s_{\sigma_2} + \sum_{I' \in \mathcal{I}_2 : \mathsf{seq}_2(I') = \sigma_2} v_{I'} + \sum_{\sigma_1 \in \Sigma_1} r_1(\sigma_1) g_2(\sigma_1, \sigma_2) \quad \forall \sigma_2 \in \Sigma_2 \qquad (9)$$

$$r_1(\emptyset) = 1 \qquad (10)$$

$$0 \leq r_1(\sigma_1) \quad \forall \sigma_1 \in \Sigma_1 \qquad (11)$$

$$r_1(\sigma_1) = \sum_{a \in A(I_1)} r_1(\sigma_1 a) \quad \forall I_1 \in \mathcal{I}_1, \sigma_1 = \mathsf{seq}_1(I_1) \qquad (12)$$

$$0 \leq s_{\sigma_2} \quad \forall \sigma_2 \in \Sigma_2 \qquad (13)$$

$$0 = s_{\sigma_2} \quad \forall \sigma_2 \in \Sigma_2^{BR} \qquad (14)$$

Note that the sequences in Σ_2^{BR} must form a complete pure realization plan of the follower; hence, the set must be non-empty (empty sequence is always in Σ_2^{BR}) and every sequence in Σ_2^{BR} must also have a continuation sequence in Σ_2^{BR} if possible. Formally, if there is a sequence $\sigma_2 \in \Sigma_2^{BR}$ and there exists an information set $I \in \mathcal{I}_2$ such that $\mathsf{seq}_2(I) = \sigma_2$ then there must exist exactly one sequence $\sigma_2' \in \Sigma_2^{BR}$ such that $\mathsf{inf}_2(\sigma_2') = I$. This condition ensures that for every information set reachable under the assumption the follower is playing realization plan corresponding to Σ_2^{BR}, there is at least one slack variable forced to be zero. Violating this condition causes the LP to return incorrect results since values v_I could be arbitrarily increased using slack variables without restricting the realization plan of the leader.

By exploiting the sequence-form representation we avoided the exponential number of constraints. Instead of having one constraint for each possible strategy (to ensure that currently selected is the best response), we have a single constraint for each sequence of the follower. The presented LP is thus of a linear size to the size of the game tree. However, the algorithm would still need to solve exponentially many of such LPs (one for each pure realization plan of the follower) to compute SSE. To further improve the algorithm, we can exploit the reformulation using mixed-integer linear programming (MILP).

Sequence-Form MILP for Computing SSE

Similarly to work in Bayesian games (Paruchuri et al. 2008), we can reformulate the problem of solving Stackelberg equilibrium as a MILP. Since the best response of the follower is only in pure strategies, we can represent the strategy of the follower as a pure realization plan with binary variables $r_2 : \Sigma_2 \to \{0, 1\}$. The realization plans are restricted as usual and the network-flow constraints (3-4) apply. We again use the same idea of enforcing certain slack variables s_{σ_2} to be equal to zero for the sequences used in the realization plan of the follower. However, since the realization plan of the follower is not fixed but it is represented with variables r_2, we need to force slack variables $s_{\sigma_2} = 0$ whenever $r_2(\sigma_2)$ equals to 1. Since r_2 are binary variables, we can use the following constraint to ensure this:

$$0 \leq s_{\sigma_2} \leq (1 - r_2(\sigma_2)) \cdot M \qquad (15)$$

where M is a large constant. Now, using the constraint (9) we restrict variables r_1 and r_2 such that $r_2 = BR_2(r_1)$.

Finally, we need to modify the objective function, since the strategy of the follower is no longer fixed. The main idea is to use a new variable p that semantically corresponds to the probability distribution over leafs in the game tree considered the strategy of both players. Such a probability corresponds to a multiplication of variables r_1 and r_2. However, since the r_2 are binary variables we do not need to use multiplication. Consider a leaf $z \in Z$ and assume sequences σ_1 and σ_2 lead to this leaf for the leader and the follower respectively (i.e., by executing the actions in these sequences, leaf z is reached). Then, the value $p(z)$ is either equal to $r_1(\sigma_1)$ (when the follower plays σ_2 with probability 1), or it is equal to 0 otherwise (the follower is not playing σ_2). We can ensure this behavior with a set of linear constraints (see below, constraints (21-23)). Putting all together, we arrive to the formulation of the problem of computing SSE for EFGs as a MILP (note, that the presented MILP has linear size to the size of the game tree, with only $|\Sigma_2|$ binary variables):

$$\max_{p,r,v,s} \sum_{z \in Z} p(z) u_1(z) \mathcal{C}(z) \tag{16}$$

$$v_{\mathsf{inf}_2(\sigma_2)} = s_{\sigma_2} + \sum_{I' \in \mathcal{I}_2 : \mathsf{seq}_2(I') = \sigma_2} v_{I'} + \sum_{\sigma_1 \in \Sigma_1} r_1(\sigma_1) g_2(\sigma_1, \sigma_2) \quad \forall \sigma_2 \in \Sigma_2 \tag{17}$$

$$r_i(\emptyset) = 1 \qquad \forall i \in N \tag{18}$$

$$r_i(\sigma_i) = \sum_{a \in A_i(I_i)} r_i(\sigma_i a) \quad \forall i \in N \; \forall I_i \in \mathcal{I}_i, \sigma_i = \mathsf{seq}_i(I_i) \tag{19}$$

$$0 \leq s_{\sigma_2} \leq (1 - r_2(\sigma_2)) \cdot M \qquad \forall \sigma_2 \in \Sigma_2 \tag{20}$$

$$0 \leq p(z) \leq r_2(\mathsf{seq}_2(z)) \qquad \forall z \in Z \tag{21}$$

$$0 \leq p(z) \leq r_1(\mathsf{seq}_1(z)) \qquad \forall z \in Z \tag{22}$$

$$1 = \sum_{z \in Z} p(z) \mathcal{C}(z) \tag{23}$$

$$r_2(\sigma_2) \in \{0, 1\} \qquad \forall \sigma_2 \in \Sigma_2 \tag{24}$$

$$0 \leq r_1(\sigma_1) \leq 1 \qquad \forall \sigma_1 \in \Sigma_1 \tag{25}$$

Experiments

We now turn to the experimental evaluation to demonstrate the scalability of our algorithm. We evaluate both variants of our algorithm – first variant with multiple LPs (denoted MULTILP), and the second variant with MILP – against the baseline approaches based on the transformation of EFGs into the normal form, denoted as EXPMULTILP (Conitzer and Sandholm 2006) and DOBBS (Paruchuri et al. 2008).

The main factor determining the complexity of the game is the size of the strategy space of the follower, since it plays a crucial part in the algorithms – one LP is solved for each pure realization plan of the follower in MULTILP, and the number of binary variables depends on the number of sequences of the follower in MILP. We analyze the scalability of the algorithms in two different settings: (1) a security-inspired search game (inspired by (Bosansky et al. 2013)) with a very large strategy space of the leader, while the strategy space of the follower is very small, and (2) randomly generated extensive-form games. While the first scenario allows us to scale to large game trees, the strategy space is more balanced in the second scenario and we analyze the scalability with respect to the increasing number of realization plans of the follower.

Experiment Settings

We use a domain-independent implementation of all algorithms and IBM CPLEX 12.5 for solving LPs and MILPs.

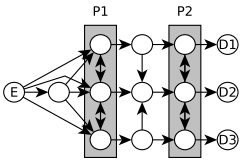
Search Game The search game is played on a directed graph (see Figure 1). The follower aims to reach one of the destination nodes (D1 – D3) from starting node (E), while the leader aims to encounter the follower with one of the two units operating in the shaded areas of the graph (P1 and P2). The follower receives different reward for reaching different destination node (the reward is randomly selected from interval $[1, 2]$). The leader receives positive reward 1 for capturing the follower. The follower leaves tracks in the visited nodes that can be discovered if the leader visits the node later in the game, but the follower can decide to erase the tracks in the current node (it takes one turn of the game).

Randomly Generated Games We use randomly generated games, where in each state of the game the number of available actions is randomly generated up to a given parameter $\{2, \ldots, \max_A\}$. Each action leads to a state where the opponent is to move and also generates *an observation* for the opponent. Observation is a number from a limited set $\{1, \ldots, \max_O\}$ and determines partitioning of the nodes into the information sets – for player i, the nodes h with the same history of moves $\mathsf{seq}_i(h)$ and the observations generated by the actions of the opponent $-i$ belong to the same information set. We generate the games of differing sizes by varying parameters $\max_A = \{3, 4, 5\}$, $\max_O = \{2, 3, 4\}$, and depth of the game (up to 5 actions for each player). The utility values for players in leafs are assigned randomly from interval $[-5, 5]$. Finally, we are also interested in the effects of the correlation between the utility values on the performance of the algorithms (i.e., the utility values are generated to have a certain correlation factor; correlation -1 corresponds to zero-sum games, 1 to identical utility functions).

Results

Search Game The results on the search game (see the table in Figure 1) show that our algorithm outperforms the baseline approaches by several orders of magnitude. We scale the game by increasing the number of steps in the game. All algorithms were able to find the solution for 5 steps – it took EXPMULTILP more than 2 hours to compute the solution, while our MILP solved the game in slightly over 11 minutes. Making another step was possible only for our MILP that solved the largest instance in 6.3 hours. The size of this instance was 8.6×10^5 nodes with 470 pure realization plans of the follower. This results demonstrate the ability of using our MILP algorithm in sequential security scenarios where a large strategy space of the leader, caused by deploying and scheduling multiple resources, is more common.

Randomly Generated Games The results on random games confirm the dominance of our algorithm (see Figure 2). The left graph compares computation times of our



Algs \ Steps	4	5	6
ExpMultiLP	89	7,764	-
DOBBS	122	3,067	-
MultiLP	41	2,485	-
MILP	19	676	22,847

Figure 1: (Left) Graph for the search game, where the leader operates two units in the shaded areas, while the follower is planning to cross the area unobserved. (Right) Computation times (in seconds) for the search game with increasing number of steps in the game.

variants with the baseline approaches. Reported values are means out of 20 games, the standard error was always very small compared to the mean value and the differences between the algorithms (not visible in the graph). The results show that even a slight increase in the number of realization plans causes the baseline approaches to perform extremely slowly (note the logarithmic y-scale). Even relatively small games with 4000 realization plans of the follower take ExpMultiLP more than 80 minutes to solve, while our MILP solves such games in 225 miliseconds.

Therefore we further scaled the games comparing only our variants (the right graph in Figure 2, note that both scales are logarithmic). The depicted results are means of at least 100 different games, the standard error was again marginal. The results show that MILP is on average at least a magnitude faster compared to the MultiLP variant. The largest instances MILP was able to solve contained more than 10^7 realization plans of the follower taking up to 2 days to solve.

The reported results are for the correlation factor set to -0.5. When the correlation factor is decreased, it is more difficult for the leader to maximize the utility, which is strongly (but not absolutely) negatively correlated to the utility of the follower. With the correlation set to -0.8 the computation time of our MILP increases to 410ms for 4000 realization plans. On the other hand, increasing the correlation factor makes the games easier to solve (games with the correlation set to -0.2 are solved in 193ms by our MILP).

Discussion

The experimental evaluation shows that our novel algorithm dramatically outperforms the baseline approaches based on the transformation of EFGs into the normal form. This advantage is not surprising, however, the experiments show that our MILP algorithm is able to scale to moderately large sequential games with up to 10^5 nodes, or more than 10^6 pure realization plans of the follower, which can be sufficient for some real-world scenarios.

The comparison between the two variants of our algorithm is strongly in favor of the MILP variant. The bottleneck of the MultiLP variant is the evaluation of each LP for every pure realization plan of the follower, since the algorithm currently does not use any pruning techniques, or any iterative methods (e.g., branch-and-price or column/constraint generation methods). Using such techniques is, however, the key behind the success of the state-of-the-art algorithms for solving Bayesian Stackelberg games (Jain, Tambe, and Kiekintveld 2011; Yin and Tambe 2012). Unfortunately, exploiting the same principles in EFGs is not straightforward due to a substantially different reason for the exponential number of the strategies of the follower.

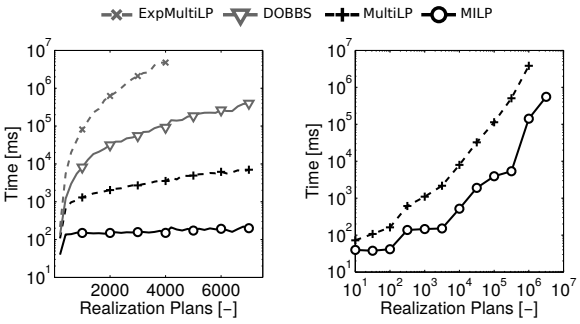


Figure 2: Comparison of the scalability of the exponential baseline approaches and the variants of our algorithm.

The main challenge stems from computing an upper bound on the Stackelberg value for a partially instantiated strategy of the follower (the lower bound is provided by solving a LP for any complete strategy of the follower). Recall that described LPs do not work if the strategy of the follower is not a complete realization plan. In the Bayesian setting, a partially instantiated strategy of the follower corresponds to fixing the strategy for certain types of the follower. This provides a reasonable upper bound, since the remaining unfixed follower types can alter the value only with respect of their probability of appearing. The situation is different in EFGs. Partially fixing the strategy of the follower only restricts the achievable leafs in the game tree, but without further assumptions on the utility structure it does not restrict the possible Stackelberg value for the leader in general. A naïve upper bound, the best response of the leader, provides a very optimistic upper bound that does not improve the computation times of MultiLP enhanced with a branch-and-bound technique.

Conclusion

This paper presents a novel algorithm for computing Strong Stackelberg equilibria in extensive-form games (EFGs) by exploiting the compact sequence-form representation of strategies. We provide two formulations of our algorithm: one based on solving multiple linear programs (LPs), and one based on mixed-integer linear program. Our novel algorithm dramatically outperforms existing approaches based on the exponential transformation of EFGs into the normal form and allow us to solve significantly larger games.

Our work opens two important lines of research. First of all, the presented algorithm can be a basis for creating domain-dependent algorithms to solve large games, for example in the security domain. Second, our variant based on solving multiple LPs can be further improved by using iterative and decomposition techniques. To allow this, a new (or domain-specific) algorithm for computing a tight upper bound on the Stackelberg value in EFGs based on a partially instantiated strategy of the follower must be designed.

Acknowledgments

This research was supported by the Czech Science Foundation (grant no. P202/12/2054). Bosansky also acknowledges support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61361136003) for the Sino-Danish Center for the Theory of Interactive Computation, and the support from the Center for Research in Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council.

References

Basilico, N.; Gatti, N.; and Amigoni, F. 2009. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, 57–64.

Bosansky, B.; Kiekintveld, C.; Lisy, V.; Cermak, J.; and Pechoucek, M. 2013. Double-oracle Algorithm for Computing an Exact Nash Equilibrium in Zero-sum Extensive-form Games. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 335–342.

Conitzer, V., and Sandholm, T. 2006. Computing the optimal strategy to commit to. In *Proceedings of the 7th ACM conference on Electronic commerce*, 82–90. ACM.

Jain, M.; Tambe, M.; and Kiekintveld, C. 2011. Quality-bounded solutions for finite Bayesian Stackelberg games: Scaling up. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, 997–1004.

Karp, R. M. 1971. Reducibility among combinatorial problems. 85–103.

Kiekintveld, C.; Jain, M.; Tsai, J.; Pita, J.; Ordóñez, F.; and Tambe, M. 2009. Computing optimal randomized resource allocations for massive security games. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, 689–696.

Koller, D.; Megiddo, N.; and von Stengel, B. 1996. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior* 14(2):247–259.

Leitmann, G. 1978. On generalized stackelberg strategies. *Optimization Theory and Applications* 26:637–643.

Letchford, J., and Conitzer, V. 2010. Computing optimal strategies to commit to in extensive-form games. In *Proceedings of the 11th ACM conference on Electronic commerce*, 83–92. New York, NY, USA: ACM.

Paruchuri, P.; Pearce, J.; Marecki, J.; Tambe, M.; Ordonez, F.; and Kraus, S. 2008. Playing games for security: an efficient exact algorithm for solving Bayesian Stackelberg games. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, 895–902.

Pita, J.; Jain, M.; Marecki, J.; Ordóñez, F.; Portway, C.; Tambe, M.; Western, C.; Paruchuri, P.; and Kraus, S. 2008. Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles International Airport. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, 125–132.

Shieh, E.; An, B.; Yang, R.; Tambe, M.; Baldwin, C.; Direnzo, J.; Meyer, G.; Baldwin, C. W.; Maule, B. J.; and Meyer, G. R. 2012. PROTECT : A Deployed Game Theoretic System to Protect the Ports of the United States. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 13–20.

Shoham, Y., and Leyton-Brown, K. 2009. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.

Tambe, M. 2011. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press.

Tsai, J.; Rathi, S.; Kiekintveld, C.; Ordóñez, F.; and Tambe, M. 2009. IRIS - A Tool for Strategic Security Allocation in Transportation Networks Categories and Subject Descriptors. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, 37–44.

von Stackelberg, H. 1934. Marktform und gleichgewicht.

von Stengel, B., and Zamir, S. 2004. Leadership with commitment to mixed strategies.

von Stengel, B. 1996. Efficient computation of behavior strategies. *Games and Economic Behavior* 14:220–246.

Vorobeychik, Y.; An, B.; and Tambe, M. 2012. Adversarial patrolling games. In *Proceedings of the AAAI Spring Symposium*.

Yin, Z., and Tambe, M. 2012. A unified method for handling discrete and continuous uncertainty in bayesian stackelberg games. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 855–862.

Yin, Z.; Korzhyk, D.; Kiekintveld, C.; Conitzer, V.; and Tambe, M. 2010. Stackelberg vs. Nash in security games: Interchangeability, equivalence, and uniqueness. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, 1139–1146.

Appendix G

Computation of Stackelberg Equilibria of Finite Sequential Games

Computation of Stackelberg Equilibria of Finite Sequential Games

BRANISLAV BOŠANSKÝ, Czech Technical University in Prague
SIMINA BRÂNZEI, Hebrew University of Jerusalem
KRISTOFFER ARNSFELT HANSEN, Aarhus University
TROELS BJERRE LUND, IT-University of Copenhagen
PETER BRO MILTERSEN, Aarhus University

The Stackelberg equilibrium is a solution concept that describes optimal strategies to commit to: Player 1 (*the leader*) first commits to a strategy that is publicly announced, then Player 2 (*the follower*) plays a best response to the leader's choice. We study the problem of computing Stackelberg equilibria in finite sequential (i.e., extensive-form) games and provide new exact algorithms, approximation algorithms, and hardness results for finding equilibria for several classes of such two-player games.

CCS Concepts: • **Theory of computation → Algorithmic game theory**; **Exact and approximate computation of equilibria**; **Problems, reductions and completeness**;

Additional Key Words and Phrases: Algorithmic game theory, extensive-form games, finite sequential games, stackelberg equilibrium, Extensive-Form Correlated Equilibrium

ACM Reference format:
Branislav Bošanský, Simina Brânzei, Kristoffer Arnsfelt Hansen, Troels Bjerre Lund, and Peter Bro Miltersen. 2017. Computation of Stackelberg Equilibria of Finite Sequential Games. *ACM Trans. Econ. Comput.* 5, 4, Article 23 (December 2017), 24 pages.
https://doi.org/10.1145/3133242

The authors acknowledge support from the Danish National Research Foundation and The National Science Foundation of China CEFC (under the grant 61361136003 CEFC) for the Sino-Danish Center for the Theory of Interactive Computation and from the, supported by the Danish Strategic Research Council. Branislav Bošanský was also supported by the Czech Science Foundation (grant no. 15-23235S). Simina Brânzei was also supported by ISF grant 1435/14 administered by the Israeli Academy of Sciences and Israel-USA Bi-national Science Foundation (BSF) grant 2014389 and the I-CORE Program of the Planning and Budgeting Committee and The Israel Science Foundation. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 740282).
A short version of this article was published in the *Proceedings of WINE 2015*. This manuscript contains all the proofs not previously published, as well as an extended introduction, discussion of related literature, and an extended example.
Authors' addresses: B. Bošanský, Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague; email: bosansky@agents.fel.cvut.cz; S. Brânzei, Hebrew University of Jerusalem; email: simina.branzei@gmail.com; K. A. Hansen, Department of Computer Science, Aarhus University; email: arnsfelt@cs.au.dk; P. B. Miltersen, Department of Computer Science, Aarhus University; email: bromille@cs.au.dk; T. B. Lund, IT-University of Copenhagen; email: trbj@itu.dk.

https://doi.org/10.1145/3133242

1 INTRODUCTION

The Stackelberg competition is a game theoretic model introduced by von Stackelberg (1934) for studying market structures. The original formulation of a Stackelberg duopoly captures the scenario of two firms that compete by selling homogeneous products. One firm—*the leader*—first decides the quantity to sell and announces it publicly, while the second firm—*the follower*—decides its own production only after observing the announcement of the first firm. The leader firm must have commitment power (e.g., holds the monopoly in an industry) and cannot undo its publicly announced strategy, while the follower firm (e.g., a new competitor) plays a best response to the leader's chosen strategy.

The Stackelberg competition has been an important model in economics ever since (see, e.g., (Amir and Grilo 1999; Etro 2007; Hamilton and Slutsky 1990; Matsumura 2003; Sherali 1984; van Damme and Hurkens 1999)), while the solution concept of a *Stackelberg equilibrium* has been studied in a rich body of literature in computer science, with a number of important real-world applications developed in the past decade (Tambe 2011). The Stackelberg equilibrium concept can be applied to any game with two players (e.g., in normal or extensive form) and stipulates that the leader first commits to a strategy, while the follower observes the leader's choice and best responds to it. The leader must have commitment power; in the context of firms, the act of moving first in an industry, such as by opening a shop, requires financial investment and is evidently a form of commitment. In other scenarios, the leader's commitment refers to ways of responding to future events should certain situations be reached, and, in such cases, the leader must have a way of enforcing credible threats. The leader can always commit to a Nash equilibrium strategy; however, it can often obtain a better payoff by choosing some other strategy profile. We focus on the two-player setting with one leader and one follower since this setting has attained the most attention in the real-world applications. One of the reason is that computing a Stackelberg equilibrium in a multiplayer game with three or more players requires finding a specific Nash equilibrium in a general-sum game among the followers, which is already a computationally hard task.

One of the notable applications using the conceptual framework of Stackelberg equilibrium has been the development of algorithms for protecting airports and ports in the United States (deployed so far in Boston, Los Angeles, New York). More recent ongoing work (Nguyen et al. 2015), explores additional problems such as protecting wildlife, forests, and fisheries. The general task of defending valuable resources against attacks can be cast in the Stackelberg equilibrium model as follows. The role of the leader is taken by the defender (e.g., police forces), who commits to a strategy, such as the allocation of staff members to a patroling schedule of locations to check. The role of the follower is played by a potential attacker, who monitors the empirical distribution (or even the entire schedule) of the strategy chosen by the defender and then best responds by devising an optimal attack given this knowledge. The crucial question is how to minimize the damage from potential threats by computing an optimal schedule for the defender. Solving this problem in practice involves several nontrivial steps, such as estimating the payoffs of the participants for the resources involved (e.g., the attacker's reward for destroying a section of an airport) and computing the optimal strategy that the defender should commit to. While the applied models do not consider strategic sequential interactions among the players, new applications may appear with the development of new scalable algorithms. In fact, many currently modeled scenarios are essentially sequential: Police forces can commit to a security protocol describing not only their allocation to targets but also their strategy in case of an attack (or some other event), and this results in a strategic response. Similarly, sequential models would allow rangers in national parks to react to immediate observations made in the field.

In this article, we are interested in the following fundamental question:

Table 1. Overview of the Computational Complexity Results Containing Both Existing and New Results Provided by This Article (Marked with *)

	Inf.	Chance	Graph	Str.	Complexity	Source
1.*	TB	✗	DAG	P	$O(\|\mathcal{S}\| \cdot (\|\mathcal{S}\| + \|\mathcal{Z}\|))$	*Theorem 3.1*
2.	TB	✗	Tree	B	$O\left(\|\mathcal{S}\| \cdot \|\mathcal{Z}\|^2\right)$	(Letchford and Conitzer 2010)
3.*	TB	✗	Tree	C	$O(\|\mathcal{S}\| \cdot \|\mathcal{Z}\|)$	*Theorem 3.2*
4.	TB	✓	Tree	P/B	NP-hard	(Letchford and Conitzer 2010)
5.*	TB	✓	Tree	P	FPTAS	*Theorem 5.3*
6.*	TB	✓	Tree	B	FPTAS	*Theorem 5.1*
7.*	TB	✓	Tree	C	$O(\|\mathcal{S}\| \cdot \|\mathcal{Z}\|)$	*Theorem 3.3*
8.*	CM	✗	Tree	B	NP-hard	*Theorem 4.1*
9.*	CM	✓	Tree	C	polynomial	*Theorem 4.2*

Information column: TB stands for *Turn-based* and CM for *Concurrent Moves*. Strategies: P stands for *pure*, B for *Behavioral*, and C for *Correlated*. Finally, $|\mathcal{S}|$ denotes the number of decision points in the game and $|\mathcal{Z}|$ the number of terminal states.

> *Given the description of a game in extensive form, what is the optimal Stackelberg strategy that the leader should commit to?*

We study this problem for multiple classes of two-player extensive-form games (EFGs) and variants of the Stackelberg solution concept that differ in kinds of strategies to commit to, and we provide both efficient algorithms and computational hardness results. We emphasize the positive results in the main text of the article and fully state technical hardness results in the appendix.

1.1 Our Results

The problem of computing a Stackelberg equilibrium in EFGs can be classified by the following parameters:

- *Information.* Information captures how much a player knows about the opponent's moves (past and present). We study *turn-based games* (TB), where for each state there is a unique player that can perform an action, and *concurrent-move games* (CM), where the players act simultaneously in at least one state.
- *Chance.* A game with chance nodes allows stochastic transitions between states; otherwise, the transitions are deterministic (made through actions of the players).
- *Graph.* We focus on *trees* and *directed acyclic graphs* (DAGs) as the main representations. Given such a graph, each node represents a different state in the game, while the edges represent the transitions between states.
- *Strategies.* We study several major types of strategies that the leader can commit to, namely *pure* (P), *behavioral* (B), and *correlated behavioral* (C).

The results are summarized in Table 1 and can be divided into three categories.[1]

First, we design a more efficient algorithm for computing optimal pure strategies to commit to for turn-based games on DAGs. Compared to the previous state of the art (due to Letchford and Conitzer (2010), (Letchford 2013)), we reduce the complexity by a factor proportional to the

[1]We stated a theorem for NP-hardness for the correlated case on DAGs that was similar to the original theorem for behavioral strategies (Letchford 2013) in an earlier version of this article. Due to an error, the theorem has not been correctly proven, and the computational complexity for this case (i.e., computing optimal correlated strategies to commit to on DAGs) remains currently open.

number of terminal states (see row 1 in Table 1). The main idea behind the improvement is the exploitation of algorithms for solving the widest-path problem on a DAG from a single source to multiple destinations. Note that the problem of computing mixed behavioral strategies to commit to is shown to be NP-hard due to a reduction from SAT (Letchford 2013).

Second, we investigate the impact of commitments to correlated strategies. In this setting, the leader can send the follower signals about which action the follower should play, and following these signals must be a best response for the follower. Contrary to behavioral strategies where the best response of the follower is a pure strategy, the leader can randomize among the signals (and thus best responses of the follower). This significantly modifies the space of strategies that can also affect the computational complexity of the problem. Indeed, we show that correlation often reduces the computational complexity of finding optimal strategies, and we design several new polynomial-time algorithms for computing the optimal correlated strategy to commit to for both turn-based and concurrent-move games (see rows 3, 7, 9).

Third, we study approximation algorithms for the NP-hard problems in this framework and provide fully polynomial-time approximation schemes (FPTAS) for finding pure and behavioral Stackelberg equilibria for turn-based games on trees with chance nodes (see rows 5, 6). The hardness proof uses a reduction from Knapsack (Letchford and Conitzer 2010), and we exploit this connection since our algorithm resembles the classical approximation scheme for Knapsack. We leave open the question of finding an approximation for concurrent-move games on trees without chance nodes for which we have only the negative result (see row 8).

1.2 Related Work

There is a rich body of literature studying the problem of computing Stackelberg equilibria. The computational complexity of the problem is known for one-shot games (Conitzer and Sandholm 2006), Bayesian games (Conitzer and Sandholm 2006), and selected subclasses of extensive-form games (Letchford and Conitzer 2010) and infinite stochastic games (Gupta 2015; Gupta et al. 2015; Letchford et al. 2012). Similarly, many practical algorithms are also known and typically based on solving multiple linear programs (Conitzer and Sandholm 2006), or mixed-integer linear programs for Bayesian (Paruchuri et al. 2008) and extensive-form games (Bošanský and Čermák 2015).

For one-shot games, the problem of computing a Stackelberg equilibrium is polynomial (Conitzer and Sandholm 2006) in contrast to the PPAD-completeness of a Nash equilibrium (Chen et al. 2009; Daskalakis et al. 2006b). The situation changes in extensive-form games where Letchford and Conitzer showed (2010) that for many cases the problem is NP-hard, while it still remains PPAD-complete for a Nash equilibrium (Daskalakis et al. 2006a). More specifically, computing Stackelberg equilibria is polynomial only for:

- games with perfect information with no chance on DAGs, where the leader commits to a pure strategy,
- games with perfect information with no chance on trees.

Introducing chance or imperfect information leads to NP-hardness. However, several cases were unexplored by the existing work; namely, extensive-form games with perfect information and concurrent moves. We address this subclass in this work.

The computational complexity can also change when the leader commits to correlated strategies. This extension of the Stackelberg notion to correlated strategies appeared in several works (Conitzer and Korzhyk 2011; Letchford et al. 2012; Xu et al. 2015). Conitzer and Korzhyk (2011) analyzed correlated strategies in one-shot games providing a single linear program for their computation. Letchford et al. (2012) showed that the problem of finding optimal correlated

strategies to commit to is NP-hard in infinite discounted stochastic games.[2] Xu et al. (2015) focused on using correlated strategies in a real-world security-based scenario.

The detailed analysis of the impact when the leader can commit to correlated strategies has, however, not been investigated sufficiently in the existing work. We address this extension and study the complexity for multiple subclasses of extensive-form games. Our results show that, for many, cases the problem of computing Stackelberg equilibria in correlated strategies is polynomial compared to the NP-hardness in behavioral strategies. Finally, these theoretical results have also practical algorithmic implications. An algorithm that computes a Stackelberg equilibrium in correlated strategies can be used to compute a Stackelberg equilibrium in behavioral strategies, allowing a significant speed-up in computation time (Čermák et al. 2016).

2 PRELIMINARIES

We consider finite two-player sequential games. Note that for every finite set K, $\Delta(K)$ denotes probability distributions over K and $\mathcal{P}(K)$ denotes the set of all subsets of K.

Definition 2.1 (2-player sequential game). A *two-player sequential game* is given by a tuple $G = (\mathcal{N}, \mathcal{S}, \mathcal{Z}, \rho, \mathcal{A}, u, \mathcal{T}, C)$, where:

- $\mathcal{N} = \{1, 2\}$ is a set of two players;
- $\mathcal{S}$ is a set of nonterminal states;
- $\mathcal{Z}$ is a set of terminal states;
- $\rho : \mathcal{S} \to \mathcal{P}(\mathcal{N}) \cup \{c\}$ is a function that defines which player(s) act in a given state, or whether the node is a chance node (case in which $\rho(s) = c$);
- $\mathcal{A}$ is a set of actions; we overload the notation to restrict the actions only for a single player as $\mathcal{A}_i$ and for a single state as $\mathcal{A}(s)$;
- $\mathcal{T} : \mathcal{S} \times \prod_{i \in \rho(s)} \mathcal{A}_i \to \{\mathcal{S} \cup \mathcal{Z}\}$ is a transition function between states depending on the actions taken by all the players that act in this state. Overloading notation, $\mathcal{T}(s)$ also denotes the children of a state s: $\mathcal{T}(s) = \{s' \in \mathcal{S} \cup \mathcal{Z} \mid \exists a \in \mathcal{A}(s);\ \mathcal{T}(s, a) = s'\}$;
- $C : \mathcal{A}_c \to [0, 1]$ are the chance probabilities on the edges outgoing from each chance node $s \in \mathcal{S}$, such that $\sum_{a \in \mathcal{A}_c(s)} C(a) = 1$;
- Finally, $u_i : \mathcal{Z} \to \mathbb{R}$ is the utility function for player $i \in \mathcal{N}$.

In this article, we study Stackelberg equilibria; thus, player 1 will be referred to as the *leader* and player 2 as the *follower*.

We say that a game is *turn-based* if there is a unique player acting in each state (formally, $|\rho(s)| = 1\ \forall s \in \mathcal{S}$) and with *concurrent moves* if both players can act simultaneously in some state. Moreover, the game is said to have *no chance* if there exist no chance nodes; otherwise, the game is *with chance*.

A *pure strategy* $\pi_i \in \Pi_i$ of a player $i \in \mathcal{N}$ is an assignment of an action to play in each state of the game ($\pi_i : \mathcal{S} \to \mathcal{A}_i$), and Π_i denotes the set of all pure strategies of player i. A *behavioral strategy* $\sigma_i \in \Sigma_i$ is a probability distribution over actions in each state $\sigma_i : \mathcal{A} \to [0, 1]$ such that $\forall s \in S, \forall i \in \rho(s)\ \sum_{a \in \mathcal{A}_i(s)} \sigma_i(a) = 1$, Σ_i denotes the set of all behavioral strategies of player i.

The expected utility of player i given a pair of strategies (σ_1, σ_2) is defined as follows:

$$u_i(\sigma_1, \sigma_2) = \sum_{z \in \mathcal{Z}} u_i(z) p_\sigma(z),$$

where $p_\sigma(z)$ denotes the probability that leaf z will be reached if both players follow the strategy from σ and due to stochastic transitions corresponding to C.

[2] More precisely, that work assumes that the correlated strategies can use a finite history.

A strategy σ_i of player i is said to represent a *best response* to the opponent's strategy σ_{-i} if $u_i(\sigma_i, \sigma_{-i}) \geq u_i(\sigma_i', \sigma_{-i})\ \forall \sigma_i' \in \Sigma_i$. Denote by $\mathcal{BR}(\sigma_{-i}) \subseteq \Pi_i$ the set of all the pure best responses of player i to strategy σ_{-i}. We can now introduce formally the Stackelberg Equilibrium solution concept:

Definition 2.2 (Stackelberg Equilibrium). A strategy profile $\sigma = (\sigma_1, \sigma_2)$ is a *Stackelberg Equilibrium* if σ_1 is an optimal strategy of the leader given that the follower best-responds to its choice. Formally, a Stackelberg equilibrium in *pure* strategies is defined as

$$(\sigma_1, \sigma_2) = \underset{\sigma_1' \in \Pi_1, \sigma_2' \in \mathcal{BR}(\sigma_1')}{\arg\max} u_1(\sigma_1', \sigma_2')$$

while a Stackelberg equilibrium in *behavioral* strategies is defined as

$$(\sigma_1, \sigma_2) = \underset{\sigma_1' \in \Sigma_1, \sigma_2' \in \mathcal{BR}(\sigma_1')}{\arg\max} u_1(\sigma_1', \sigma_2').$$

Next, we describe the notion of a Stackelberg equilibrium where the leader can commit to a correlated strategy in a sequential game. The concept was suggested and investigated by Letchford et al. (Letchford et al. 2012), but no formal definition exists. Formalizing such a definition here, we observe that the definition is essentially the "Stackelberg analogue" of the notion of *Extensive-Form Correlated Equilibria (EFCE)* introduced by von Stengel and Forges (2008). This parallel turns out to be technically relevant as well.

Definition 2.3 (Stackelberg Extensive-Form Correlated Equilibrium). A probability distribution ϕ on pure strategy profiles Π is called a *Stackelberg Extensive-Form Correlated Equilibrium (SEFCE)* if it maximizes the leader's utility (that is, $\phi = \arg\max_{\phi' \in \Delta(\Pi)} u_1(\phi')$) subject to the constraint that whenever the play reaches a state s where the follower can act, the follower is recommended an action a according to ϕ such that the follower cannot gain by unilaterally deviating from a in state s (and possibly in all succeeding states), given the posterior on the probability distribution of the strategy of the leader, defined by the actions taken by the leader so far.

The variants of the Stackelberg solution concept with pure and behavioral strategies are guaranteed to exist since we assume that the follower breaks ties in favor of the leader (von Stengel and Zamir 2010). The existence of the correlated variant is guaranteed by the existence of EFCE (von Stengel and Forges 2008). We give an example to illustrate the variants of the Stackelberg solution concept.

Example 2.4. Consider the game in Figure 1, where the follower moves first (in states s_1, s_2) and the leader second (in states s_3, s_4). By committing to a behavioral strategy, the leader can gain utility 1 in the optimal case: Leader commits to play *left* in state s_3 and *right* in s_4. The follower will then prefer playing *right* in s_2 and *left* in s_1, reaching the leaf with utilities $(1, 3)$. Note that the leader cannot gain more by committing to strictly mixed behavioral strategies.

Now, consider the case when the leader commits to correlated strategies. We interpret the probability distribution over strategy profiles ϕ as signals sent to the follower in each node where the follower acts, whereas the leader is committing to play with respect to ϕ and the signals sent to the follower. This can be shown in node s_2, where the leader sends one of two signals to the follower, each with probability 0.5. In the first case, the follower receives the signal to move *left*, while the leader commits to play the uniform strategy in s_3 and action *left* in s_4, reaching the utility value $(2, 1)$ if the follower plays according to the signal. In the second case, the follower receives the signal to move *right*, while the leader commits to play *right* in s_4 and *left* in s_3, reaching the utility value $(1, 3)$ if the follower plays according to the signal. By using this correlation, the leader is

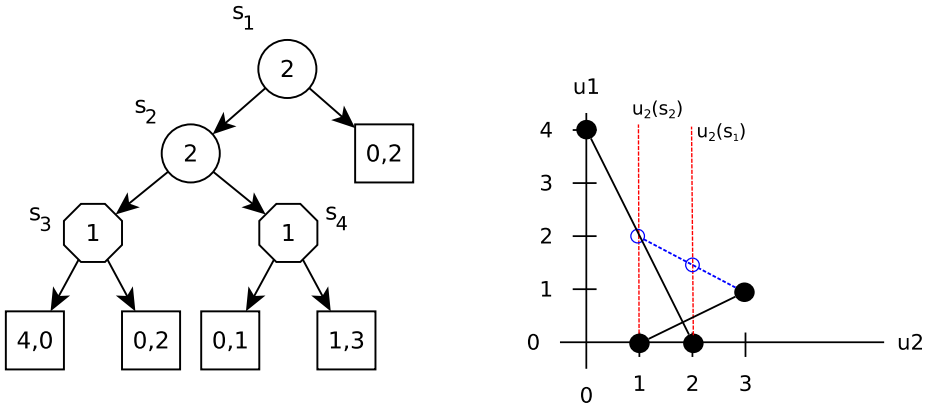


Fig. 1. (Left) An example game with different outcomes depending on whether the leader commits to behavioral or to correlated strategies. The leader acts in nodes s_3 and s_4, the follower acts in nodes s_1 and s_2. Utility values are shown in the terminal states: First value is the utility for the leader, second value is the utility of the follower. (Right) A visualization of the outcomes of the example game in the two-dimensional utility space of the players: The horizontal axis corresponds to the utility of the follower, the vertical axis corresponds to the utility of the leader. Red vertical lines visualize the minimal value the follower can guarantee in s_2 or s_1, respectively; blue lines and points correspond to new outcomes that can be achieved if the leader commits to correlated strategies.

able to get the utility of 1.5, while ensuring the utility of 2 for the follower; hence, the follower will follow the only recommendation in node s_1 to play *left*.

The situation can be visualized using a two-dimensional space, where the x-axis represents the utility of the follower and the y-axis represents the utility of the leader. This type of visualization was also used in Letchford and Conitzer (2010), and we use it further in the proof of Theorem 3.2. While the black nodes correspond to the utility points of the leafs, the solid black lines correspond to outcomes when the leader randomizes between the leafs. The follower plays a best-response action in each node; hence, in order to force the follower to play action *left* in s_2, the leader must guarantee the follower the utility of at least 1 in the subgame rooted in node s_3 since the follower can get at least this value by playing *right* in s_2. Therefore, each state of the follower restricts the set of possible outcomes of the game. These restrictions are visualized as the vertical dashed lines: One corresponds to the described situation in node s_2 and the second one due to the leaf following node s_1. Considering only commitments to behavioral strategies, the best of all possible outcomes for the leader is the point $(u_2 = 3, u_1 = 1)$. With correlation, however, the leader can achieve a mixture of points $(u_2 = 1, u_1 = 2)$ and $(u_2 = 3, u_1 = 1)$ (the blue dashed line). This can also be interpreted as forming a convex hull over all possible outcomes in the subtree rooted in node s_2. Note, that without correlation, the set of all possible outcomes is not generally a convex set. Finally, after restricting this set of possible solutions due to leaf in node s_1, the intersection point $(u_2 = 2, u_1 = 1.5)$ represents the expected utility for the Stackelberg Extensive-Form Correlated Equilibrium solution concept.

The example gives an intuition about the structure of the probability distribution ϕ in SEFCE. In each state of the follower, the leader sends a signal to the follower and commits to follow the correlated strategy if the follower admits the recommendation while simultaneously committing to punish the follower for each deviation. This punishment is simply a strategy that minimizes the follower's utility and will be useful in many proofs; next, we introduce some notation for it.

Let σ^m denote a behavioral strategy profile where, in each subgame, the leader plays a minmax behavior strategy based on the utilities of the follower, and the follower plays a best response.

Moreover, for each state $s \in \mathcal{S}$, we denote by $\mu(s)$ the expected utility of the follower in the subgame rooted in state s if both players play according to σ^m (i.e., the value of the corresponding zero-sum subgame defined by the utilities of the follower).

Note that being a probability distribution over pure strategy profiles, a SEFCE is, a priori, an object of exponential size in the size of the description of the game when it is described as a tree. This has to be dealt with before we can consider computing it. The following lemma gives a compact representation of the correlated strategies in a SEFCE, and the proof yields an algorithm for constructing the probability distribution ϕ from the compact representation. It is this compact representation that we seek to compute.

Lemma 2.5. *For any turn-based or concurrent-move game in tree form, there exists a SEFCE $\phi \in \Delta(\Pi)$ that can be compactly represented as a behavioral strategy profile $\sigma = (\sigma_1, \sigma_2)$ such that $\forall z \in \mathcal{Z}\ p_\phi(z) = p_\sigma(z)$ and ϕ corresponds to the following behavior:*

- *the follower receives signals in each state s according to $\sigma_2(a)$ for each action $a \in \mathcal{A}_2(s)$*
- *the leader chooses the action in each state s according to $\sigma_1(a)$ for each action $a \in \mathcal{A}_1(s)$ if the state s was reached by following the recommendations*
- *both players switch to the minmax strategy σ^m after a deviation by the follower.*

Proof. Let ϕ' be a SEFCE. We construct the behavioral strategy profile σ from ϕ' and then show how an optimal strategy ϕ can be constructed from σ and σ^m.

To construct σ, it is sufficient to specify a probability $\sigma(a)$ for each action $a \in \mathcal{A}(s)$ in each state s. We use the probability of state s being reached (denoted $\phi'(s)$) that corresponds to the sum of pure strategy profiles $\phi'(\pi)$ such that the actions in strategy profile π allow state s to be reached.

Formally, there exists a sequence $s_0, a_0, \ldots, a_{k-1}, s_k$ of states and actions (starting at the root), such that for every $j = 0, \ldots, k-1$ it holds that $a_j = \pi(s_j)$, $s_{j+1} = \mathcal{T}(s_j, a_j)$ (or s_{j+1} is the next decision node of some player if $\mathcal{T}(s_j, a_j)$ is a chance node), $s_0 = s_{root}$, and $s_k = s$. Let $\Pi(s)$ denote a set of pure strategy profiles for which such a sequence exists for state s, and $\Pi(s, a) \subseteq \Pi(s)$ the strategy profiles that not only reach s, but also prescribe action a to be played in state s. We have:

$$\sigma(a) = \frac{\sum_{\pi' \in \Pi(s,a)} \phi'(\pi')}{\phi'(s)}, \text{ where } \phi'(s) = \sum_{\pi' \in \Pi(s)} \phi'(\pi').$$

In case $\phi'(s) = 0$, we set the behavior strategy in σ arbitrarily.

Next, we construct a strategy ϕ that corresponds to the desired behavior and show that it is indeed an optimal SEFCE strategy. We need to specify a probability for every pure strategy profile $\pi = (\pi_1, \pi_2)$. Consider the sequence of states and actions that corresponds to executing the actions from the strategy profile π. Let $s^l_0, a^l_0, \ldots, a^l_{k_l-1}, s_{k_l}$ be one of q possible sequences of states and actions (there can be multiple such sequences due to chance nodes), such that $j = 0, \ldots, k_l - 1$, $a^l_j = \pi(s^l_j)$, $s^l_{j+1} = \mathcal{T}(s^l_j, a^l_j)$ (or s^l_{j+1} is one of the next decision nodes of some player immediately following the chance node(s) $\mathcal{T}(s^l_j, a^l_j)$), $s^l_0 = s_{root}$, and $s^l_{k_l} \in \mathcal{Z}$. The probability for the strategy profile π corresponds to the probability of executing the sequences of actions multiplied by the probability that the remaining actions prescribe minmax strategy σ^m in case the follower deviates:

$$\phi(\pi) = \left(\prod_{l=1}^{q} \prod_{j=0}^{k_l-1} \sigma(a^l_j) \right) \cdot \prod_{a' = \pi(s') | s' \in \mathcal{S} \setminus \{s^1_0, \ldots, s^1_{k_0-1}, s^2_0, \ldots, s^q_{k_q-1}\}} \sigma^m(a').$$

Correctness. By construction of σ and ϕ, it holds that probability distribution over leafs remains the same as in ϕ'; hence, $\forall z \in \mathcal{Z}\ p_{\phi'}(z) = p_\sigma(z) = p_\phi(z)$ and thus the expected utility of ϕ for the players is the same as in ϕ'.

Second, we have to show that the follower has no incentive to deviate from the recommendations in ϕ. By deviating to some action a' in state s, the follower gains $\mu(\mathcal{T}(s,a'))$ since both players play according to σ^m after a deviation. In ϕ', the follower can get for the same deviation at best some utility value $v_2(\mathcal{T}(s,a'))$, which by the definition of the minmax strategies σ^m is greater than or equal to $\mu(\mathcal{T}(s,a'))$. Since the expected utility of the follower for following the recommendations is the same in ϕ as in ϕ', and the follower has no incentive to deviate in ϕ' because of the optimality, the follower has no incentive to deviate in ϕ either. □

3 COMPUTING EXACT STRATEGIES IN TURN-BASED GAMES

We start our computational investigation with turn-based games.

Theorem 3.1. *There is an algorithm that takes as input a turn-based game in DAG form with no chance nodes and outputs a Stackelberg equilibrium in pure strategies. The algorithm runs in time* $O(|\mathcal{S}|(|\mathcal{S}|+|\mathcal{Z}|))$.

Proof. Our algorithm performs three passes through all the nodes in the graph.

First, the algorithm computes the minmax values $\mu(s)$ of the follower for each node in the game by backward induction.

Second, the algorithm computes a *capacity* for each state in order to determine which states of the game are reachable (i.e., there exists a commitment of the leader and a best response of the follower such that the state can be reached by following their strategies). The capacity of state s, denoted $\gamma(s)$, is defined as the minimum utility of the follower that needs to be guaranteed by the outcome of the subgame starting in state s in order to make this state reachable. We initially set $\gamma(\mathcal{S}\cup\mathcal{Z}\setminus\{s_{root}\})=\infty$ and mark them as *open*, while we set $\gamma(s_{root})=-\infty$ and mark the root state as *closed*.

Next, the algorithm evaluates each open state s, for which all parents have been marked as *closed*. We distinguish whether the leader or the follower makes the decision:

- *s is a leader node*: The algorithm sets $\gamma(s')=\min(\gamma(s'),\gamma(s))$ for all children $s'\in\mathcal{T}(s)$;
- *s is a follower node*: The algorithm sets $\gamma(s')=\min(\gamma(s'),\max(\gamma(s),\max_{s''\in\mathcal{T}(s)\setminus\{s'\}}\mu(s'')))$ for all children $s'\in\mathcal{T}(s)$.

Finally, we mark state s as *closed*.

We say that leaf $z\in\mathcal{Z}$ is a *possible outcome*, if $\mu(z)=u_2(z)\geq\gamma(z)$. Now, the solution is such a possible outcome that maximizes the utility of the leader; that is, $\arg\max_{z\in\mathcal{Z}\ u_2(z)\geq\gamma(z)}u_1(z)$. The strategy is now constructed in the third pass by following nodes from leaf z back to the root while using nodes s' with capacities $\gamma(s')\leq\mu(z)$. Due to the construction of capacities, such a path exists and forms a part of the Stackelberg strategy. The leader commits to the strategy leading to max min utility for the follower in the remaining states that are not part of this path.

Correctness. To show the correctness of the algorithm, we must show that capacities for each node are correctly computed in the second pass. Whenever the algorithm evaluates a state s, the capacities of its parents are final; hence, the capacity value $\gamma(s)$ is also determined and final. Next, since the graph is a finite DAG, at least one vertex is marked as *closed* in each iteration, and therefore the second phase terminates and each vertex is evaluated exactly once.

A capacity for state s is defined as the minimum utility of the follower that needs to be guaranteed by the outcome of the subgame starting in state s. Recall that all nodes have the initial capacity set to ∞. Now, the capacities are updated iteratively. If a state s' can be immediately reached from a leader's node s, the capacity of s' can be decreased only if $\gamma(s)<\gamma(s')$: Since node s can be reached while guaranteeing the value of $\gamma(s)$ for the follower, the leader can then just commit to

play the action that leads to s'. If a state s' can be immediately reached from a follower's node s, the capacity of s' can be decreased due to other choices in s: The action that leads from s to s' must be a best response for the follower; hence, the follower must be guaranteed the best utility she can get by deviating in s, which is ensured by the computation of capacities. As a direct consequence of the construction of capacities, for each node s there exists a pure strategy of the leader π_1 that she can commit to and a best response $\pi_2 \in \mathcal{BR}_2(\pi_1)$ of the follower such that node s is reached when players follow these strategies.

Now, selecting the leaf with the maximum utility for the leader among all possible outcomes is a direct application of the definition of Stackelberg equilibria.

Complexity Analysis. Computing the max min values can be done in $O(|\mathcal{S}|(|\mathcal{S}| + |\mathcal{Z}|))$ by backward induction due to the fact the graph is a DAG. In the second pass, the algorithm solves the widest-path problem from a single source to all leafs. In each node, the algorithm calculates capacities for every child. In nodes where the leader acts, there is a constant-time operation performed for each child. However, we need to be more careful in nodes where the follower acts. For each child $s' \in \mathcal{T}(s)$, the algorithm computes a maximum value $\mu(s')$ of all of the siblings. We can do this efficiently by computing two maximal values of $\mu(s')$ for all $s' \in \mathcal{T}(s)$ (say s^1, s^2), and for each child then the term $\max_{s'' \in \mathcal{T}(s) \setminus \{s'\}} \mu(s'')$ equals either to s^1 if $s' \neq s^1$, or to s^2 if $s' = s^1$. Therefore, the second pass can again be done in $O(|\mathcal{S}|(|\mathcal{S}| + |\mathcal{Z}|))$. Finally, finding the optimal outcome and constructing the optimal strategy is again at most linear in the size of the graph. Therefore, the algorithm takes at most $O(|\mathcal{S}|(|\mathcal{S}| + |\mathcal{Z}|))$ steps. □

Next, we provide an algorithm for computing a Stackelberg extensive-form correlated equilibrium for turn-based games with no chance nodes.

Theorem 3.2. *There is an algorithm that takes as input a turn-based game in tree form with no chance nodes and outputs an SEFCE in the compact representation. The algorithm runs in time* $O(|\mathcal{S}||\mathcal{Z}|)$.

Proof. We improve the algorithm from the proof of Theorem 4 in Letchford and Conitzer (2010). The algorithm contains two steps: (i) a bottom-up dynamic program that for each node s computes the set of possible outcomes, and (ii) a downward pass constructing the optimal correlated strategy in the compact representation.

For each node s we keep set of points H_s in two-dimensional space, where the x-dimension represents the utility of the follower and the y-dimension represents the utility of the leader. These points define the convex set of all possible outcomes of the subgame rooted in node s (we assume that H_s contains only the points on the boundary of the convex hull). We keep each set H_s sorted by polar angle.

Upward pass. In leaf $z \in \mathcal{Z}$, we set $H_z = \{z\}$. In nodes s, where the leader acts, the set of points H_s is equal to the convex hull of the corresponding sets of the children H_w. That is, $H_s = \text{Conv}(\cup_{w \in \mathcal{T}(s)} H_w)$.

In nodes s, where the follower acts, the algorithm performs two steps. First, the algorithm removes from each set H_w of child w the outcomes from which the follower has an incentive to deviate. To do this, the algorithm uses the maxmin u_2 values of all other children of s except w and creates a new set $\hat{H}_w$ that we call the *restricted set*. The restricted set $\hat{H}_w$ is defined as an intersection of the convex set representing all possible outcomes H_w and all outcomes defined by the halfspace restricting the utility x of the follower by the inequality:

$$x \geq \max_{w' \in \mathcal{T}(s); w' \neq w} \min_{p' \in H_{w'}} u_2(p').$$

Second, the algorithm computes the set H_s by creating a convex hull of the corresponding restricted sets $\hat{H}_w$ of the children w. That is, $H_s = \text{Conv}(\cup_{w \in \mathcal{T}(s)} \hat{H}_w)$.

Finally, in the root of the game tree, the outcome of the SEFCE is the point with maximal payoff of player 1: $p_{SE} = \arg\max_{p \in H_{s_{root}}} u_1(p)$.

Downward pass. We now construct the compact representation of commitment to correlated strategies that ensures the outcome p_{SE} calculated in the upward pass. The method for determining the optimal strategy in each node is similar to the method strategy(s, p'') used in the proof of Theorem 4 in Letchford and Conitzer (2010).

Given a node s and a point p'' that lies on the boundary of H_s, this method specifies how to commit to correlated strategies in the subtree rooted in node s. Moreover, the proof in Letchford and Conitzer (2010) also showed that it is sufficient to consider mixtures of at most two actions in each node, and allowing correlated strategies does violate their proof. We consider separately leader and follower nodes:

- For each node s, where the leader acts, the algorithm needs to find two points p, p' in the boundaries of children H_w and $H_{w'}$, such that the desired point p'' is a convex combination of $p \in H_w$ and $p' \in H_{w'}$. If $w = w'$, then the strategy in node s is to commit to pure strategy leading to node w. If $w \neq w'$, then the strategy to commit to in node s is a mixture: with probability α to play action leading to w and with probability $(1 - \alpha)$ to play action leading to w', where $\alpha \in [0, 1]$ is such that $p'' = \alpha p + (1 - \alpha)p'$. Finally, for every child $s' \in \mathcal{T}(s)$ we call the method strategy with appropriate p (or p') in case $s' = w$ (or w'), and with the threat value corresponding to $\mu(s')$ for every other child.
- For each node s where the follower acts, the algorithm again needs to find two points p, p' in the *restricted* boundaries of children $\hat{H}_w$ and $\hat{H}_{w'}$, such that the desired point p'' is a convex combination of $p \in \hat{H}_w$ and $p' \in \hat{H}_{w'}$. The reason for using the restricted sets is because the follower must not have an incentive to deviate from the recommendation.

  Similarly to the previous case, if $w = w'$, then the correlated strategy in node s is to send the follower signal leading to node w while committing further to play strategy(w, p) in the subtree rooted in node w, and to play the minmax strategy in every other child s' corresponding to value $\mu(s')$.

  If $w \neq w'$, then there is a mixture of possible signals: With probability α, the follower receives a signal to play the action leading to w, and, with probability $(1 - \alpha)$, a signal to play the action leading to w', where $\alpha \in [0, 1]$ is again such that $p'' = \alpha p + (1 - \alpha)p'$. As before, by sending the signal to play a certain action, the leader commits to play method strategy(w, p) (or strategy(w', p')) in the subtree rooted in node w (or w') and commits to play the minmax strategy leading to value $\mu(s')$ for every other child s'.

Correctness. Due to the construction of the set of points H_s that are maintained for each node s, these points correspond to the convex hull of all possible outcomes in the subgame rooted in node s. In leafs, the algorithm adds the point corresponding to the leaf. In the leader's nodes, the algorithm creates a convex combination of all possible outcomes in the children of the node. The only places where the algorithm removes some outcomes from these sets are nodes of the follower. If a point is removed from H_w in node s, there exists an action of the follower in s that guarantees the follower a strictly better expected payoff than the expected payoff of the outcome that correspond to the removed point. Therefore, such an outcome is not possible as the follower will have an incentive to deviate. The outcome selected in the root node is the possible outcome that maximizes the payoff of the leader on all possible outcomes; hence, it is optimal for the leader. Finally, the downward

pass constructs the compact representation of the optimal correlated strategy to commit to that reaches the optimal outcome.

Complexity Analysis. Computing the boundary of the convex hull H_s takes $O(|\mathcal{Z}|)$ time in each level of the game tree since the children sets H_w are already sorted (De Berg et al. 2000, p. 6). Moreover, since we keep only nodes on the boundary of the convex hull, the inequality $\sum_{s \in \mathcal{S}} |H_s| \leq |\mathcal{Z}|$ for all nodes in a single level of the game tree also bounds the number of lines that need to be checked in the downward pass. Therefore, each pass takes at most $O(|\mathcal{S}||\mathcal{Z}|)$ time. □

Interestingly, the algorithm described in the proof of Theorem 3.2 can be modified also in cases where the game contains chance, as shown in the next theorem. This is in contrast to computing a Stackelberg equilibria that is NP-hard with chance.

Theorem 3.3. *There is an algorithm that takes as input a turn-based game in tree form with chance nodes and outputs the compact form of an SEFCE for the game. The algorithm runs in time* $O(|\mathcal{S}||\mathcal{Z}|)$.

Proof. We can use the proof from Theorem 3.2, but we need to analyze what happens in chance nodes in the upward pass. The algorithm computes in chance nodes the Minkowski sum of all convex sets in child nodes, and, since all sets are sorted and this is a planar case, this operation can be again performed in linear time (De Berg et al. 2000, p. 279). The size of set H_s is again bounded by the number of all leafs (Gritzmann and Sturmfels 1993). □

4 COMPUTING EXACT STRATEGIES IN CONCURRENT-MOVE GAMES

Next, we analyze concurrent-move games and show that while the problem of computing a Stackelberg equilibrium in behavior strategies is NP-hard (even without chance nodes), the problem of computing an SEFCE can be solved in polynomial time.

Theorem 4.1. *Given a concurrent-move game in tree form with no chance nodes and a number* α*, it is NP-hard to decide if the leader achieves payoff at least* α *in a Stackelberg equilibrium in behavior strategies.*

The proof for the preceding hardness result is included in Appendix A.1; the proof uses a reduction from the NP-complete problem Knapsack and constructs a two-step concurrent-move game. Note that computing a Stackelberg equilibrium in a single-shot concurrent-move game is polynomial (Conitzer and Sandholm 2006).

Theorem 4.2. *For a concurrent-move games in tree form, the compact form of an SEFCE for the game can be found in polynomial time by solving a single linear program.*

Proof. We construct a linear program (LP) based on the LP for computing EFCE (von Stengel and Forges 2008). We use the compact representation of SEFCE strategies (described by Lemma 2.5) represented by variables $\delta(s)$ that denote a joint probability that state s is reached when both players, and chance, play according to SEFCE strategies.

The size of the original EFCE LP—both the number of variables and constraints—is quadratic in the number of sequences of players. However, the LP for EFCE is defined for a more general class of imperfect-information games without chance. In our case, we can exploit the specific structure of a concurrent-move game and, together with the Stackelberg assumption, reduce the number of constraints and variables.

First, the deviation from a recommended strategy causes the game to reach a different subgame in which the strategy of the leader can be chosen (almost) independently of the subgame that follows the recommendation.

Second, the strategy that the leader should play according to the deviations is a minmax strategy, with which the leader punishes the follower by minimizing the utility of the follower as much as possible. Thus, by deviating to action a' in state s, the follower can get at best the minmax value of the subgame starting in node $\mathcal{T}(s, a')$ that we denote as $\mu(\mathcal{T}(s, a'))$. The values $\mu(s)$ for each state $s \in \mathcal{S}$ can be computed beforehand using backward induction.

The linear program is:

$$\max_{\delta, v_2} \sum_{z \in \mathcal{Z}} \delta(z) u_1(z) \tag{1}$$

subject to:

$$\delta(s_{root}) = 1 \tag{2}$$

$$0 \leq \delta(s) \leq 1 \qquad \forall s \in \mathcal{S} \tag{3}$$

$$\delta(s) = \sum_{s' \in \mathcal{T}(s)} \delta(s') \qquad \forall s \in \mathcal{S};\ \rho(s) = \{1, 2\} \tag{4}$$

$$\delta(\mathcal{T}(s, a_c)) = \delta(s) C(s, a_c) \qquad \forall s \in \mathcal{S}\ \forall a \in \mathcal{A}_c(s); \rho(s) = \{c\} \tag{5}$$

$$v_2(z) = u_2(z)\delta(z) \qquad \forall z \in \mathcal{Z} \tag{6}$$

$$v_2(s) = \sum_{s' \in \mathcal{T}(s)} v_2(s') \qquad \forall s \in \mathcal{S} \tag{7}$$

$$\sum_{a_1 \in \mathcal{A}_1(s)} v_2(\mathcal{T}(s, a_1 \times a_2)) \geq \sum_{a_1 \in \mathcal{A}_1(s)} \delta(\mathcal{T}(s, a_1 \times a_2)) \mu(\mathcal{T}(s, a_1 \times a_2')) \qquad \forall s \in \mathcal{S}\ \forall a_2, a_2' \in \mathcal{A}_2(s) \tag{8}$$

The interpretation is as follows. Variables δ represent the compact form of the correlated strategies. Equation (2) ensures that the probability of reaching the root state is 1, while Equation (3) ensures that for each state s, we have $\delta(s)$ between 0 and 1.

Network-flow constraints: The probability of reaching a state equals the sum of probabilities of reaching all possible children (Equation (4)), and it must correspond with the probability of actions in chance nodes (Equation (5)). The objective function ensures that the LP finds a correlated strategy that maximizes the leader's utility.

The follower has no incentive to deviate from the recommendations given by δ: To this end, variables $v_2(s)$ represent the expected payoff for the follower in a subgame rooted in node $s \in \mathcal{S}$ when played according to δ, defined by Equations (6–7). Each action that is recommended by δ must guarantee the follower at least the utility she gets by deviating from the recommendation. This is ensured by Equation (8), where the expected utility for recommended action a_2 is expressed by the left side of the constraint, while the expected utility for deviating is expressed by the right side of the constraint.

Note that the expected utility on the right-hand side of Equation (8) is calculated by considering the posterior probability after receiving the recommendation a_2 and the minmax values of children states after playing a_2'; $\mu(\mathcal{T}(s, a_1 \times a_2'))$.

Therefore, the variables δ found by solving this linear program correspond to the compact representation of the optimal SEFCE strategy. □

5 APPROXIMATING OPTIMAL STRATEGIES

In this section, we describe fully polynomial-time approximation schemes for finding a Stackelberg equilibrium in behavioral strategies as well as in pure strategies for turn-based games on trees with chance nodes. The notion of approximation we consider is *additive* approximation, and, for this to make sense, we assume that the utility function of the leader is bounded to the range $[0, 1]$.

We start with the problem of computing behavioral strategies for turn-based games on trees with chance nodes.

THEOREM 5.1. *There is an algorithm that takes as input a turn-based game on a tree with chance nodes, leader utilities in the range* $[0, 1]$, *and a parameter* ϵ *and computes a behavioral strategy for the leader. That strategy, combined with some best response of the follower, achieves a payoff that differs by at most* ϵ *from the payoff of the leader in a Stackelberg equilibrium in behavioral strategies. The algorithm runs in time* $O(T(H_T/\epsilon)^3)$, *where* T *is the size of the game tree and* H_T *is its height.*

The exact version of this problem was shown to be NP-hard by Letchford and Conitzer (2010). Their hardness proof was a reduction from KNAPSACK and our algorithm is closely related to the classical approximation scheme for this problem. We present here the algorithm and delegate the proof of correctness to the appendix.

Our scheme uses dynamic programming to construct a table of values for each node in the tree. Each table contains a discretized representation of the possible tradeoffs between the utility that the leader can get and the utility that can, at the same time, be offered to the follower. In the appendix, we show that the cumulative error in the leader's utility is bounded additively by the height of the tree. This error only depends on the height of the tree and not the utility. By an initial scaling of the leader utility by a factor D, this error can be made arbitrarily small, at the cost of extra computation time. The scaling is equivalent to discretizing the leader's payoff to multiples of some small $\delta = 1/D$. For simplicity, we only describe the scheme for binary trees, since nodes with higher branching factors can be replaced by small equivalent binary trees.

An important property is that only the leader's utility is discretized, since we need to be able to reason correctly about the follower's actions. The tables are indexed by the leader's utility and contain values that are the follower's utility.

We first present the dynamic programming algorithm as a separate statement.

PROPOSITION 5.2. *There is an algorithm that takes as input a turn-based game on a tree with chance nodes and leader utilities in the range* $[U_1, U_2]$, *where* U_1 *and* U_2 *are integers, and computes for each subtree* T *of the game a table* A_T *with the following properties guaranteed in each table:*

(1) If $A_T[k] > -\infty$ *the leader has a behavioral strategy for the game tree* T *that offers the follower utility* $A_T[k]$ *while securing utility* at least k *to the leader.*
(2) No behavioral strategy of the leader can (starting from subtree T*) offer the follower utility* strictly more *than* $A_T[k]$ *while securing utility* at least $k + H_T$ *to the leader, where* H_T *is the height of the tree* T.
(3) The entries of A_T *are nonincreasing,* $A_T[U_1]$ *is the largest utility the leader can help the follower to obtain, and* $A_T[U_2 + 1] = -\infty$.

Last, the algorithm runs in time $O(TU^3)$, *where* T *is the size of the game and* $U = U_2 - U_1$.

(SKETCH). We will now examine each type of node and, for each, show how the table is constructed. For each node T, we let L and R denote the two successors (if any), and we let A_T, A_L, and A_R denote their respective tables. Each table will have $U + 2$ entries, indexed by $k \in \{U_1, \ldots, U_2 + 1\}$.

Leaf nodes. If T is a leaf with utility (u_1, u_2), the table can be filled directly from the definition:

$$A_T[k] := \begin{cases} u_2 & \text{, if } k \leq u_1 \\ -\infty & \text{, otherwise.} \end{cases}$$

Leader nodes. If T is a leader node, and the leader plays L with probability p followed up by the strategies that gave the guarantees for $A_L[i]$ and $A_R[j]$, then the leader would get at least an

expected $pi + (1-p)j$ while being able to offer $pA_L[i] + (1-p)A_R[j]$ to the follower. For a given k, the optimal combination of the computed tradeoffs becomes:

$$A_T[k] := \max_{i,j,p}\{pA_L[i] + (1-p)A_R[j] \quad | \quad pi + (1-p)j \geq k\}.$$

This table can be computed in time $O(U^3)$ by looping over all i, j, k, and taking the maximum with the extremal feasible values of p.

Chance nodes. If T is a chance node, where the probability of L is p, and the leader combines the strategies that gave the guarantees for $A_L[i]$ and $A_R[j]$, then the leader would get at least an expected $pi + (1-p)j$ while being able to offer $pA_L[i] + (1-p)A_R[j]$ to the follower. For a given k, the optimal combination of the computed tradeoffs becomes: $A_T[k] := \max_{i,j}\{pA_L[i] + (1-p)A_R[j] \quad | \quad pi + (1-p)j \geq k\}$. The table A_T can thus be filled in time $O(U^3)$ by looping over all i, j, k, and this can even be improved to $O(U^2)$ by a simple optimization.

Follower nodes. If T is a follower node, then if the leader combines the strategy for $A_L[k]$ in L with the minmax strategy for R, this ensures leader utility k if $A_L[k] \geq \mu(R)$, and, similarly, if the leader combines the strategy for $A_R[k]$ in R with the minmax strategy for L it ensures leader utility k if $A_R[j] \geq \mu(L)$. Thus, the optimal combination becomes

$$A_T[k] := \max(A_L[k]\downarrow_{\mu(R)}, A_R[k]\downarrow_{\mu(L)}) \qquad x\downarrow_\mu := \begin{cases} x, \text{ if } x \geq \mu \\ -\infty, \text{ otherwise.} \end{cases}$$

The table A_T can be filled in time $O(U)$.

Putting it all together, each table can be computed in time $O(U^3)$, and there is one table for each node in the tree, which gives the desired running time.

With this in place, the approximation algorithm is immediate.

(Theorem 5.1). Let G be the given game and let u_{Opt} be the leader utility in a Stackelberg equilibrium in G. Let G' be obtained from G by scaling all leader utilities by a factor $(H_T + 1)/\epsilon$ (we do not round to integer utilities; this is already done implicitly by the dynamic programming algorithm).

Next, apply the algorithm of Proposition 5.2 to G'. Let A_T be the table of the root node, let $k = \max\{i \mid A_T[i] > -\infty\}$, and let σ_1 be the associated leader strategy given by property (1). Note that k is well-defined, since $A_T[0] > -\infty$ by property (3). By property (2), we have that $((H_T + 1)/\epsilon)u_{\text{Opt}} < k + 1 + H_T$, since $A_T[k+1] = -\infty$. Thus, σ_1 approximates the leader payoff of a Stackelberg equilibrium with additive error $H_T + 1$ in G', which means the additive error in the original game G is $(H_T + 1)/((H_T + 1)/\epsilon) = \epsilon$. The running time is $O(T(H_T/\epsilon)^3)$ as stated.

Next, we state the analogous statement for the case of pure strategies. Again, the exact problem was shown to be NP-hard by Conitzer and Letchford.

Theorem 5.3. *There is an algorithm that takes as input a turn-based game on a tree with chance nodes and leader utilities in the range* $[0, 1]$ *and a parameter* ϵ *and computes a pure strategy for the leader. That strategy, combined with some best response of the follower, achieves a payoff that differs by at most* ϵ *from the payoff of the leader in a Stackelberg equilibrium in pure strategies. The algorithm runs in time* $O(T(H_T/\epsilon)^2)$*, where* T *is the size of the game tree and* H_T *is its height.*

The proof of this theorem is the same as the proof of Theorem 5.1, except for the use of a modified dynamic programming algorithm as stated here:

Proposition 5.4. *There is an algorithm that takes as input a turn-based game on a tree with chance nodes and leader utilities in the range* $[U_1, U_2]$*, where* U_1 *and* U_2 *are integers, and computes for each subtree* T *of the game a table* A_T *with the following properties guaranteed in each table:*

(1) *If $A_T[k] > -\infty$ the leader has a pure strategy for the game tree T that offers the follower utility $A_T[k]$ while securing utility* at least *k to the leader.*
(2) *No pure strategy of the leader can (starting from subtree T) offer the follower utility* strictly more *than $A_T[k]$ while securing utility* at least *$k + H_T$ to the leader, where H_T is the height of the tree T.*
(3) *The entries of A_T are nonincreasing, $A_T[U_1]$ is the largest utility the leader can help the follower to obtain, and $A_T[U_2 + 1] = -\infty$.*

Last, the algorithm runs in time $O(TU^2)$, where T is the size of the game and $U = U_2 - U_1$.

(SKETCH). The algorithm is essentially the same as the one for behavioral strategies, except that leader nodes only have $p \in \{0, 1\}$. For a given k, the optimal combination of the computed tradeoffs becomes:

$$A_T[k] := \max\{A_L[k], A_R[k]\}.$$

The table A_T can thus be computed in time $O(U)$.

Since now the leader nodes can now be handled in linear time, the chance nodes are now the limiting case, and each table can thus be computed in time $O(U^2)$, giving the desired total running time $O(TU^2)$.

6 DISCUSSION AND CONCLUSION

Our article settles several open questions in the problem of complexity of computing a Stackelberg equilibrium in finite sequential games. Very often, the problem is NP-hard for many subclasses of extensive-form games, and we show that the hardness holds also for games in the tree form with concurrent moves. However, there are important subclasses that admit either an efficient polynomial algorithm or fully polynomial-time approximation schemes (FPTAS); we provide an FPTAS for games on trees with chance. The question unanswered within the scope of this article is whether there exists a (fully) polynomial-time approximation scheme for games in the tree form with concurrent moves. Our conjecture is that the answer is negative.

Second, we formalize a Stackelberg variant of the Extensive-Form Correlated Equilibrium solution concept (SEFCE) where the leader commits to correlated strategies. We show that the complexity of the problem is often reduced (to polynomial) compared to NP-hardness when the leader commits to behavioral strategies.

Our article does not address many other variants of computing a Stackelberg equilibrium where the leader commits to correlated strategies. First, we consider only two-player games with one leader and one follower. Even though computing an EFCE in games with multiple players is solvable in polynomial time, a recent result showed that computing a SEFCE on trees with no chance with three or more players is NP-hard (Černý 2016). Second, we consider only behavioral strategies (or memoryless strategies) in games on DAGs. Extending the concept of SEFCE to strategies that can use some fixed-size memory is a natural continuation of the present work.

A APPENDIX

In this section, we provide all the missing proofs.

A.1 Computing Exact Strategies in Concurrent-Move Games

For the analysis in this section, we use a variant of the NP-complete problem KNAPSACK, which we call KNAPSACK WITH UNIT-ITEMS:

root node

		f_0	f_1	f_2	...	f_N
l_1	$\mathcal{I}_1$		$(0, NM - W - M)$	$(0, -W - M)$	...	$(0, -W - M)$
l_2	$\mathcal{I}_2$		$(0, -W - M)$	$(0, NM - W - M)$	...	$(0, -W - M)$
$\vdots$					$\ddots$	
l_N	$\mathcal{I}_N$		$(0, -W - M)$	$(0, -W - M)$	...	$(0, NM - W - M)$

$\mathcal{I}_i$

	L	R
$\oplus$	$(Nv_i, -Nw_i)$	$(0, -Nw_i)$
$\ominus$	$(Nv_i - Nw_i)$	$(0, 0)$

Fig. 2. Game tree for reduction in proof of Theorem 4.1.

Knapsack with unit-items: *Given N items with positive integer weights $w_1, \ldots, w_N$ and values $v_1, \ldots, v_N$, a weight budget W, and a target value K, and such that at least W of the items have weight and value 1, does there exist $J \in \mathcal{P}(N)$ such that $\sum_{i \in J} w_i \leq W$ and $\sum_{i \in J} v_i \geq K$?*

The following lemma will be useful.

Lemma A.1. *The Knapsack with unit-items problem is NP-complete.*

Proof. We can reduce from the ordinary Knapsack problem. Given N items with weights $w_1, \ldots, w_N$ and values $v_1, \ldots, v_N$, and weight budget W and target K, we form $N + W$ items. The weight and values of the first N items are given by w_i and $(W + 1)v_i$, for $i = 1, \ldots, N$. The next W items are given weight and value 1. The weight budget is unchanged W, but the new target value is $(W + 1)K$. □

We can now prove the main result of this section.

Theorem 4.1 (Restated). *Given a concurrent-move game in tree form with no chance nodes and a number α, it is NP-hard to decide if the leader achieves payoff at least α in a Stackelberg equilibrium in behavior strategies.*

Proof. Consider an instance of Knapsack with unit-items. We define a concurrent-move extensive-form game in a way so that the optimal utility attainable by the leader is equal to the optimal solution value of the Knapsack with unit-items instance.

The game tree consists of two levels (see Figure 2): the root node consisting of N actions of the leader and $N + 1$ actions of the follower. M denotes a large constant that we use to force the leader to select a uniform strategy in the root node. More precisely, we choose M as the smallest integer such that $M > WNv_i$ and $M > Nw_i$ for $i = 1, \ldots, N$. In the second level, there is a state $\mathcal{I}_i$ corresponding to item i that models the decision of the leader to include items in the subset (action $\oplus$) or not (action $\ominus$).

Consider a feasible solution J to the Knapsack with unit-items problem with unit-items. This translates into a strategy for the leader as follows. In the root node, she plays the uniform strategy and, in the subgame $\mathcal{I}_i$ plays $\oplus$ with probability 1 if $i \in J$ and plays $\ominus$ with probability 1 otherwise. We can now observe that the follower plays L in subgames $\mathcal{I}_i$ where $i \in J$, since ties are broken in favor of the leader, and the follower plays R in subgames $\mathcal{I}_i$ where $i \notin J$. In the root node, action f_0 for the follower thus leads to payoff $-\sum_{i \in J} w_i \geq -W$. Actions f_k for $k \geq 1$ lead to payoff

$$\frac{1}{N}(NM - W - M) + \frac{N-1}{N}(-W - M) = -W.$$

Since ties are broken in favor of the leader, the follower plays action f_0, which means that the leader receives payoff $\sum_{i \in J} v_i$, which is the value of the KNAPSACK WITH UNIT-ITEMS solution.

Consider, on the other hand, an *optimal* strategy for the leader. By the structure of the game, we have the following lemma:

CLAIM 1. *Without loss of generality, the leader plays using a pure strategy in each subgame $\mathcal{I}_i$.*

PROOF. If in subgame $\mathcal{I}$ the leader commits to playing $\oplus$ with probability 1, the follower will choose to play L due to ties being broken in favor of the leader. If, on the other hand, the leader plays $\oplus$ with probability strictly lower than 1, the follower will choose to play R, leading to utility 0 for the leader and at most 0 for the follower. Since the leader can only obtain positive utility if the follower plays action f_0 in the root node, there is thus no benefit for the leader in decreasing the utility for the follower by committing to a strictly mixed strategy. In other words, if the leader plays $\oplus$ with probability strictly lower than 1, the leader might as well play $\oplus$ with probability 0. □

Thus, from now on, we assume that the leader plays using a pure strategy in each sub-game $\mathcal{I}_i$. Let $J \in \mathcal{P}(N)$ be such that the set of indices i of the subgames $\mathcal{I}_i$ where the leader commits to action $\oplus$.

CLAIM 2. *If the strategy of the leader ensures positive utility, it chooses an action uniformly at random in the root node.*

PROOF. Let $\varepsilon_i \in [-\frac{1}{N}, 1 - \frac{1}{N}]$ be such that the leader commits to playing action l_i with probability $\frac{1}{N} + \varepsilon_i$. Then, if the follower plays action f_0, the leader obtains payoff

$$\sum_{i \in J} v_i + N \sum_{i \in J} \varepsilon_i v_i$$

and the follower obtains payoff

$$-\sum_{i \in J} w_i - N \sum_{i \in J} \varepsilon_i w_i.$$

If the follower plays action f_k, for $k \geq 1$, the leader obtains payoff 0 and the follower obtains payoff $\varepsilon_k NM - W$.

Let k be such that $\varepsilon_k = \max_i \varepsilon_i$ and assume to the contrary that $\varepsilon_k > 0$. Note that

$$\varepsilon_k \geq \frac{1}{N} \sum_{i:\varepsilon_i > 0} \varepsilon_i = -\frac{1}{N} \sum_{i:\varepsilon_i < 0} \varepsilon_i.$$

We now proceed by case analysis.

Case 1 ($\sum_{i \in J} w_i \geq W$). By definition of ε_k and M we have

$$\begin{aligned} \varepsilon_k M &\geq \left(-\frac{1}{N} \sum_{i \in J:\varepsilon_i < 0} \varepsilon_i\right) M > -\frac{1}{N} \sum_{i \in J:\varepsilon_i < 0} \varepsilon_i (N w_i) \\ &= -\sum_{i \in J:\varepsilon_i < 0} \varepsilon_i w_i \geq -\sum_{i \in J} \varepsilon_i w_i. \end{aligned}$$

Multiplying both sides of the inequality by N and using the inequality: $-W \geq -\sum_{i \in J} w_i$, we have

$$\varepsilon_k NM - W > -\sum_{i \in J} w_i - N \sum_{i \in J} \varepsilon_i w_i,$$

which means that action f_k is preferred by the follower. Thus, the leader receives payoff 0.

Case 2 ($\sum_{i \in J} w_i < W$). Since we have a Knapsack with unit-items instance, there is a knapsack solution that obtains value $1 + \sum_{i \in J} v_i$, which corresponds to a strategy for the leader that obtains the same utility. Since the current strategy is optimal for the leader, we must have $\sum_{i \in J} v_i + N \sum_{i \in J} \varepsilon_i v_i \geq 1 + \sum_{i \in J} v_i$, which means that $1 \leq N \sum_{i \in J} \varepsilon_i v_i \leq (N^2 \max_i v_i)\varepsilon_k$, and thus $\varepsilon_k \geq 1/(N^2 \max_i v_i)$. We then have by definition of M that

$$\varepsilon_k NM - W \geq \frac{NM}{N^2 \max_i v_i} - W > 0.$$

Thus, the payoff for the follower is strictly positive for the action f_k, and this is thus preferred to f_0, thus leading to payoff 0 to the leader. □

Since there is a strategy for the leader that obtains strictly positive payoff, we can thus assume that the strategy for the leader chooses an action uniformly at random in the root node, and the follower chooses action f_0. Since f_0 is preferred by the follower to any other action, this means that $\sum_{i \in J} w_i \leq W$, and the leader obtains payoff $\sum_{i \in J} v_i$. Thus, this corresponds exactly to a feasible solution to the Knapsack with unit-items instance of the same value.

B APPROXIMATING OPTIMAL STRATEGIES

In this section, we provide the full analysis for the dynamic programming algorithms used for our approximation algorithm.

Proposition 5.2 (Restated). *There is an algorithm that takes as input a turn-based game on a tree with chance nodes and leader utilities in the range $[U_1, U_2]$, where U_1 and U_2 are integers, and computes for each subtree T of the game a table A_T with the following properties guaranteed in each table:*

(1) If $A_T[k] > -\infty$ the leader has a behavioral strategy for the game tree T that offers the follower utility $A_T[k]$ while securing utility at least k to the leader.
(2) No behavioral strategy of the leader can (starting from subtree T) offer the follower utility strictly more than $A_T[k]$ while securing utility at least $k + H_T$ to the leader, where H_T is the height of the tree T.
(3) The entries of A_T are nonincreasing, $A_T[U_1]$ is the largest utility the leader can help the follower to obtain, and $A_T[U_2 + 1] = -\infty$.

Last, the algorithm runs in time $O(TU^3)$, where T is the size of the game and $U = U_2 - U_1$.

We have provided the algorithm in the main body of this article; its correctness and running time will follow from the next lemma.

Lemma B.1. *The algorithm of Proposition 5.2 is correct and has the desired running time.*

Proof. We will prove the properties hold for the computed tables by induction. These properties also form our induction hypothesis. In the proof , we will for convenience index the tables with k also outside the range $\{U_1, \ldots, U_2 + 1\}$, noting that the tables would have $A_T[k] = A_T[U_1]$ for $k < U_1$ and $A_T[k] = A_T[U_2 + 1]$ for $k > U_2 + 1$ following the definitions of the algorithm.

We consider each type of node in the tree. The easy case is the case of leaf nodes, where the properties hold trivially by construction.

Leader nodes. Let T be a leader node, with successors L and R, each with tables A_L and A_R. If the leader plays L with probability p and plays R with the remaining probability $(1 - p)$, followed up by the strategies that gave the guarantees for $A_L[i]$ and $A_R[j]$, then the leader would get at

least an expected $pi + (1-p)j$, while being able to offer $pA_L[i] + (1-p)A_R[j]$ to the follower. For a given k, the optimal combination of the computed tradeoffs becomes:

$$A_T[k] := \max_{i,j,p} \{pA_L[i] + (1-p)A_R[j] \quad | \quad pi + (1-p)j \geq k\}$$

For property (1) of the induction hypothesis, the strategy that guarantees $A_T[k]$ simply combines the strategies for the maximizing $A_L[i]$ and $A_R[j]$ along with the probability p at node T. For a given i, j, and k, finding the optimal value p amounts to maximizing a linear function over an interval (i.e., it will attain its maximum at one of the end points of the interval). The table A_T can thus be filled in time $O(U^3)$ by looping over all i, j, k.

For property (2) of the induction hypothesis, assume for contradiction that some strategy σ yields utilities (u_1^σ, u_2^σ) with

$$u_1^\sigma \geq k + H_T \quad \text{and} \quad u_2^\sigma > A_T[k]. \tag{9}$$

Let p_σ be the probability that σ assigns to the action L, and let $(u_1^{\sigma,L}, u_2^{\sigma,L})$ and $(u_1^{\sigma,R}, u_2^{\sigma,R})$ be the utilities from playing σ and the corresponding follower strategy in the left and right child, respectively. By definition,

$$u_l^\sigma = p_\sigma \cdot u_l^{\sigma,L} + (1-p_\sigma) \cdot u_l^{\sigma,R}, \quad \forall l \in \{1,2\}. \tag{10}$$

By the induction hypothesis,

$$u_2^{\sigma,c} \leq A_c[\lfloor u_1^{\sigma,c} \rfloor - H_T + 1], \quad \forall c \in \{L,R\}. \tag{11}$$

Thus,

$$A_T[k] < u_2^\sigma \leq p_\sigma \cdot A_L[\lfloor u_1^{\sigma,L} \rfloor - H_T + 1] + (1-p_\sigma) \cdot A_R[\lfloor u_1^{\sigma,R} \rfloor - H_T + 1]. \tag{12}$$

But

$$p_\sigma \cdot (\lfloor u_1^{\sigma,L} \rfloor - H_T + 1) + (1-p_\sigma) \cdot (\lfloor u_1^{\sigma,R} \rfloor - H_T + 1) \tag{13}$$

$$\geq p_\sigma \cdot (u_1^{\sigma,L} - H_T) + (1-p_\sigma) \cdot (u_1^{\sigma,R} - H_T) \tag{14}$$

$$= u_1^\sigma - H_T \geq k, \tag{15}$$

meaning that $i = \lfloor u_1^{\sigma,L} \rfloor - H_T + 1$ and $j = \lfloor u_1^{\sigma,R} \rfloor - H_T + 1$ satisfy the constraints in the definition of $A_T[k]$, which contradicts the assumption that $u_2^\sigma > A_T[k]$.

Finally, property (3) holds trivially by the definition of $A_T[k]$ and the induction hypothesis.

Chance nodes. Let T be a chance node, with successors L and R, each with tables A_L and A_R, and let p be the probability that chance picks L. If the leader combines the strategies that gave the guarantees for $A_L[i]$ and $A_R[j]$, then the leader would get at least an expected $pi + (1-p)j$ while being able to offer $pA_L[i] + (1-p)A_R[j]$ to the follower. For a given k, the optimal combination of the computed tradeoffs becomes:

$$A_T[k] := \max_{i,j} \{pA_L[i] + (1-p)A_R[j] \quad | \quad pi + (1-p)j \geq k\}.$$

For property (1) of the induction hypothesis, the strategy that guarantees $A_T[k]$ simply combines the strategies for the maximizing $A_L[i]$ and $A_R[j]$. The table A_T can thus be filled in time $O(U^3)$ by looping over all i, j, k, and this can even be improved to $O(U^2)$ by a simple optimization.

For property (2) of the induction hypothesis, assume for contradiction that some strategy σ yields utilities (u_1^σ, u_2^σ) with

$$u_1^\sigma \geq k + H_T \quad \text{and} \quad u_2^\sigma > A_T[k]. \tag{16}$$

Let $(u_1^{\sigma,L}, u_2^{\sigma,L})$ and $(u_1^{\sigma,R}, u_2^{\sigma,R})$ be the utilities from playing σ and the corresponding follower strategy in the left and right child, respectively. By definition,

$$u_l^\sigma = p \cdot u_l^{\sigma,L} + (1-p) \cdot u_l^{\sigma,R}, \quad \forall l \in \{1,2\}. \tag{17}$$

By the induction hypothesis,

$$u_2^{\sigma,c} \leq A_c[\lfloor u_1^{\sigma,c} \rfloor - H_T + 1], \quad \forall c \in \{L,R\}. \tag{18}$$

Thus,

$$A_T[k] < u_2^\sigma \leq p \cdot A_L[\lfloor u_1^{\sigma,L} \rfloor - H_T + 1] + (1-p) \cdot A_R[\lfloor u_1^{\sigma,R} \rfloor - H_T + 1]. \tag{19}$$

But

$$p \cdot (\lfloor u_1^{\sigma,L} \rfloor - H_T + 1) + (1-p) \cdot (\lfloor u_1^{\sigma,R} \rfloor - H_T + 1) \tag{20}$$

$$\geq p \cdot (u_1^{\sigma,L} - H_T) + (1-p) \cdot (u_1^{\sigma,R} - H_T) \tag{21}$$

$$= u_1^\sigma - H_T \geq k, \tag{22}$$

meaning that $i = \lfloor u_1^{\sigma,L} \rfloor - H_T + 1$ and $j = \lfloor u_1^{\sigma,R} \rfloor - H_T + 1$ satisfy the constraints in the definition of $A_T[k]$, which contradicts the assumption that $u_2^\sigma > A_T[k]$.

Again, property (3) holds trivially by the definition of $A_T[k]$ and the induction hypothesis.

Follower Nodes. Let T be a follower node, with successors L and R, each with tables A_L and A_R, and let τ_L and τ_R be the minmax value for the follower in L and R, respectively. If the leader combines the strategy for $A_L[k]$ in L with the minmax strategy for R, this ensures the leader utility k if $A_L[k] \geq \tau_R$, and, similarly, if the leader combines the strategy for $A_R[k]$ in R with the minmax strategy for L, it ensures leader utility k if $A_R[j] \geq \tau_L$. Thus, if we let

$$x \downarrow_\tau := \begin{cases} x & \text{, if } x \geq \tau \\ -\infty & \text{, otherwise,} \end{cases}$$

then the optimal combination becomes

$$A_T[k] := \max(A_L[k] \downarrow_{\tau_R}, A_R[k] \downarrow_{\tau_L}).$$

For property (1) of the induction hypothesis, the strategy that guarantees $A_T[k]$ simply combines the strategies for the maximizing $A_L[i]$ or $A_R[j]$ in one branch and playing minmax in the other. The table A_T can thus be filled in time $O(U)$.

For property (3), it follows trivially that the entries of A_T are nonincreasing and that $A_T[U_2 + 1] = -\infty$ from the definition of $A_T[k]$ and the induction hypothesis. For the remaining part, note that $A_L[U_1] \geq \tau_L$ and $A_R[U_1] \geq \tau_R$. Thus, $\max(A_L[U_1], A_R[U_1]) \geq \max(\tau_L, \tau_R)$, which means $A_T[U_1] = \max(A_L[U_1], A_R[U_1])$.

Finally, for property (2) of the induction hypothesis, let H_T be the height of the tree T. Suppose that some strategy σ yields (u_1^σ, u_2^σ) with

$$u_1^\sigma \geq k + H_T \quad \text{and} \quad u_2^\sigma > A_T[k]. \tag{23}$$

Assume without loss of generality that the follower plays L. Let $(u_1^{\sigma,L}, u_2^{\sigma,L})$ be the utilities from playing σ and the corresponding follower strategy in the left child. Combined with the induction hypothesis, we get

$$A_T[k] < u_2^{\sigma,T} = u_2^{\sigma,L} \leq A_L[\lfloor u_1^{\sigma,L} \rfloor - H_T + 1] = A_T[\lfloor u_1^\sigma \rfloor - H_T + 1]. \tag{24}$$

But this is a contradiction, since $A_T[k]$ is nonincreasing in k and

$$\lfloor u_1^\sigma \rfloor - H_T + 1 \geq u_1^\sigma - H_T \geq k. \tag{25}$$

This completes the proof. □

For the case of commitment to pure strategies, we had a very similar algorithm.

Proposition 5.4 (Restated). *There is an algorithm that takes as input a turn-based game on a tree with chance nodes and leader utilities in the range $[U_1, U_2]$, where U_1 and U_2 are integers, and computes for each subtree T of the game a table A_T with the following properties guaranteed in each table:*

(1) *If $A_T[k] > -\infty$ the leader has a pure strategy for the game tree T that offers the follower utility $A_T[k]$ while securing utility* at least *k to the leader.*
(2) *No pure strategy of the leader can (starting from subtree T) offer the follower utility* strictly more*than $A_T[k]$ while securing utility* at least *$k + H_T$ to the leader, where H_T is the height of the tree T.*
(3) *The entries of A_T are nonincreasing, $A_T[U_1]$ is the largest utility the leader can help the follower to obtain, and $A_T[U_2 + 1] = -\infty$.*

Last, the algorithm runs in time $O(TU^2)$, where T is the size of the game and $U = U_2 - U_1$.

In essence, the algorithm for Theorem 5.3 is the same, except leader nodes only consider $p \in \{0, 1\}$. The induction hypothesis is the same, except the quantifications are over pure strategies instead. We argue the correctness of this algorithm formally in the following lemma.

Lemma B.2. *The algorithm of Proposition 5.4 is correct and has the desired running time.*

Proof. We have the same construction and induction hypothesis as in Proposition 5.2. Let T be a leader node, with successors L and R, each with tables A_L and A_R. If the leader plays L (or R), followed up by the strategies that gave the guarantees for $A_L[k]$ (or $A_R[k]$), then the expected leader utility would be k while being able to offer $A_L[k]$ (or $A_R[k]$ respectively) to the follower. For a given k, the optimal combination of the computed tradeoffs becomes:

$$A_T[k] := \max\{A_L[k], A_R[k]\}.$$

For property (1) of the induction hypothesis, we simply use the move that maximizes the expression combined with the strategies that guarantee $A_L[k]$ and $A_R[k]$ in the successors. The table A_T can thus be filled in time $O(U)$.

For property (2) of the induction hypothesis, assume for contradiction that some pure strategy π yields utilities (u_1^π, u_2^π) with

$$u_1^\pi \geq k + H_T \quad \text{and} \quad u_2^\pi > A_T[k]. \tag{26}$$

Assume without loss of generality that π plays L at T, and let $(u_1^{\pi,L}, u_2^{\pi,L})$ be the utilities from playing π and the corresponding follower strategy in L. By definition,

$$u_l^\pi = u_l^{\pi,L}, \quad \forall l \in \{1, 2\}. \tag{27}$$

By the induction hypothesis,

$$u_2^{\pi,L} \leq A_L[\lfloor u_1^{\pi,L} \rfloor - H_T + 1]. \tag{28}$$

Thus,

$$A_T[k] < u_2^\pi \leq A_L[\lfloor u_1^{\pi,L} \rfloor - H_T + 1]. \tag{29}$$

But

$$\lfloor u_1^{\pi,L} \rfloor - H_T + 1 \geq u_1^{\pi,L} - H_T = u_1^\pi - H_T \geq k, \tag{30}$$

meaning that $i = \lfloor u_1^{\pi,L} \rfloor - H_T + 1$ satisfies the constraints in the definition of $A_T[k]$, which contradicts the assumption that $u_2^\pi > A_T[k]$.

Finally, property (3) holds trivially by the definition of $A_T[k]$ and the induction hypothesis.

Note that the analysis for the other nodes is identical to those for behavioral strategies, which completes the proof. □

REFERENCES

Rabah Amir and Isabel Grilo. 1999. Stackelberg versus Cournot equilibrium. *Games and Economic Behavior* 26, 1 (1999), 1–21.

Branislav Bošanský and Jiří Čermák. 2015. Sequence-form algorithm for computing Stackelberg equilibria in extensive-form games. In *Proceedings of AAAI Conference on Artificial Intelligence*. AAAI, 805–811.

Jiří Čermák, Branislav Bošanský, Karel Durkota, Viliam Lisý, and Christopher Kiekintveld. 2016. Using correlated strategies for computing Stackelberg equilibria in extensive-form games. In *Proceedings of AAAI Conference on Artificial Intelligence*. AAAI, 439–445.

Jakub Černý. 2016. *Stackelberg Extensive-Form Correlated Equilibrium with Multiple Followers*. Master's thesis. Czech Technical University in Prague.

Xi Chen, Xiaotie Deng, and Shang-Hua Teng. 2009. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM* 56, Article 14 (May 2009), 57 pages. Issue 3.

Vincent Conitzer and Dmytro Korzhyk. 2011. Commitment to correlated strategies. In *Proceedings of AAAI Conference on Artificial Intelligence*. AAAI, 632–637.

Vincent Conitzer and Tuomas Sandholm. 2006. Computing the optimal strategy to commit to. In *Proceedings of ACM Conference on Economics and Computation (EC)*. ACM, 82–90.

Constantinos Daskalakis, Alex Fabrikant, and Christos H. Papadimitriou. 2006a. The game world is flat: The complexity of Nash equilibria in succinct games. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*. Springer Berlin Heidelberg, 513–524.

Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. 2006b. The complexity of computing a Nash equilibrium. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*. ACM, 71–78.

Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. 2000. *Computational Geometry* (2nd ed.). Springer.

Federico Etro. 2007. *Stackelberg Competition and Endogenous Entry*. Springer Berlin, 91–129.

Peter Gritzmann and Bernd Sturmfels. 1993. Minkowski addition of polytopes: Computational complexity and applications to Grbner bases. *SIAM Journal on Discrete Mathematics (SIDMA)* 6, 2 (1993), 246–269.

Anshul Gupta. 2015. *Equilibria in Finite Games*. Ph.D. Dissertation. University of Liverpool.

Anshul Gupta, Sven Schewe, and Dominik Wojtczak. 2015. Making the best of limited memory in multi-player discounted sum games. In *Proceedings of the 6th International Symposium on Games, Automata, Logics and Formal Verification*. Electronic Proceedings in Theoretical Computer Science (EPTCS), 16–30.

Jonathan H. Hamilton and Steven M. Slutsky. 1990. Endogenous timing in duopoly games: Stackelberg or cournot equilibria. *Games and Economic Behavior* 2, 1 (1990), 29–46.

Joshua Letchford. 2013. *Computational Aspects of Stackelberg Games*. Ph.D. Dissertation. Duke University.

Joshua Letchford and Vincent Conitzer. 2010. Computing optimal strategies to commit to in extensive-form games. In *Proceedings of ACM Conference on Economics and Computation (EC)*. ACM, 83–92.

Joshua Letchford, Liam MacDermed, Vincent Conitzer, Ronald Parr, and Charles L. Isbell. 2012. Computing optimal strategies to commit to in stochastic games. In *Proceedings of AAAI Conference on Artificial Intelligence*. AAAI, 1380–1386.

Toshihiro Matsumura. 2003. Stackelberg mixed duopoly with a foreign competitor. *Bulletin of Economic Research* 55 (2003), 275–287.

Thanh H. Nguyen, Francesco M. Delle Fave, Debarun Kar, Aravind S. Lakshminarayanan, Amulya Yadav, Milind Tambe, Noa Agmon, Andrew J. Plumptre, Margaret Driciru, Fred Wanyama, and Aggrey Rwetsiba. 2015. *Making the Most of Our Regrets: Regret-Based Solutions to Handle Payoff Uncertainty and Elicitation in Green Security Games*. Springer International Publishing, 170–191.

Praveen Paruchuri, Jonathan P. Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez, and Sarit Kraus. 2008. Playing games for security: An efficient exact algorithm for solving Bayesian Stackelberg games. In *Proceedings of International Conference on Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 895–902.

Hanif D. Sherali. 1984. A multiple leader Stackelberg model and analysis. *Operations Research* 32, 2 (1984), 390–404.

Milind Tambe. 2011. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press.

Eric van Damme and Sjaak Hurkens. 1999. Endogenous Stackelberg leadership. *Games and Economic Behavior* 28, 1 (1999), 105-129.

Heinrich von Stackelberg. 1934. Marktform und gleichgewicht. *Springer-Verlag*.

Bernhard von Stengel and Francoise Forges. 2008. Extensive-form correlated equilibrium: Definition and computational complexity. *Mathematics of Operations Research* 33, 4 (2008), 1002–1022.

Bernhard von Stengel and Shmuel Zamir. 2010. Leadership games with convex strategy sets. *Games and Economic Behavior* 69, 2 (2010), 446–457.

Haifeng Xu, Zinovi Rabinovich, Shaddin Dughmi, and Milind Tambe. 2015. Exploring information asymmetry in two-stage security games. In *Proceedings of AAAI Conference on Artificial Intelligence*. AAAI, 1057–1063.

Received July 2016; revised December 2016; accepted June 2017

Appendix H

Using Correlated Strategies for Computing Stackelberg Equilibria in Extensive-Form Games

Using Correlated Strategies for Computing Stackelberg Equilibria in Extensive-Form Games

Jiří Čermák[1], **Branislav Bošanský**[1,2], **Karel Durkota**[1], **Viliam Lisý**[1,3], **Christopher Kiekintveld**[4]
[1] Agent Technology Center, Faculty of Electrical Engineering, Czech Technical University in Prague
[2] Department of Computer Science, Aarhus University
[3] Department of Computing Science, University of Alberta
[4] Department of Computer Science, University of Texas at El Paso
jiri.cermak@agents.fel.cvut.cz, branislav.bosansky@agents.fel.cvut.cz, karel.durkota@agents.fel.cvut.cz,
lisy@ualberta.ca, cdkiekintveld@utep.edu

Abstract

Strong Stackelberg Equilibrium (SSE) is a fundamental solution concept in game theory in which one player commits to a strategy, while the other player observes this commitment and plays a best response. We present a new algorithm for computing SSE for two-player extensive-form general-sum games with imperfect information (EFGs) where computing SSE is an NP-hard problem. Our algorithm is based on a correlated version of SSE, known as Stackelberg Extensive-Form Correlated Equilibrium (SEFCE). Our contribution is therefore twofold: (1) we give the first linear program for computing SEFCE in EFGs without chance, (2) we repeatedly solve and modify this linear program in a systematic search until we arrive to SSE. Our new algorithm outperforms the best previous algorithms by several orders of magnitude.

Introduction

The roles of players in many games are often asymmetric. One example is the ability of one player (*the leader*) to commit to a strategy, to which the other players (*the followers*) react by playing their best response. In real-world scenarios, the leader can model a market leader with the power to set the price for items or services, or a defense agency committing to a security protocol to protect critical facilities. Optimal strategies for the players in such situations are described by the Strong Stackelberg Equilibrium (SSE) (Leitmann 1978; von Stengel and Zamir 2004). There are many examples of successful applications of SSE in infrastructure protection (Tambe 2011) as well as protecting the environment and wildlife (Fang, Stone, and Tambe 2015).

In most of the existing works, the game models are simplified and do not consider the sequential interaction among players (Pita et al. 2008; Tsai et al. 2009; Shieh et al. 2012; Jiang et al. 2013). One reason is computational complexity, since computing SSE is often NP-hard when sequential interaction is allowed (Letchford and Conitzer 2010). Another reason is the lack of practical algorithms. The only algorithm designed specifically for computing SSE in two-player imperfect-information general-sum extensive-form games (EFGs) was only introduced recently (Bosansky and Cermak 2015) and formulates the problem as a mixed-integer variant of sequence-form linear program (referred to as BC15). However, the scalability of BC15 is limited as it contains a binary variable for each sequence of actions of the follower.

Our main contribution is a novel algorithm computing SSE in EFGs that offers a dramatic speed-up in computation time compared to BC15. The key idea behind our algorithm is in computing a variant of SSE, where the leader commits to correlated strategies – i.e., the leader can send signals to the follower, and the best response of the follower is to follow these signals. We use this variant to find the original SSE by systematically restricting which signals the leader can send to the follower. This variant of SSE has previously been studied in single step games (Conitzer and Korzhyk 2011), finite turn-based and concurrent-move games (Bosansky et al. 2015), infinite concurrent-move stochastic games (Letchford et al. 2012), and security games (Xu et al. 2015; Rabinovich et al. 2015; Durkota et al. 2015). Formally, it has been defined as Stackelberg Extensive-Form Correlated Equilibrium (SEFCE) in (Bosansky et al. 2015). While it was shown that the utility value for the leader in SSE cannot be closely approximated by SEFCE (Letchford et al. 2012), we show that one can use SEFCE to compute SSE. Since there was no previously known algorithm for computing SEFCE in EFGs, we also show that SEFCE can be found in polynomial time in EFGs without chance.

The paper is structured as follows. After introducing the formalism of EFGs, we formally define both SSE and SEFCE, and give an example of a game where these concepts differ. Next, we present the linear program (LP) for computing SEFCE in EFGs without chance (we describe a modified LP for EFGs with chance in the appendix). Afterwards, we present our algorithm for computing SSE in EFGs that iteratively solves the LP for SEFCE with additional constraints until SSE is reached. Finally, we provide three variants of our algorithm and show that each variant significantly improves the computation time compared to BC15 on randomly generated games and an example of a search game.

Technical Background

Extensive-form games model sequential interactions between players and can be visually represented as game trees. Formally, a two-player EFG is defined as a tuple $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, u, \mathcal{C}, \mathcal{I})$: $\mathcal{N} = \{l, f\}$ is a set of players, the leader and the follower. We use i to refer to one of the players, and $-i$ to refer to his opponent. $\mathcal{H}$ denotes a finite set of *nodes* in the game tree. Each node corresponds to a unique *history* of actions taken by all players and chance from the root of the game; hence, we use the terms history and node interchangeably. We say that h is a *prefix* of h' ($h \sqsubseteq h'$) if h lies on a path from the root of the game tree to h'. $\mathcal{A}$ denotes the set of all actions. $\mathcal{Z} \subseteq \mathcal{H}$ is the set of all *terminal nodes* of the game. For each $z \in \mathcal{Z}$ we define a *utility function* for each player i ($u_i : \mathcal{Z} \to \mathbb{R}$). Chance player selects actions based on a fixed probability distribution known to all players. Function $\mathcal{C} : \mathcal{H} \to [0, 1]$ denotes the probability of reaching node h due to chance; $\mathcal{C}(h)$ is the product of chance probabilities of all actions in history h.

Imperfect observation of player i is modeled via *information sets* $\mathcal{I}_i$ that form a partition over $h \in \mathcal{H}$ where i takes action. Player i cannot distinguish between nodes in any information set $I \in \mathcal{I}_i$. We overload the notation and use $A(I_i)$ to denote possible actions available in each node from information set I_i. We assume that action a uniquely identifies the information set where it is available. We assume *perfect recall*, which means that players remember history of their own actions and all information gained during the course of the game. As a consequence, all nodes in any information set I_i have the same history of actions for player i.

Pure strategies Π_i assign one action for each $I \in \mathcal{I}_i$. A more efficient representation in the form of *reduced pure strategies* Π_i^* assigns one action for each $I \in \mathcal{I}_i$ reachable while playing according to this strategy. A *mixed strategy* $\delta_i \in \Delta_i$ is a probability distribution over Π_i. For any pair of strategies $\delta \in \Delta = (\Delta_l, \Delta_f)$ we use $u_i(\delta) = u_i(\delta_i, \delta_{-i})$ for the expected outcome of the game for player i when players follow strategies δ. A *best response* of player i to the opponent's strategy δ_{-i} is a strategy $\delta_i^{BR} \in BR_i(\delta_{-i})$, where $u_i(\delta_i^{BR}, \delta_{-i}) \geq u_i(\delta_i', \delta_{-i})$ for all $\delta_i' \in \Delta_i$.

Strategies in EFGs with perfect recall can be compactly represented by using the sequence form (Koller, Megiddo, and von Stengel 1996). A *sequence* $\sigma_i \in \Sigma_i$ is an ordered list of actions taken by a single player i in history h. $\emptyset$ stands for the empty sequence (i.e., a sequence with no actions). A sequence $\sigma_i \in \Sigma_i$ can be extended by a single valid action a taken by player i, written as $\sigma_i a = \sigma_i'$. We say that σ_i is a *prefix* of σ_i' ($\sigma_i \sqsubseteq \sigma_i'$) if σ_i' is obtained by finite number (possibly zero) of extensions of σ_i. We use $\sigma_i(I_i)$ and $\sigma_i(h)$ to denote the sequence of i leading to I_i and h, respectively. We use the function $I_i(\sigma_i')$ to obtain the information set in which the last action of the sequence σ_i' is taken. For an empty sequence, function $I_i(\emptyset)$ returns the information set of the root node. A mixed strategy of a player can now be represented as a *realization plan* ($r_i : \Sigma_i \to \mathbb{R}$). A realization plan for a sequence σ_i is the probability that player i will play σ_i under the assumption that the opponent plays to allow the actions specified in σ_i to be played. By $g_i : \Sigma_l \times \Sigma_f \to \mathbb{R}$ we denote the *extended utility function*, $g_i(\sigma_l, \sigma_f) = \sum_{z \in \mathcal{Z} | \sigma_l(z) = \sigma_l \wedge \sigma_f(z) = \sigma_f} u_i(z) C(z)$. If no leaf is reachable with pair of sequences σ, value of g_i is 0.

Solution Concepts in EFGs

Here we provide a formal definition of Strong Stackleberg Equilibrium (SSE) (e.g., in (Leitmann 1978)) and Stackelberg Extensive-Form Correlated Equilibrium (SEFCE) (Bosansky et al. 2015) and give the intuition on an example game.

Definition 1. *A strategy profile $\delta = (\delta_l, \delta_f)$ is a* Strong Stackelberg Equilibrium *if δ_l is an optimal strategy of the leader given that the follower best-responds. Formally:*

$$(\delta_l, \delta_f) = \underset{\delta_l' \in \Delta_l, \delta_f' \in BR_i(\delta_l')}{\arg\max} u_l(\delta_l', \delta_f'). \quad (1)$$

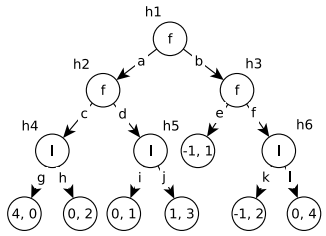
The SSE of the game in Figure 1[1] (the first utility in every leaf is for the leader, second for the follower) prescribes the leader to commit to playing g in h_4, j in h_5, and k in h_6. The strategy of the follower is then to play a in h_1 and d in h_2, leading to the expected utility of 1 for the leader.

In SEFCE we allow the leader to send signals to the follower and condition his strategy on sent signals. More specifically, the leader chooses $\pi_f^* \in \Pi_f^*$ as the recommendations for the follower according to SEFCE before the game starts. The actual recommendation to play some action $a \in A(I_f)$ is revealed to the follower only after he reaches I_f. Therefore, the follower only knows the past and current recommendations, and the probability distribution from which the recommendations are drawn in the future.

Definition 2. *A probability distribution λ on reduced pure strategy profiles Π^* is called a* Stackelberg Extensive-Form Correlated Equilibrium *if it maximizes the leader's utility subject to the constraint that whenever play reaches an information set I where the follower can act, the follower is recommended an action a according to λ such that the follower cannot gain by unilaterally deviating from a in I and possibly in all succeeding information sets given the posterior on the probability distribution of the strategy of the leader, defined by the actions taken by the leader so far.*

The middle table in Figure 1 represents the distribution λ forming the SEFCE of the example game (rows are labeled by Π_l^*, columns by Π_f^*). The leader chooses the signals to the follower to be either $\{a, c\}$ or $\{a, d\}$ based on the probability distribution depicted in the table. Afterwards, the corresponding column defines a valid mixed strategy for the leader and the signals the follower receives in his information sets. More specifically, the follower can receive either c or d in h_2. When the follower receives the recommendation to play c, the leader commits to mix uniformly between g and h in h_4 and to play i in h_5. When the follower receives d as the recommendation, the leader commits to playing g in h_4 and j in h_5. Finally in h_1 the follower is recommended to play a, while the leader commits to play k in h_6 to ensure that the follower does not deviate from playing a. The expected utility of the leader is 1.5 in SEFCE. Note that SSE

[1] This is a corrected version of the example published on AAAI 2016.



	$\{a,c\}$	$\{a,d\}$	$\{b,e\}$	$\{b,f\}$
$\{g,i,k\}$	0.25	0	0	0
$\{g,i,l\}$	0	0	0	0
$\{g,j,k\}$	0	0.5	0	0
$\{g,j,l\}$	0	0	0	0
$\{h,i,k\}$	0.25	0	0	0
$\{h,i,l\}$	0	0	0	0
$\{h,j,k\}$	0	0	0	0
$\{h,j,l\}$	0	0	0	0

	$\emptyset$	(a)	(b)	(a,c)	(a,d)	(b,e)	(b,f)
$\emptyset$	1	1	0	0.5	0.5	0	0
(g)	0.75	0.75	0	0.25	0.5	-	-
(h)	0.25	0.25	0	0.25	0	-	-
(i)	0.5	0.5	0	0.5	0	-	-
(j)	0.5	0.5	0	0	0.5	-	-
(k)	1	1	0	-	-	0	0
(l)	0	0	0	-	-	0	0

Figure 1: (Left) EFG with different SEFCE and SSE. (Middle) SEFCE distribution over Π^*. (Right) SEFCE correlation plan.

in this representation always corresponds to a table where only a single column of the follower has non-zero values.

LP for Computing SEFCE

To compactly represent the behavior described in the middle table in Figure 1, we use a *correlation plan* (von Stengel and Forges 2008) that is quadratic in the size of the game tree, instead of the exponential representation using Π^*. A correlation plan for a sequence pair $p(\sigma_l, \sigma_f)$ represents the expected probability that σ_l will be played if actions from σ_f are recommended to the follower. In order to model SEFCE strategies using the correlation plan, the follower must be able to determine whether following the signal is the best response. Therefore, we need to specify the plan for so called *relevant sequences*. Consider our game in Figure 1; when the follower receives, for example, signal c in h_2, the follower needs to know the commitment of the leader in the related part of the tree – i.e, in both nodes h_4, h_5 – to evaluate whether following the signal is the best response.

Definition 3. *A pair of sequences (σ_l, σ_f) is termed* relevant *if and only if either $\sigma_f = \emptyset$, or $\exists h, h' \in \mathcal{H}, h' \sqsubseteq h; \sigma_l = \sigma_l(h) \wedge h' \in I_f(\sigma_f)$.*

By $rel(\sigma_i)$ we denote the set of sequences of $-i$ which form a relevant pair with σ_i. In our example $rel((a)) = rel((b)) = \Sigma_l$, $rel((b,e)) = rel((b,f)) = \{\emptyset, (k), (l)\}$, and $rel((a,c)) = rel((a,d)) = \{\emptyset, (g), (h), (i), (j)\}$.

Definition 4. *A* correlation plan *(von Stengel and Forges 2008) is a partial function $p : \Sigma_l \times \Sigma_f \to \mathbb{R}$ such that there is a probability distribution λ on the set of reduced strategy profiles Π^* so that for each relevant sequence pair (σ_l, σ_f), the term $p(\sigma_l, \sigma_f)$ is defined and fulfills $p(\sigma_l, \sigma_f) = \sum_{(\pi_l, \pi_f) \in \Pi^*} \lambda(\pi_l, \pi_f)$ where π_l, π_f prescribe playing all of the actions in σ_l and σ_f, respectively.*

Let us now describe the SEFCE strategies (middle table in Figure 1) using the correlation plan (the right table; rows are labeled by Σ_l, columns by Σ_f). Every column of the table corresponds to an expected probability of the occurrence of the leader's sequences when the follower follows his recommendations in the future, and gets the recommendation to play the σ_f corresponding to this column. We use '-' to mark irrelevant sequence pairs. The entry for every σ_l, σ_f is the sum of all the entries corresponding to the pure strategies containing all the actions from σ_l and σ_f in the middle table of Figure 1. The behavior in columns corresponding to sequences (a,c) and (a,d) matches the behavior discussed in the previous section. The behavior in columns for sequence $\emptyset$ and (a) corresponds to the expected probability of playing sequences of the leader given the probability of recommendations, e.g., the probability of playing g for the recommendation (a) is equal to $p((g),(a,c)) + p((g),(a,d))$ (in this case the leader will play uniformly either g with probability 0.5 according to the column for (a,c) or g with probability 1 according to the column for (a,d)). Probabilities in column for (a) allow the follower to evaluate his choices in h_1.

Now we are ready to describe the LP for computing SEFCE in EFGs without chance that uses correlation plan:

$$\max_{p,v} \sum_{\sigma_l \in \Sigma_l} \sum_{\sigma_f \in \Sigma_f} p(\sigma_l, \sigma_f) g_l(\sigma_l, \sigma_f) \tag{2}$$

$$\text{s.t.} \quad p(\emptyset, \emptyset) = 1; \quad 0 \le p(\sigma_l, \sigma_f) \le 1 \tag{3}$$

$$p(\sigma_l(I), \sigma_f) = \sum_{a \in A(I)} p(\sigma_l(I)a, \sigma_f) \;\; \forall I \in \mathcal{I}_l, \forall \sigma_f \in rel(\sigma_l) \tag{4}$$

$$p(\sigma_l, \sigma_f(I)) = \sum_{a \in A(I)} p(\sigma_l, \sigma_f(I)a) \;\; \forall I \in \mathcal{I}_f, \forall \sigma_l \in rel(\sigma_f) \tag{5}$$

$$v(\sigma_f) = \sum_{\sigma_l \in rel(\sigma_f)} p(\sigma_l, \sigma_f) g_f(\sigma_l, \sigma_f) + \sum_{I \in \mathcal{I}_f;\, \sigma_f(I) = \sigma_f} \sum_{a \in A_f(I)} v(\sigma_f a) \;\; \forall \sigma_f \in \Sigma_f \tag{6}$$

$$v(I, \sigma_f) \ge \sum_{\sigma_l \in rel(\sigma_f)} p(\sigma_l, \sigma_f) g_f(\sigma_l, \sigma_f(I)a) + \sum_{I' \in \mathcal{I}_f;\, \sigma_f(I') = \sigma_f(I)a} v(I', \sigma_f)$$
$$\forall I \in \mathcal{I}_f, \forall \sigma_f \in \bigcup\nolimits_{h \in I} rel(\sigma_l(h)), \forall a \in A(I) \tag{7}$$

$$v(\sigma_f(I)a) = v(I, \sigma_f(I)a) \qquad \forall I \in \mathcal{I}_f, \forall a \in A(I) \tag{8}$$

The LP is derived from the computation of Extensive-Form Correlated Equilibrium (von Stengel and Forges 2008) by omitting the incentive constraints for the leader and maximizing the expected utility of the leader (this is similar to the approach in normal-form games (Conitzer and Korzhyk 2011)). Constraints (3) to (5) ensure that p is a well-formed correlation plan. Constraint (6) ensures that $v(\sigma_f)$ represents the expected utility of playing σ_f for the follower, when he follows his recommendations. The first sum represents the expected utility of the leaves reached by playing according to σ_l and σ_f, the second sum represents the contribution of the expected utility from information sets reached by the continuations of σ_f. Constraint (7) ensures that $v(I_f, \sigma_f)$ is the maximum over all possible sequences leaving I_f (denoted as $\sigma_f(I_f)a$ for all $a \in A(I_f)$) after the follower has

received recommendation σ_f. Finally, constraint (8) forces the move recommended to the follower in I_f to be optimal.

Definition 5. *We say that p uses* inconsistent recommendation *in $I \in \mathcal{I}_f$ if and only if p defines two different recommendations for the follower in I. Formally, $\exists a, a' \in A(I), a \neq a', \exists \sigma_l, \sigma'_l \in \bigcup_{h \in I} \sigma_l(h)\ p(\sigma_l, \sigma_f(I)a) > 0 \wedge p(\sigma'_l, \sigma_f(I)a') > 0$. If there exists no such information set we say that p uses only* consistent recommendations.

Theorem 1. *Assume a solution of the LP as described in eqs.* (2) *to* (8) *such that there are only consistent recommendations for the follower. There is a SSE strategy that can be found in polynomial time from the p variables.*

Proof. First, we show how the strategy of the leader is constructed. In every $I \in \mathcal{I}_l$ there is a subset Σ_r of relevant sequences $rel(\sigma_l(I))$ played with a positive probability. The behavior in I is specified by $p(\sigma_l(I)a, \sigma_f)$ for all $a \in A(I)$ for arbitrary $\sigma_f \in \Sigma_r$, as the behavior is the same for all $\sigma'_f \in \Sigma_r$. This is guaranteed by constraint (5) – it forces the probability of $p(\sigma_l, \sigma'_f)$ to be equal to the sum of $p(\sigma_l, \sigma''_f)$ over all extensions σ''_f of σ'_f. Since there can be only a single extension played with positive probability (assumed consistent recommendations) the probabilities must be equal.

For every $I \in \mathcal{I}_f$ there exists at most one action $a \in A(I)$ with $p(\sigma_l, \sigma_f a) > 0$ for some $\sigma_l \in \Sigma_l$ and $\sigma_f = \sigma_f(I)$ (consistent recommendations). By taking these actions and arbitrary actions in information sets where there is no such action a, we obtain a pure strategy for the follower. Finally, due to the correctness of the strategy of the leader proved in the previous step and constraints (6–8), this pure strategy is a best response of the follower. □

Theorem 2. *Assume a solution of the LP as described in eqs.* (2) *to* (8)*. The objective value is greater than or equal to the expected utility of the leader in SSE.*

Proof. Theorem 1 shows that in case the leader uses only consistent recommendations, the value of the LP corresponds to the expected utility of the leader in SSE. If the leader can also use inconsistent recommendations, the value of the LP can be only greater or equal. □

Algorithm Computing SSE

In this section we describe the algorithm for computing SSE. The algorithm uses the linear program for computing SEFCE in case the game is without chance. Otherwise, a slightly modified version of this LP is required, however, the algorithm remains the same. Due to the space constraints, we describe this modified LP in details in the appendix of the paper and refer to one of these two LPs as UB-SSE-LP.

The high level idea of our algorithm (depicted in Algorithm 1) is a recursive application of the following steps: (1) solve the UB-SSE-LP, (2) detect the set of information sets of the follower with inconsistent recommendations $\mathcal{I}_{in}$, (3) restrict the leader to use only consistent recommendations in $\mathcal{I}_{in}$ by adding new constraints to the UB-SSE-LP. Restrictions are added cumulatively, until we arrive at a restricted UB-SSE-LP yielding only consistent p. The expected utility for the leader and the correlation plan p in this

Input: An UB-SSE-LP P
Output: leader's expected utility and strategy profile in SSE

```
1  M ← {(∞, ∅)}; LB ← −∞; p_c ← ∅
2  while M ≠ ∅ do
3      (UB, m) ← max(M)
4      if UB < LB then
5          return (LB, p_c)
6      apply(m, P)
7      if feasible(P) then
8          (value, p) ← solve(P)
9          I_in ← inconsistentRecommendations(p)
10         if I_in = ∅ then
11             if value > LB then LB ← value; p_c ← p
12         else addRestrictions((UB, m), M, I_in, value)
13     revert(m, P)
14 return (LB, p_c)
```

Algorithm 1: Algorithm for computing the SSE.

solution correspond to a candidate for the expected utility of the leader and the strategies of players in SSE. It is only a solution candidate, since we have enforced actions, which may not be a part of SSE.

In more details, Algorithm 1 assumes as the input the UB-SSE-LP P for the game we want to solve. By *modification* $mod = (UB, m)$ we denote a pair of constraints m, to be added to P, and the upper bound UB on the value of P after adding the constraints. M is the set of modifications to be explored during the search for the SSE sorted in descending order of UB. The variable LB stores the expected utility for the leader in the best solution candidate found so far p_c. The main cycle of the algorithm starts on line 2, we iterate until there are no possible modifications of P left. On line 3 we remove the modification with the highest UB from M. We choose such $mod = (UB, m)$ in order to first explore modifications with the potential to lead to solution candidates with the highest expected utility for the leader. The algorithm verifies whether the modification mod (the one with the highest UB value) can improve the current best solution candidate (line 4). If not, the algorithm terminates and the best candidate found so far is the SSE. Otherwise, we add the constraints in m to P (line 6). If the modified P is feasible, the algorithm solves the LP (line 8) obtaining the expected utility of the leader and the correlation plan p. We find the set of information sets where the follower gets an inconsistent recommendation in p (line 9). If p uses only consistent recommendations, this solution corresponds to a solution candidate. If the expected utility for the leader is higher than for the best solution candidate found so far, we replace it (line 11). If $\mathcal{I}_{in}$ is not empty, we generate new modifications to be applied to P and add them to M (line 12). The function $addRestrictions$ will be discussed in more detail in the next subsection, as we explored several options of the modification generation. Finally, we revert the changes in m (line 13). If there are modifications left to be explored in M we continue with the next iteration. Every modification contains all the constraints added to the original P given as an input, therefore after $revert$ we again obtain the original P.

When we enforce the whole strategy of the follower to be consistent with the SSE, the solution of the LP corre-

```
1  addRestrictions((UB, m), M, I_in, value)
2      I ← getShallowest(I_in)
3      for a ∈ A(I) do
4          UB_a ← value; m_a ← m
5          for σ_l ∈ rel(σ_f(I)a) do
6              m_a ← m_a ∪ {p(σ_l, σ_f(I)) = p(σ_l, σ_f(I)a)}
7          M ← M ∪ {(UB_a, m_a)}
```

Algorithm 2: SI-LP.

sponds to the SSE, as it now maximizes the expected utility of the leader under the restriction the follower gets recommended the pure best response to the strategy of the leader. It remains to be shown, that we are guaranteed to add modifications to M which force the correct set of actions of the follower in every version of $addRestrictions$. The final correctness arguments will be provided after discussing the $addRestrictions$ versions used.

Rules for Restricting Follower's Behavior

We examine different approaches for method $addRestrictions$ that generates new modifications of UB-SSE-LP resulting in three different variants of our new algorithm.

In Algorithm 2 we describe the first version of the function $addRestrictions$, labeled SI-LP. On line 2 we find the shallowest information set I, where the follower receives an inconsistent recommendation. We generate modification for every $a \in A(I)$. Every such modification enforces corresponding $a \in A(I)$ to be recommended deterministically. The upper bound is set to the value of P computed before invoking $addRestrictions$. The shallowest information set is chosen to avoid unnecessary modifications in deeper parts of the game tree, which might end up not being visited at all. This version of $addRestrictions$ transforms Algorithm 1 to branch and bound algorithm.

By adding $mod = (UB, m)$ to M for every action in the shallowest information set with inconsistent recommendations, until no such set exists, we ensure that all of the actions which might form a part of SSE will be tried. The behavior in information sets with consistent recommendation need not be restricted, as we are sure that the follower has no incentive to deviate and therefore plays his best response maximizing the expected utility of the leader. Finally, since we assign to UB a value which forms an upper bound on the solution of P after adding constraints m, we are sure that if we terminate the algorithm on line 4 in Algorithm 1, there is indeed no possibility to encounter a solution candidate better than the one yielding the current LB.

A second option, presented in Algorithm 3, chooses $\mathcal{I}_c \subseteq \mathcal{I}_{in}$ (line 2) and restricts the recommendations in every $I \in \mathcal{I}_c$. The restriction in I is done in the following way. First, detect the subset A_c of $A(I)$ of actions which are recommended with positive probability (line 5) and make the recommendation mutually exclusive using binary variables (lines 8 and 9), converting the LP P to a mixed integer linear program (MILP). We use two options of creating $\mathcal{I}_c$. First, we create a singleton containing only the shallowest $I \in \mathcal{I}_{in}$, we refer to this algorithm as SI-MILP. Second, we let $\mathcal{I}_c = \mathcal{I}_{in}$, we refer to this algorithm as AI-MILP. Algo-

```
1  addRestrictions((UB, m), M, I_in, value)
2      I_c ← chooseSets(I_in)
3      UB' ← ∞; m' ← m
4      for I ∈ I_c do
5          A_c ← {a ∈ A(I) | ∃σ_l ∈ Σ_l p(σ_l, σ_f(I)a) > 0}
6          for a ∈ A_c do
7              for σ_l ∈ rel(σ_f(I)a) do
8                  m' ← m' ∪ {p(σ_l, σ_f(I)a) ≤ b_a}
9              m' ← m' ∪ {Σ_{a∈A_c} b_a = 1}
10         M ← M ∪ {(UB', m')}
```

Algorithm 3: MILP.

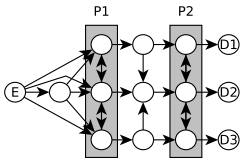
rithm 1 using both SI-MILP and AI-MILP closely resembles constraint generation, with the difference that additional binary variables are also added in every iteration.

If we introduce a binary variable for every action of the follower in the game, we are guaranteed to obtain the SSE, as the MILP then finds a strategy profile maximizing the expected utility of the leader (ensured by the objective), while the follower plays a pure best response to the strategy of the leader (breaking ties in favor of the leader due to the objective), which is the definition of the SSE. If we create some partial enforcement of consistent recommendations using the binary variables and we obtain a pure strategy for the follower then this is again SSE, since the enforcement in the rest of the game would not make any difference as the follower already gets consistent recommendations there. Finally, since we restrict the follower's recommendations until consistent recommendations are obtained, both MILP based rules indeed guarantee to find the SSE.

Experimental Evaluation

We now turn to the experimental evaluation of the three described variants of our algorithm for computing an SSE. We use BC15 (Bosansky and Cermak 2015) as a baseline, state-of-the-art algorithm for computing SSE in EFGs. Single-threaded IBM CPLEX 12.5 solver was used to compute all the (MI)LPs. We use two different domains previously used for evaluation of BC15: a search game representing the scenario where security units defend several targets against an attacker, and randomly generated games.

Search Game. The search game is played on a directed graph (see Figure 2). The follower aims to reach one of the destination nodes (D1 – D3) from starting node (E) in a given number of moves, while the leader aims to catch the follower with one of the two units operating in the shaded areas of the graph (P1 and P2). The follower receives different reward for reaching different destination node (the reward is randomly selected from the interval $[1, 2]$). The leader receives positive reward 1 for capturing the follower. Once the follower runs out of moves without reaching any goal or being captured, both players receive 0. The follower leaves tracks in the visited nodes that can be discovered if the leader visits the node. The follower can erase the tracks in the current node (it takes one turn of the game). The follower does not know the position of the patrolling units, the leader observes only the tracks left by the follower.



Algs \ Steps	4	5	6
BC15	25	863	42,261
SI-LP	9	212	10,790
AI-MILP	6	257	11,997
SI-MILP	5	272	11,263

Figure 2: (Left) Search game graph. (Right) Runtimes in seconds for the search game with increasing depth.

Table 1: Number of games solved in given time intervals.

Algs\Runtime	1s	5s	30s	2min	25min	4h
BC15	2	12	137	245	393	55
SI-LP	583	191	54	12	4	0
AI-MILP	529	259	51	5	0	0
SI-MILP	483	279	72	9	1	0

Randomly Generated Games. We use randomly generated games, where in each state of the game the number of available actions is randomly generated up to a given parameter $\{2, \ldots, \max_A\}$. Each action leads to a state where the opponent is to move and also generates *an observation* for the opponent. An observation is a number from a set $\{1, \ldots, \max_O\}$ and determines partitioning of the nodes into the information sets – for player i, the nodes h with the same history of moves $\sigma_i(h)$ and the observations generated by the actions of the opponent $-i$ belong to the same information set. We generate games of differing sizes by varying parameters $\max_A = \{3, 4\}$, $\max_O = \{2, 3\}$, and depth of the game (up to 5 actions for each player). The utility for the players is randomly generated in the interval $[-100, 100]$. The utilities are correlated with factor set to -0.5 (1 represents identical utilities, -1 zero-sum utilities).

Results

The runtime results on random games are depicted in the top graph of Figure 3. The x-axis shows the number of realization plans of the follower, while the y-axis depicts the time in seconds needed to solve a given instance (both axes are logarithmic). The number of realization plans of the follower is a good estimate of how difficult the game is to solve as it reflects both the size of the game as well as the structure of information sets. Each point represents the mean time needed for every algorithm to solve the instances from a given time interval (at least 600 different instances). The standard errors of the mean values are very small compared to the differences between algorithms and not visible in the graph.

The results show that each of the variants significantly outperforms the previous state-of-the-art algorithm BC15. It typically takes around 10 minutes for BC15 to solve games with 10^6 realization plans of the follower, while our algorithms were often able to find solutions under a second. AI-MILP performs best on average, since it is the least sensitive to the different structure of the instances solved. AI-MILP fixes the behavior in a higher number of information sets and so it preforms a successful trade off between the complexity of solving a single MILP and the number of MILP invocations. The second best approach on average is the SI-MILP.

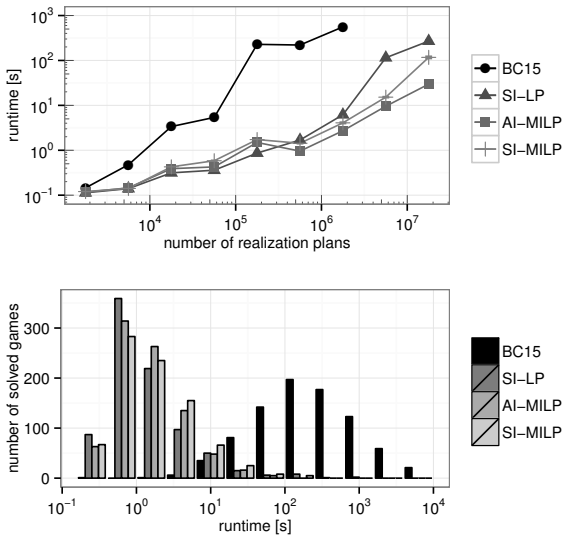


Figure 3: (Top) Runtimes on randomly generated games. (Bottom) Number of solved games in given time intervals.

The average performance is slightly worse due to a higher number of MILP invocations needed to solve more difficult instances (i.e., the ones with many inconsistent recommendations). Finally, the SI-LP has the worst average performance out of the variants of our new algorithm, since SI-LP needs even more LP invocations on more difficult instances in comparison to the previous variants of our algorithm.

Additionally, we provide in the bottom graph of Figure 3 a histogram for number of instances solved (y-axis) within a time interval (x-axis). The results were calculated on random games with the number of realization plans from interval $[3 \cdot 10^5, 3 \cdot 10^6]$. Despite a slightly worse averge performance of SI-LP on these instances, it solved the highest number of instances very fast compared to the other two variants (SI-LP solves 69% of the instances under 1 second, while AI-MILP solves 62% and SI-MILP 57%). The histogram also shows the reason behind the worse average performance of SI-LP. There are multiple instances that SI-LP solves in more then 200 seconds, while such outliers are not present for the latter two variants (the worst outlier for SI-LP took 773 seconds, while the longest time for SI-MILP was 146 seconds, for AI-MILP 57 seconds and for BC15 2.5 hours). For clarity we provide the same data in coarser intervals in Table 1, where the outliers are clearly visible (the label of column represents the upper bound of the corresponding time interval, the label of the column to the left the lower bound of the time interval). The results show that SI-LP is more efficient on instances where the advantage in using the correlated strategies is marginal and there are only few information sets with inconsistent recommendations. On the other hand, AI-MILP offers higher robustness across different instances.

Finally, in Figure 2 we present the results on the search game. All the new approaches performed similarly, outperforming the BC15 in every setting. This shows that our al-

gorithm outperforms BC15 even in an unfavorable setting. The search game has a specific structure, where the strategy space of the leader is large (joint actions of the patrolling units), while the strategy space of the follower is significantly smaller. This structure is favorable for BC15, since it implies a relatively small number of binary variables (the MILP contains one binary variables for each sequence of the follower), with the overall size of the MILP being linear in the size of the game, while the size of our underlying LP is quadratic due to the correlation plan.

Conclusion

We present a novel domain-independent algorithm for computing Strong Stackelberg Equilibria (SSE) in extensive-form games that uses the correlated variant of Stackelberg Equilibria (Stackelberg Extensive-Form Correlated Equilibrium). This work opens several areas for future research. First, our algorithm can be adapted and applied for solving specific domains since its scalability is significantly better in comparison to the existing algorithms. Second, the scalability can most-likely be further improved by employing iterative approaches for solving the underlying linear program. Third, several question were not addressed in our approach and remain open: Is it possible to generalize presented algorithm for computing SSE with multiple followers? Can we relax the assumption of perfect recall?

Acknowledgments

This research was supported by the Czech Science Foundation (grant no. 15-23235S), by the Danish National Research Foundation and The National Science Foundation of China (under the grant 61361136003) for the Sino-Danish Center for the Theory of Interactive Computation and Office of Naval Research Global (grant no. N62909-13-1-N256). This material is based upon work supported by the National Science Foundation (grant no. IIS-1253950). This research was supported by Alberta Innovates Technology Futures through the Alberta Innovates Centre for Machine Learning and Reinforcement Learning and AI Lab and by the Grant Agency of the Czech Technical University in Prague, grant No. SGS15/205/OHK3/3T/13 and SGS15/206/OHK3/3T/13.

References

Bosansky, B., and Cermak, J. 2015. Sequence-Form Algorithm for Computing Stackelberg Equilibria in Extensive-Form Games. In *AAAI Conference on Artificial Intelligence*, 805–811.

Bosansky, B.; Branzei, S.; Hansen, K. A.; Miltersen, P. B.; and Sorensen, T. B. 2015. Computation of Stackelberg Equilibria of Finite Sequential Games. In *11th Conference on Web and Informatics (WINE)*.

Conitzer, V., and Korzhyk, D. 2011. Commitment to Correlated Strategies. In *AAAI Conference on Artificial Intelligence*.

Durkota, K.; Lisy, V.; Bosansky, B.; and Kiekintveld, C. 2015. Approximate solutions for attack graph games with imperfect information. In *Decision and Game Theory for Security*, 228–249. Springer.

Fang, F.; Stone, P.; and Tambe, M. 2015. When Security Games Go Green: Designing Defender Strategies to Prevent Poaching and Illegal Fishing. In *Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, 2589–2595.

Jiang, A. X.; Yin, Z.; Zhang, C.; Tambe, M.; and Kraus, S. 2013. Game-theoretic Randomization for Security Patrolling with Dynamic Execution Uncertainty. In *12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 207–214.

Koller, D.; Megiddo, N.; and von Stengel, B. 1996. Efficient Computation of Equilibria for Extensive two-person Games. *Games and Economic Behavior* 247–259.

Leitmann, G. 1978. On generalized Stackelberg strategies. *Journal of Optimization Theory and Applications* 26(4):637–643.

Letchford, J., and Conitzer, V. 2010. Computing Optimal Strategies to Commit to in Extensive-Form Games. In *11th ACM conference on Electronic commerce*, 83–92.

Letchford, J.; MacDermed, L.; Conitzer, V.; Parr, R.; and Isbell, C. L. 2012. Computing Optimal Strategies to Commit to in Stochastic Games. In *AAAI Conference on Artificial Intelligence*.

Pita, J.; Jain, M.; Marecki, J.; Ordóñez, F.; Portway, C.; Tambe, M.; Western, C.; Paruchuri, P.; and Kraus, S. 2008. Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles International Airport. In *7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 125–132.

Rabinovich, Z.; Jiang, A. X.; Jain, M.; and Xu, H. 2015. Information Disclosure as a Means to Security. In *14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 645–653.

Shieh, E.; An, B.; Yang, R.; Tambe, M.; Baldwin, C.; DiRenzo, J.; Maule, B.; and Meyer, G. 2012. Protect: A deployed game theoretic system to protect the ports of the united states. In *11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 13–20.

Tambe, M. 2011. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press.

Tsai, J.; Kiekintveld, C.; Ordóñez, F.; Tamble, M.; and Rathi, S. 2009. IRIS - A Tool for Strategic Security Allocation in Transportation Networks Categories and Subject Descriptors. In *8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 37–44.

von Stengel, B., and Forges, F. 2008. Extensive-form Correlated Equilibrium: Definition and Computational Complexity. *Mathematics of Operations Research* 33(4):1002–1022.

von Stengel, B., and Zamir, S. 2004. Leadership with Commitment to Mixed Strategies. Technical report, CDAM Research Report LSE-CDAM-2004-01.

Xu, H.; Rabinovich, Z.; Dughmi, S.; and Tambe, M. 2015. Exploring Information Asymmetry in Two-Stage Security Games. In *AAAI Conference on Artificial Intelligence*.

Appendix

Extensive-Form Games with Chance

In this section we analyze the situation when EFGs contain chance. With chance, one cannot directly use the correlation plan as defined in Definition 4 to compute SEFCE – von Stengel and Forges showed (2008) that the presence of chance disrupts the structure of relevant sequences. More specifically, by using correlation plan p, one can describe a richer set of strategies that does not correspond to Extensive-Form Correlated Equilibrium, defined as a probability distribution over (reduced) pure strategy profiles.

We are, however, interested in finding SSE for a given EFG with chance – i.e., such a solution of the LP in eqs. (2) to (8) which recommends only a single pure strategy to the follower. We can exploit this restriction and modify the LP that uses correlation plan according to Definition 4. We show that this new LP (1) returns SSE if all recommendations for the follower are consistent; (2) computes an upper bound on the expected utility of the leader in SSE.

The main difference caused by chance is that multiple nodes in information sets of the leader can now be reached with a non-zero probability in p even if the recommendation for the follower is always consistent. This is not the case in games without chance, since each node in an information set of the leader has the same history for the leader (due to *perfect recall*) and a different sequence of actions of the follower. Therefore, we need to add constraints ensuring that the strategy of the leader cannot depend on actions of chance unobserved by the leader (i.e., they spawn different nodes in an information set of the leader).

Definition 6. *Let $I \in \mathcal{I}_l$ be an information set of the leader and $h \in I$ nodes in this information set. Define $\Gamma_I = (\gamma_1, \gamma_2, \ldots)$ to be a partitioning of the nodes in I based on the history of chance. Every node h belongs to exactly one set γ_k such that all nodes in γ_k share the same sequence of chance actions. Formally,*

$$\forall h \in I \; \exists \gamma_k \in \Gamma_I \; h \in \gamma_k$$
$$\forall h, h' \in \gamma_k \; \sigma_{\mathcal{C}}(h) = \sigma_{\mathcal{C}}(h')$$
$$\forall h \in \gamma_k \; \forall h' \in \gamma_j \; k \neq j \wedge \sigma_{\mathcal{C}}(h) \neq \sigma_{\mathcal{C}}(h')$$

Next, we extend the definition of relevant sequences to a restricted set such that the sequences are relevant due to a specific node in the information set of the leader h:

$$rel(h) = \{\sigma_f \mid \exists h' \in \mathcal{H}, h' \sqsubseteq h; h' \in I_f(\sigma_f)\},$$

additionally there can be no pair of sequences $\sigma_f, \sigma'_f \in rel(h)$ such that one is a strict prefix of another: $\sigma_f \sqsubseteq \sigma'_f$ and $\sigma_f \neq \sigma'_f$. Now we can give a modified version of LP used in our algorithm:

$$\max_{p,v} \sum_{\sigma_l \in \Sigma_l} \sum_{\sigma_f \in \Sigma_f} p_{\sigma_l,\sigma_f} g_l(\sigma_l, \sigma_f) \tag{9}$$

$$\text{s.t.} \quad \text{constraints (3)-(8)} \tag{10}$$

$$s_{I,a_l} = \sum_{h \in \gamma_k} \sum_{\sigma_f \in rel(h)} p(\sigma_l(I)a_l, \sigma_f) \qquad \forall I \in \mathcal{I}_l \forall a_l \in A_l(I) \forall \gamma_k \in \Gamma_I \tag{11}$$

$$0 \leq s_{I,a_l} \leq 1 \qquad \forall I \in \mathcal{I}_l \forall a_l \in A_l(I) \tag{12}$$

We added a new variable s_{I,a_l} that represents the realization probability of action a_l being played in the information set I. Constraint (11) now ensures that the strategy in this information set cannot depend on chance, as shown by the following Lemma and Theorem.

Lemma 1. *Assume a solution of the LP described in eqs. (9-12) such that there are only consistent recommendations for the follower. Then, for every information set of the leader $I \in \mathcal{I}_l$ and every partition $\gamma_k \in \Gamma_I$ there exists at most one sequence of the follower $\sigma_f \in rel(h)$ for some node $h \in \gamma_k$ such that $p(\sigma_l(h), \sigma_f) > 0$.*

Proof. For contradiction, let σ_f, σ'_f be such different relevant sequences for which $p(\sigma_l(I), \sigma_f) > 0$ and $p(\sigma_l(I), \sigma'_f) > 0$ and let $h, h' \in \gamma_k$ (not necessarily different), such that $\sigma_f \in rel(h)$ and $\sigma'_f \in rel(h')$.

Now, thanks to constraint (5) we know that there must be a predecessor of these nodes with strictly positive value of p. Let σ''_f be the longest common prefix of the two sequences of the follower σ_f and σ'_f. By construction of $rel(h)$ we now that $\sigma_f \neq \sigma'_f \neq \sigma''_f$ We distinguish two cases: (1) either the first different action in sequences σ_f and σ'_f after the longest common prefix σ''_f is taken in the same information set, or (2) there are two different information sets in which these different actions are taken. The first case contradicts the assumption that there are no inconsistent recommendations for the follower. In the second case, these two information sets must be reached either due to a different action of the leader or chance, since σ''_f is the longest common prefix and the first different actions in the sequences of the follower are played in these information sets. However, this contradicts the fact that the history for the leader is the same due to the assumption of perfect recall, as well as the history of chance is the same for these two nodes due to the partitioning of Γ_I. □

Theorem 3. *Assume a solution of the LP as described in eqs. (9-12) such that there are only consistent recommendations for the follower. There is a SSE strategy that can be found in polynomial time from variables p.*

Proof. We need to show how $p(\sigma_l, \sigma_f)$ is translated to a valid mixed strategy that corresponds to SSE. First, we show how the strategy of the leader is built. There is only a single relevant sequence $\sigma_f \in \bigcup_{h \in \gamma_k} rel(h)$ with a strictly positive probability in every information set of the leader I (thanks to Lemma 1). Since the history of actions of the leader is the same for all nodes in this information set (perfect recall) and thanks to the constraint (5), the value of the positive probability is the same for each relevant sequence of the follower (only the prefixes of σ_f have positive probability) for each partition γ_k. Therefore, constraint (11) now ensures that the behavior of the leader is unique for all relevant sequences of the follower. And so, for every information set of the leader variables p define a valid strategy that maximizes the expected utility of the leader and that can be computed in the same way as in the standard sequence form in EFGs.

For the case of the follower, there is no inconsistent recommendations by assumption; hence, for every information set I of the follower there exists at most one action $a_f \in A(I)$ with $p(\sigma_l, \sigma_f a_f) > 0$ for some $\sigma_l \in \Sigma_l$ and $\sigma_f = \sigma_f(I)$. By taking these actions and arbitrary actions in information sets where there is no such action a_f, we obtain a pure strategy for the follower. Finally, thanks to the constraints (6–8), this pure strategy is a best response of the follower to the strategy of the leader obtained in the previous step. □

Theorem 4. *Assume a solution of the LP as described in eqs. (9-12). The objective value of the LP is greater or equal to the expected utility of the leader in SSE.*

Proof. The theorem holds as a direct consequence of Theorem 3. We know that in case there are no inconsistent recommendations for the follower, the value of the objective corresponds to SSE. If we do not restrict the recommendations, the value can be only greater or equal. □

Appendix I

Incremental Strategy Generation for Stackelberg Equilibria in Extensive-Form Games

Incremental Strategy Generation for Stackelberg Equilibria in Extensive-Form Games

JAKUB ČERNÝ, Czech Technical University in Prague, Czech Republic
BRANISLAV BOŠANSKÝ, Czech Technical University in Prague, Czech Republic
CHRISTOPHER KIEKINTVELD, University of Texas at El Paso, USA

Dynamic interaction appears in many real-world scenarios where players are able to observe (perhaps imperfectly) the actions of another player and react accordingly. We consider the baseline representation of dynamic games—the extensive form—and focus on computing Stackelberg equilibrium (SE), where the leader commits to a strategy to which the follower plays a best response. For one-shot games (e.g., security games), strategy-generation (SG) algorithms offer dramatic speed-up by incrementally expanding the strategy spaces. However, a direct application of SG to extensive-form games (EFGs) does not bring a similar speed-up since it typically results in a nearly-complete strategy space. Our contributions are twofold: (1) for the first time we introduce an algorithm that allows us to incrementally expand the strategy space to find a SE in EFGs; (2) we introduce a heuristic variant of the algorithm that is theoretically incomplete, but in practice allows us to find exact (or close-to optimal) Stackelberg equilibrium by constructing a significantly smaller strategy space. Our experimental evaluation confirms that we are able to compute SE by considering only a fraction of the strategy space that often leads to a significant speed-up in computation times.

CCS Concepts: • **Theory of computation** → **Algorithmic game theory**; **Exact and approximate computation of equilibria**;

Additional Key Words and Phrases: Extensive-form games; strong Stackelberg equilibrium; correlated equilibrium; strategy generation

1 INTRODUCTION

Many game-theoretic models inspired by real-world scenarios are dynamic, in that they model sequences of interacting moves and observations by the players. The players may have

This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes not with standing any copyright notation here on.
Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

Authors' addresses: Jakub Černý, Czech Technical University in Prague, Technická 2, Prague, 16627, Czech Republic, jakub.cerny@agents.fel.cvut.cz; Branislav Bošanský, Czech Technical University in Prague, Technická 2, Prague, 16627, Czech Republic, bosansky@fel.cvut.cz; Christopher Kiekintveld, University of Texas at El Paso, 500 West University Ave. El Paso, Texas, 79968-0518, USA, cdkiekintveld@utep.edu.

ACM EC'18, June 18–22, 2018, Ithaca, NY, USA. ACM ISBN 978-1-4503-5829-3/18/06…$15.00
https://doi.org/10.1145/3219166.3219219

uncertainty about the effects of actions and may receive only imperfect observations of the actions chosen by other players. The baseline formalism for reasoning about dynamic games with a limited horizon is *extensive form games* (EFGs), which can be represented using game trees. Many scenarios can be modeled as EFG, including card games like poker [5, 15], security games with patrols (e.g., examples in [4, 11]), computer network attacks [6, 7], and synthetic biology games in medicine [16].

The roles of the players in many real-world games are asymmetric. One player (*the leader*) has the power to commit to a strategy and the other player (*the follower*) plays a best response. For example, the leader can correspond to a market leader with the power to set the price for items or services, or a defense agency committing to a security protocol to protect critical facilities. Optimal strategies for the players in such situations are described by the Stackelberg Equilibrium (SE) [12, 22]. We follow the common assumption in the literature that the follower break ties in favor of the leader; hence, we compute a Strong Stackelberg Equilibrium (SSE)[1].

The problem of computing SSE in EFGs is known to be NP-complete [4, 13]. There are several existing algorithms for computing SSE in EFGs. Bošanský and Čermák [4] introduced a mixed-integer linear program (MILP) that extends the sequence form linear program for computing NE in EFGs to SSE [10, 20]. The scalability of this approach was improved by first computing a correlated version of the SSE (called Stackelberg Extensive-Form Correlated Equilibrium, SEFCE) and then using a search to refine the SEFCE into a SSE [19]. Finally, Kroer et al. [11] introduced a novel MILP formulation that incorporates interval uncertainty in the utility of the follower, as well as a limited lookahead approach assuming that the follower is not perfectly rational and can only reason to a limited depth.

We introduce a novel strategy-generation (SG) technique that starts from a small restricted game and incrementally expands the game tree of a two-player extensive-form game to compute SSE. The inspiration for the algorithm is based on previous algorithms that have been very successful for one-shot games (e.g., many types of security games [8, 9, 23]), as well as for zero-sum sequential [2, 14, 18] and EFGs [3]. However, none of these previous SG approaches translates directly to a practical algorithm for EFGs. In the first case, the main step in each iteration is to add sequences of actions of the leader that can potentially increase the objective of a (MI)LP. This is done by computing the reduced costs for variables that are not included in the program yet [8]. We investigate this approach for EFG in Section 3, but encounter a fundamental problem in that this approach adds many unnecessary sequences, leading to "reduced" games that are comparable in size to the original. The SG algorithm for zero-sum EFGs relies heavily on the zero-sum assumption and expands the game tree by adding best-response sequences into the game tree. Unfortunately, this approach does not converge to SSE in a general sum EFG.

We can now state the two main challenges we must solve to develop an effective SG algorithm for computing SSE in EFGs: *(1) What is the representation of the restricted game, and in particular, the abstracted parts of the game tree? (2) What is the methodology for expanding the restricted game?* We address both of these challenges and show that (1) the abstracted parts of the game tree can be effectively represented using a subset of pareto-optimal outcomes; (2) the expansion can be done when an outcome from the abstracted part of the game tree is used in the current solution of the restricted game. Our technique is independent and can be combined with any of the existing algorithms

[1]We expect that our general approach would be applicable for computing other variations of Stackelberg equilibrium, but leave this to future work.

for computing SSE. We use the SEFCE-based algorithm [19] since it has the best-known scalability. The LP for computing SEFCE is quadratic in the size of the game tree, so constructing a smaller game tree should also have a significant performance benefit. In addition to an exact algorithm, we also introduce heuristic variants that lose theoretical guarantees on convergence in exchange for much smaller games; our experimental results show that the heuristic variant can compute exact Stackelberg equilibrium of games with more than 10^7 states by constructing only 8% of the original SEFCE LP. More aggressive heuristics are able to find near-optimal solutions considering only 3% of the LP, while the outcome for the leader is on 6.38% worse compared to the optimum. Often, such a dramatic reduction in the size of the LP leads to significant speed-up in computation times.

2 EXTENSIVE-FORM GAMES AND STACKELBERG SOLUTION CONCEPTS

Extensive-form games model sequential interactions between players and can be visually represented as game trees. Formally, a two-player EFG is defined as a tuple $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, u, \mathcal{C}, \mathcal{I})$: $\mathcal{N} = \{l, f\}$ is a set of players, the leader and the follower. We use i to refer to one of the players, and $-i$ to refer to his opponent. $\mathcal{H}$ denotes a finite set of *nodes* in the game tree. Each node corresponds to a unique *history* of actions taken by all players and chance from the root of the game; hence, we use the terms history and node interchangeably. We say that h is a *prefix* of h' ($h \sqsubseteq h'$) if h lies on a path from the root of the game tree to h'. $\mathcal{A}$ denotes the set of all actions. $\mathcal{Z} \subseteq \mathcal{H}$ is the set of all *terminal nodes* of the game. For each $z \in \mathcal{Z}$ we define a *utility function* for each player i ($u_i : \mathcal{Z} \to \mathbb{R}$). A chance player selects actions based on a fixed probability distribution known to all players. Function $\mathcal{C} : \mathcal{H} \to [0, 1]$ denotes the probability of reaching node h due to chance; $\mathcal{C}(h)$ is the product of chance probabilities of all actions in history h.

Imperfect observation of player i is modeled via *information sets* $\mathcal{I}_i$ that form a partition over $h \in \mathcal{H}$ where i takes action. Player i cannot distinguish between nodes in any information set $I \in \mathcal{I}_i$. We overload the notation and use $A(I_i)$ to denote possible actions available in each node from an information set I_i. We assume that action a uniquely identifies the information set where it is available. We assume *perfect recall*, which means that players remember the history of their own actions and all information gained during the course of the game. As a consequence, all nodes in any information set I_i have the same history of actions for player i.

Pure strategies Π_i assign one action for each $I \in \mathcal{I}_i$. A more efficient representation in the form of *reduced pure strategies* Π_i^* assigns one action for each $I \in \mathcal{I}_i$ reachable while playing according to this strategy. A *mixed strategy* $\delta_i \in \Delta_i$ is a probability distribution over Π_i. For any pair of strategies $\delta \in \Delta = (\Delta_l, \Delta_f)$ we use $u_i(\delta) = u_i(\delta_i, \delta_{-i})$ for the expected outcome of the game for player i when players follow strategies δ. A *best response* of player i to the opponent's strategy δ_{-i} is a strategy $\delta_i^{BR} \in BR_i(\delta_{-i})$, where $u_i(\delta_i^{BR}, \delta_{-i}) \geq u_i(\delta_i', \delta_{-i})$ for all $\delta_i' \in \Delta_i$.

Strategies in EFGs with perfect recall can be compactly represented by using the sequence form [10]. A *sequence* $\sigma_i \in \Sigma_i$ is an ordered list of actions taken by a single player i in history h. $\emptyset$ stands for the empty sequence (i.e., a sequence with no actions). A sequence $\sigma_i \in \Sigma_i$ can be extended by a single valid action a taken by player i, written as $\sigma_i a = \sigma_i'$. We say that σ_i is a *prefix* of σ_i' ($\sigma_i \sqsubseteq \sigma_i'$) if σ_i' is obtained by finite number (possibly zero) of extensions of σ_i. We use $seq_i(I_i)$ and $seq_i(h)$ to denote the sequence of i leading to I_i and h, respectively. We use the function $inf_i(\sigma_i')$ to obtain the information set in which the last action of the sequence σ_i' is taken. For an empty sequence, function $inf_i(\emptyset)$ returns the information set of the root node. A mixed strategy of a player can now be

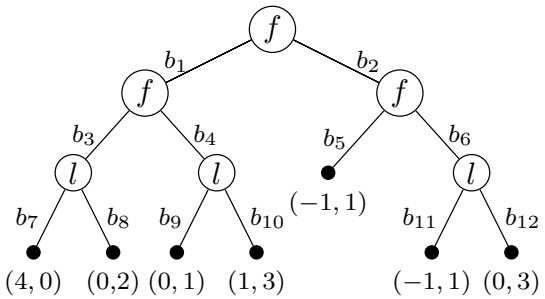
represented as a *realization plan* ($r_i : \Sigma_i \to \mathbb{R}$). A realization plan for a sequence σ_i is the probability that player i will play σ_i under the assumption that the opponent plays to allow the actions specified in σ_i to be played. By $g_i : \Sigma_l \times \Sigma_f \to \mathbb{R}$ we denote the *extended utility function*, $g_i(\sigma_l, \sigma_f) = \sum_{z \in \mathcal{Z} | seq_l(z)=\sigma_l \wedge seq_f(z)=\sigma_f} u_i(z)C(z)$. If no leaf is reachable with a pair of sequences σ, the value of g_i is 0.

Stackelberg Solution Concepts in EFGs. We provide a formal definition of Strong Stackelberg Equilibrium (SSE) (e.g., in [12]) and Stackelberg Extensive-Form Correlated Equilibrium (SEFCE) [1, 13] and give the intuition on an example game.

Definition 2.1. A strategy profile $\delta = (\delta_l, \delta_f)$ is a *Strong Stackelberg Equilibrium* if δ_l is an optimal strategy of the leader given that the follower best-responds. Formally:

$$(\delta_l, \delta_f) = \underset{\delta'_l \in \Delta_l, \delta'_f \in BR_i(\delta'_l)}{\arg\max} u_l(\delta'_l, \delta'_f). \tag{1}$$

The SSE of the game in Figure 1[2] (the first utility in every leaf is for the leader, second for the follower) prescribes the leader to commit to playing actions b_7 and b_{10} in the information sets in the left subtree and b_{11} in the right subtree. The strategy of the follower is then to play b_1 in the root of the game and b_4 in the left subtree, leading to the expected utility of 1 for the leader.



	$\{b_1, b_3\}$	$\{b_1, b_4\}$	$\{b_2, b_5\}$	$\{b_2, b_6\}$
$\{b_7, b_9, b_{11}\}$	0.25	0	0	0
$\{b_7, b_9, b_{12}\}$	0	0	0	0
$\{b_7, b_{10}, b_{11}\}$	0	0.5	0	0
$\{b_7, b_{10}, b_{12}\}$	0	0	0	0
$\{b_8, b_9, b_{11}\}$	0.25	0	0	0
$\{b_8, b_9, b_{12}\}$	0	0	0	0
$\{b_8, b_{10}, b_{11}\}$	0	0	0	0
$\{b_8, b_{10}, b_{12}\}$	0	0	0	0

	$\emptyset$	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}
$\emptyset$	1	1	0	0.5	0.5	0	0
b_1	0.75	0.75	0	0.25	0.5	-	-
b_2	0.25	0.25	0	0.25	0	-	-
$b_1 b_3$	0.5	0.5	0	0.5	0	-	-
$b_1 b_4$	0.5	0.5	0	0	0.5	-	-
$b_2 b_5$	1	1	0	-	-	0	0
$b_2 b_6$	0	0	0	-	-	0	0

Fig. 1. (Left) An EFG with different SEFCE and SSE. Each internal node is labeled by a player who acts in this node, while under every terminal node is a tuple of utilities obtained by the first player and the second player, respectively. Every edge is labeled by an action performed on a way from the node above to the node below. The direction of the edges is omitted, but the tree is assumed to be traversed from top to bottom. (Right Up) The SEFCE distribution over Π^*. (Right Down) The SEFCE correlation plan.

In SEFCE we allow the leader to send signals to the follower and condition his strategy on sent signals. More specifically, the leader chooses $\pi^*_f \in \Pi^*_f$ as the recommendations for the follower according to SEFCE before the game starts. The actual recommendation to play some action $a \in A(I_f)$ is revealed to the follower only after he reaches I_f. Therefore, the follower only knows the past and current recommendations, and the probability distribution from which the recommendations are drawn in the future.

[2]The example is from [19].

Definition 2.2. A probability distribution λ on reduced pure strategy profiles Π^* is called a *Stackelberg Extensive-Form Correlated Equilibrium* if it maximizes the leader's utility subject to the constraint that whenever play reaches an information set I where the follower can act, the follower is recommended an action $\tilde{a}$ according to λ such that the follower cannot gain by unilaterally deviating from $\tilde{a}$ in I and possibly in all succeeding information sets given the posterior on the probability distribution of the strategy of the leader, defined by the actions taken by the leader so far.

SEFCE of the EFG in Figure 1 is shown in the top table, with the rows being labeled by the leader's strategies Π_l^* and the columns by the follower's strategies Π_f^*). At the beginning of the game, the leader decides to send either $\{b_1, b_3\}$ or $\{b_1, b_4\}$ as a recommendation to the follower according to the probabilities in the table. In both cases, the follower commits to playing action b_{11} in order to incentivize the follower to play b_1. If the follower receives b_3, the leader plays action b_9 and mixes uniformly between b_7 and b_8 in his right-most information set. If the follower is recommended b_4, the leader commits to playing b_7 and b_{10}. The expected utility of the leader in this equilibrium is 1.5, which is strictly better than in the SSE.

Using SEFCE for computing SSE. The correlated variant of the Stackelberg equilibrium can be used for computing SSE. The main idea is to find SEFCE and then iteratively add constraints so the recommendations for the follower are such that in each information set, the follower can receive only a single action to be played as a recommendation. In that case, the follower is playing a pure best response and the strategy of the leader is the same as in the SSE.

We now describe the linear program for computing SEFCE. In this linear program, the strategies of the players are represented using a correlation plan for relevant sequences. The sequences are termed *relevant* when decisions at the information sets reachable by one of the sequences can affect the decisions at the information sets reachable by the other sequence.

Definition 2.3 (Relevant sequences [21]). A pair of sequences (σ_1, σ_2) is termed *relevant* if and only if $\exists i \in \{1, 2\}$ either $\sigma_i = \emptyset$ or $\exists h, h' \in H, h' \sqsubseteq h; \sigma_i = seq_i(h) \wedge \sigma_{-i} = seq_{-i}(h')$.

The set of sequences of player $-i$ which form a relevant pair with σ_i is denoted $rel(\sigma_i)$. For the EFG depicted in Figure 1 $rel(b_1) = rel(b_2) = \Sigma_l$, $rel(b_2b_5) = rel(b_2b_6) = \{\emptyset, b_{11}, b_{12}\}$, and $rel(b_1b_3) = rel(b_1b_4) = \{\emptyset, b_7, b_8, b_9, b_{10}\}$. Now it is possible to define a generalization of a realization plan suitable for joint probabilities of pairs of sequences, called a *correlation plan*. We have to ensure that in mutually relevant information sets the consistency of recommendations cannot be violated.

Definition 2.4 (A correlation plan [21]). A partial function $p : \Sigma_1 \times \Sigma_2 \to \mathbb{R}$ is a correlation plan if there is a probability distribution λ on the set of reduced strategy profiles Π^* so that for each relevant sequence pair (σ_1, σ_2), the term $p(\sigma_1, \sigma_2)$ equals to $p(\sigma_1, \sigma_2) = \sum_{(\pi_1, \pi_2) \in \Pi^*} \lambda(\pi_1, \pi_2)$ where π_1, π_2 prescribe playing all of the actions in σ_1 and σ_2, respectively.

The correlation plans describe a joint realization probability $p(\sigma_l, \sigma_f)$ that a sequence σ_f is recommended to the follower, in case the leader plays sequence σ_l. The bottom table of Figure 1 represents the correlation plan of the SEFCE strategies. The rows of the table are labeled by Σ_f, while columns are labeled by Σ_l. In every row identified by σ_f is depicted the probability of the leader playing the corresponding column sequence in case the follower is recommended the sequence σ_f and follows his recommendations. The irrelevant pairs of

sequences are denoted '-'. The correlation plan of every relevant pair (σ_l, σ_f) is the sum of all probabilities of pure strategies containing actions from σ_l and σ_f in the top table.

In [21] the authors proved that correlation plans are sufficient to characterize the set of extensive-form correlated equilibrium (EFCE) in two-player games with no chance nodes. In [19] the authors used this characterization to formulate the following linear program computing SEFCE.

THEOREM 2.5 (STACKELBERG EXTENSIVE-FORM CORRELATED EQUILIBRIUM IN TWO–PLAYER GAME WITHOUT CHANCE MOVES [19]). *The distribution λ on Π^* defines a SEFCE if and only if λ is a solution of the following linear program that maximizes leader's expected utility*

$$\max_{p,v} \sum_{\sigma_l \in \Sigma_l} \sum_{\sigma_f \in \Sigma_f} p(\sigma_l, \sigma_f) g_l(\sigma_l, \sigma_f) \tag{2}$$

and the respective correlation plan p satisfies

$$p(\emptyset, \emptyset) = 1; \quad 0 \leq p(\sigma_l, \sigma_f) \leq 1 \tag{3}$$

$$p(seq_l(I), \sigma_f) = \sum_{a \in A(I)} p(seq_l(I)a, \sigma_f) \quad \forall I \in \mathcal{I}_l, \forall \sigma_f \in rel(\sigma_l) \tag{4}$$

$$p(\sigma_l, seq_f(I)) = \sum_{a \in A(I)} p(\sigma_l, seq_f(I)a) \quad \forall I \in \mathcal{I}_f, \forall \sigma_l \in rel(\sigma_f) \tag{5}$$

$$v(\sigma_f) = \sum_{\sigma_l \in rel(\sigma_f)} p(\sigma_l, \sigma_f) g_f(\sigma_l, \sigma_f) + \\ + \sum_{I \in \mathcal{I}_f;\ seq_f(I) = \sigma_f} \sum_{a \in A_f(I)} v(\sigma_f a) \quad \forall \sigma_f \in \Sigma_f \tag{6}$$

$$v(I, \sigma_f) \geq \sum_{\sigma_l \in rel(\sigma_f)} p(\sigma_l, \sigma_f) g_f(\sigma_l, seq_f(I)a) + \sum_{I' \in \mathcal{I}_f;\ seq_f(I') = seq_f(I)a} v(I', \sigma_f) \\ \forall I \in \mathcal{I}_f, \forall \sigma_f \in \bigcup_{h \in I} rel(seq_l(h)), \forall a \in A(I) \tag{7}$$

$$v(seq_f(I)a) = v(I, seq_f(I)a) \qquad \forall I \in \mathcal{I}_f, \forall a \in A(I) \tag{8}$$

The first three constraints enforce the consistency of the correlation plan. The following constraint ensures that v_{σ_f} is a representation of an expected payoff of the follower when he plays σ_f, assuming he follows his recommendations. The constraint consists of two parts – the first sum computes the expected utility of the leafs reached by playing according to σ_l and σ_f, the second sum adds the contribution of the expected utility of information sets reachable by all the extensions of σ_f. The next constraint guarantees that the expected payoff $v(I, \sigma_f)$ is the maximum over all possible sequences leaving the information set I (denoted as $seq_f(I)a$ for all possible actions $a \in A(I)$) after the follower is recommended to play σ_f. Finally, the last constraint forces the move which is recommended to the follower in the information set I to be optimal. The value of SEFCE is always greater or equal than the value of SSE.

THEOREM 2.6 ([19]). *Assume a solution of the LP as described in Theorem 2.5. The objective value is greater than or equal to the expected utility of the leader in SSE.*

As stated before, SSE can be reached by a branch-and-bound algorithm [19] so that the recommendations for the follower are unique – the authors define *inconsistent recommendations* that are then fixed by a branch-and-bound type of search algorithm (BnB).

Definition 2.7 (Inconsistent recommendations [19]). We say that p uses *inconsistent recommendation* in $I \in \mathcal{I}_f$ if and only if p defines two different recommendations for the follower in I. Formally, $\exists a, a' \in A(I), a \neq a', \exists \sigma_l, \sigma'_l \in \bigcup_{h \in I} seq_l(h)\ p(\sigma_l, seq_f(I)a) > 0 \wedge p(\sigma'_l, seq_f(I)a') > 0$. If there exists no such information set we say that p uses only *consistent recommendations*.

ALGORITHM 1: BnB-Based Algorithm for computing SSE.

Input: An UB-SSE-LP P
Output: leader's expected utility and strategy profile in SSE
$M \leftarrow \{(\infty, \emptyset)\}$; $LB \leftarrow -\infty$; $p_c \leftarrow \emptyset$
while $M \neq \emptyset$ **do**
  $(UB, m) \leftarrow \max(M)$
  if $UB < LB$ **then**
    return $(LB,\ p_c)$
  apply(m, P)
  if feasible(P) **then**
    $(value, p) \leftarrow$ solve(P)
    $\mathcal{I}_{in} \leftarrow$ inconsistentRecommendations(p)
    if $\mathcal{I}_{in} = \emptyset$ **then**
      if $value > LB$ **then** $LB \leftarrow value$; $p_c \leftarrow p$
    else addRestrictions((UB, m), M, $\mathcal{I}_{in}$, $value$)
  revert(m, P)
return $(LB,\ p_c)$

The Algorithm 1 can be decomposed into iterative applications of the following steps: (1) solve an LP P in line 2, (2) find the set of information sets of the follower with inconsistent recommendations $\mathcal{I}_{in}$ in line 1, and (3) restrict the leader's strategy to use only consistent recommendations in $\mathcal{I}_{in}$ by adding new constraints m to the LP P in line 1. The restrictions are added recursively until a restricted LP for SEFCE uses only consistent correlation plan p. At the beginning, the algorithm is initiated with an LP computing SEFCE.

3 CHALLENGES IN STRATEGY GENERATION FOR COMPUTING SSE IN EFGS

We now describe the general framework of a strategy generation (SG) technique for computing a Strong Stackelberg Equilibrium (SSE) in extensive-form games (EFGs). We formally define a restricted game as a subgraph of the original game tree and describe the baseline SG algorithm that extends the algorithm for one-shot games [8]. However, we show on a simple example that such a direct adaptation can easily generate the full game tree.

3.1 Strategy Generation for SSE Using Reduced Costs

The main idea of the previous approach [8] is to decompose the mathematical program for computing SSE into a master problem and a slave problem. In the master problem, we solve for SSE of a smaller restricted game (RG), where the RG is a subset of the original unrestricted game. In the slave problem, we search for such strategies of the leader to be added into the RG which have the largest positive impact on the objective, given the solution of the master problem. The impact is measured using *reduced costs*, calculated from the dual solution.

We translate this idea for computing SSE in EFGs while exploiting SEFCE solution concept. Therefore, our master program corresponds to the LP formulation of SEFCE (LP

in Theorem 2.5) and the slave problem identifies which strategies of the leader should be added to the restricted game.

After the SEFCE is found, the BnB algorithm is called in order to compute SSE. The only change from the original version depicted in Algortihm 1 is that it solves for SEFCE in the restricted game and after adding every restriction, the algorithm checks whether the RG should be expanded.

Iterative Construction of SEFCE LP. We now detail how we translate this idea for computing SSE in EFGs. First, our master program – the LP formulation of SEFCE – will be solved for a restricted game that can be fully specified by the subset of sequences of the leader (called *allowed sequences*). Similarly to [3], we define the sets of nodes, actions, and information sets as subsets of the original unrestricted sets based on the allowed sequences. The RG is denoted as $G' = (\mathcal{N}, \mathcal{H}', \mathcal{Z}', \mathcal{A}', u, \mathcal{C}, \mathcal{I}')$ and the set of all sequences in the RG is denoted Σ'.

We assume the RG is always closed on prefixes of leader's sequences, such that for any leader's sequence in RG, all follower's relevant sequences are in RG. The leader's sequences hence fully define the RG. The set $\mathcal{H}'$ can be derived as

$$\mathcal{H}' \leftarrow \{h \in \mathcal{H} : \forall i \in \mathcal{N}\ seq_i(h) \in \Sigma'\}, \tag{9}$$

and the restriction $\mathcal{A}'$ of $\mathcal{A}$ to the sequences in the RG is constructed as

$$\mathcal{A}'(h) \leftarrow \{a \in \mathcal{A}(h) : ha \in \mathcal{H}'\} \quad \forall h \in \mathcal{H}'. \tag{10}$$

The leafs in the RG are $\mathcal{Z}' = \mathcal{Z} \cap \mathcal{H}'$. Finally, the information sets of the RG are derived as

$$\mathcal{I}'_i \leftarrow \{I_i \in \mathcal{I}_i : \exists h \in \mathcal{H}' \cap I_i\} \quad \forall i \in \mathcal{N}. \tag{11}$$

In the slave problem, we search for the leader's sequences not contained in the RG. The dual solution as for computing the reduced costs follows:

$$rc(\sigma_l) = \max_{\sigma_f \in rel(\sigma_l)} rc(p(\sigma_l, \sigma_f)), \tag{12}$$

where

$$\begin{aligned} rc(p(\sigma_l, \sigma_f)) \ = g_l(\sigma_l, \sigma_f) &- \sum_{I_l \in \mathcal{I}'_l : seq(I_l) = \sigma_l} d_4(I_l, \sigma_f) + d_4(inf(\sigma_l), \sigma_f) - \\ &- \sum_{I_f \in \mathcal{I}'_f : seq(I_f) = \sigma_f} d_5(I_f, \sigma_l) + d_5(inf(\sigma_f), \sigma_l) - d_6(\sigma_f) + \\ &+ \sum_{I_f \in \mathcal{I}'_f : \sigma_f \in \bigcup_{h \in I} rel(seq_l(h))} \ \sum_{a \in \mathcal{A}'(I_f)} g_f(\sigma_l, seq_f(I_f)a) d_7(I_f, \sigma_f, a), \end{aligned} \tag{13}$$

and the $d_j(\dots)$, $j \in \{4, 5, 6, 7\}$, are the dual values of the constraints 4, 5, 6 and 7 in the current solution of the LP computing SEFCE. In order for the algorithm to guarantee a convergence to an optimum, the RG does not contain any temporary leafs, in contrast to [3]. After enumerating the costs for all leader's sequences outside the RG, the sequences with the positive reduced costs $\hat{\Sigma}$ are added to the RG. Simultaneously, we include also all relevant sequences for the follower. The sequences Σ' hence change as

$$\Sigma' \leftarrow \Sigma' \cup \hat{\Sigma} \cup rel\left(\hat{\Sigma}\right). \tag{14}$$

The LP for the restricted game is then extended by generating new correlation plan constraints 4 and 5; and constraints 6, 7 and 8 in case a new follower's sequence enter the RG. The

new correlation pairs $p(\sigma_l, \sigma_f)$ are added into the already existing constraints and to the objective.

At the beginning of the algorithm, the initial restricted game is constructed using the sequences forming an arbitrary reduced pure strategy of the leader. In practice, we started from the left-most branch of the game tree and continued adding the sequences of the leader until we arrived to the full reduced pure strategy. The process of solving the master problem and the slave problems is then repeated until all remaining leader sequences out of the RG have a non-positive cost.

3.2 Limitations of Reduced Costs for SSE in EFGs

However, the practical experiments with reduced costs show that the final sizes of the RGs often reach the size of the original game.

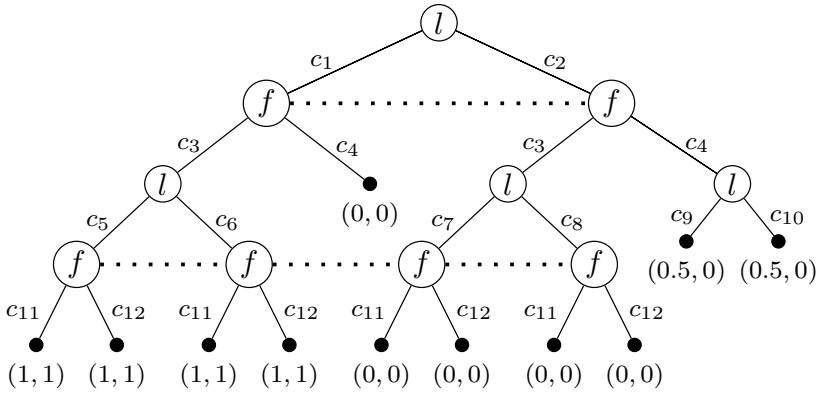


Fig. 2. An example of an EFG where the final RG for the strategy generation with reduced costs is equal to the complete game. The nodes which belong to the same information set are connected by a dashed line. Otherwise, the figure follows a standard denotation of an extensive-form game.

Example 3.1. As an example, consider an EFG depicted in Figure 2. At the beginning, the RG is initialized as

$$\Sigma' \leftarrow \{\emptyset, c_1, c_1c_5, c_3, c_4, c_{11}, c_{12}\}. \tag{15}$$

Note that Σ' already describes a SEFCE in this game. After the first iteration, the sequences $\hat{\Sigma} = \{c_1c_6, c_2, c_2c_9, c_2c_{10}\}$ are added into the RG. c_1c_6, c_2c_9 and c_2c_{10} are included, because they lead to the leafs with the positive utility and no dual variable is able to induce a non-positive reduced cost. The sequence c_2 is added because of the positive value of the dual variable $(4)(inf(c_2), \emptyset)$ used in computing $rc(p(c_2, \emptyset))$. The RG is hence extended into

$$\begin{aligned}\Sigma' \leftarrow \{&\emptyset, c_1, c_1c_5, c_1c_6, c_2, c_2c_9, c_2c_{10},\\ &c_3, c_3c_{11}, c_3c_{12}, c_4, c_{11}, c_{12}\}.\end{aligned} \tag{16}$$

In the second iteration, the positive value of the dual variable $(4)(inf(c_2c_7), c_4)$ enforces adding also sequences c_2c_7, c_2c_8 and $\Sigma = \Sigma'$, i.e. the RG in equal to the complete game.

The algorithm based on reduced costs expands the whole game tree, even though the SSE is clearly located in the left subtree. Our experiments gave comparable results even when the algorithm was modified to add only the sequences with the highest costs, or by initializing the restricted game using either a leaf with the highest utility for the leader, or a Nash equilibrium in the zero-sum variant of the games where the follower's utilities are set to be complementary to the leader's utilities.

ALGORITHM 2: Gadget-Based Algorithm for computing SEFCE.

Input: An EFG G
Output: leader's expected utility and strategy profile in SEFCE
$(M, \Sigma') \leftarrow$ expand(root(G), $\emptyset$)
$expansionNeeded \leftarrow$ size(M) > 0
while *expansionNeeded* **do**
  for *state* h *in* M **do**
    $\Sigma' \leftarrow \Sigma' \cup$ createGadget(h)
  $(value, p) \leftarrow$ solve(Σ')
  $M \leftarrow \emptyset$, $E \leftarrow$ getGadgetsToExpand(p)
  $expansionNeeded \leftarrow False$
  for *state* h *in* E **do**
    $(M', \Sigma') \leftarrow$ expand(h, Σ')
    $M \leftarrow M \cup M'$
    if *size*(M') > 0 **then** $expansionNeeded \leftarrow True$
return ($value$, p)

The main problem of this approach is in the inability of the reduced costs to evaluate the true impact of a sequence because the utilities are located only in the leafs. In order to design more effective strategy-generation methods, it is necessary to introduce generalized temporary leafs with utilities for non-terminal sequences. When designing the temporary leafs in the Stackelberg setting, we have to keep in mind that the leader can use the individual leafs to create threats against the follower. He can either intentionally lure the follower into a specific subtree or to punish him in another. In contrast to the double-oracle methods, the temporary leafs have to consist of more complex tree structures than just the single game states.

4 GADGET-BASED STRATEGY GENERATION FOR EFGS

We introduce a novel strategy-generation algorithm that follows the intuition and avoids adding complete sequences into the game tree by abstracting selected parts of the game tree. We term these abstractions as *gadgets*: a multi-level subtree is replaced with a gadget consisting of a node of the leader and actions leading directly to leafs that correspond to a subset of leafs of the original subtree. The overall structure of the algorithm is the same as in Section 3. In the master problem, we solve for SEFCE in the RG with the gadgets acting as the temporary leafs while ensuring that the gadget-based restricted game always provides an upper bound on SEFCE of the original game. In the slave problem, we identify reachable gadgets and expand the RG. If no gadget is reachable by a current strategy, we reached the equilibrium and it follows that the subtrees rooted in gadgets are not a part of the equilibrium. Algorithm 2 depicts the main steps of the algorithm. We now describe these steps in more details.

Construction of Gadgets. We explain how gadgets are constructed. Let h be the state of the leader which should serve as the root of the gadget. We find all leafs Z_h reachable from h. The leafs can be visualized as points in a two-dimensional space where one dimension correspond to the utility of the leader and the second one corresponds to the utility of the follower (see Figure 4 for an example of this visualization). Since these points represent possible outcomes in the abstracted game tree and the goal is to maximize the utility for the leader, it is sufficient to keep only a subset of Z_h that correspond to the upper convex

hull of these points – for each reachable utility of the follower we seek maximal utility of the leader. We add new sequences, such that every gadget leaf is reachable from h by one leader's action. Note that once we add these new sequences to the set of sequences in the RG Σ', by the definition of information set, h is no longer a part of the same information set as in the original unrestricted game.

ALGORITHM 3: Creating the gadget.

Input: A state h
Output: sequences added to RG
$\hat{\Sigma} \leftarrow \emptyset$
$Z_h \leftarrow$ getLeafsUnder(h)
$Z_h \leftarrow$ getUpperConvexHull(Z_h)
for *state* z *in* Z_h **do**
  $a_{z,h} \leftarrow$ createNewAction(z, h)
  $\hat{\Sigma} \leftarrow \hat{\Sigma} \cup seq_l(h)a_{z,h}$
return $\hat{\Sigma}$

ALGORITHM 4: Expanding the gadget.

Input: A gadget root h, a set of sequences Σ'
Output: new gadget roots, updated set Σ'
$\Sigma' \leftarrow \Sigma' \backslash \{\sigma \in \Sigma' : \exists z\ \sigma = seq_l(h)a_{z,h}\}$
$H_h \leftarrow$ getShallowestLeaderStates(h)
for *state* g *in* H_h **do**
  $\hat{\Sigma}_l \leftarrow \{\sigma \in \Sigma_l : \sigma \sqsubseteq \sigma_l(g)\}$
  $\hat{\Sigma}_f \leftarrow \{\sigma \in \Sigma_f : \sigma \sqsubseteq \sigma_f(g)\}$
  $\Sigma' \leftarrow \Sigma' \cup \hat{\Sigma}_f \cup \hat{\Sigma}_l$
return (H_h, Σ')

Expansion of Gadgets. Now we describe the expansion of the RG. Let h be the state of the leader which should be expanded and Σ' the current set of sequences in the RG. In case h is a root of the gadget, we first delete from Σ' all gadget sequences associated with h. We search the subtree in the unrestricted game under h and find the set H_h of the shallowest leader's states in each branch, as

$$H_h \leftarrow \{g \in \mathcal{H}_l : \sigma(h) \sqsubseteq seq(g); \nexists g' \in \mathcal{H}_l\ \sigma(h) \sqsubseteq seq(g') \sqsubseteq seq(g)\} \tag{17}$$

For each state g in H_h we add to Σ' all prefixes of $seq_l(g)$ and $seq_f(g)$. Finally, all states in H_h are identified as the new gadget roots.

It remains to explain which gadgets have to be expanded, based on the correlation plan p of the current solution. First, we find the set R of all gadgets reachable by p as

$$R \leftarrow \{h \in \mathcal{H}_l : \exists z\ \exists \sigma_f\ p(seq_l(h)a_{z,h}, \sigma_f) > 0\}. \tag{18}$$

We have to ensure the gadgets provide an upper bound on SEFCE. To ensure that, the RG has to be *closed on the information sets of the follower* – whenever a node h where the follower acts is added into the restricted game, all nodes from the information set I, such that h belongs to this information set, also have to be added to the restricted game.

Lemma 4.1. *Let G' be a RG closed on the information sets of the follower. Then for the utility of the leader in SEFCE of G' (denoted as $SEFCE(G')$) it holds that $SEFCE(G') \geq SEFCE(G)$, where G is the complete game.*

Proof. Because G' is closed on the information sets of the follower, the belief of the follower that he is located in a given state of his information set in RG is fully determined by the strategy of the leader in the RG. Therefore, the strategy of the follower in this information set cannot be affected by any strategy of the leader in the subtrees rooted in the gadgets. Because gadgets assume that leader can always obtain any outcome in the subtree under the gadget, the abstraction represented by a gadget overestimates leader's strategizing ability in the subtree, hence providing an upper bound on $SEFCE(G)$. □

The gadgets which have to be expanded in the current iteration of the algorithm are hence both all the reachable gadgets and also the minimum set of gadgets which will retain the

property of the RG being closed on the information sets, given that all gadgets in R are expanded.

The gadget algorithm creates a restricted game similarly to the strategy-generation with the reduced costs. The RG is again fully defined by the sequences of the leader. However, the set Σ' is now updated in a more complex manner, as described in the previous paragraphs about creating the gadgets and expanding the gadgets. Let $\tilde{\Sigma} = \Sigma' \backslash \Sigma$. The sets $\mathcal{H}', \mathcal{Z}', \mathcal{A}'$ and $\mathcal{I}'$ can be derived as follows:

$$\mathcal{H}' \leftarrow \{h \in \mathcal{H} : \forall i \in \mathcal{N}\ seq_i(h) \in \Sigma'\} \cup \{z : \exists h\ seq_l(h)a_{z,h} \in \tilde{\Sigma}\} \tag{19}$$

$$\mathcal{Z}' \leftarrow \mathcal{H}' \cap \mathcal{Z} \tag{20}$$

$$\mathcal{A}'(h) \leftarrow \{a \in \mathcal{A}(h) : ha \in \mathcal{H}'\} \cup \{a_{z,h} : \exists z\ seq_l(h)a_{z,h} \in \tilde{\Sigma}\} \quad \forall h \in \mathcal{H}' \tag{21}$$

$$\mathcal{I}'_i \leftarrow \{I_i \in \mathcal{I}_i : \exists h \in \mathcal{H}' \cap I_i; \nexists z\ seq_l(h)a_{z,h} \in \tilde{\Sigma}\} \cup \{I_h : \exists z\ seq_l(h)a_{z,h} \in \tilde{\Sigma}\} \quad \forall i \in \mathcal{N}. \tag{22}$$

The LP for computing SEFCE is altered according to the current set Σ'.

Lemma 4.2. *Let $\hat{G}$ be a modified game constructed from the original game G by forbidding some actions of the leader. Then $SEFCE(\hat{G}) \leq SEFCE(G)$.*

Proof. Because some actions of the leader are forbidden, the set of strategies available to the leader is restricted, while the set of strategies of the follower remains unconstrained. In case the excluded strategies are not a part of the SEFCE support in G, then $SEFCE(\hat{G}) = SEFCE(G)$. Otherwise some leafs of SEFCE become unreachable, the commitment advantage of the leader is hence reduced, resulting in a worse strategy for the leader and $SEFCE(\hat{G}) \leq SEFCE(G)$. □

Now we are ready to prove the algorithm converges to SEFCE.

Theorem 4.3. *Assume a solution p of Algorithm 2. Then p is a correlation plan of a SEFCE in the original unrestricted game. Algorithm 2 terminates in a finite time.*

Proof. Since every EFG is finite, there is a finite number of gadgets which can be created. Once a gadget is expanded, it can never be created again, because the game tree is expanded top to bottom. Therefore, there is a finite number of LPs to solve. Algorithm 2 hence terminates in a finite time.

In every iteration, Algorithm 2 finds SEFCE in the current RG, such that the utility of the leader is maximal. In the worst case, the restricted game equals the complete game and it cannot be extended any further and the algorithm terminates, returning a SEFCE. Otherwise, there are states in the complete game, which are not a part of the final RG G'. Algorithm 2 terminates once no gadget is reachable by p in the current solution. G' can be hence seen as a modified game in which some actions (those not added to the RG) are forbidden. Therefore, by Lemma 4.2 it holds that $SEFCE(G') \leq SEFCE(G)$. Because RG is closed on the information sets of the follower, by Lemma 4.1 $SEFCE(G') \geq SEFCE(G)$. The SEFCE in the RG is hence a SEFCE in the original unrestricted game. □

4.1 Heuristic Gadget-Based Strategy Generation for EFGs

The effectiveness of the complete algorithm described in the previous section is still limited. However, at the cost of losing theoretical guarantees, the algorithm can be turned into a heuristic one and its performance can be measured experimentally. To this end, we introduce three main heuristics: (1) we relax the requirement that the restricted game (RG) has to be

closed on the information sets of the follower; (2) the upper convex hull of points in a gadget can be approximated using fewer points; (3) the leader utility can be artificially decreased in order to compensate for the overestimation caused by the fact that the maximum-value leafs are rarely reached if the follower plays the best response. We applied all these heuristics and we now describe each of the heuristics in more detail:

Restricted Game Not Closed on Information Sets of the Follower. In order to guarantee the convergence to the equilibrium, the requirement for the RG to be closed on follower's information sets is necessary. Consider a variant of our algorithm, where we relax the condition for maintaining the closed RG on the information sets of the follower and we expand only those gadgets that are reachable in the current solution. The following example shows a game where such a modification leads to a suboptimal solution.

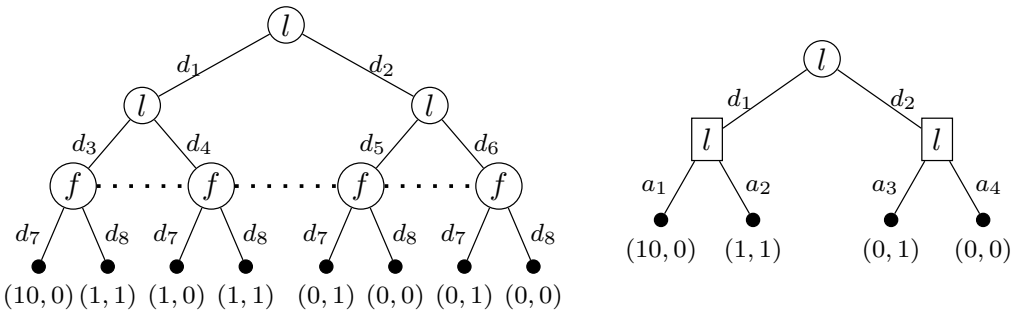


Fig. 3. (Left) An example of EFG where the heuristic algorithms does not converge. The figure follows a standard denotation of an extensive-form game. (Right) An initial RG of the gadget algorithm. By rectangles are denoted the roots of the gadgets. The full gadget algorithm would expand both gadgets after the first iteration. The heuristic gadget algorithm expands only the left gadget.

Example 4.4. Consider the game depicted in Figure 3, where SEFCE (which is the same as the SSE) is as follows: the leader commits to playing actions d_1 and d_2 with the same probability 0.5 and actions d_3 and d_5 in his lower information sets. The follower is, therefore, indifferent between d_7 and d_8 and by the definition of SEFCE plays d_7. The follower's strategy guarantees the leader the expected utility of 5.0 in the equilibrium. The right part of Figure 3 shows the initial RG of the heuristic gadget algorithm with gadget actions a_1, a_2, a_3 and a_4. Because the optimal strategy of the leader is to move into the left gadget, it is expanded into the full subtree. In the second iteration, the leader has no intention of playing d_2 neither, because his utility is strictly greater in the left subtree. The algorithm therefore terminates. Because the RG is not closed on the follower's information sets, the leader is unaware of the possibility to make the follower indifferent and the optimum is not reached.

Approximation of Upper Convex Hulls. This approximation addresses the problem of creating gadgets for subtrees with large upper convex hull of leafs. Assume that there are m sequences of the follower and n sequences of the leader in the subtree. The number of leafs is hence upper bounded by $m \times n$. If the number of leafs added to the gadget is close to this bound and $mn > m + n$, the number of the sequences which are added into the RG is even larger then number of sequences in the whole subtree and the corresponding LP can be prohibitively large. We solve this issue by excluding from the upper convex hull those leafs, which do not provide a significant increase in the leader's utility when compared with

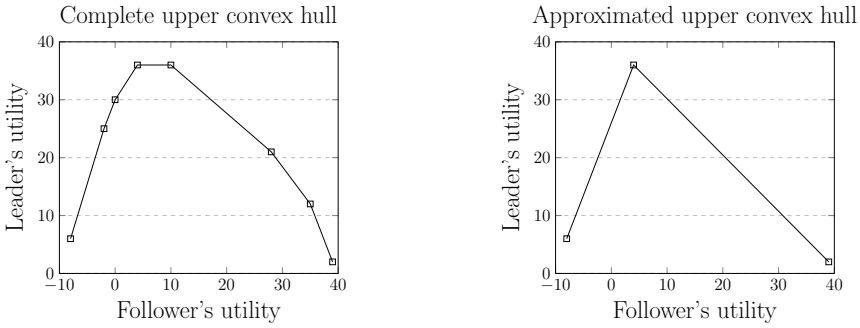


Fig. 4. An example of upper convex hull approximation. (Left) The complete upper convex hull, which is used in the full gadget algorithm. (Right) The approximated upper convex hull with parameter $\delta = 0.4$.

the affine combination of the neighboring leafs. Formally, we delete a leaf $z = (u_l, u_f)$ in between the leafs $z^1 = (u_l^1, u_f^1)$ and $z^2 = (u_l^2, u_f^2)$ in case that

$$\left| u_l^1 + \frac{(u_l^1 - u_l^2)(u_f - u_f^1)}{u_f^1 - u_f^2} - u_l \right| < \delta u_l, \tag{23}$$

where the greater the $\delta > 0$ gets, the more strict the approximation is.

Penalization of Leader's Utilities. Finally, we penalize the leader's utility in order to compensate the optimism when the upper convex hull is created from the leafs directly. Also, during the iterations, there might be several SEFCE with the same optimum, but with different support. In case the solution returned from the LP solver uses the gadgets even though the same optimum can be reached with sequences leading to the ordinary leafs in the RG, the algorithm makes unnecessary iterations. Consequently, both the size of the final RG and the computational time are unnecessarily increased. To speed up the convergence of the algorithm we subtract $\epsilon|maxUtility_l|$, where $\epsilon > 0$, from all leader's utilities in leafs in the gadgets in order to enforce the LP solver into not using the gadgets if not necessary.

5 EXPERIMENTS

We performed experiments to evaluate the performance of our heuristic algorithm using strategy generation to compute SSE in EFGs. For comparison, we use the state-of-the-art SEFCE-based algorithm [19], which we refer to as FULL since it uses the full strategy space. All algorithms were implemented in Java 1.8 and all LP computations were completed by a single-threaded IBM CPLEX 12.8 solver using the barrier method. Based on an initial exploration of the parameter space we set the parameter δ that controls the approximation of the upper convex hulls to value $\delta = 0.3$ since it gave the best results. However, we note that the difference in performance between different values of δ was relatively small.

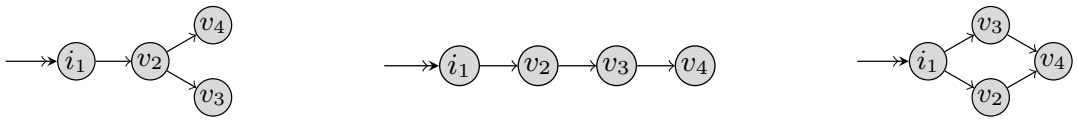
Flip It Games. The domain we used for the experiments is the "Flip It" game [17]. This game is motivated by a cybersecurity scenario where an attacker can perform a stealthy attack to gain control of a resource (e.g., install malware on a host or steal a password) that may not be immediately detected by the defender. However, the defender can take actions to restore control to the defender (e.g., performing a virus scan or resetting a password). Flip It can be viewed as a general model of players competing over resources, and many variations have been proposed in the literature. We use it for evaluation here because it has a symmetric structure that makes it particularly difficult to compute SSE; specifically, the vast majority of follower strategies are best responses to some strategy for the leader. This

makes it a particularly challenging test for our approach since strategy generation methods would perform even better in games with many irrelevant strategies.

A two-player Flip It game is defined as a tuple $F = (V, E, t, \rho, \gamma)$. The game is played by a defender and an attacker on a directed graph (V, E) for a finite number of simultaneous rounds t. The graph represents a typical structure for a computer network where not all nodes are publicly accessible and an attacker may need to move deeper into the network. There is a positive reward $\rho : V \to \mathbb{R}^+$ and a positive cost $\gamma : V \to \mathbb{R}^+$associated with each node $v \in V$. At the beginning of the game, the defender controls all of the nodes. In each round, each player selects one node to flip, i.e to attempt to gain control of. The flipping action is successful when two conditions are met. First, the current owner of the node does not also flip it; and second, the player has control over at least one predecessor of the node. We assume that the source nodes in (V, E) can be flipped in any round (i.e., they are public nodes). For every flipping action, the players pay the cost assigned to the node. At the end of every round the players collect the total rewards from all nodes they control:

$$u_j^i(V_j^i) \leftarrow -\gamma(v_j^i) + \sum_{v \in V_j^i} \rho(v) \quad \forall i \in \mathcal{N}, \tag{24}$$

where V_j^i are the nodes the player i owns after round j. After t rounds the game ends and the final utilities are the sum of the rewards collected in the individual rounds. We consider two versions of the game with different amounts of information provided to the players. In the All-Points (AP) version the players learn whether their action succeeded and how many points they have in total after each round. In the No-Info (NI) version the only information the players observe is the sequence of actions they play. Moreover, we assume that the graph can also contain a single disconnected *p*ass node with zero reward and zero cost, simulating a pass action. The directed graphs used in the experiments are depicted in Figure 5. All of them include the pass node, which is not depicted in the figure.



(a) Graph 4a (b) Graph 4b (c) Graph 4c

Fig. 5. Graphs used in the experiments with the Flip It game.

We assume the defender acts as a leader in this game, while the attacker assumes the role of the follower. For graphs 4a and 4c in Figure 5 we solved 40 instances of the Flip It games: 20 All-Points games and 20 No-Info games. For graphs 4b we computed 60 instances: 30 All-Points games and 30 No-Info games. Each node in the graph was randomly assigned to one of the following types: (1) high reward, high cost, (2) high reward, low cost, (3) low reward, high cost, and (4) low reward, low cost. The reward and cost were generated uniformly randomly from the intervals depicted in Table 6 to generate representative instances of Flip It games.

The number of rounds was fixed at $t = 5$, so the number of nodes in the EFG representation of the Flip It game is approximately 11×10^6. Since the SI-LP variant of FULL algorithm was reported to be fastest in [19], we use this variant as a baseline algorithm for computing SSE in Flip It games.

Interval\Type	(1)	(2)	(3)	(4)
Reward interval	6..10	6..10	3..6	1..4
Cost interval	6..10	3..6	5..9	1..4

Fig. 6. The intervals from which the rewards and costs were generated for individual types of nodes.

5.1 Results

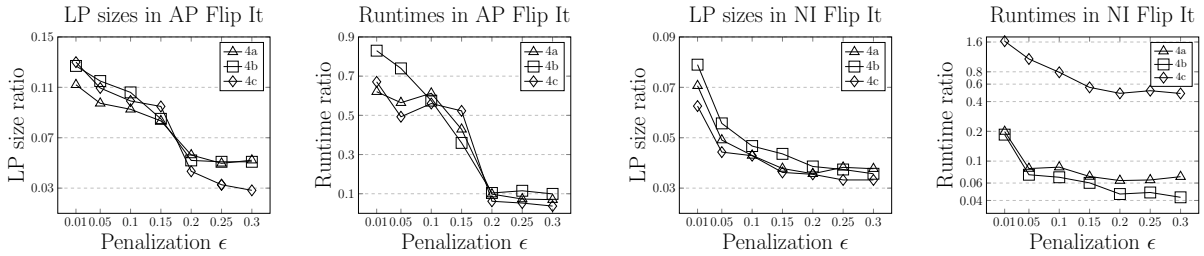


Fig. 7. The median LP sizes and median runtimes in both variants of Flip It for different values of ϵ.

The sizes of the LPs generated for the final restricted games (RG) in the All-Points version of the Flip It games are presented in the leftmost graph of Figure 7. The x-axis shows the penalization parameter ϵ, while the y-axis depicts the ratio of the number of coefficients in the LP describing SSE in the final RG to the number of coefficients in the complete LP used in FULL. Every point in the graph corresponds to the median ratio over the sampled instances. The results show that even with small ϵ, the final LPs are more than eight times smaller than in FULL. As the ϵ is set higher, the final RGs become even smaller. For the graph $4c$ and $\epsilon = 0.3$, the final LP reaches a size of less than 3% of the full LP. The runtime results for the All-Points version are depicted in the second graph of Figure 7. The x-axis varies the penalization parameter and the y-axis shows the median speedup, which we calculate as the runtime of the heuristic gadget algorithm divided by the runtime of the FULL algorithm.

In the same Figure 7, we present also the results for the final LP sizes and runtimes on the No-Info version of the Flip It game. The graphs follow the same format as the graphs for the All-Points version. Interestingly, when given no information about the progression of the game, the more complex structure of the graph $4c$ slows down the convergence of the heuristic algorithm. The reason is that with larger information sets than in the All-Points version, the upper convex hulls are also larger and contain similar leafs. The algorithm expands only a few gadgets in every iteration and the number of LP invocations is greater. Note that even though the speedup is not that impressive, the sizes of final LPs of all graphs are still less than 4% for the largest value of the parameter ϵ.

Finally, in Table 8 we present the median deviations from the optimum solution achieved by the gadget algorithm. The deviations are measured as the absolute difference in the game values computed by FULL and the heuristic gadget algorithm divided by the SSE game value. We see in the results that the heuristic gadget algorithm achieves nearly optimal results even with higher values of the penalization parameter ϵ. For example, for the graph $4c$ the deviation is 6.38%, with the size of the final LP only 2.8% in the All-Points version, and 1.75% with the size of the final LP 3.3% in the No-Info version.

Instance \ ϵ	0.01	0.05	0.1	0.15	0.2	0.25	0.3
4a All-Points	0%	0%	0%	0.9%	2.42%	3.01%	3.23%
4a No-Info	0%	0.35%	0.72%	1.29%	1.77%	2.5%	2.55%
4b All-Points	0%	0%	0%	0.56%	0.8%	2.39%	2.48%
4b No-Info	0%	0.16%	0.67%	1.27%	2.15%	2.42%	2.86%
4c All-Points	0%	0%	0.033%	0.79%	3.47%	4.8%	6.38%
4c No-Info	0%	0.24%	0.89%	1.75%	1.75%	1.75%	1.75%

Fig. 8. The deviations achieved with the heuristic gadget algorithm under different setting of the penalization parameter ϵ.

6 CONCLUSION

In this paper, we introduce the first algorithm that successfully exploits the ideas of incremental strategy generation for computing Stackelberg equilibria in extensive-form games. Our algorithm is based on a novel way of representing abstracted parts of the game tree, as well methods for expanding the representation systematically to compute Stackelberg equilibrium. An advantage of our representation is that it leads to three different heuristic approaches that trade off theoretical guarantees for greatly improved practical performance. The heuristic methods can compute exact (or near-optimal) outcomes in practice while constructing only 3% of the original linear program compared to the current state-of-the-art algorithm, often achieving significant speed-up in computation times even in a very challenging class of games.

The current experimental results show that even in a game that is structurally difficult for strategy generation and the Stackelberg solution concept, it is not necessary to consider the whole game tree and all the possibilities. Our algorithm suggests additional directions towards more scalable and approximate algorithms for computing (approximate) Stackelberg equilibrium in large extensive-form games, and it would be interesting to explore other types of games that may have more favorable structure.

REFERENCES

[1] Branislav Bošanský, Simina Brânzei, Kristoffer Arnsfelt Hansen, Troels Bjerre Lund, and Peter Bro Miltersen. 2017. Computation of Stackelberg Equilibria of Finite Sequential Games. *ACM Transactions on Economics and Computation* 5, 4, Article 23 (Dec. 2017), 24 pages.

[2] Branislav Bošanský, Albert Xin Jiang, Milind Tambe, and Christopher Kiekintveld. 2015. Combining Compact Representation and Incremental Generation in Large Games with Sequential Strategies. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[3] Branislav Bošanský, Christopher Kiekintveld, Viliam Lisý, and Michal Pěchouček. 2014. An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information. *Journal of Artificial Intelligence Research* 51 (2014), 829–866.

[4] Branislav Bošanský and Jiří Čermák. 2015. Sequence-Form Algorithm for Computing Stackelberg Equilibria in Extensive-Form Games. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. 805–811.

[5] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. 2015. Heads-up limit holdem poker is solved. *Science* 347, 6218 (2015), 145–149.

[6] Karel Durkota, Viliam Lisý, Christopher Kiekintveld, Branislav Bošanský, and Michal Pěchouček. 2016. Case Studies of Network Defense with Attack Graph Games. *IEEE Intelligent Systems* 31, 5 (2016), 24–30.

[7] Karel Durkota, Viliam Lisý, Christopher Kiekintveld, Karel Horák, Branislav Bošanský, and Tomáš Pevný. 2017. Optimal Strategies for Detecting Data Exfiltration by Internal and External Attackers. In *GameSec 2017, LNCS 10575*. 171–192.

[8] Manish Jain, Erim Kardes, Christopher Kiekintveld, Fernando Ordónez, and Milind Tambe. 2010. Security Games with Arbitrary Schedules: A Branch and Price Approach.. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*.

[9] Manish Jain, Milind Tambe, and Christopher Kiekintveld. 2011. Quality-bounded solutions for finite Bayesian Stackelberg games: Scaling up. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*. 997–1004.

[10] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. 1996. Efficient Computation of Equilibria for Extensive two-person Games. *Games and Economic Behavior* (1996), 247–259.

[11] Christian Kroer, Gabriele Farina, and Tuomas Sandholm. 2018. Robust Stackelberg Equilibria in Extensive-Form Games and Extension to Limited Lookahead. In *Proceedings of Thirty-Second AAAI Conference on Artificial Intelligence*.

[12] George Leitmann. 1978. On generalized Stackelberg strategies. *Journal of Optimization Theory and Applications* 26, 4 (1978), 637–643.

[13] Joshua Letchford and Vincent Conitzer. 2010. Computing Optimal Strategies to Commit to in Extensive-Form Games. In *Proceedings of the 11th ACM conference on Electronic commerce*. 83–92.

[14] H. Brendan McMahan, Geoffrey J. Gordon, and Avrim Blum. 2003. Planning in the Presence of Cost Functions Controlled by an Adversary. In *Proceedings of the International Conference on Machine Learning*. 536–543.

[15] Matěj Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. 2017. DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker. *Science* (2017).

[16] Tuomas Sandholm. 2015. Steering evolution strategically: computational game theory and opponent exploitation for treatment planning, drug design, and synthetic biology. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI Press, 4057–4061.

[17] Marten Van Dijk, Ari Juels, Alina Oprea, and Ronald L Rivest. 2013. FlipIt: The game of stealthy takeover. *Journal of Cryptology* 26, 4 (2013), 655–713.

[18] Ondřej Vaněk. 2013. *Computational Methods for Transportation Security*. Ph.D. Dissertation. Czech Technical University in Prague.

[19] Jiří Čermák, Branislav Bošanský, Karel Durkota, Viliam Lisý, and Christopher Kiekintveld. 2016. Using Correlated Strategies for Computing Stackelberg Equilibria in Extensive-Form Games. In *Proceedings of Thirtieth AAAI Conference on Artificial Intelligence*. 439–445.

[20] Bernhard von Stengel. 1996. Efficient Computation of Behavior Strategies. *Games and Economic Behavior* 14 (1996), 220–246.

[21] Bernhard von Stengel and Françoise Forges. 2008. Extensive-Form Correlated Equilibrium: Definition and Computational Complexity. *Mathematics of Operations Research* 33, 4 (2008), 1002–1022.

[22] Bernhard von Stengel and Shmuel Zamir. 2004. *Leadership with Commitment to Mixed Strategies*. Technical Report. CDAM Research Report LSE-CDAM-2004-01.

[23] Haifeng Xu, Rupert Freeman, Vincent Conitzer, Shaddin Dughmi, and Milind Tambe. 2016. Signaling in bayesian stackelberg games. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 150–158.