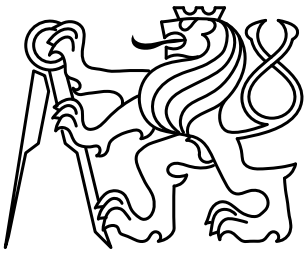
Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science



Learning to Play Large Sequential Games

Habilitation Thesis

Viliam Lisý

Prague, April 2019

Copyright

The works presented in this habilitation thesis are protected by the copyright of AAAS, Elsevier, IFAAMAS, AAAI, Neural Information Processing Systems Foundation, IOS Press, and IEEE. They are presented and reprinted in accordance with the copyright agreements with the respective publishers. Further copying or reprinting can be done exclusively with the permission of the respective publishers.

Abstract

This habilitation thesis summarizes relevant advancements in approximating optimal strategies in large extensive form games (EFGs). It argues that limited scalability of pre-existing methods is one of the main reasons why modeling relevant real world problems as extensive form games has been infeasible. This habilitation thesis summarizes relevant advancements in approximating optimal strategies in large extensive-form games (EFGs). It argues that limited scalability of pre-existing methods is one of the main reasons why modeling relevant real-world problems as extensive form games has often been infeasible.

The thesis first focuses of more classical abstraction-based methods to increase scalability. It seeks more efficient solution techniques for compactly represented games and afterwards focusses on automated construction of compact game abstractions guided by the process of their solving.

Even with recent advancements, the abstraction-based solution techniques have severe limitations, as we demonstrate in one of the attached papers. Therefore, we later focus on online game solving methods, which do not create a strategy for all possible decision points in the game in advance, but compute the strategy only for the situations that actually occur while playing the game. This type of solution methods was always preferred for perfect information games, but only our recent results allow for generalizing these methods to imperfect information games in a principled and theoretically sound way. We present variants of depth-limited look-ahead search as well as Monte Carlo tree search techniques for imperfect information games with formal performance guarantees.

Contents

Chapter 1

Introduction

Game theory is a formal framework for analyzing behavior of self-interested agents interacting in a shared environment. It describes the possible outcomes of these interactions and the computational branch of the field has recently provided algorithms for approximating the optimal strategies of the agents for increasingly complex games. These advancements led to expert level game playing algorithms for a range of recreational games (such as Chess [Hsu06], Go[SHM$^+$16], or Poker[BBJT15]), but also to creating strategies for real-world problems, such as auctions [CSS06], various physical or network security problems, and law enforcement scenarios [Tam11]. Strategies computed by game-theoretic algorithms are in daily use by airport security patrols [JTP$^+$10] and coast guards [SAY$^+$12].

Despite this success, there are fundamental limitations in applicability of game theoretic methods to the real world. Two of the most apparent are **limited scalability** of the approaches to larger settings and the complexity of **modelling** the real-world situations. These two problems are often closely interconnected. Modelling a real world problem in more details leads to larger models that are not solvable by existing techniques. Limiting the size of the model makes it less similar to the actual real-world problem and therefore the strategies computed in this model are less likely to perform well in practice.

The scalability is a major problem, even though an optimal strategy in many games can be found in a time polynomial in the size of the game [KMVS96]. Games are often defined by compact rules, but the explicit representation of the game is usually exponential in several parameters. For example, No-Limit Texas Hold'em played at the Annual Computer Poker Competition has more than 10^{160} distinct decision points [Joh13]. A robotic visibility-based pursuit-evasion game with two pursuers and one evader on a four-connected graph and 100 moves time limit has more than 10^{180} decision points. These numbers are many orders of magnitude larger than the believed number of atoms in the observable universe. Therefore, it is not likely that we will ever be able to create an explicit representation of the optimal strategy for these games.

This is not a major problem in **perfect information** games, such as chess or go. A common approach applicable in these games, which is explained in basic AI textbooks, is to reason about each situation only when it arises. In perfect information games, we can create a short look-ahead tree starting from the current position and use a heuristic evaluation function after few actions to keep

the size of the tree manageable (Chapter 5, [RN10]). Additionally, we can use Monte Carlo simulations from the current state of the game to evaluate available actions and choose the most promising one [KS06]. These are the methods that created the expert level programs that has beaten best human professionals in perfect information games, such as, backgammon, checkers, chess or go.

However, real-world problems usually do not have perfect information. Network administrators do not know when the hackers strike and what tools they use. An airport security officer does not know which passengers are carrying a contraband. Security scenarios are generally games of **imperfect information**, more similar to poker. There has been a large body of research on poker and imperfect-information extensive-form games (EFGs) in general in last two decades. Since players do not know the current state of the game, it was long believed that it is not possible to compute a strategy only for the current decision in an imperfect information game and provide any performance guarantees. Until the end of the year 2016, the most successful approach for creating strategies for large extensive-form games was **abstraction**. For example, a game with 10^{160} terminal states would be abstracted to up to 10^{13} states by assuming that distinct situations in the game are sufficiently similar to be considered the same. The smaller version of the game would then be solved **offline** and the solution would be heuristically mapped to the original game. This approach may produce meaningful strategies, and requires only minimal computation power at the time of play. On the other hand, our previous results show that even the most advanced poker bots created by this methodology loose four times more than always blindly folding against an exploitative adversary [LB17].

A main disadvantage of offline solution techniques is the requirement to store a strategy for any situation that may occur in the (possible abstracted) game. However, in large games, even if we play them for several human life-times, we will encounter only a tiny proportion of the possible states in the game. It is desirable to be able to compute the strategy only for the situations that we encounter. There has been previous work on adapting the **online** solution methods from perfect information games, such as MCTS and expectimax search to imperfect information games [CPW12, RNK$^+$10]. Even though this leads to strong play in some domains [LSBF10, NW12], there are no performance guarantees on these methods and we have shown that their performance can even get worse with increasing thinking time per move [Lis14a]. Therefore, we have created the first theoretically sound online solution method for imperfect information games - online outcome sampling [LLB15]. The algorithm guarantees that as we increase the thinking time per move, the empirical play of the algorithm is closer to the Nash equilibrium.

The guarantee of convergence to Nash equilibrium assured by OOS is relevant in smaller games, but in larger games, the thinking time required to guarantee near equilibrium play is impractically long. Without the ability to generalize among similar states, it is impossible to guarantee good performance in an arbitrary game, if the thinking time allows us to see only a tiny fraction of all possible situations in the game. In [MSB$^+$17], we have shown that the knowledge learned about a small fraction of situations in a game can be generalized to the whole game at least in games similar to poker. We have proposed a theoretically sound **heuristic look-ahead search** algorithm for poker. The algorithm, called continual resolving, is able to compute strategies online only for the situations that occur in a game. We have shown that the extra infor-

mation necessary to start reasoning about the strategy from a current situation can be compactly summarized in a vector of probabilities and a vector of counterfactual values. These vectors can be soundly updated after all events that can occur during the gameplay. Furthermore, we have derived a special form of heuristic evaluation function that can be used for imperfect-information games and we have shown that this function is learnable by deep neural network based on sample solutions of relatively small number of simpler situations. This algorithm was the first that was able to outperform professional poker players in no-limit Texas hold'em.

This success inspired our current research on generalization of continual resolving to be applicable in domains beyond poker. In our recent paper [SKL19], we show that after minor modification, continual resolving can be soundly applied in any two-player zero-sum extensive form game. The representation and learnability of a suitable heuristic evaluation function in other domains is the focus of our current work.

In Chapter 2, we present a brief high level overview of the main game theoretic concepts necessary to understand this text. In Section 3, we focus on related work and our contribution to creating and solving abstractions of large games. Afterwards, we will explain the context and our contributions in online game playing algorithms in domains beyond finite perfect information settings. Next, we conclude the thesis and present future research directions. Finally, we append selected works as they have been published in peer-reviewed journals and conferences in Appendix A.

Chapter 2

Background

This chapter briefly introduces the intuition behind the most important game theoretic concepts used in this text. The papers in the appendix include the thorough formal definitions.

2.1 Extensive Form game

Extensive form games (EFG) represents the space of all possible developments of a game. In perfect information games, they are also known as game trees. Nodes in the tree represent possible states in the game and the root of an EFG represents the initial state of the game. In each node of the tree, a single player chooses and action to play next. These actions are represented as edges in EFG. Besides the regular players, the game also includes a special chance (or nature) player, whose actions represent stochastic events happening in the game. The leaf nodes in an EFG represent terminal states of the game and assign a utility value to each player. If the utility of one player is always minus the utility of the other player, hence always when one player is winning the other one is losing, the game is called zero-sum.

The imperfect information in an EFG is represented by information sets, which group the nodes which the acting player cannot tell apart. As a result, the same player has to act in all nodes of an information set and the same actions have to be available in all of them. The strategy the player plays also has to be consistent over all these nodes.

Figure 2.1 presents the EFG representation of a simple poker game, called Kuhn poker. There are two players: Ann (red) and Beth (blue). Initially, each of the players ante one chip (not present in the figure). The chance player (green) plays first and deals each player one of cards {1,2,3}. The last card stays face down. Next, Ann can either check or raise by adding another chip. If Ann checks, Beth has the options to raise or check. After the raise, Ann can fold, or call and add the second chip as well. Afterwards, the player with a higher card wins the two chips and the other player loses them, as it is indicated by the numbers at the leaves of the game. The information sets are indicated by the dashed lines. After the initial chance move, Ann knows the card she got (the first number at the chance action), but does not know which of the remaining two cards Beth got. Similarly, when Beth moves, she knowns her

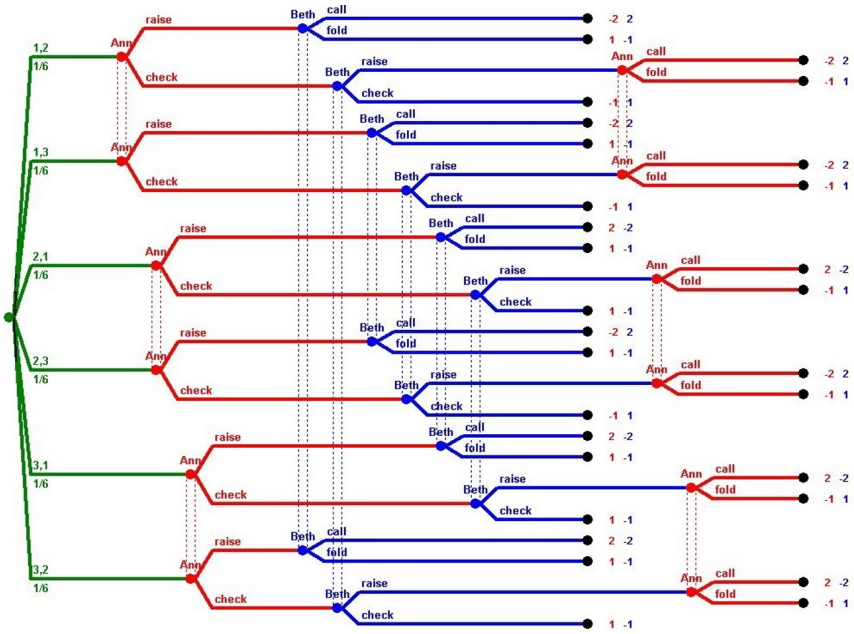


Figure 2.1: Extensive form game representation of a simple poker game, called Kuhn poker.

own card and Ann's previous move, but both options for Ann's card are in the same information set.

Optimal strategies in EFGs often require randomization. It is not enough to choose the best action in each information set, we need to choose the best probability distribution over the actions. The most common notion of optimality used in games is the notion of *Nash equilibrium*. It is a tuple of strategies, one for each player, such that no player can improve her expected utility by deviating to a different strategy.

2.2 Standard Solution Techniques

Nash equilibrium in two player zero-sum games can be found in polynomial time by linear programming [KMVS96]. However, substantially better performance in large games can be achieved by iterative algorithms design particularly for games, such as excessive gap technique (EGT) [GHPS07] and counterfactual regret minimization (CFR) [ZJBP08]. These algorithms are, in essence, similar to gradient descent in individual player's strategies, but they need to be designed more carefully to guarantee the convergence to a Nash equilibrium. The implementation simplicity led to the wide adaptation of CFR and its modifications that allowed solving increasingly large games [LWZB09, JBBB12, TBJB15]. It resulted in finding an essentially optimal solution to the smallest variant of poker played by people [BBJT15]. This relatively simple variant of poker has approximately 10^{13} decision points. The pre-computed strategy required 900 CPU-years of computation and is 11 TB large, even after extremely efficient domain-specific compression.

Chapter 3

Solving Large Games Using Abstractions

Solving imperfect information games is substantially more complicated than solving perfect information games. The players need to reason not only about the possible future actions of the opponent, but also about the possible current states the game, opponent's belief about the states of the game, opponent's beliefs about player's own beliefs, etc. This has two important consequences. First, the optimal strategies in imperfect information games are typically randomized. Instead of a single action in each situation, the strategies prescribe a probability distribution over the available actions. These strategies need to be found by continuous optimization methods and not a simple search [KMVS96]. Second, the imperfect information makes the strategies in distant parts of the game interdependent. Optimal strategy following one sequence of actions depends on the strategy planned for completely different sequences of actions [LLB15]. Therefore, it is impossible to reason about different decisions completely independently of each other.

For a long time, the only possibility for approximating the optimal strategy for an imperfect information game was to compute the optimal strategy for each possible decision point in the game **offline**, prior to playing the game, using continuous optimization techniques. For sequential games, this initially meant solving an exponentially large normal-form representation of the game. The first algorithm that scaled beyond toy problems is the sequence form linear program [KMVS96], which is polynomial in the size of the game tree. On the other hand, the size of the game tree is still exponential in the number of players, and the number of consequent decisions in the game. Therefore, this algorithm is still not sufficient to compute strategies for realistic problems, such as poker games played by people.

3.1 Abstraction Methodology

The most prominent method for playing larger games was creating an abstraction, solving it, and using a heuristic mapping to find a situation from the abstract game that is closest to the situation reached in the gameplay [GS13]. For almost a decade, the community of Annual Computer Poker Competition

(ACPC)[1] adopted this methodology and focused on creating algorithms capable of solving larger and larger abstract versions of poker games.

The first large step was moving from general purpose LP solvers to iterative algorithms design particularly for games, such as excessive gap technique (EGT) [GHPS07] and counterfactual regret minimization (CFR) [ZJBP08]. The implementation simplicity led to the wide adaptation of CFR and its modifications that allowed solving increasingly large games [LWZB09, JBBB12, TBJB15]. It resulted in finding an essentially optimal solution to the smallest variant of poker played by people [BBJT15]. This relatively simple variant of poker has approximately 10^{13} decision points. The pre-computed strategy required 900 CPU-years of computation and is 11 TB large, even after extremely efficient domain-specific compression. Hence, computing the complete strategy offline cannot scale to substantially larger games.

The size of games commonly encountered by people can be many orders of magnitude larger than the 10^{13} decision points. For example, two player no-limit Texas hold'em variant used at ACPC has approximately 10^{160} decision points [Joh13]. The abstraction methodology is able to compress the game to the solvable 10^{13}, but it requires merging 10^{147} distinct decision points from the original game to a single decision point in the abstract game. As a result, the exact same strategy would be played in all of these decision points, which is not likely to be close to the optimal play.

3.2 Compact Game Representation

One way abstractions of a size that is only a tiny fraction of the full size of the game can be representative is if the original game is fundamentally *sparse*. For example, assume a player can make one of ten irrelevant actions at the beginning of the game, such as choosing the color of their pieces. This action has no impact on the strategy in the game, but makes the explicit representation of the game ten times larger. An abstraction merging the same states with different colors together is sufficient to find the optimal strategy of the complete game.

This trivial example of sparsity can be avoided by not modeling the irrelevant decision in the game. However, more complex forms of sparsity, such as irrelevance of the order of execution of some of player's actions, or sparsity of interaction between the strategies of different players are more difficult to capture. Therefore, it is often captured in the form of compact representation of the models of the game. Compact representations model a problem using substantially less parameters and a set of structural assumptions. An extreme case of compact representation of no-limit Texas hold'em with 10^{160} decision points is a short computer program encoding the game rules in combination with an interpreter for the used programming language.

Using a compact representation of a problem does not necessarily make it computationally simpler. The asymptotically optimal algorithm to solve the problem may still use generation of its explicit representation as the first step. If solving the original problem requires linear time and its compact representation is logarithmic in the size of the original problem, solving the compact representation of the problem may be exponential.

[1] http://www.computerpokercompetition.org/

3.2.1 Normal Form Games with Sequential Strategies

Normal Form Games with Sequential Strategies (NFGSS) are a particularly useful compact representation of extensive form games introduced in [BJTK15]. They allow capturing a limited dependence of players' rewards on the history of their actions. Furthermore, they assume that players do not learn any information about their opponent's actions before the game is over. These restrictions allow describing extensive form games with substantially less parameters, compared to their explicit representation, but still keep the computational complexity of finding the Nash equilibrium of the game polynomial in the size of the compact representation. NFGSS has been used to model border protection scenarios as well as ticket inspection problems.

The Nash equilibrium of a NFGSS can be found by linear programming [BJTK15]. However, linear programming has relatively high memory and computation time requirements. In computational poker research, CFR has been shown to be much more efficient.

In [LDB16], we show that CFR-based algorithms are preferable to LPs also for NFGSS. We propose an adaptation of the CFR+ algorithm for NFGSS. We prove that it is guaranteed to converge in this class of games and compare its performance to the standard methods based on linear programming and incremental game generation. We validate our approach on two security-inspired domains. We show that with a negligible loss in precision, CFR+ can compute a Nash equilibrium with five times less computation than its competitors. This paper represents the first successful demonstration that the CFR-based algorithms, developed for computational poker, can improve the state of the art in security games modeling real world security problems. The paper describing these results is attached in Appendix A.

3.3 Simultaneous Abstraction and Solving

If a game model we want to solve is too large, we can sometimes find an efficient compact representation of the game as explained in the previous section. However, efficiently solvable compact representations often have very restrictive assumptions. Previous poker research, therefore, often focused on creating abstractions by merging large sets of distinct decision points to the same decision points in the abstract game. First abstractions were created by hand. Later, methods based on automated clustering of similar decision points based on domain-dependent metrics have been introduced [GSS07a]. It is not clear how to generalize these abstraction techniques to domains beyond poker.

The abstraction techniques developed for poker construct the abstract game *before* solving the game and then use standard solution techniques to solve the abstract game. As a result, the abstraction is not optimized to a particular solution of the game in any way. This is clearly suboptimal, since the granularity of the abstraction after an action never played in the optimal solution is typically the same as after an action that is always played.

In [CLB18], we reduce the memory required for computing and representing a (near) optimal strategy for a game using automatically-constructed imperfect-recall abstractions created by domain-independent algorithms with respect to the solution being computed.

Domain independent Most existing methods for automatically constructing abstractions in extensive-form games were designed primarily for poker. They explicitly work with concepts like cards and rounds of the game [SL00, BBD$^+$03], or at least assume that the actions are publicly observable [BS15] and ordered [GSS07b]. This is not true in many other domains (e.g., in security). The algorithms we proposed are completely domain-independent and applicable to any extensive-form game. They only require a definition of the game and a desired distance of the solution from the equilibrium in the original game.

Imperfect recall Computationally efficient algorithms for computing (near) optimal strategies in extensive-form games [vS96, ZJBP08, HGPnS10] require players to remember all the information gained during the game: a property denoted as *perfect recall*. Therefore, the automated abstraction methods designed to be used with these algorithms [GSS07b, BS15] must construct perfect-recall abstractions to provide performance guarantees. Requiring perfect recall has, however, a significant disadvantage – the number of decision points and hence both the memory required during the computation and the memory required to store the resulting strategy grows exponentially with the number of moves. To achieve additional memory savings, the assumption of perfect recall may need to be violated in the abstract game resulting in *imperfect recall*. Using imperfect recall abstractions can bring exponential savings in memory, and these abstractions are particularly useful in games in which the exact knowledge about the past is not required for playing optimally. While it can be easy to identify specific examples of imperfect recall abstractions for some games, it is very difficult to systematically and algorithmically identify which information is required for solving the original game and which can be removed only form the structure of the game. For example, in imperfect information card games, it is usually important to estimate the cards the opponent currently holds in her hand. While past events generally reveal some information, it is not clear which exact event is relevant or not.

The only pre-existing method for automatically constructing imperfect-recall abstractions with qualitative bounds is presented in [KS16]. This method considers only a very restricted class of imperfect-recall abstractions. In short, the information sets can be merged only if they satisfy strict properties on the history of actions and there is a mapping between the applicable actions in these information sets, such that future developments of the game and possible rewards are similar (see [KS16] for all the details). n our work, we take a different approach and instead of constraining which information sets can be merged, we design algorithms that similarly to *inflation* operation [Dal53] refine information sets where necessary. Our approach, however, does not require any specific structure of the abstract game or the information sets being refined. We introduce two domain-independent algorithms, which can start with an arbitrary imperfect recall abstraction of the solved two-player zero-sum perfect recall EFG. The algorithms simultaneously solve the abstracted game, detect the missing information causing problems and refine the abstraction to include this information. This process is repeated until provable convergence to the desired approximation of the Nash equilibrium of the **original game**.

Our algorithms can be initialized by an arbitrary abstraction since the choice

of the initial abstraction does not affect the convergence guarantees of the algorithms. Hence, for example in poker, we can use the existing state-of-the-art abstractions used by the top poker bots. Even though these abstractions have no guarantees that they allow solving the original poker to optimality, our algorithms will further refine these abstractions where necessary and provide the desired approximation of the Nash equilibrium in the original game. If there is no suitable abstraction available for the solved game, the algorithms can start with a simple coarse imperfect recall abstraction (we provide a domain-independent algorithm for constructing such abstraction) and again update the abstraction until it allows approximation of the Nash equilibrium of the original game to the desired precision. The initial paper describing a subset of these methods [CBL17] is attached in Appendix A. The full paper describing the details of this approach is under review and the preprint is available on arXiv [CLB18].

3.4 Abstractions Summary and Future Work

Abstraction are a common solution to solving games that are too large to be solved in their original form. If the game model includes enough additional structure, compact representations can be used to solve it optimally. However, many problems do not allow for efficient compact representations or at least it is not obvious how to find them. Therefore, a lot of poker research has focused on creating lossy abstractions that aspire to capture the most important strategic aspects of the original game in a substantially smaller abstract game. Regardless of some success in smaller games [RFR08, CLB18], the top no-limit poker programs created using this methodology are four times worse than not looking and cards and folding all hands. The paper [LB17] presenting the method for evaluation of the quality of large poker abstraction along with the results on the state of the art poker bots is attached in Appendix A.

Abstraction methodology with abstractions constructed statically before solving the problem does not seem to be promising for solving very large games. However, the abstraction methodology can still be useful. The recent research [BLGS18, LHG+18] shows initial promising results in using neural networks to create dynamic abstractions, which in addition to reducing the number of parameters allow non-trivial generalization of computation results between similar decision points in the game. Another use for abstractions is introduced in [BSA18], where even very week abstractions have been shown to provide useful approximations of the values achievable by the players in the game. Therefore, I believe abstraction can still be a useful component of strong game playing algorithms.

Chapter 4

Playing Large Games with Online Computation

The interdependence of optimal strategies in different decision points of imperfect information games caused that for a long time, algorithms for computing the (near) optimal strategies in extensive form games computed the strategy for all decision points in a game at once. The requirement to pre-compute and store the strategy for all decisions is extremely limiting. In perfect information games, we avoid this problem by computing a strategy for a particular game situation **online**, only when it occurs in the game. This property of game playing algorithms is so compelling, that many researchers gave up the optimality guarantees and developed online algorithms that empirically perform well in some classes of imperfect information games [LSBF10, CPW12, LBP12]. These algorithms are often effective, but may make easily exploitable mistakes in some situation. The lack of theoretic guarantees makes them hard to use for example in real world security settings, where a motivated attacker might exploit their flaws.

There are only few online algorithms for imperfect information games with performance guarantees. To the best of my knowledge, the first was online outcome sampling (OOS) [LLB15], developed as a part of my PhD studies. It combines the incremental game generation of Monte Carlo tree search with a sound variant of CFR. This algorithm is guaranteed to produce an optimal strategy after sufficient computation time. However, the bounds are not practical for realistically long computations. Furthermore, it is often outperformed by the non-sound alternatives in experiments.

The second, and much more practically successful theoretically sound online algorithm for imperfect information EFGs is the one we used in DeepStack, first no-limit Texas hold'em poker bot that outperformed human professional players [MSB$^+$17]. This algorithm summarizes all it needs to know about the part of the game that already cannot occur to a vector of probabilities and counterfactual values, and limits the depth of reasoning about the game using a special form of evaluation function, represented by an artificial neural network. The high-level idea of the algorithm is quite general, but in its current form, it heavily relays on the specific structure of poker. Since the all hidden information in Texas hold'em is created at the beginning of the game and all future information

is public, the maintained summary vectors of probabilities and counterfactual values have fixed length and very clear semantics. Since the game is divided to clear rounds, these rounds create natural places where to replace the rest of the game by the neural network.

The success of DeepStack then inspired further development of algorithms using either only online computations [SKL19] or a combination of offline pre-computation with online reasoning [BSA18].

4.1 Monte Carlo Tree Search in Imperfect Information Games

A classic example of using online computation in perfect-information games is the minimax search with heuristic evaluation function, which is typically used in chess programs [RN09]. However, a significant jump in the performance of the state-of-the-art solvers for many perfect-information problems, such as the game of Go [GS11], or domain-independent planning under uncertainty [KE12] was recently caused by Monte Carlo Tree Search (MCTS) algorithm. The main idea of Monte Carlo Tree Search is running a large number of randomized simulations of the problem and learning the best actions to choose from the experience. The earlier simulations are generally used to create statistics that help to guide the later simulations to more important parts of the search space and eventually decide on the best action to play in the current state of the game.

Games with imperfect information are fundamentally more complicated than perfect-information games, because of the need for randomized strategies and high interdependence of the optimal strategy in distinct decision points. Game theory provides means to deal with all these complications, but previous attempts to adapt MCTS to imperfect-information games generally did not try to ensure optimality of the strategies from the perspective of game theory. The algorithms were developed mainly based on heuristics and analogies with the perfect information case. In [Lis14a], we show that the lack of theoretical foundations of the algorithms causes pathologies, such as more computation time per move leading to worse, more exploitable, strategies. There are several fundamental concepts in game theory, which can help in designing algorithms for creating strategies in complex games. The notion of Nash equilibrium defines the optimal strategy against the strongest possible opponent with well-understood quality guarantees against suboptimal opponents. Our contribution in MCTS algorithms for imperfect information games is mainly in creating variants of the algorithms that are guaranteed to converge to the Nash equilibrium.

4.1.1 Simultaneous Move Games

Our contribution can be divided to two parts. In the first, we address the extensive-form games with simultaneous moves, but otherwise perfect information. Even this class of games includes some of the challenges present in fully imperfect information games, such as the need for randomized strategies. In [LKLB13], we have shown that the MCTS algorithm for this class of games can be easily modified to guarantee convergence to an approximate equilibrium with a large class of selection functions. We made the theoretical analysis

substantially more extensive and further improved the performance of the modified MCTS algorithm in [KL18]. Since the modifications introduced to MCTS in order to guarantee its convergence made the empirical speed of the convergence slower, we have also investigated an alternative direction of creating sound MCTS algorithm. We have developed an online version of Monte Carlo Counterfactual Regret Minimization for simultaneous move games, which we first introduced in [LLW13] and further developed and thoroughly evaluated in [BLL$^+$16]. The key difference between this algorithm and the standard MCTS framework is that the updates in different states of the game are not performed independently from the strategy in the rest of the game, but they take into account information about the contribution of individual players to the probabilities that the updated state of the game is visited. Furthermore, it uses important sampling corrections to remove the bias in the state value estimates that would otherwise be caused by exploration. The details of the algorithm are described [BLL$^+$16] included in Appendix A.

4.1.2 Generic Imperfect Information Games

Our second line of contributions are MCTS algorithms for generic imperfect information games. Since creating theoretically sound MCTS algorithms for generic imperfect information games is more challenging. Our earlier algorithms for these games also lack guarantees of convergence to the optimal strategy.

In [LBP12], we have demonstrated that using MCTS leads to more efficient play in visibility-based pursuit evasion, compared to the fixed-depth search algorithm constituting the state of the art for that domain. Afterwards, we were gradually improving the aspects of the algorithm that prevented establishing the convergence guarantees. We started modelling the exact information both players have in a game [LPS$^+$12], we explored selection functions that guarantee convergence at least in the smallest games [Lis14a], and we added modelling of the probability distribution over the possible opponent's states after some moves are played [Lis14b]. These improvements proved to be helpful in particular domains, but they were not sufficient to guarantee the convergence to the optimal strategies in general.

Removing limitations of MCTS may lead to a theoretically sound algorithm, however, it proved to be difficult to provide convergence guarantees even if they may hold [KL18]. Therefore, we changed our strategy. We started with a game solving algorithm with well-understood theoretical guarantees and we modified it to a MCTS-style algorithm. The original algorithm, Monte Carlo Counterfactual Regret Minimization (MCCFR)[LWZB09], is an offline algorithm, which explicitly represents a strategy for all decision points in the game. Starting form a uniform strategy, it repetitively updates the strategy for the game based on traversing a subset of the game tree. In the extreme case, called outcome sampling, it traverses only a single path form the root of the game to a leaf in each iteration. In order to turn this algorithm to a MCTS-style online algorithm, we had to make the following modifications, while making sure we are not losing the convergence guarantees. First, we added the incremental tree building. Initially, only the root node and the strategy in that node is explicitly present in memory. The strategy for the remaining decision points in implicitly assumed to be uniform. In further iterations, the explicitly stored part of the game tree is extended in the parts where the players are likely to play and the strategy is

updated only in these parts of the game. Second, after some moves are already played in the game and the game reaches an information set of the player using our algorithm, we want to improve the strategy in this information set before playing a next move. Running additional samples from the root of the game is unlikely to reach the current information sets. However, forcing the samples to update it may leak the information about the current information of the player to the opponent and prevent convergence to the optimal strategy. Therefore, we have designed two targeting schemes based on importance sampling corrections, which improve the strategy without breaking the guarantees. The empirical evaluation on a range of different games shows that while our novel algorithm, Online Outcome Sampling, outperforms the heuristic alternatives on small games, it does not perform well on larger games. The weaker practical performance was later linked to high variance of the updates in [BLL$^+$16]. Both [LLB15] and [BLL$^+$16] are included in Appendix A.

4.2 Continual Resolving

The MCTS algorithms introduced in the previous section always start their samples from the root of the game. They focus their attention towards the current information set reached by a player in the game, but can do so only to a limited extend if they want to keep the optimality guarantees. This is wasteful and it increases variance of the strategy updates. Therefore, they are not efficient in larger games. In order to play large games online, it seems to be important to start reasoning about the current situation, with a limited consideration of the past. This is achieved by the algorithm called continual resolving. We have first developed the algorithm specifically for poker as a part of DeepStack. Afterwards, we have shown how this algorithm can be generalized to any extensive form game.

Continual resolving is heavily based on the CFR-D algorithm [BJB14]. CFR-D assumes the game can be split into a part played in the first few moves, called *trunk*, and multiple independent sub-games, which start after the trunk is finished. These subgames are more complex than subgames in perfect information games, since they must be composed of whole collections of subtrees instead of a single subtree. CFR-D allows computing the optimal (Nash equilibrium) strategy for the trunk with keeping only the trunk and one of the subgames in memory at any time during the run of the algorithm. CFR-D is an iterative algorithm, which in each iteration (1) performs first half of an iteration of CFR in the trunk, (2) uses the current strategy in the trunk to consistently initialize the subgames, (3) solves each of the subgames separately, storing only a limited amount of information at the root of the subgame, and (4) completes the iteration of CFR in the trunk based on the values computed in the subgames. After hundreds of iterations of CFR-D, the strategy in the trunk converges close to the optimal strategy. However, the algorithm does not store the strategy in the subgames. Therefore, [BJB14] defines how to recover the optimal strategy in a subgame once it is reached during the game. This process is called resolving. It requires computing the probability of reaching all information sets of the solving player in the root of the subgame based on the strategy in the trunk. Furthermore, it has to estimate the best possible reward the opponent can achieve after reaching his information sets in the root of the subgame. These estimates can be

computed form the solution of the reached subgame in each iteration of CFR-D.

Continual resolving is a recursive application of the resolving process. The game is solved from the root. While solving the game, the necessary estimates of the bounds on the opponent values are computed for all decision points, where the acting player can move next. Once the decision point is reached, these values are used to start the resolving process from this position, again estimating the required values for the possible next states. Estimating the values from the exact solution of the subgames is not feasible for larger games. Therefore, we have proposed two alternative ways how to do it. DeepStack uses depth-limited game solving based on CFR-D, Monte Carlo Continual Resolving uses Monte Carlo simulations.

4.2.1 DeepStack

Monte Carlo tree search techniques initially became successful in the game of Go, where it is hard to create a strong heuristic evaluation function [GS11]. However, only after strong evaluation functions were realized using deep neural networks, the combination of MCTS and evaluation functions outperformed best humans in this game [SHM+16]. The paper describing DeepStack poker bot [MSB+17] was the first to introduce the concept of continual resolving. Furthermore, it makes it practical for very large domains by proposing depth-limited game solving. The main idea comes from CFR-D. When DeepStack decides about the next move to play, it expands the game tree only for few next moves in the game. It uses the CFR-D algorithm to solve this look-ahead tree. However, instead of solving the subgames at the end of the look-ahead tree, which would be infeasible, it uses a pre-trained neural network to estimate values provided by their optimal strategy.

The neural networks in DeepStack are trained by bootstrapping. First, small subgames close to the end of the game, initialized by random trunk strategies, are solved. A neural network is trained to represent the mapping form the probabilities of reaching the information sets in these subgames to their resulting values. Second, randomly initialized subgames further form the end of the game are solved using depth-limited game solving utilizing the first neural network. Next, games even further from the end are solved using the second network, and so on.

Depth-limited continual resolving converges to the optimal strategy if the neural networks are able to learn the required targets with high precision. In practice, with only 3 seconds per move, this algorithm outperformed ten professional poker players with high statistical significance. The paper describing DeepStack is attached in Appendix A

4.2.2 Monte Carlo Continual Resolving

The most important element of the structure of a poker game used by DeepStack is a public tree, which defines the tree of well-defined subgames. In Texas hold'em, all hidden information in the game is created at the beginning. The later actions by both players and chance are public (observable by everyone) and the hidden information is revealed only after he game is finished. The public tree decomposes the game to largely independent parts. It allows defining the data necessary for resolving in the form of two relatively small constant-length

vectors. These vectors are stored and maintained based on the actions in the public tree and their length is equal to the length of the hidden information created at the beginning of the game.

General extensive-form games often contain actions unobservable by the players. In a visibility-based pursuit-evasion game, all player's actions are unobservable if the players do not see each other. Furthermore, the game contains observations related to player's own actions, such as the absence of opponent at a location, which do not have any equivalent in poker. The size of the hidden information may grow to a size comparable to the number of leaves in the game three. In other games, a public tree may not be present at all. To apply DeepStack's algorithm to an arbitrary EFG, we have to allow the summary vectors to have very large and variable size. Moreover, we have to define sound update rules for (partially) unobservable actions and observations other than current actions of the opponent.

The DeepStack's algorithm also uses poker structure to decide where to end the look-ahead tree and use the heuristic evaluation function. It is always at the beginning of the new round of the game after additional public cards are revealed. These points in the game are convenient for several reasons. First, it is trivial to identify them when building the look-ahead tree. Second, they are sufficiently close to each other so that the look-ahead tree can always be quickly solved in a relatively small memory. Third, they share sufficient strategic similarity, so that their solutions can be generalized using deep neural networks.

In [SKL19], we have generalized continual resolving to be applicable in any extensive-form game, not just poker. Furthermore, instead of using a neural network, we construct the strategy and estimate the values necessary for resolving using Monte Carlo simulations. This allows easy evaluation of continual resolving on arbitrary domains, but it does not reach the performance achievable using a pre-trained evaluation function. The paper describing Continual Resolving for arbitrary domains and Monte Carlo Continual Resolving [SKL19] is included in Appendix A.

Chapter 5

Conclusions and Future Work

Most of my recent research focused on approximating optimal strategies in large extensive form games. The main properties of the algorithms I strived for are minimal use of domain specific knowledge and well understood theoretical guarantees on the performance of the algorithms. I believe these properties are important in order to understand the essence of the solved problems and maximize future impact of the produced solutions.

In my work, I have contributed to the state of the art in more classical algorithms based on abstracting the large games to manageable size, but I believe my main contribution lies in participating in creation of theoretically sound online methods for approximating optimal strategies in large games only for the decision points that occur during a specific game. This removes the memory and computation intensive requirement of pre-computing the optimal strategy for the whole game in advance and vastly increases the applicability of game solving in real world problems.

DeepStack has demonstrated that we can scale online game solving to human size problems if the structure of the game is sufficiently simple. No-limit Texas hold'em has small amount of hidden information, which does not change during the game. Also, it has well defined "rounds", which are suitable for placing the evaluation function. We have shown that this approach can be extended to games beyond poker, but in many larger games, the general algorithm would not be practical. The main problem is an efficient representation of the statistics required by continual resolving in games with large and dynamically changing set of possible hidden states of the game. Novel data representations and algorithms for their updating likely need to be developed in order to deploy the algorithm in these games.

Even if managing the large sets of hidden information is possible, the algorithm would still be able to play only zero-sum games with rational players. Therefore, an interesting future direction, which has been explored in the offline algorithm, but not in online algorithms is using models of bounded rationality of the opponents. These models can be learned from historical data or based on psychological assumptions. In both cases, it is not clear, how to incorporate them into online algorithms in a principled way. A completely new set of

challenges comes also from considering more than two players, or non-zero-sum structure of rewards in the game. Game theory is no longer able to prescribe the optimal strategy to play in the game and the players must more carefully adapt to the play of their opponents.

Next interesting line of research is using the novel algorithms for solving extensive form games in practical applications. Most sequential decision-making problems in network and physical security scenarios can be modeled as extensive form games. However, existing applications of game theory in these domains usually do not utilize this model and rather ignore the sequential nature of the problem. Notable exceptions are out previous works on sequential reconfiguration of detectors of malicious behavior in network security in order to maximize the chance of detecting an ongoing attack [LPS+12] and selecting optimal configuration of honeypots to protect a specific computer network against a sequential attack [DLK+16]. Both these works are included in Appendix A. One of the main limitations of using EFGs in practical applications has been their limited scalability. With the recent advancements and their further generalizations, these or similar models can be revisited and solved in greater details.

Bibliography

[BBD+03] Darse Billings, Neil Burch, Aaron Davidson, Robert Holte, Jonathan Schaeffer, Terence Schauenberg, and Duane Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *IJCAI*, pages 661–668, 2003.

[BBJT15] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit holdem poker is solved. *Science*, 347(6218):145–149, 2015.

[BJB14] Neil Burch, Michael Johanson, and Michael Bowling. Solving imperfect information games using decomposition. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[BJTK15] Branislav Bosansky, Albert Xin Jiang, Milind Tambe, and Christopher Kiekintveld. Combining compact representation and incremental generation in large games with sequential strategies. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[BLGS18] Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. Deep counterfactual regret minimization. *arXiv preprint arXiv:1811.00164*, 2018.

[BLL+16] Branislav Bosansky, Viliam Lisy, Marc Lanctot, Jiri Cermak, and M.H.M. Winands. Algorithms for Computing Strategies in Two-Player Simultaneous Move Games. *Artificial Intelligence*, 2016.

[BS15] Noam Brown and Tuomas Sandholm. Simultaneous abstraction and equilibrium finding in games. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 489–496. AAAI Press, 2015.

[BSA18] Noam Brown, Tuomas Sandholm, and Brandon Amos. Depth-limited solving for imperfect-information games. In *Advances in Neural Information Processing Systems*, pages 7663–7674, 2018.

[CBL17] Jiri Cermak, Branislav Bosansky, and Viliam Lisy. An algorithm for constructing and solving imperfect recall abstractions of large extensive-form games. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 936–942. AAAI Press, 2017.

[CLB18] Jiri Cermak, Viliam Lisy, and Branislav Bosansky. Constructing imperfect recall abstractions to solve large extensive-form games. *arXiv preprint arXiv:1803.05392*, 2018.

[CPW12] Peter I Cowling, Edward J Powley, and Daniel Whitehouse. Information set monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):120–143, 2012.

[CSS06] Peter Cramton, Yoav Shoham, and Richard Steinberg. *Combinatorial auctions*. MIT press, 2006.

[Dal53] Norman Dalkey. Equivalence of information patterns and essentially determinate games. *Contributions to the Theory of Games*, 2:217, 1953.

[DLK+16] Karel Durkota, Viliam Lisy, Christopher Kiekintveld, Branislav Bosansky, and Michal Pechoucek. Case studies of network defense with attack graph games. *IEEE Intelligent Systems*, 31(5):24–30, 2016.

[GHPS07] Andrew Gilpin, Samid Hoda, Javier Pena, and Tuomas Sandholm. Gradient-based algorithms for finding nash equilibria in extensive form games. In *International Workshop on Web and Internet Economics*, pages 57–69. Springer, 2007.

[GS11] Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[GS13] Samuel Ganzfried and Tuomas W Sandholm. Action translation in extensive-form games with large action spaces: Axioms, paradoxes, and the pseudo-harmonic mapping. AAAI, 2013.

[GSS07a] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre 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*, volume 22, page 50. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[GSS07b] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre 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*, volume 22, page 50, 2007.

[HGPnS10] Samid Hoda, Andrew Gilpin, Javier Peña, and Tuomas Sandholm. Smoothing Techniques for Computing Nash Equilibria of Sequential Games. *Mathematics of Operations Research*, 35(2):494–512, May 2010.

[Hsu06] F. Hsu. *Behind Deep Blue: Building the Computer that Defeated the World Chess Championship*. Princeton University Press, 2006.

[JBBB12] Michael Johanson, Nolan Bard, Neil Burch, and Michael Bowling. Finding optimal abstract strategies in extensive-form games. In *AAAI*, 2012.

[Joh13] Michael Johanson. Measuring the size of large no-limit poker games. *IarXiv:1302.7008*, 2013.

[JTP+10] Manish Jain, Jason Tsai, James Pita, Christopher Kiekintveld, Shyamsunder Rathi, Milind Tambe, and Fernando Ordonez. Software Assistants for Randomized Patrol Planning for the LAX Airport Police and the Federal Air Marshals Service. *Interfaces*, 40:267–290, 2010.

[KL18] Vojtech Kovarik and Viliam Lisy. Analysis of hannan consistent selection for monte carlo tree search in simultaneous move games. *arXiv preprint arXiv:1804.09045*, 2018.

[KMVS96] Daphne Koller, Nimrod Megiddo, and Bernhard Von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2):247–259, 1996.

[KS06] Levente Kocsis and Csaba Szepesvari. Bandit based monte-carlo planning. In *ECML 2006*, pages 282–293. 2006.

[KS16] Christian Kroer and Tuomas Sandholm. Imperfect-Recall Abstractions with Bounds in Games. In *Proceedings of the seventeenth ACM conference on Economics and computation*, pages 459–476. ACM, 2016.

[LB17] Viliam Lisy and Michael Bowling. Eqilibrium approximation quality of current no-limit poker bots. In *AAAI-17 Workshop on Computer Poker and Imperfect Information Games*, 2017.

[LBP12] Viliam Lisy, Branislav Bošansky, and Michal Pěchouček. Anytime algorithms for multi-agent visibility-based pursuit-evasion games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '12, pages 1301–1302, 2012.

[LDB16] Viliam Lisy, Trevor Davis, and Michael Bowling. Counterfactual regret minimization in sequential security games. In *Proceedings of the Thirtieth Conference on Artificial Intelligence (AAAI)*, pages 544–550, 2016.

[LHG+18] Hui Li, Kailiang Hu, Zhibang Ge, Tao Jiang, Yuan Qi, and Le Song. Double neural counterfactual regret minimization. *arXiv preprint arXiv:1812.10607*, 2018.

[Lis14a] Viliam Lisy. Alternative selection functions for information set monte carlo tree search. *Acta Polytechnica*, 54(5):333–340, 2014.

[Lis14b] Viliam Lisy. *Monte Carlo Tree Search in Imperfect-Information Games*. PhD thesis, Faculty of Electrical Engineering, Czech Technical University in Prague, 2014.

[LKLB13] Viliam Lisy, Vojtech Kovarik, Marc Lanctot, and Branislav Bosansky. Convergence of Monte Carlo Tree Search in Simultaneous Move Games. In *Advances in Neural Information Processing Systems (NIPS)*, volume 26, pages 2112–2120, 2013.

[LLB15] Viliam Lisy, Marc Lanctot, and Michael Bowling. Online monte carlo counterfactual regret minimization for search in imperfect information games. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, pages 27–36, 2015.

[LLW13] Marc Lanctot, Viliam Lisy, and Mark HM Winands. Monte carlo tree search in simultaneous move games with applications to goofspiel. In *Workshop on Computer Games*, pages 28–43. Springer, 2013.

[LPS+12] Viliam Lisy, Radek Pibil, Jan Stiborek, Branislav Bosansky, and Michal Pechoucek. Game-theoretic approach to adversarial plan recognition. In *Proceedings of the 20th European Conference on Artificial Intelligence*, pages 546–551. IOS Press, 2012.

[LSBF10] Jeffrey Richard Long, Nathan R Sturtevant, Michael Buro, and Timothy Furtak. Understanding the success of perfect information monte carlo sampling in game tree search. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[LWZB09] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte carlo sampling for regret minimization in extensive games. In *Advances in Neural Information Processing Systems 22 (NIPS)*, pages 1078–1086, 2009.

[MSB+17] Matej Moravcik, Martin Schmid, Neil Burch, Viliam Lisy, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in no-limit poker. *Science*, page aam6960, 2017.

[NW12] Pim Nijssen and Mark HM Winands. Monte carlo tree search for the hide-and-seek game scotland yard. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(4):282–294, 2012.

[RFR08] Julie Rehmeyer, Nathan Fox, and Renzi Rico. Ante up, human: The adventures of polaris the poker-playing robot. *Wired*, 16.12:186–191, December 2008.

[RN10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[RNK+10] Eric Raboin, Dana Nau, Ugur Kuter, Satyandra K Gupta, and Petr Svec. Strategy generation in multi-agent imperfect-information pursuit games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 947–954. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[SAY+12] Eric Shieh, Bo An, Rong Yang, Milind Tambe, Craig Baldwin, Joseph Direnzo, Garrett Meyer, Craig W Baldwin, Ben J Maule, and Garrett R Meyer. PROTECT : A Deployed Game Theoretic System to Protect the Ports of the United States. *AAMAS*, 2012.

[SHM+16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[SKL19] Michal Sustr, Vojtech Kovarik, and Viliam Lisy. Monte carlo continual resolving for online strategy computation in imperfect information games. In *Proceedings of the 2019 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '19, 2019.

[SL00] Jiefu Shi and Michael L Littman. Abstraction methods for game theoretic poker. *Computers and Games*, 2063:333–345, 2000.

[Tam11] Milind Tambe. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press, 2011.

[TBJB15] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. Solving heads-up limit texas hold'em. In *IJCAI*, pages 645–652, 2015.

[vS96] Bernhard von Stengel. Efficient Computation of Behavior Strategies. *Games and Economic Behavior*, 14:220–246, 1996.

[ZJBP08] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. *Advances in Neural Information Processing Systems*, 20:1729–1736, 2008.

Appendix A

Attached Published Papers

The following peer reviewed publications are attached in this appendix:

MSB+17 Matej Moravcik, Martin Schmid, Neil Burch, Viliam Lisy, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in no-limit poker. *Science*, page aam6960, 2017.

BLL+16 Branislav Bosansky, Viliam Lisy, Marc Lanctot, Jiri Cermak, and M.H.M. Winands. Algorithms for Computing Strategies in Two-Player Simultaneous Move Games. *Artificial Intelligence*, 2016.

DLK+16 Karel Durkota, Viliam Lisy, Christopher Kiekintveld, Branislav Bosansky, and Michal Pechoucek. Case studies of network defense with attack graph games. *IEEE Intelligent Systems*, 31(5):24–30, 2016.

SKL19 Michal Sustr, Vojtech Kovarik, and Viliam Lisy. Monte carlo continual resolving for online strategy computation in imperfect information games. In *Proceedings of the 2019 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '19, 2019.

LLB15 Viliam Lisy, Marc Lanctot, and Michael Bowling. Online monte carlo counterfactual regret minimization for search in imperfect information games. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, pages 27–36, 2015.

CBL17 Jiri Cermak, Branislav Bosansky, and Viliam Lisy. An algorithm for constructing and solving imperfect recall abstractions of large extensive-form games. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 936–942. AAAI Press, 2017.

LDB16 Viliam Lisy, Trevor Davis, and Michael Bowling. Counterfactual regret minimization in sequential security games. edings of the 30th Conference on Artificial Intelligence (AAAI) In *Proceedings of the Thirtieth Conference on Artificial Intelligence (AAAI)*, pages 544-550 2016.

LB17 Viliam Lisy and Michael Bowling. Eqilibrium approximation quality of current no-limit poker bots. In *AAAI-17 Workshop on Computer Poker and Imperfect Information Games*, 2017.

LKLB13 Viliam Lisy, Vojtech Kovarik, Marc Lanctot, and Branislav Bosansky. Convergence of Monte Carlo Tree Search in Simultaneous Move Games. In *Advances in Neural Information Processing Systems (NIPS)*, volume 26, pages 2112–2120, 2013.

LPS+12 Viliam Lisy, Radek Pibil, Jan Stiborek, Branislav Bosansky, and Michal Pechoucek. Game-theoretic approach to adversarial plan recognition. In *Proceedings of the 20th European Conference on Artificial Intelligence*, pages 546–551. IOS Press, 2012.

arXiv:1701.01724v3 [cs.AI] 3 Mar 2017

DeepStack: Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker

Matej Moravčík[♠,♥,†], Martin Schmid[♠,♥,†], Neil Burch[♠], Viliam Lisý[♠,♣],
Dustin Morrill[♠], Nolan Bard[♠], Trevor Davis[♠],
Kevin Waugh[♠], Michael Johanson[♠], Michael Bowling[♠,*]

[♠]Department of Computing Science, University of Alberta,
Edmonton, Alberta, T6G2E8, Canada

[♥]Department of Applied Mathematics, Charles University,
Prague, Czech Republic

[♣]Department of Computer Science, FEE, Czech Technical University,
Prague, Czech Republic

[†]These authors contributed equally to this work and are listed in alphabetical order.

[*]To whom correspondence should be addressed; E-mail: bowling@cs.ualberta.ca

Artificial intelligence has seen several breakthroughs in recent years, with games often serving as milestones. A common feature of these games is that players have perfect information. Poker is the quintessential game of imperfect information, and a longstanding challenge problem in artificial intelligence. We introduce DeepStack, an algorithm for imperfect information settings. It combines recursive reasoning to handle information asymmetry, decomposition to focus computation on the relevant decision, and a form of intuition that is automatically learned from self-play using deep learning. In a study involving 44,000 hands of poker, DeepStack defeated with statistical significance professional poker players in heads-up no-limit Texas hold'em. The approach is theoretically sound and is shown to produce more difficult to exploit strategies than prior approaches. [1]

Games have long served as benchmarks and marked milestones of progress in artificial intelligence (AI). In the last two decades, computer programs have reached a performance that

[1]This is the author's version of the work. It is posted here by permission of the AAAS for personal use, not for redistribution. The definitive version was published in *Science*, (March 02, 2017), doi: 10.1126/science.aam6960.

exceeds expert human players in many games, e.g., backgammon (*1*), checkers (*2*), chess (*3*), Jeopardy! (*4*), Atari video games (*5*), and go (*6*). These successes all involve games with information symmetry, where all players have identical information about the current state of the game. This property of perfect information is also at the heart of the algorithms that enabled these successes, e.g., local search during play (*7, 8*).

The founder of modern game theory and computing pioneer, von Neumann, envisioned reasoning in games without perfect information. "Real life is not like that. Real life consists of bluffing, of little tactics of deception, of asking yourself what is the other man going to think I mean to do. And that is what games are about in my theory." (*9*) One game that fascinated von Neumann was poker, where players are dealt private cards and take turns making bets or bluffing on holding the strongest hand, calling opponents' bets, or folding and giving up on the hand and the bets already added to the pot. Poker is a game of imperfect information, where players' private cards give them asymmetric information about the state of game.

Heads-up no-limit Texas hold'em (HUNL) is a two-player version of poker in which two cards are initially dealt face-down to each player, and additional cards are dealt face-up in three subsequent rounds. No limit is placed on the size of the bets although there is an overall limit to the total amount wagered in each game (*10*). AI techniques have previously shown success in the simpler game of heads-up limit Texas hold'em, where all bets are of a fixed size resulting in just under 10^{14} decision points (*11*). By comparison, computers have exceeded expert human performance in go (*6*), a perfect information game with approximately 10^{170} decision points (*12*). The imperfect information game HUNL is comparable in size to go, with the number of decision points exceeding 10^{160} (*13*).

Imperfect information games require more complex reasoning than similarly sized perfect information games. The correct decision at a particular moment depends upon the probability distribution over private information that the opponent holds, which is revealed through their past actions. However, how our opponent's actions reveal that information depends upon their knowledge of our private information and how our actions reveal it. This kind of recursive reasoning is why one cannot easily reason about game situations in isolation, which is at the heart of heuristic search methods for perfect information games. Competitive AI approaches in imperfect information games typically reason about the entire game and produce a complete strategy prior to play (*14–16*). Counterfactual regret minimization (CFR) (*14,17,18*) is one such technique that uses self-play to do recursive reasoning through adapting its strategy against itself over successive iterations. If the game is too large to be solved directly, the common response is to solve a smaller, abstracted game. To play the original game, one translates situations and actions from the original game to the abstract game.

Although this approach makes it feasible for programs to reason in a game like HUNL, it does so by squeezing HUNL's 10^{160} situations down to the order of 10^{14} abstract situations. Likely as a result of this loss of information, such programs are behind expert human play. In 2015, the computer program Claudico lost to a team of professional poker players by a margin of 91 mbb/g (*19*), which is a "huge margin of victory" (*20*). Furthermore, it has been recently shown that abstraction-based programs from the Annual Computer Poker Competition have

massive flaws (*21*). Four such programs (including top programs from the 2016 competition) were evaluated using a local best-response technique that produces an approximate lower-bound on how much a strategy can lose. All four abstraction-based programs are beatable by over 3,000 mbb/g, which is four times as large as simply folding each game.

DeepStack takes a fundamentally different approach. It continues to use the recursive reasoning of CFR to handle information asymmetry. However, it does not compute and store a complete strategy prior to play and so has no need for explicit abstraction. Instead it considers each particular situation as it arises during play, but not in isolation. It avoids reasoning about the entire remainder of the game by substituting the computation beyond a certain depth with a fast approximate estimate. This estimate can be thought of as DeepStack's intuition: a gut feeling of the value of holding any possible private cards in any possible poker situation. Finally, DeepStack's intuition, much like human intuition, needs to be trained. We train it with deep learning (*22*) using examples generated from random poker situations. We show that DeepStack is theoretically sound, produces strategies substantially more difficult to exploit than abstraction-based techniques, and defeats professional poker players at HUNL with statistical significance.

DeepStack

DeepStack is a general-purpose algorithm for a large class of sequential imperfect information games. For clarity, we will describe its operation in the game of HUNL. The state of a poker game can be split into the players' private information, hands of two cards dealt face down, and the public state, consisting of the cards laying face up on the table and the sequence of betting actions made by the players. Possible sequences of public states in the game form a public tree with every public state having an associated public subtree (Fig. 1).

A player's strategy defines a probability distribution over valid actions for each decision point, where a decision point is the combination of the public state and the hand for the acting player. Given a player's strategy, for any public state one can compute the player's range, which is the probability distribution over the player's possible hands given that the public state is reached.

Fixing both players' strategies, the utility for a particular player at a terminal public state, where the game has ended, is a bilinear function of both players' ranges using a payoff matrix determined by the rules of the game. The expected utility for a player at any other public state, including the initial state, is the expected utility over reachable terminal states given the players' fixed strategies. A best-response strategy is one that maximizes a player's expected utility against an opponent strategy. In two-player zero-sum games, like HUNL, a solution or Nash equilibrium strategy (*23*) maximizes the expected utility when playing against a best-response opponent strategy. The exploitability of a strategy is the difference in expected utility against its best-response opponent and the expected utility under a Nash equilibrium.

The DeepStack algorithm seeks to compute and play a low-exploitability strategy for the

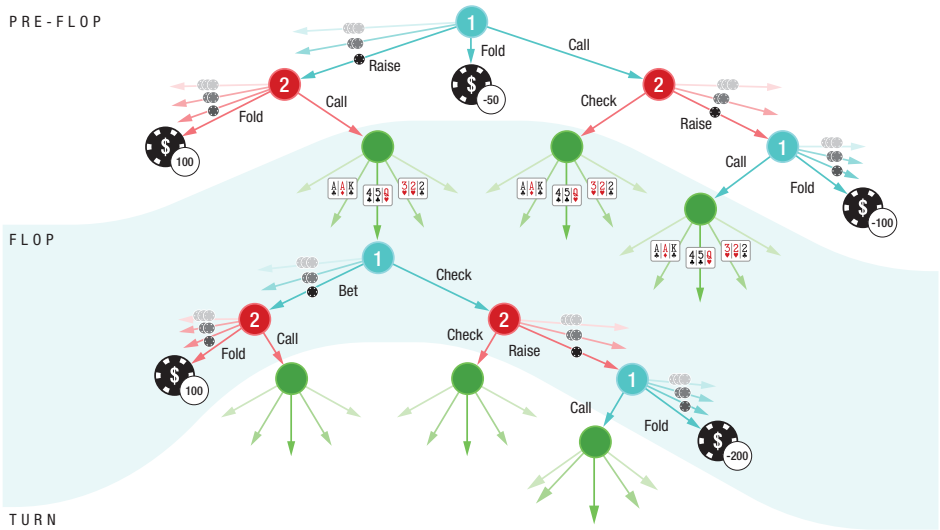


Figure 1: A portion of the public tree in HUNL. Nodes represent public states, whereas edges represent actions: red and turquoise showing player betting actions, and green representing public cards revealed by chance. The game ends at terminal nodes, shown as a chip with an associated value. For terminal nodes where no player folded, the player whose private cards form a stronger poker hand receives the value of the state.

game, i.e., solve for an approximate Nash equilibrium. DeepStack computes this strategy during play only for the states of the public tree that actually arise. Although computed during play, DeepStack's strategy is static, albeit stochastic, because it is the result of a deterministic computation that produces a probability distribution over the available actions.

The DeepStack algorithm (Fig. 2) is composed of three ingredients: a sound local strategy computation for the current public state, depth-limited lookahead using a learned value function to avoid reasoning to the end of the game, and a restricted set of lookahead actions. At a conceptual level these three ingredients describe heuristic search, which is responsible for many of AI's successes in perfect information games. Until DeepStack, no theoretically sound application of heuristic search was known in imperfect information games. The heart of heuristic search methods is the idea of "continual re-searching", where a sound local search procedure is invoked whenever the agent must act without retaining any memory of how or why it acted to reach the current state. At the heart of DeepStack is continual re-solving, a sound local strategy computation which only needs minimal memory of how and why it acted to reach the current public state.

Continual re-solving. Suppose we have taken actions according to a particular solution strategy but then in some public state forget this strategy. Can we reconstruct a solution strategy for the subtree without having to solve the entire game again? We can, through the process of

re-solving (*17*). We need to know both our range at the public state and a vector of expected values achieved by the opponent under the previous solution for each opponent hand (*24*). With these values, we can reconstruct a strategy for only the remainder of the game, which does not increase our overall exploitability. Each value in the opponent's vector is a counterfactual value, a conditional "what-if" value that gives the expected value if the opponent reaches the public state with a particular hand. The CFR algorithm also uses counterfactual values, and if we use CFR as our solver, it is easy to compute the vector of opponent counterfactual values at any public state.

Re-solving, however, begins with a strategy, whereas our goal is to avoid ever maintaining a strategy for the entire game. We get around this by doing continual re-solving: reconstructing a strategy by re-solving every time we need to act; never using the strategy beyond our next action. To be able to re-solve at any public state, we need only keep track of our own range and a suitable vector of opponent counterfactual values. These values must be an upper bound on the value the opponent can achieve with each hand in the current public state, while being no larger than the value the opponent could achieve had they deviated from reaching the public state. This is an important relaxation of the counterfactual values typically used in re-solving, with a proof of sufficiency included in our proof of Theorem 1 below (*10*).

At the start of the game, our range is uniform and the opponent counterfactual values are initialized to the value of being dealt each private hand. When it is our turn to act we re-solve the subtree at the current public state using the stored range and opponent values, and act according to the computed strategy, discarding the strategy before we act again. After each action, either by a player or chance dealing cards, we update our range and opponent counterfactual values according to the following rules: (i) Own action: replace the opponent counterfactual values with those computed in the re-solved strategy for our chosen action. Update our own range using the computed strategy and Bayes' rule. (ii) Chance action: replace the opponent counterfactual values with those computed for this chance action from the last re-solve. Update our own range by zeroing hands in the range that are impossible given new public cards. (iii) Opponent action: no change to our range or the opponent values are required.

These updates ensure the opponent counterfactual values satisfy our sufficient conditions, and the whole procedure produces arbitrarily close approximations of a Nash equilibrium (see Theorem 1). Notice that continual re-solving never keeps track of the opponent's range, instead only keeping track of their counterfactual values. Furthermore, it never requires knowledge of the opponent's action to update these values, which is an important difference from traditional re-solving. Both will prove key to making this algorithm efficient and avoiding any need for the translation step required with action abstraction methods (*25, 26*).

Continual re-solving is theoretically sound, but by itself impractical. While it does not ever maintain a complete strategy, re-solving itself is intractable except near the end of the game. In order to make continual re-solving practical, we need to limit the depth and breadth of the re-solved subtree.

Limited depth lookahead via intuition. As in heuristic search for perfect information games, we would like to limit the depth of the subtree we have to reason about when re-solving. However, in imperfect information games we cannot simply replace a subtree with a heuristic or precomputed value. The counterfactual values at a public state are not fixed, but depend on how players play to reach the public state, i.e., the players' ranges (*17*). When using an iterative algorithm, such as CFR, to re-solve, these ranges change on each iteration of the solver.

DeepStack overcomes this challenge by replacing subtrees beyond a certain depth with a learned counterfactual value function that approximates the resulting values if that public state were to be solved with the current iteration's ranges. The inputs to this function are the ranges for both players, as well as the pot size and public cards, which are sufficient to specify the public state. The outputs are a vector for each player containing the counterfactual values of holding each hand in that situation. In other words, the input is itself a description of a poker game: the probability distribution of being dealt individual private hands, the stakes of the game, and any public cards revealed; the output is an estimate of how valuable holding certain cards would be in such a game. The value function is a sort of intuition, a fast estimate of the value of finding oneself in an arbitrary poker situation. With a depth limit of four actions, this approach reduces the size of the game for re-solving from 10^{160} decision points at the start of the game down to no more than 10^{17} decision points. DeepStack uses a deep neural network as its learned value function, which we describe later.

Sound reasoning. DeepStack's depth-limited continual re-solving is sound. If DeepStack's intuition is "good" and "enough" computation is used in each re-solving step, then DeepStack plays an arbitrarily close approximation to a Nash equilibrium.

Theorem 1 *If the values returned by the value function used when the depth limit is reached have error less than ϵ, and T iterations of CFR are used to re-solve, then the resulting strategy's exploitability is less than $k_1\epsilon + k_2/\sqrt{T}$, where k_1 and k_2 are game-specific constants. For the proof, see (10).*

Sparse lookahead trees. The final ingredient in DeepStack is the reduction in the number of actions considered so as to construct a sparse lookahead tree. DeepStack builds the lookahead tree using only the actions fold (if valid), call, 2 or 3 bet actions, and all-in. This step voids the soundness property of Theorem 1, but it allows DeepStack to play at conventional human speeds. With sparse and depth-limited lookahead trees, the re-solved games have approximately 10^7 decision points, and are solved in under five seconds using a single NVIDIA GeForce GTX 1080 graphics card. We also use the sparse and depth-limited lookahead solver from the start of the game to compute the opponent counterfactual values used to initialize DeepStack's continual re-solving.

Relationship to heuristic search in perfect information games. There are three key challenges that DeepStack overcomes to incorporate heuristic search ideas in imperfect information

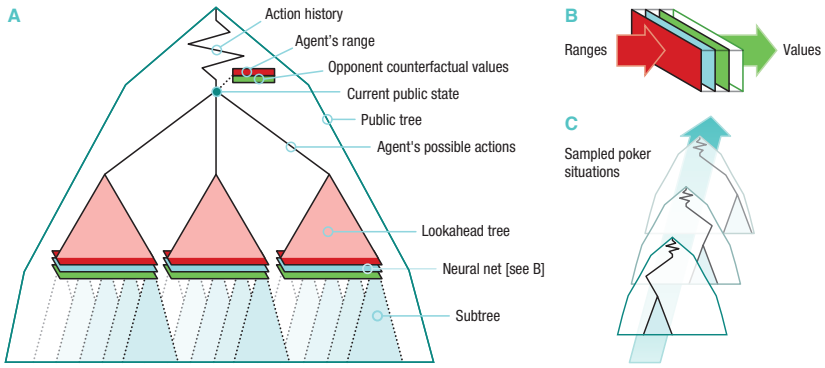


Figure 2: **DeepStack overview.** **(A)** DeepStack reasons in the public tree always producing action probabilities for all cards it can hold in a public state. It maintains two vectors while it plays: its own range and its opponent's counterfactual values. As the game proceeds, its own range is updated via Bayes' rule using its computed action probabilities after it takes an action. Opponent counterfactual values are updated as discussed under "Continual re-solving". To compute action probabilities when it must act, it performs a re-solve using its range and the opponent counterfactual values. To make the re-solve tractable it restricts the available actions of the players and lookahead is limited to the end of the round. During the re-solve, counterfactual values for public states beyond its lookahead are approximated using DeepStack's learned evaluation function. **(B)** The evaluation function is represented with a neural network that takes the public state and ranges from the current iteration as input and outputs counterfactual values for both players (Fig. 3). **(C)** The neural network is trained prior to play by generating random poker situations (pot size, board cards, and ranges) and solving them to produce training examples. Complete pseudocode can be found in Algorithm S1 (*10*).

games. First, sound re-solving of public states cannot be done without knowledge of how and why the players acted to reach the public state. Instead, two additional vectors, the agent's range and opponent counterfactual values, must be maintained to be used in re-solving. Second, re-solving is an iterative process that traverses the lookahead tree multiple times instead of just once. Each iteration requires querying the evaluation function again with different ranges for every public state beyond the depth limit. Third, the evaluation function needed when the depth limit is reached is conceptually more complicated than in the perfect information setting. Rather than returning a single value given a single state in the game, the counterfactual value function needs to return a vector of values given the public state and the players' ranges. Because of this complexity, to learn such a value function we use deep learning, which has also been successful at learning complex evaluation functions in perfect information games (*6*).

Relationship to abstraction-based approaches. Although DeepStack uses ideas from abstraction, it is fundamentally different from abstraction-based approaches. DeepStack restricts the number of actions in its lookahead trees, much like action abstraction (*25, 26*). However,

each re-solve in DeepStack starts from the actual public state and so it always perfectly understands the current situation. The algorithm also never needs to use the opponent's actual action to obtain correct ranges or opponent counterfactual values, thereby avoiding translation of opponent bets. We used hand clustering as inputs to our counterfactual value functions, much like explicit card abstraction approaches (*27, 28*). However, our clustering is used to estimate counterfactual values at the end of a lookahead tree rather than limiting what information the player has about their cards when acting. We later show that these differences result in a strategy substantially more difficult to exploit.

Deep Counterfactual Value Networks

Deep neural networks have proven to be powerful models and are responsible for major advances in image and speech recognition (*29, 30*), automated generation of music (*31*), and game-playing (*5, 6*). DeepStack uses deep neural networks with a tailor-made architecture, as the value function for its depth-limited lookahead (Fig. 3). Two separate networks are trained: one estimates the counterfactual values after the first three public cards are dealt (flop network), the other after dealing the fourth public card (turn network). An auxiliary network for values before any public cards are dealt is used to speed up the re-solving for early actions (*10*).

Architecture. DeepStack uses a standard feedforward network with seven fully connected hidden layers each with 500 nodes and parametric rectified linear units (*32*) for the output. This architecture is embedded in an outer network that forces the counterfactual values to satisfy the zero-sum property. The outer computation takes the estimated counterfactual values, and computes a weighted sum using the two players' input ranges resulting in separate estimates of the game value. These two values should sum to zero, but may not. Half the actual sum is then subtracted from the two players' estimated counterfactual values. This entire computation is differentiable and can be trained with gradient descent. The network's inputs are the pot size as a fraction of the players' total stacks and an encoding of the players' ranges as a function of the public cards. The ranges are encoded by clustering hands into 1,000 buckets, as in traditional abstraction methods (*27, 28, 33*), and input as a vector of probabilities over the buckets. The output of the network are vectors of counterfactual values for each player and hand, interpreted as fractions of the pot size.

Training. The turn network was trained by solving 10 million randomly generated poker turn games. These turn games used randomly generated ranges, public cards, and a random pot size (*10*). The target counterfactual values for each training game were generated by solving the game with players' actions restricted to fold, call, a pot-sized bet, and an all-in bet, but no card abstraction. The flop network was trained similarly with 1 million randomly generated flop games. However, the target counterfactual values were computed using our depth-limited

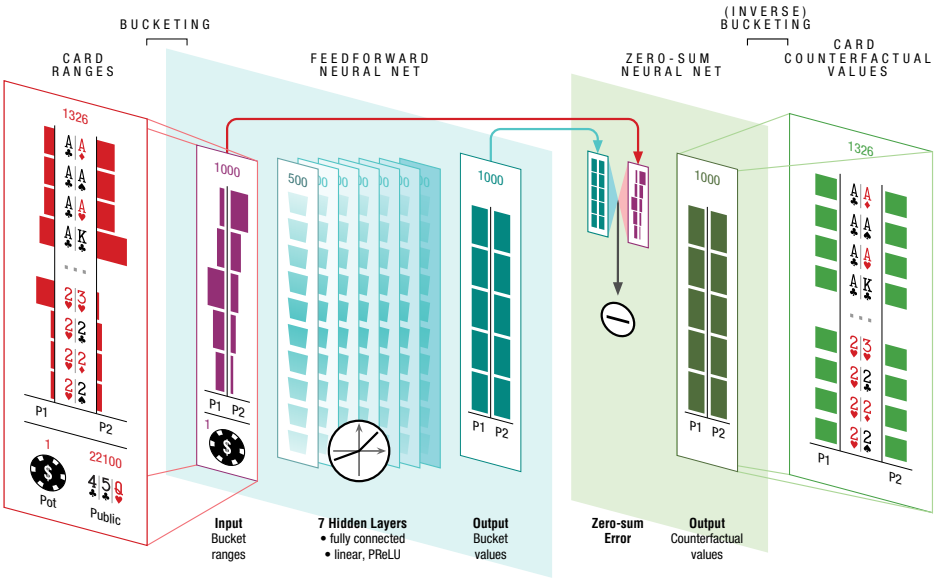


Figure 3: **Deep counterfactual value network.** The inputs to the network are the pot size, public cards, and the player ranges, which are first processed into hand clusters. The output from the seven fully connected hidden layers is post-processed to guarantee the values satisfy the zero-sum constraint, and then mapped back into a vector of counterfactual values.

solving procedure and our trained turn network. The networks were trained using the Adam gradient descent optimization procedure (*34*) with a Huber loss (*35*).

Evaluating DeepStack

We evaluated DeepStack by playing it against a pool of professional poker players recruited by the International Federation of Poker (*36*). Thirty-three players from 17 countries were recruited. Each was asked to complete a 3,000 game match over a period of four weeks between November 7th and December 12th, 2016. Cash incentives were given to the top three performers ($5,000, $2,500, and $1,250 CAD).

Evaluating performance in HUNL is challenging because of the large variance in per-game outcomes owing to randomly dealt cards and stochastic choices made by the players. The better player may lose in a short match simply because they were dealt weaker hands or their rare bluffs were made at inopportune times. As seen in the Claudico match (*20*), even 80,000 games may not be enough to statistically significantly separate players whose skill differs by a considerable margin. We evaluate performance using AIVAT (*37*), a provably unbiased low-variance technique for evaluating performance in imperfect information games based on carefully constructed control variates. AIVAT requires an estimated value of holding each hand in each public state, and then uses the expected value changes that occur due to chance actions and actions of

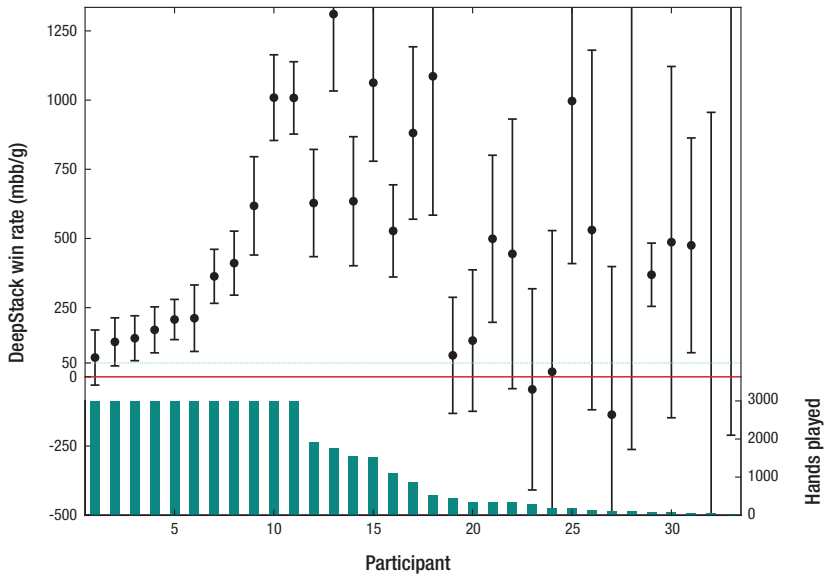


Figure 4: **Performance of professional poker players against DeepStack.** Performance estimated with AIVAT along with a 95% confidence interval. The solid bars at the bottom show the number of games the participant completed.

players with known strategies (i.e., DeepStack) to compute the control variate. DeepStack's own value function estimate is perfectly suited for AIVAT. Indeed, when used with AIVAT we get an unbiased performance estimate with an impressive 85% reduction in standard deviation. Thanks to this technique, we can show statistical significance (*38*) in matches with as few as 3,000 games.

In total 44,852 games were played by the thirty-three players with 11 players completing the requested 3,000 games. Over all games played, DeepStack won 492 mbb/g. This is over 4 standard deviations away from zero, and so, highly significant. Note that professional poker players consider 50 mbb/g a sizable margin. Using AIVAT to evaluate performance, we see DeepStack was overall a bit lucky, with its estimated performance actually 486 mbb/g. However, as a lower variance estimate, this margin is over 20 standard deviations from zero.

The performance of individual participants measured with AIVAT is summarized in Figure 4. Amongst those players that completed the requested 3,000 games, DeepStack is estimated to be winning by 394 mbb/g, and individually beating 10 out of 11 such players by a statistically significant margin. Only for the best performing player, still estimated to be losing by 70 mbb/g, is the result not statistically significant. More details on the participants and their results are presented in (*10*).

Table 1: **Exploitability bounds from Local Best Response.** For all listed programs, the value reported is the largest estimated exploitability when applying LBR using a variety of different action sets. Table S2 gives a more complete presentation of these results (*10*). ‡: LBR was unable to identify a positive lower bound for DeepStack's exploitability.

Program	LBR (mbb/g)
Hyperborean (2014)	4675
Slumbot (2016)	4020
Act1 (2016)	3302
Always Fold	750
DeepStack	**0** ‡

Exploitability

The main goal of DeepStack is to approximate Nash equilibrium play, i.e., minimize exploitability. While the exact exploitability of a HUNL poker strategy is intractable to compute, the recent local best-response technique (LBR) can provide a lower bound on a strategy's exploitability (*21*) given full access to its action probabilities. LBR uses the action probabilities to compute the strategy's range at any public state. Using this range it chooses its response action from a fixed set using the assumption that no more bets will be placed for the remainder of the game. Thus it best-responds locally to the opponent's actions, providing a lower bound on their overall exploitability. As already noted, abstraction-based programs from the Annual Computer Poker Competition are highly exploitable by LBR: four times more exploitable than folding every game (Table 1). However, even under a variety of settings, LBR fails to exploit DeepStack at all — itself losing by over 350 mbb/g to DeepStack (*10*). Either a more sophisticated lookahead is required to identify DeepStack's weaknesses or it is substantially less exploitable.

Discussion

DeepStack defeated professional poker players at HUNL with statistical significance (*39*), a game that is similarly sized to go, but with the added complexity of imperfect information. It achieves this goal with little domain knowledge and no training from expert human games. The implications go beyond being a milestone for artificial intelligence. DeepStack represents a paradigm shift in approximating solutions to large, sequential imperfect information games. Abstraction and offline computation of complete strategies has been the dominant approach for almost 20 years (*33, 40, 41*). DeepStack allows computation to be focused on specific situations that arise when making decisions and the use of automatically trained value functions. These are two of the core principles that have powered successes in perfect information games, albeit conceptually simpler to implement in those settings. As a result, the gap between the largest

perfect and imperfect information games to have been mastered is mostly closed.

With many real world problems involving information asymmetry, DeepStack also has implications for seeing powerful AI applied more in settings that do not fit the perfect information assumption. The abstraction paradigm for handling imperfect information has shown promise in applications like defending strategic resources (*42*) and robust decision making as needed for medical treatment recommendations (*43*). DeepStack's continual re-solving paradigm will hopefully open up many more possibilities.

References and Notes

1. G. Tesauro, *Communications of the ACM* **38**, 58 (1995).
2. J. Schaeffer, R. Lake, P. Lu, M. Bryant, *AI Magazine* **17**, 21 (1996).
3. M. Campbell, A. J. Hoane Jr., F. Hsu, *Artificial Intelligence* **134**, 57 (2002).
4. D. Ferrucci, *IBM Journal of Research and Development* **56**, 1:1 (2012).
5. V. Mnih, *et al.*, *Nature* **518**, 529 (2015).
6. D. Silver, *et al.*, *Nature* **529**, 484 (2016).
7. A. L. Samuel, *IBM Journal of Research and Development* **3**, 210 (1959).
8. L. Kocsis, C. Szepesvári, *Proceedings of the Seventeenth European Conference on Machine Learning* (2006), pp. 282–293.
9. J. Bronowski, The ascent of man, Documentary (1973). Episode 13.
10. See Supplementary Materials.
11. In 2008, Polaris defeated a team of professional poker players in heads-up limit Texas hold'em (*44*). In 2015, Cepheus essentially solved the game (*18*).
12. V. L. Allis, Searching for solutions in games and artificial intelligence, Ph.D. thesis, University of Limburg (1994).
13. M. Johanson, Measuring the size of large no-limit poker games, *Technical Report TR13-01*, Department of Computing Science, University of Alberta (2013).
14. M. Zinkevich, M. Johanson, M. Bowling, C. Piccione, *Advances in Neural Information Processing Systems 20* (2008), pp. 905–912.
15. A. Gilpin, S. Hoda, J. Peña, T. Sandholm, *Proceedings of the Third International Workshop On Internet And Network Economics* (2007), pp. 57–69.

16. End-game solving (*17, 45, 46*) is one exception to computation occurring prior to play. When the game nears the end, a new computation is invoked over the remainder of the game. Thus, the program need not store this part of the strategy or can use a finer-grained abstraction aimed to improve the solution quality. We discuss this as re-solving when we introduce DeepStack's technique of continual re-solving.

17. N. Burch, M. Johanson, M. Bowling, *Proceedings of the Twenty-Eighth Conference on Artificial Intelligence* (2014), pp. 602–608.

18. M. Bowling, N. Burch, M. Johanson, O. Tammelin, *Science* **347**, 145 (2015).

19. We use milli-big-blinds per game (mbb/g) to measure performance in poker, where a milli-big-blind is one thousandth of the forced big blind bet amount that starts the game. This normalizes performance for the number of games played and the size of stakes. For comparison, a win rate of 50 mbb/g is considered a sizable margin by professional players and 750 mbb/g is the rate that would be lost if a player folded each game. The poker community commonly uses big blinds per one hundred games (bb/100) to measure win rates, where 10 mbb/g equals 1 bb/100.

20. J. Wood, Doug Polk and team beat Claudico to win $100,000 from Microsoft & The Rivers Casino, *Pokerfuse,* http://pokerfuse.com/news/media-and-software/26854-doug-polk-and-team-beat-claudico-win-100000-microsoft/ (2015).

21. V. Lisý, M. Bowling, *Proceedings of the AAAI-17 Workshop on Computer Poker and Imperfect Information Games* (2017). https://arxiv.org/abs/1612.07547.

22. DeepStack is not the first application of deep learning to the game of poker. Previous applications of deep learning, though, are either not known to be theoretically sound (*47*), or have only been applied in small games (*48*).

23. J. F. Nash, *Proceedings of the National Academy of Sciences* **36**, 48 (1950).

24. When the previous solution is an approximation, rather than an exact Nash equilibrium, sound re-solving needs the expected values from a best-response to the player's previous strategy. In practice, though, using expected values from the previous solution in self-play often works as well or better than best-response values (*10*).

25. A. Gilpin, T. Sandholm, T. B. Sørensen, *Proceedings of the Seventh International Conference on Autonomous Agents and Multi-Agent Systems* (2008), pp. 911–918.

26. D. Schnizlein, M. Bowling, D. Szafron, *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence* (2009), pp. 278–284.

27. A. Gilpin, T. Sandholm, T. B. Sørensen, *Proceedings of the Twenty-Second Conference on Artificial Intelligence* (2007), pp. 50–57.

28. M. Johanson, N. Burch, R. Valenzano, M. Bowling, *Proceedings of the Twelfth International Conference on Autonomous Agents and Multi-Agent Systems* (2013), pp. 271–278.

29. A. Krizhevsky, I. Sutskever, G. E. Hinton, *Advances in Neural Information Processing Systems 25* (2012), pp. 1106–1114.

30. G. Hinton, *et al.*, *IEEE Signal Processing Magazine* **29**, 82 (2012).

31. A. van den Oord, *et al.*, *CoRR* **abs/1609.03499** (2016).

32. K. He, X. Zhang, S. Ren, J. Sun, *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 1026–1034.

33. J. Shi, M. L. Littman, *Proceedings of the Second International Conference on Computers and Games* (2000), pp. 333–345.

34. D. P. Kingma, J. Ba, *Proceedings of the Third International Conference on Learning Representations* (2014).

35. P. J. Huber, *Annals of Mathematical Statistics* **35**, 73 (1964).

36. International Federation of Poker. http://pokerfed.org/about/ (Accessed January 1, 2017).

37. N. Burch, M. Schmid, M. Moravcik, M. Bowling, *Proceedings of the AAAI-17 Workshop on Computer Poker and Imperfect Information Games* (2017). http://arxiv.org/abs/1612.06915.

38. Statistical significance where noted was established using a two-tailed Student's t-test at the 0.05 level with $N \geq 3000$.

39. Subsequent to our study, the computer program Libratus, developed at CMU by Tuomas Sandholm and Noam Brown, defeated a team of four professional heads-up poker specialists in a HUNL competition held January 11-30, 2017. Libratus has been described as using "nested endgame solving" (*49*), a technique with similarities to continual re-solving, but developed independently. Libratus employs this re-solving when close to the end of the game rather than at every decision, while using an abstraction-based approach earlier in the game.

40. D. Billings, *et al.*, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* (2003), pp. 661–668.

41. T. Sandholm, *AI Magazine* **31**, 13 (2010).

42. V. Lisy, T. Davis, M. Bowling, *Proceedings of the Thirtieth Conference on Artificial Intelligence* (2016), pp. 544–550.

43. K. Chen, M. Bowling, *Advances in Neural Information Processing Systems 25* (2012), pp. 2078–2086.

44. J. Rehmeyer, N. Fox, R. Rico, *Wired* **16.12**, 186 (2008).

45. S. Ganzfried, T. Sandholm, *Proceedings of the Fourteenth International Conference on Autonomous Agents and Multi-Agent Systems* (2015), pp. 37–45.

46. M. Moravcik, M. Schmid, K. Ha, M. Hladík, S. J. Gaukrodger, *Proceedings of the Thirtieth Conference on Artificial Intelligence* (2016), pp. 572–578.

47. N. Yakovenko, L. Cao, C. Raffel, J. Fan, *Proceedings of the Thirtieth Conference on Artificial Intelligence* (2016), pp. 360–367.

48. J. Heinrich, D. Silver, *arXiv preprint arXiv:1603.01121* (2016).

49. N. Brown, T. Sandholm, *Proceedings of the AAAI-17 Workshop on Computer Poker and Imperfect Information Games* (2017).

50. M. Zinkevich, M. Littman, *Journal of the International Computer Games Association* **29**, 166 (2006). News item.

51. D. Morrill, Annual computer poker competition poker GUI client, https://github.com/dmorrill10/acpc_poker_gui_client/tree/v1.2 (2012).

52. O. Tammelin, N. Burch, M. Johanson, M. Bowling, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence* (2015), pp. 645–652.

53. R. Collobert, K. Kavukcuoglu, C. Farabet, *BigLearn, NIPS Workshop* (2011).

54. S. Ganzfried, T. Sandholm, *Proceedings of the Twenty-Eighth Conference on Artificial Intelligence* (2014), pp. 682–690.

Acknowledgements. The hand histories of all games played in the human study as well as those used to generate the LBR results against DeepStack are in (*10*). We would like to thank the IFP, all of the professional players who committed valuable time to play against DeepStack, the anonymous reviewers' invaluable feedback and suggestions, and R. Holte, A. Brown, and K. Blažková for comments on early drafts of this article. We especially would like to thank IBM for their support of this research through an IBM faculty grant. The research was also supported by Alberta Innovates through the Alberta Machine Intelligence Institute, the Natural Sciences and Engineering Research Council of Canada, and Charles University (GAUK) Grant no. 391715. This work was only possible thanks to computing resources provided by Compute Canada and Calcul Québec. MM and MS are on leave from IBM Prague. MJ serves as a Research Scientist, and MB as a Contributing Brain Trust Member, at Cogitai, Inc. DM owns 200 shares of Gamehost Inc.

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.

© 2016 Elsevier B.V. All rights reserved.

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/© 2016 Elsevier B.V. All rights reserved.

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.

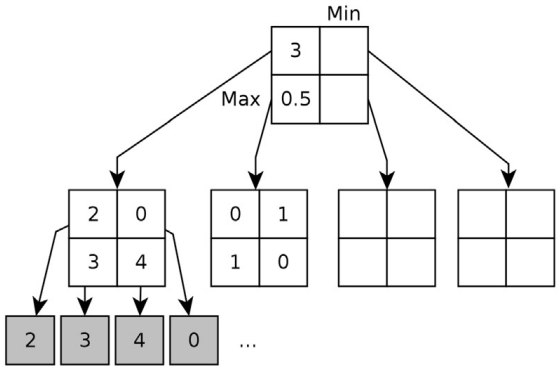


Fig. 1. An example of a two-player simultaneous move game. Each white matrix corresponds to a state of the game where both players (a maximizing player with actions in rows and a minimizing player with actions in columns) act simultaneously. The dark squares are terminal states. The values shown in the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

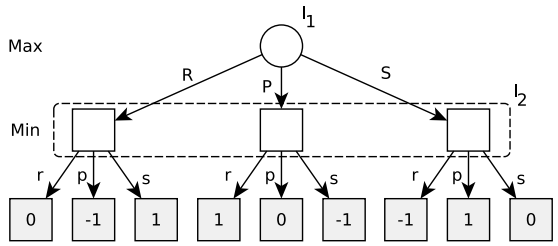


Fig. 3. The matrix game of Rock, Paper, Scissors (left) and its equivalent extensive-form game representation (right). The extensive game has four states, two information sets (I_1 and I_2), and nine terminal histories: $\{Rr, Rp, Rs, Pr, Pp, Ps, Sr, Sp, Ss\}$.

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

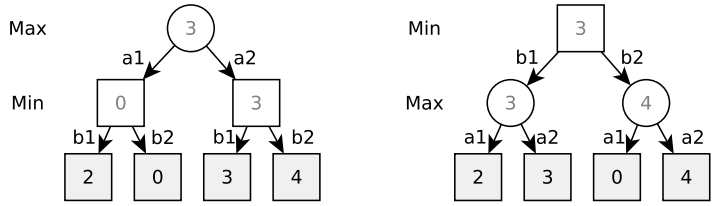


Fig. 4. Different serializations of a simple simultaneous move game. Utility values are in the leaf nodes, the gray values correspond to the value propagation when solving the serialized game.

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.

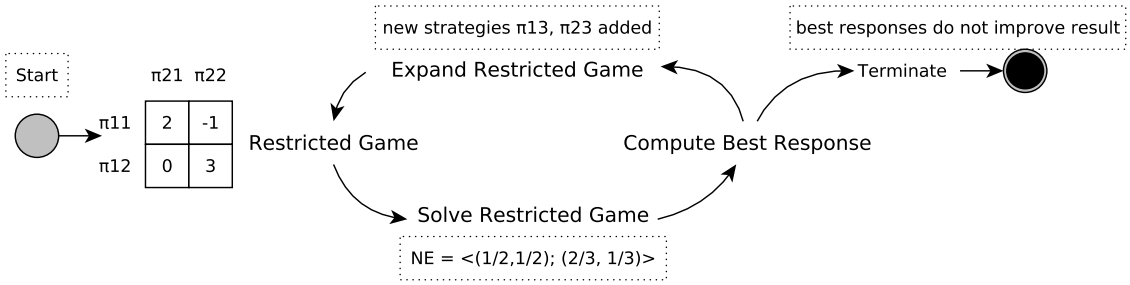


Fig. 5. Schematic of the double-oracle algorithm for a normal-form game.

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

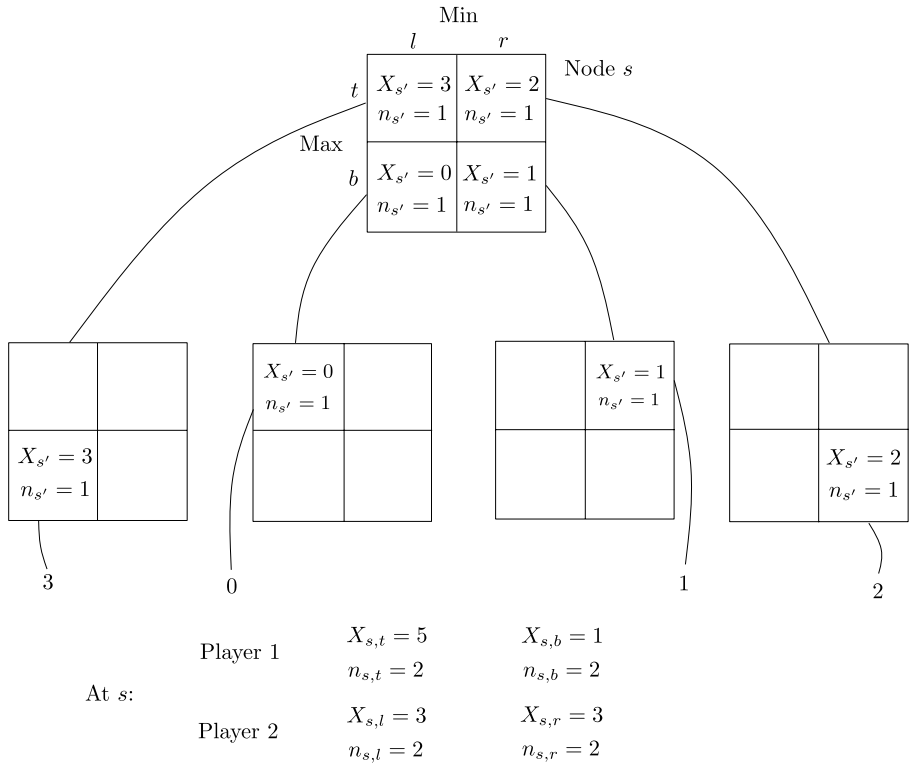


Fig. 6. Simultaneous Move MCTS example. Here, $X_{s'}$ represents the cumulative payoff of all simulations that have passed through the cell, while $n_{s'}$ represents the number of simulations that have passed through the cell.

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).

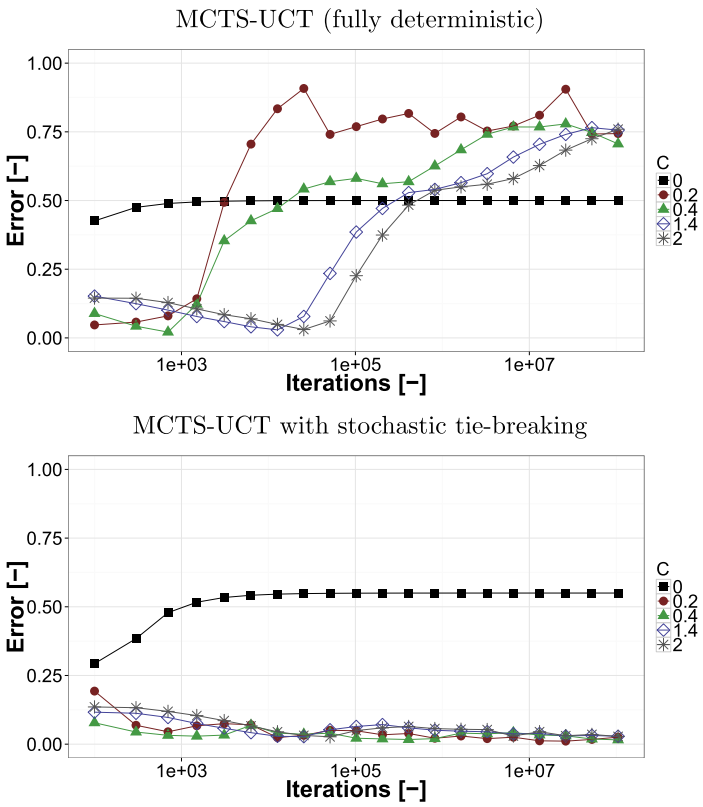


Fig. 8. Exploitability of strategies of recommended by MCTS-UCT over time in Biased Rock, Paper, Scissors. Vertical axis represents exploitability.

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

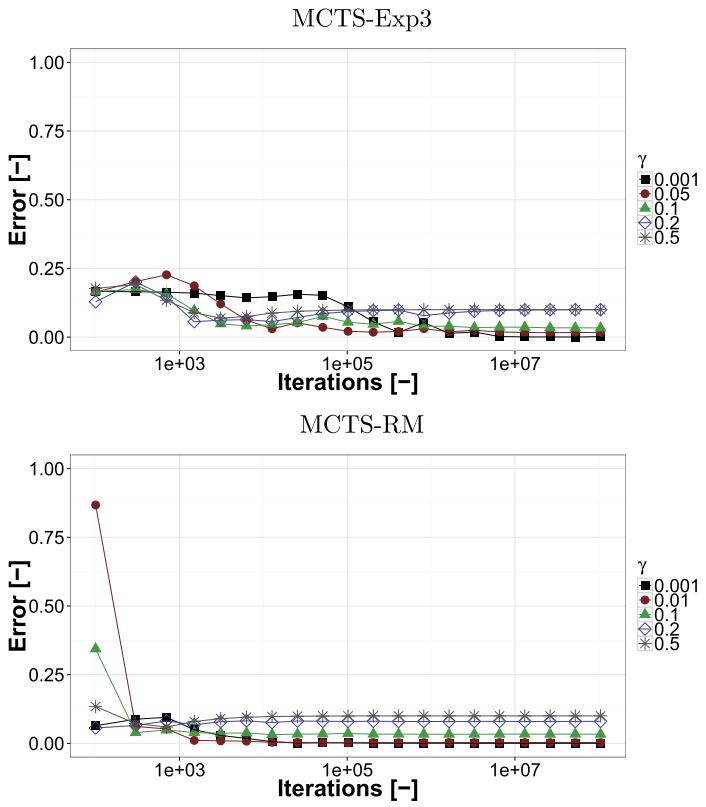


Fig. 9. Exploitability of strategies recommended by MCTS-Exp3 and MCTS-RM over time in Biased Rock, Paper, Scissors. Vertical axis represents exploitability.

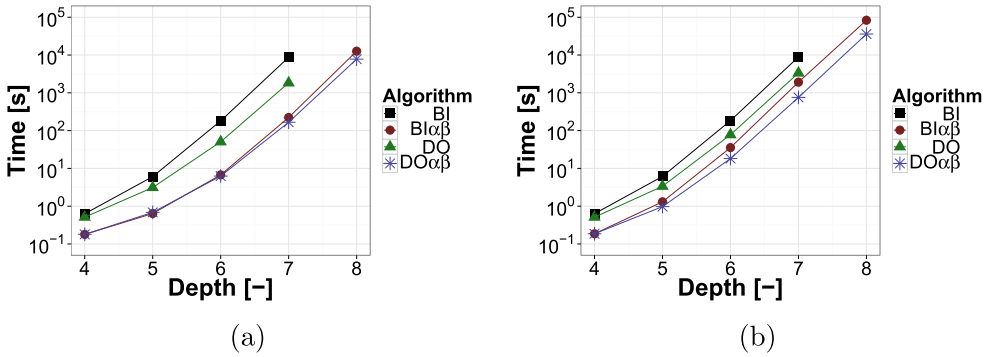


Fig. 10. Running times of the exact algorithms on Goofspiel with fixed sequences of cards for increasing size of the deck; subfigure (a) depicts the results with win–tie–lose utilities, (b) depicts the results with point difference utilities.

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.

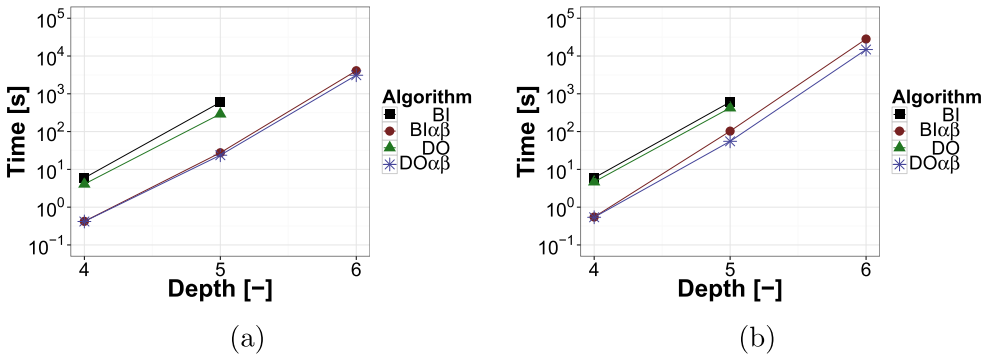


Fig. 11. Running times of exact algorithms on Goofspiel with chance nodes for increasing size of the deck; subfigure (a) depicts the results with win–tie–lose utilities, (b) depicts the results with point utilities.

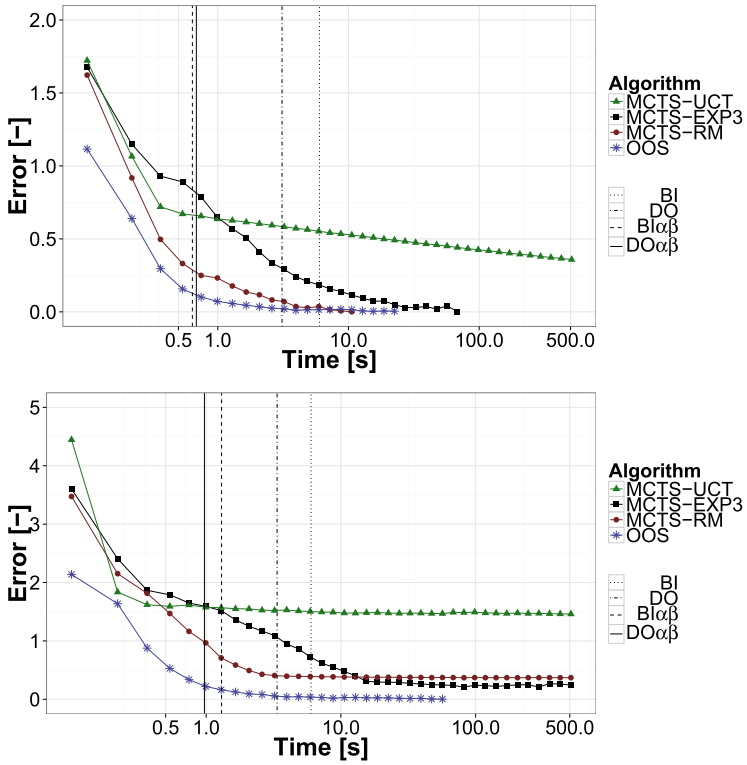


Fig. 12. Convergence of the sampling algorithms on Goofspiel with 5 cards and a fixed sequence of cards. The vertical lines correspond to the computation times for the exact algorithms. (Top) Goofspiel with win–tie–lose utility values; (bottom) Goofspiel with point utilities.

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

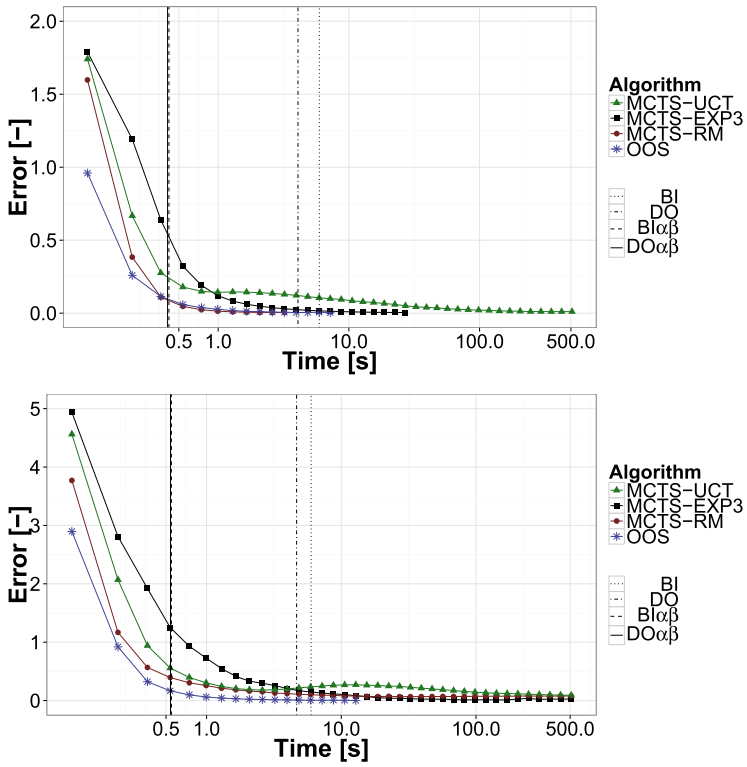


Fig. 13. Convergence of the sampling algorithms on Goofspiel with 4 cards and chance nodes. The vertical lines correspond to the computation times for the exact algorithms. (Top) Goofspiel with win–tie–lose utility values; (bottom) Goofspiel with point utilities.

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.

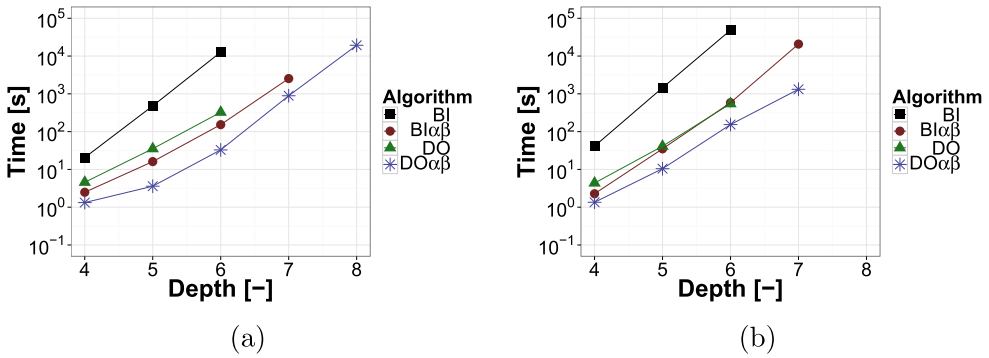


Fig. 14. Running times of exact algorithms on pursuit–evasion games with an increasing number of moves: subfigure (a) depicts the results on 4×4 grid graph, (b) depicts results for 5×5 grid.

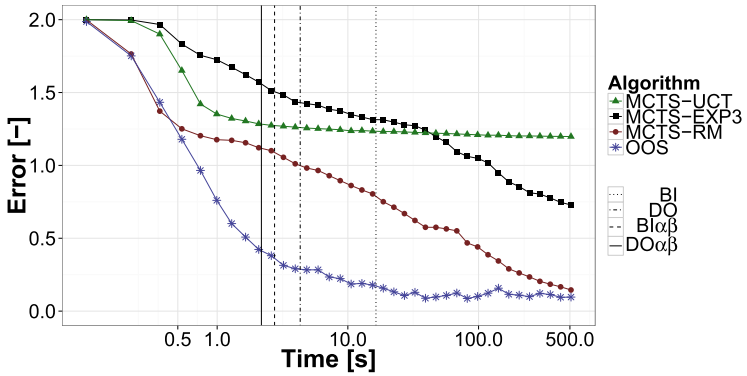


Fig. 15. Convergence of the sampling algorithms on a pursuit–evasion game, on a 4×4 graph, with depth set to 4. The vertical lines correspond to the computation times for the exact algorithms.

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

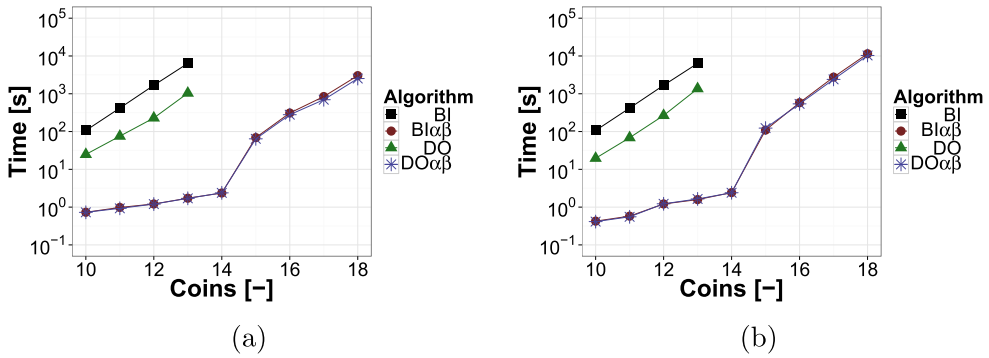


Fig. 16. Running times of the exact algorithms on Oshi-Zumo with K set to 4 and an increasing number of coins: subfigure (a) depicts the results for binary utilities, (b) depicts the results with point utilities.

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 $K = 4$.

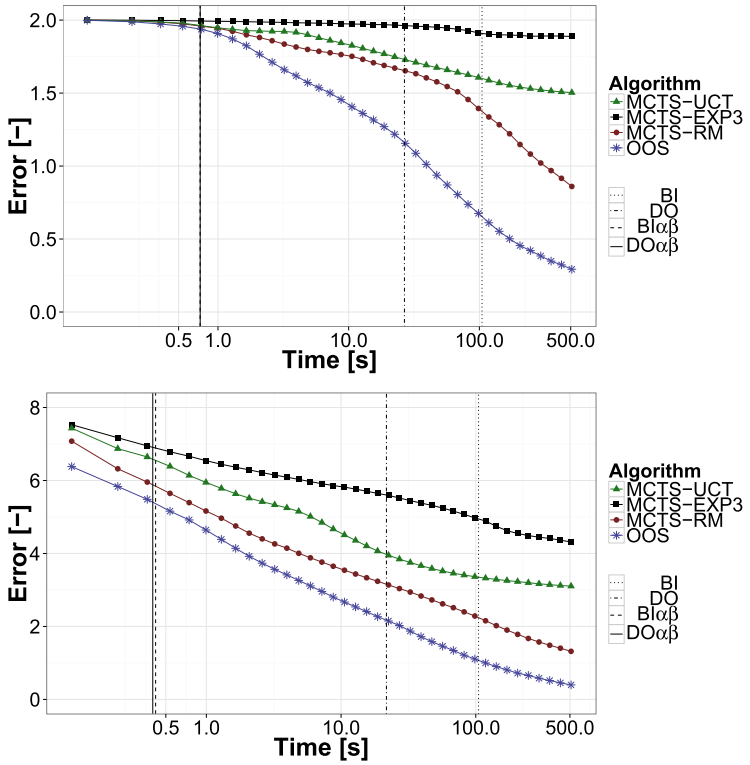


Fig. 17. Convergence of the sampling algorithms on Oshi-Zumo game with 10 coins, $K = 3$, and $M = 1$. The vertical lines correspond to the computation times for the exact algorithms. (Top) Oshi-Zumo with win–tie–lose utility values; (bottom) Oshi-Zumo with point utilities.

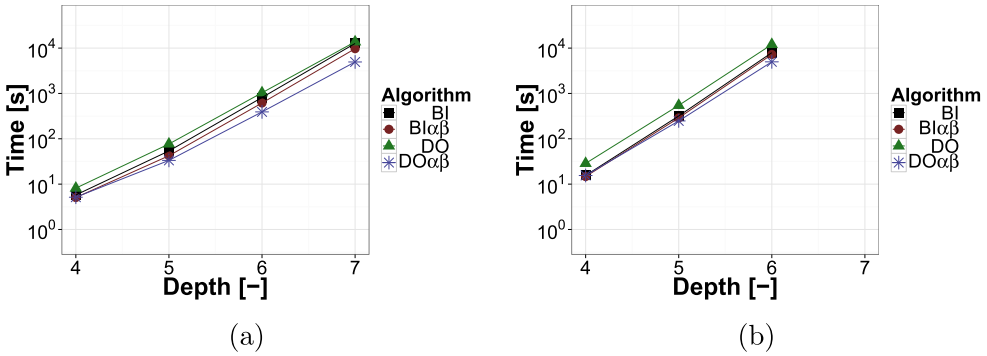


Fig. 18. Running times of the exact algorithms on randomly generated games with increasing depth: subfigure (a) depicts the results with branching factor set to 4 actions for each player, (b) depicts the results with branching factor 5.

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.

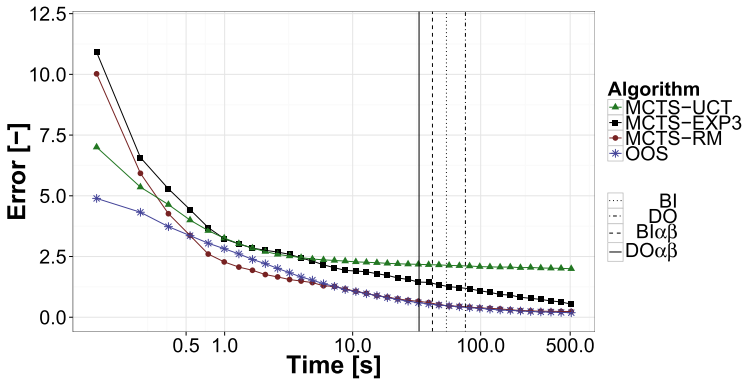


Fig. 19. Convergence of the sampling algorithms on a random game with branching factor 4 and depth 5. The vertical lines correspond to the computation times for the exact algorithms.

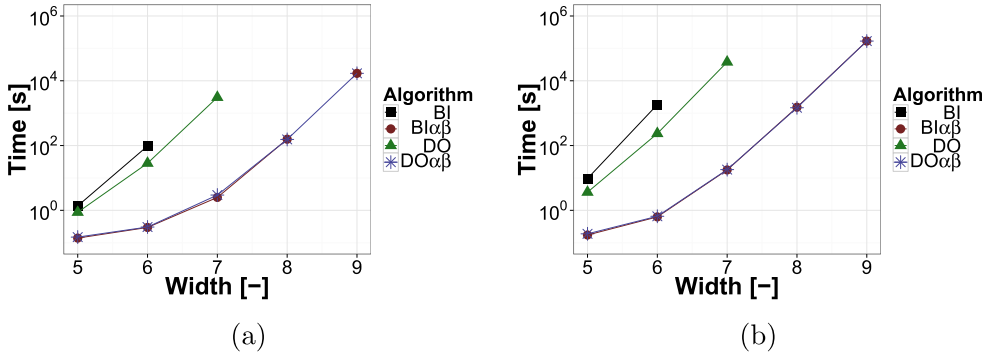


Fig. 20. Running times of the exact algorithms on Tron with increasing *width* of the grid graph: subfigure (a) depicts the results with *height* of the graph set to *width* − 1, (b) depicts the results with *height* = *width*.

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.

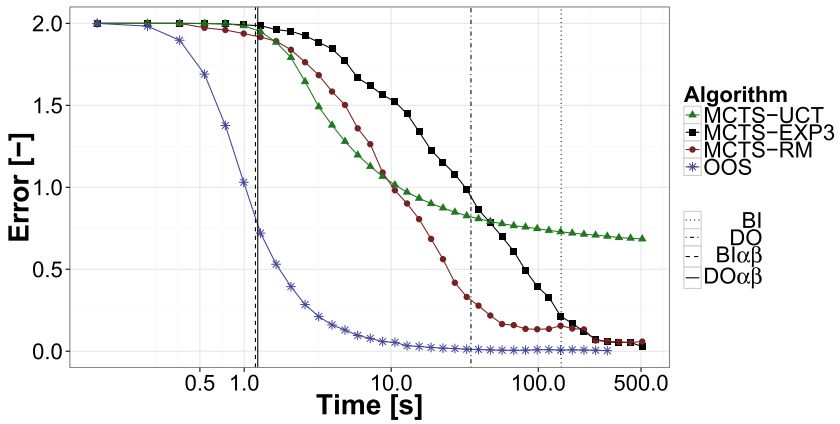


Fig. 21. Convergence comparison of different sampling algorithms on Tron on grid 5×6. The vertical lines correspond to the computation times for the exact algorithms.

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

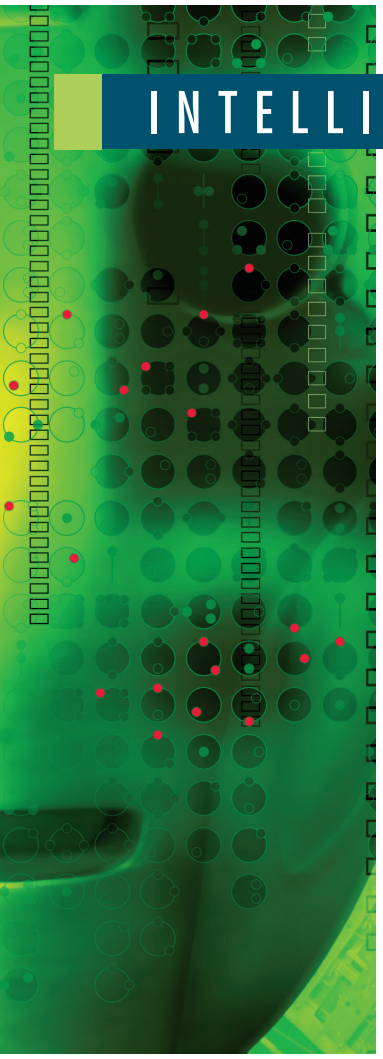
This work is funded by the Czech Science Foundation (grant Nos. P202/12/2054 and 15-23235S) and the Netherlands Organisation for Scientific Research (NWO) in the framework of the project Go4Nature, grant number 612.000.938. Branislav Bošanský also acknowledges support from the Danish National Research 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.



Case Studies of Network Defense with Attack Graph Games

Karel Durkota, *Czech Technical University*

Viliam Lisý, *Czech Technical University and University of Alberta*

Christopher Kiekintveld, *University of Texas at El Paso*

Branislav Bošanský and Michal Pěchouček, *Czech Technical University*

A challenge many existing tools fail to address is that attackers react strategically to new security measures, adapting their behaviors in response. Game theory provides a methodology for making decisions that take into account these reactions.

Computer network security is an example of asymmetric, strategic conflict between defenders and attackers, with attackers performing a wide range of intrusion actions such as scanning the network and exploiting vulnerabilities, and defenders countering with actions such as intrusion detection and filtering. Different defense mechanisms have varying costs and effectiveness against specific types of attacks, forcing network administrators to make challenging decisions about how best to optimize the ways in which they select and deploy these measures. Initial effectiveness can be mitigated in the long term as attackers adapt to security measures, making optimization even more challenging.

Game theory provides a methodology for developing new decision support tools that takes into account attackers' sophisticated responses to defense strategies. The key idea is that rational attackers will respond to security, so we can model the network defense problem as a two-player, multistage game. Solving these models lets us find an optimal defense plan to mitigate attacks and can also provide a quantitative measure of security improvement. We demonstrate this methodology for one specific type of defensive strategy: honeypots, which are fake hosts or services added to a network. Honeypots act as decoys to distract attackers from real hosts, detect the presence of intruders, and gather detailed information about an attacker's activity.

Operating believable honeypots is expensive in terms of hardware, software, and administrator time (typically spent managing the honeypot and analyzing data). In addition, many different types of honeypots could be used in any given network. We describe a game-theoretic approach for optimizing honeypot deployments as a case study for how game theory can be used to make network security decisions. Our case study on a realistic network shows the feasibility of this methodology.

1541-1672/16/$33.00 © 2016 IEEE
Published by the IEEE Computer Society

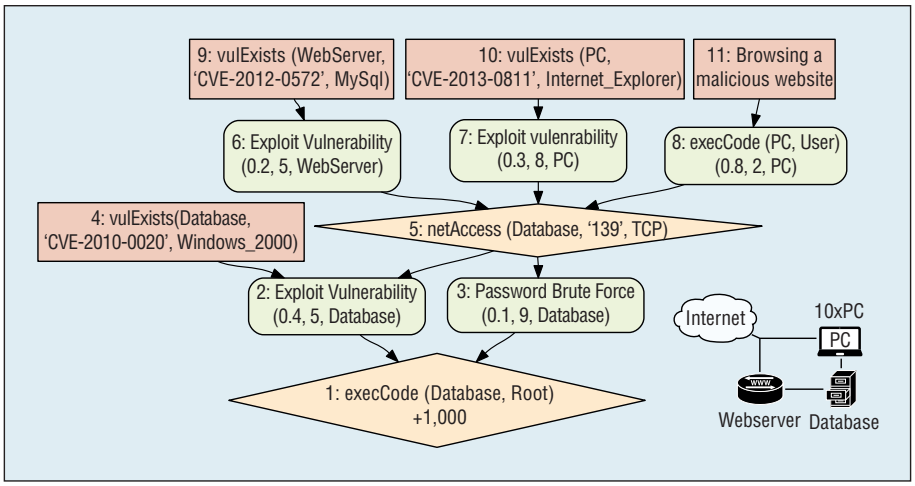


Figure 1. A network and an attack graph representing ways to gain access to the database. Each action (rounded rectangular node) has preconditions and effects, represented by incoming and outgoing edges, respectively. Initially true facts are represented with rectangle nodes and initially false facts with diamond nodes.

Background

Network administrators and security researchers use honeypots to detect and analyze attacker behaviors and tools.[1] In particular, the Honeynet Project provides a wealth of software and literature on knowledge acquired about attackers. Companies use honeypots as intrusion detection systems (IDSs)—hardware products (for example, Canary by Thinkst) that can emulate various OSs. To effectively use honeypots, network administrators must decide how many to deploy (contingent on cost) and where to place them to make them attractive targets.

Game Theory

Game theory models decision-making problems with multiple decision makers (players) in a common, often partially observable, environment. A game consists of players, strategies (actions) available to each player, and utilities for each defender depending on the joint choices of all players; players can also have incomplete information about moves made by other players or the environment. The optimal strategy for a player generally depends on other players' behavior.

Game theory provides a variety of solution concepts and algorithms for analyzing games with different characteristics. We focus on two-player games where the administrator (defender) chooses a honeypot allocation that minimizes the cost and expected loss caused by an attacker compromising the network. The attacker chooses a strategy that maximizes the value of attacking the network while avoiding honeypots and minimizing costs. We use Stackelberg game models similar to those used for physical security domains.[2] The defender acts first, taking actions to harden the network by adding honeypots. The attacker then chooses the optimal attack plan based on (limited) knowledge about the network and the defender's strategy. Solving the game means computing a strategy for each player, which describes an optimal (stochastic) action choice in every possible situation.

Attack Graph

Attack graphs (AGs) represent possible sequential attacker strategies for compromising a specific computer network. They're automatically generated based on known vulnerabilities[3] and are used to identify the minimal subset of vulnerabilities to be patched or sensors to be placed in the network to prevent known attacks, to calculate security risk measures,[4] or to find the shortest attack plan in penetration testing. We use AGs to compactly represent possible attack plans (attacker strategies) in our game models.

Model Overview

We model a network as a set of host types, such as the webserver or PC in Figure 1. Two hosts are of the same host type if they run the same services and have the same network connectivity and value. For example, a collection of workstations that run the same OS image are modeled as the same type. The host-type PC in Figure 1 has 10 equivalent hosts, all of which present the same attack opportunities. By representing each type only once in the attack graph, we can scale with the number of unique types rather than of individual hosts.

Defending

Let's say the defender places k honeypots in the network by duplicating the configurations of existing hosts (with obfuscated data) or creating new host types. The defender pays a cost that's dependent on the host type for each honeypot, so adding more honeypots of a specific host type increases the likelihood that the attacker will interact with a honeypot instead of a real host. If the attacker interacts with a honeypot during an attack, an alert is sent to the defender, who immediately stops the attack or applies other countermeasures.

Attacking

In our model, we consider exploit actions that target a specific vulnerability in a host. Successfully executing an exploit results in the attacker gaining privileged or nonprivileged access to that host.

An AG compactly represents all known sequences of exploit actions for the host in the network. Specifically, it captures the dependencies between the exploit actions and true or

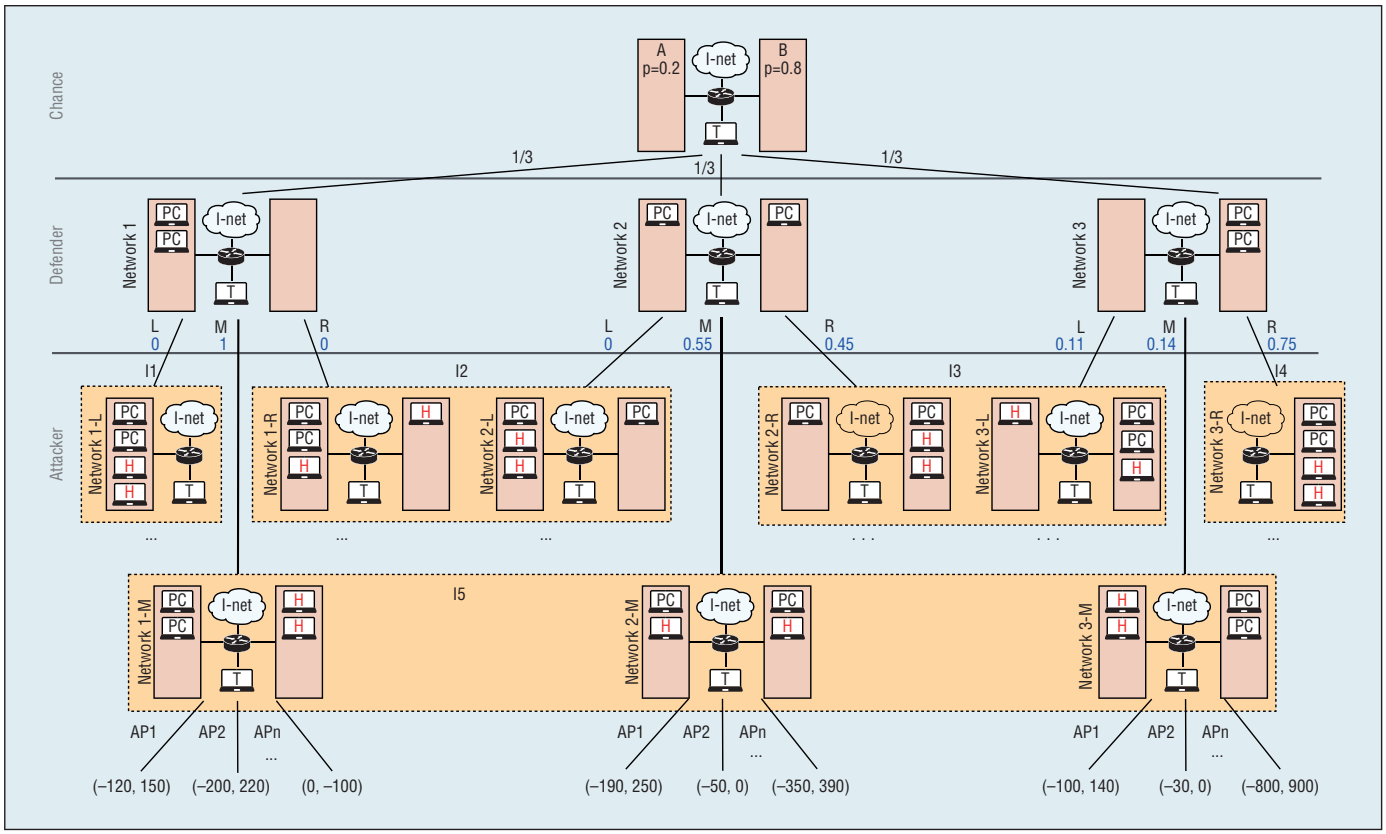


Figure 2. Game tree for a simple network of host types A and B, with attack success probabilities 0.2 and 0.8, respectively. Chance plays uniformly into three possible networks, each containing two hosts. The defender chooses possible combinations of allocating two honeypots in each possible network; the blue probability value is a possible defense strategy. The attacker selects attack policy AP_1 through AP_n according to the networks' attack graphs.

false facts that represent logical statement about the network state. Figure 1 depicts a possible AG for the network; each action (rounded rectangular node) has preconditions and effects, represented by incoming and outgoing edges, respectively. All preconditions must be true to perform the action. If an exploit action succeeds, then all its effects become true and the attacker obtains rewards. Initially true facts are represented with rectangle nodes and initially false facts with diamond nodes.

In Figure 1, action "`2:Exploit Vulnerability`" has preconditions: Fact 4, vulnerability exists, and Fact 5, the attacker can access the daase on port 139. If the attacker successfully performs the action, he will obtain root privileges to the database (Fact 1) and reward +1,000. The success probability is the likelihood that an exploit will succeed, given that all preconditions are met, and the cost represents the attacker's monetary cost and effort for attempting to perform an action, as well as the consequences of possibly disclosing the exploit. Action 2 in Figure 1 has a success probability of 0.4 and a cost of 5.

We use MulVAL[3] to automatically construct AGs from information collected by network scanning tools (such as Nessus or OpenVAS). The action costs and success probabilities can be estimated using the Common Vulnerability Scoring System[5] or other data sources.

In our game models, the attacker chooses the optimal attack policy that fully characterizes the attacker's behavior at every point in an attack—it specifies the order of the actions to be executed by a rational attacker. The problem of computing the optimal attack policy can be represented as a finite horizon Markov decision process (MDP). We use domain-specific MDP algorithms to find optimal strategies.[6]

Game Model

Honeypots in network security are a form of a deception, and we can model deception in games where one player has more information about the game state than the other (such as the network structure or honeypot locations). Predicting player behavior is more difficult in these games because it depends on players' beliefs about the likelihood of possible states. We model the honeypot allocation problem as a chance player who makes an initial random move with a probability corresponding to the attacker's belief of each network's likelihood, as shown in Figure 2. The attacker is uncertain which of the possible extended networks (networks after the chance move) is the defender's real network before the defender added honeypots.

We assume that the attacker's belief is common knowledge (if not, the defender can approximate it by analyzing real-world network frequency). In our example, the core network (at the chance layer) consists of a gateway router and a target host T. We model an attacker who knows that there are two additional hosts, either of host type A or B, with equal probability. Therefore, the chance player extends the core network by adding possible combinations of host types A and B, each with a probability of 1/3 (known to both players), which leads to Networks 1 through 3.

The defender then decides which honeypot host types to add to each extended network. Although the defender knows the actual network, the strategy specifies the honeypots to add to every possible network. We assume that the attacker knows that the defender can add up to k honeypots, therefore, he or she must reason about what the defender would do in each possible situation. This creates an interesting information structure. If the defender adds two honeypots to A in Network 1, which results in Network 1-L, the attacker can be certain that two out of four hosts in A are honeypots. However, if the defender adds one honeypot to A and one to B in Network 1 and two honeypots to A in Network 2, the resulting Networks 1-R and 2-L look exactly the same to the attacker. When the attacker observes this network, he or she can't be certain if the host in B is a honeypot or not. An information set is the set of networks that look the same to the attacker; in Figure 2, they're denoted by dashed rectangles I1 through I5. The attacker must play the same strategy for all networks in an information set because they're indistinguishable. However, in each information set, the attacker can have different attack plans to choose from (for example, in Network 1-L, only host type A can be attacked, while in Network 3-R, only host type B). The defender controls the attacker's observations about the network to a large extent, leading to the potential for deception and a complex decision problem for the attacker.

We assume that the honeypot duplicate is indistinguishable from the original host to the attacker. If the attacker goes after host type A in Network 1-L, he or she has a 50 percent chance of attacking the honeypot. Therefore, the attacker must weigh the probability of being detected against the expected benefit of the successful attack and the cost of alternative attacks. Each attack policy results in a potentially different pair of utility values specified in the leaves of the game tree (AP_1 through AP_n in I5 in Figure 2). The first value is the defender's utility, which contains honeypot costs and expected losses from attacks. The second value is the attacker's utility, which contains the expected reward, cost, and penalty for being detected.

Figure 2 shows the defender's strategy OPT_d, with probabilities highlighted in blue. The strategy prescribes the probability of playing each action in that network. The defender can influence the attacker's probabilities of observing networks—typically, the best strategies make the attacker indifferent about which host type to attack first, which in turn leads to the least effective attacks. For example, if the defender plays the OPT_d strategy, then the attacker who observes a network in I3 faces with probability $0.45/(0.45 + 0.11) \approx 0.8$ Network 2-R and with probability 0.2 Network 3-L. Because attacks are four times more likely to succeed at host type B than at A, the defender prefers adding a honeypot to B four times more likely than to A.

Computing the defender's strategy is difficult because the number of possible defender actions grows combinatorically with the number of honeypots and host types. Computing the attacker's optimal attack plan is an NP-hard problem, and computing the defender's strategy in Stackelberg equilibrium is an NP-hard problem for imperfect information games. We describe the game model in more detail along with several approximation algorithms elsewhere.[7]

Case Study I: TV Company

We now present a case study to demonstrate how we can use this framework to model a real network and feasibly compute optimal honeypot allocations. The optimal strategies for this network incorporate deception, with the defender exploiting the attacker's uncertainty about the network.

Domain Description

The case study is based on a network that the Swedish Defense Research Agency used during its cybersecurity exercises in 2012,[8] in which networks were deliberately left vulnerable to attacks. Here, we simplify the original network by representing complete subnetworks as single host types and reducing the number of vulnerabilities to three PCs and routers. The resulting attack graph has 61 actions and 102 facts.

The attacker can initiate an attack on any PC (via a malicious website visited by a user on that host) except the target studio host (T in Figure 3). The routers can't be attacked remotely, only locally. The attacker aims to gain root privileges to the studio host, which has a value of 1,000 for both players. Exploit actions have costs between 5 and 10 (7.5 on average), and success probabilities are estimated by MulVAL (0.7 on average). The attacker has a penalty of 200 if detected, and the defender's cost for creating a complex honeypot mimicking studio is 130, while any other PC costs 30.

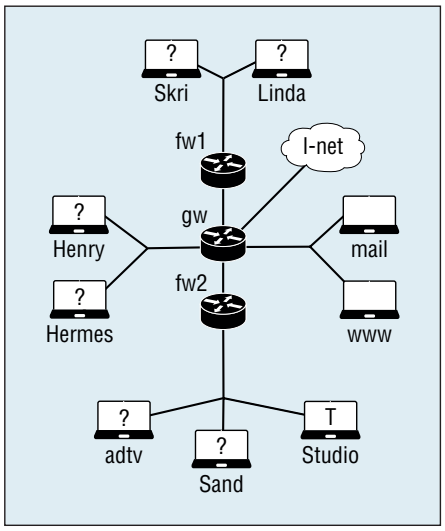


Figure 3. Network for TV company case study. The attacker knows about all routers, host mail, www, and studio (target) in the network, but he's uncertain if hosts with a question mark are present in the network. Therefore, he must consider all possible combinations and likelihood of their occurrence, to compute optimal attack strategies.

Game Analysis

The attacker knows the core network (nodes without question marks in Figure 3). Although uncertain about the hosts with question marks, the attacker knows that the attacked network contains two of the hosts with a question mark, just not which ones exactly. The defender knows his or her network, which consists of the core network and PC host types adtv and Linda. We refer to this as the real network.

Solving the game means finding the defender's optimal strategy for honeypot allocation in every possible network instance that can occur (after the chance move). The defender adds two honeypots of the studio host type to secure the target host, despite their high cost. A more interesting strategy occurs with three honeypots, where the attacker reasons about $\binom{7}{3} = 35$ possible networks and computes attack plans. Because adtv appeals to the attacker, the defender's strategy recommends adding adtv as a honeypot in networks where adtv doesn't exist (that is, in hypothetical networks other than the real one). This makes the attacker cautious about attacking adtv. The defender of the real network (with Linda and adtv) allocates honeypots

- with probability 0.75: π_1 = {fw2, adtv, studio}, and
- with probability 0.25: π_2 = {fw2, gw, Skri},

with expected utility –412, which leads to a counterintuitive strategy for the rational attacker, who attacks adtv only when two adtv hosts are present, one real and one honeypot (after action π_1). Although there's a probability of 0.5 that the attacker will interact with a honeypot, it's worth the risk. Counterintuitively, when the defender plays π_2, in which case adtv could be attacked without interacting with a honeypot, the attacker reasons that it's "too good to be true" and instead attacks host www. When the attacker observes a single adtv host, the host is more likely to be a honeypot (with a probability of 0.65) than a real host: in alternative networks, adtv is often added as a honeypot.

With three honeypots, the administrator can exploit the attacker's uncertainty by playing mixed strategies, thus reducing total costs. With more honeypots, the optimal strategies become difficult to comprehend in full detail, let alone calculated by hand. Therefore, we argue that a decision support system of this kind is valuable to administrators.

Case Study II: Human Strategies

We conducted a survey to compare the performance of game-theoretic solutions to human opponents. The 45 respondents who participated in the survey were either the attendees of a four-day forensic malware seminar, members of a multiagent technology group at Czech Technical University in Prague, or employees of computer security companies. The survey was presented as a competition to motivate respondents to create effective strategies.

To avoid overwhelming participants with too much information, we used the simple networks in Figure 2 and set all honeypot costs and attack action costs to zero. We kept the reward of 1,000 for target host T and a penalty of 200 for the attacker if detected, and the exploit success probabilities at 0.2 and 0.8 for host types A and B, respectively. Attacking routers and host type T is always successful, but the attacker can initiate attacks only from host types A or B.

Respondent Behaviors

Our analysis of 45 collected responses revealed five main clusters in the respondents' defense strategies. We compared this clustering to 500 uniformly random datasets using five standard quality metrics. The quality of our clustering was better than the 95th percentile, indicating that the clusters aren't likely to appear by chance. A strategy belongs to a cluster if its L1 distance from a hand-selected centroid strategy is less than 1/3.

For each cluster, we present the percentage of strategies in that cluster and the defender's average expected utility (u_d) of the strategies against a worst-case attacker:

- Optimal strategy (OPT_d), with 15 percent, $u_d = -236 \pm 23$, is a cluster around the game-theoretic solution.
- Perfect information (PI_d), with 26 percent, $u_d = -267 \pm 21$, defends each network in isolation by playing into I1, I4, and I5; it doesn't exploit the attacker's uncertainty.
- Single information set (SIS_d), with 11 percent, $u_d = -343 \pm 16$, always plays to I5.
- Biased uniform (BU_d), with 26 percent, $u_d = -292 \pm 22$, mostly allocates one honeypot to each side, with probabilities 0.1 and 0.2 for both honeypots to A and B, respectively.

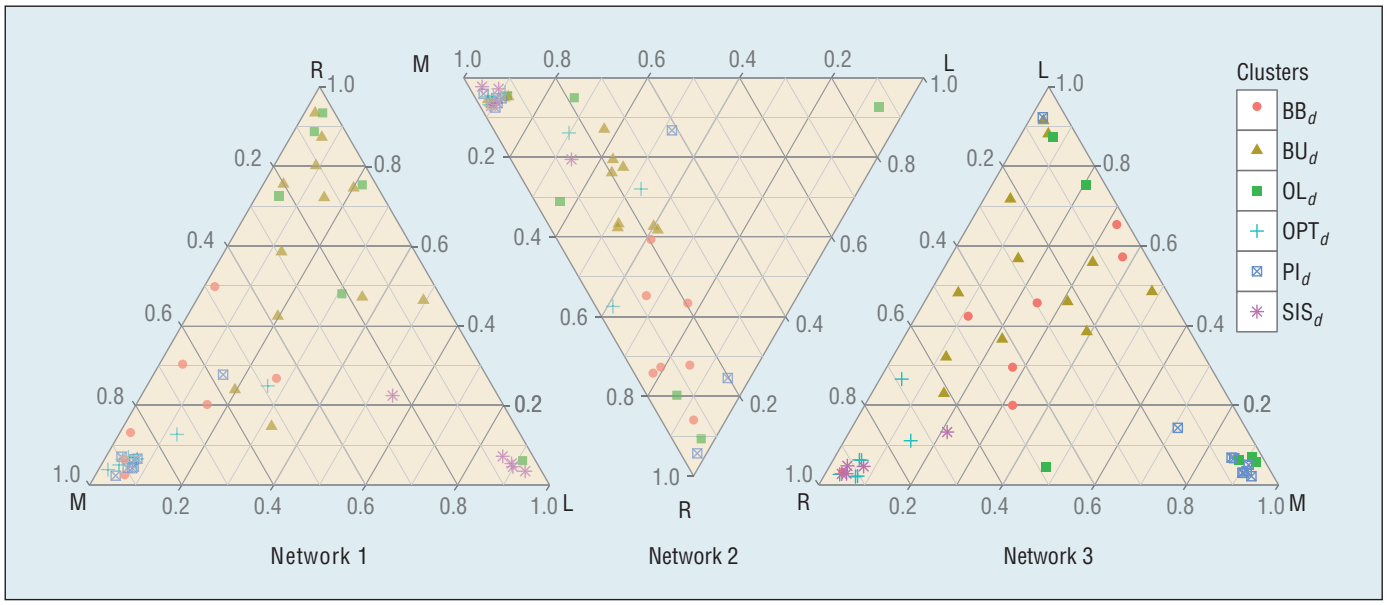


Figure 4. Respondents' defense strategies. Each triangle represents the space of possible defender strategies for the network in Figure 2 (each corner is a pure strategy).

Table 1. Defender's utility, standard deviation, 10th and 90th percentile for the defender's strategies (rows) and attacker's strategies (columns).*

Defense/attack	Best response (BR$_a$)			Mean respondent (MR$_a$)		
	μ	σ	10th, 90th percentile	μ	σ	10th, 90th percentile
Optimal (OPT$_d$)	–207	131	–350, –50	–189	181	–350, 100
Respondent (R$_d$)	–285	236	–500, 100	–167	208	–360, 100
Mean respondent (MR$_d$)	–262	266	–500, 100	–164	207	–360, 100
Baseline uniform	–317	322	–800, 100	–162	240	–500, 100
Best response (BR$_d$)	–302	187	–500, –50	–111	200	–500, 100

* BR$_a$ is the attacker's best-response strategy to a defense strategy; MR$_d$ and MR$_a$ are mean respondent defense and mean respondent attack strategies, respectively.

- Both to B (BB$_d$), with 15 percent, $u_d = -255 \pm 24$, protects the more vulnerable host type B with two honeypots.
- Outliers (OL$_d$), with 13 percent, $u_d = -334 \pm 111$, are defense strategies with a distance larger than 1/3 to any centroid strategy.

Figure 4 illustrates the individual defense strategies. Each triangle represents one possible network, and the points represent the normalized probability distributions over pure strategies (each corner represents a pure strategy).

Survey Analysis

Table 1 summarizes the strategy analysis. Each entry contains the defender's mean utility μ, standard deviation σ, and 10th (or 90th) percentile from 10^6 simulations for a pair of a defense strategy (row) against an attack strategy (column).

The OPT$_d$ strategy maximizes the defender's utility against the worst-case attacker. Column BR$_a$ is the best-response (worst-case) attack strategy against each defense strategy, which shows how exploitable a defense strategy is. R$_d$ represents the respondents' defense strategies in our dataset. We generated a single "mean" respondent strategy by averaging the individual strategies, labeled MR$_d$. By comparing these strategies to the baseline uniform strategy, we see that respondents' play gave better results than the random baseline.

The respondents' mean attack strategy MR$_a$ reveals some interesting observations. The MR$_d$ against MR$_a$ has a lower mean utility than the optimal strategy OPT$_d$ against MR$_a$, indicating that against human attackers, it might be a better defense over OPT$_d$. However, MR$_d$ might not be a good strategy from a long-term perspective. Attackers will likely learn from experience and move toward the best-response BR$_a$ by increasing the frequency of successful attacks and reducing failed ones.[8] Instead of playing OPT$_d$, it might be better to begin with the MR$_d$ strategy, but switch to OPT$_d$ as attackers start adapting.

We can develop an even better defense strategy than OPT$_d$ or MR$_d$ that exploits human behavior. BR$_d$ is the

THE AUTHORS

Karel Durkota is a PhD student and research fellow in the Artificial Intelligence Center at Czech Technical University. His research interests include the defender's decision-making problem using game theory in a complex system. Durkota received an MS in artificial intelligence from Czech Technical University. Contact him at karel.durkota@agents.fel.cvut.cz.

Viliam Lisý is a research fellow in the Artificial Intelligence Center at Czech Technical University, where he received his PhD. He's currently a postdoc at the University of Alberta. His research focuses on algorithmic and computational game theory, computing strategies in sequential games, and applications to security. Contact him at viliam.lisy@agents.fel.cvut.cz.

Christopher Kiekintveld is an assistant professor at the University of Texas at El Paso. His research interests include intelligent systems, focusing on multiagent systems and computational decision making. Kiekintveld received a PhD in strategic reasoning from the University of Michigan. He has received several best paper awards, the David Rist Prize, and an NSF CAREER award. Contact him at cdkiekintveld@utep.edu.

Branislav Bošanský is an assistant professor in the Department of Computer Science at Czech Technical University. His research interests include algorithmic and computational game theory, computing strategies in sequential games, and applications to security. Bošanský received a PhD in artificial intelligence from Czech Technical University. Contact him at branislav.bosansky@agents.fel.cvut.cz.

Michal Pěchouček is a full professor at Czech Technical University, where he also heads the Artificial Intelligence Center. His research interests include cyber security, multiagent simulation, and planning. Pěchouček received a PhD in artificial intelligence and biocybernetics from Czech Technical University. Contact him at michal.pechoucek@agents.fel.cvut.cz.

defender's best-response defense strategy against the human MR_d attack strategy, resulting in the highest utility for the defender. However, there's added risk because it's vulnerable against BR_a attackers who specifically exploit this strategy.

Our strategies have a large σ due to two factors. First, networks have different levels of security (Network 3 is more vulnerable than Network 1), and more vulnerable networks will naturally have lower utility. For example, with strategy OPT_d, the administrator of Networks 1, 2, and 3 has an expected utility –23, –247, and –350, respectively. Second, randomized strategies are necessary to minimize the strategy's predictability, but they also result in variability in outcomes. The OPT_d strategy has the lowest standard deviation against both types of attackers, which can be a desirable property.

Our results show strengths and weaknesses in both theoretical and human solutions: humans were effective at defending against human attackers, but fared poorly against worst-case attackers. Game-theoretic strategies are robust, but there are opportunities to further exploit weaknesses in human opponents. In addition, we had to simplify the game considerably so human players could understand it. In complex scenarios, humans might not be able to compute any plausible strategy with a reasonable effort. Further empirical studies are clearly needed to investigate the effectiveness of game models for network security, but we view this as a promising first step.

Our work has the potential for further research in several directions. The attacker's interaction with honeypots can be modeled in more detail, including the attacker's attempts to detect honeypots. Another direction is to include in the model network changes that can be partially solved by deploying dynamic honeypots. This model could consider sets of possible network changes and find strategies that additionally minimize the cost for honeypot reconfigurations. ■

Acknowledgments

This work was supported by the Grant Agency of the CTU in Prague (SGS16/235/OHK3/3T/13), Czech Science Foundation (15-23235S), and Cisco Systems.

References

1. L. Spitzner, "Honeypots: Catching the Insider Threat," *Proc. 19th Annual Computer Security Applications Conf.*, 2003, pp. 170–179.
2. M. Tambe, *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*, Cambridge Univ. Press, 2011.
3. X. Ou, W.F. Boyer, and M.A. McQueen, "A Scalable Approach to Attack Graph Generation," *Proc. 13th ACM Conf. Computer and Communications Security*, 2006, pp. 336–345.
4. L. Wang et al., "An Attack Graph-Based Probabilistic Security Metric," *Data and Applications Security*, vol. 5094, 2008, pp. 283–296.
5. P. Mell, K. Scarfone, and S. Romanosky, "Common Vulnerability Scoring System," *IEEE Security & Privacy*, vol. 4, no. 6, 2006, pp. 85–89.
6. K. Durkota et al., "Optimal Network Security Hardening Using Attack Graph Games," *Proc. 24th Int'l Joint Conf. Artificial Intelligence*, 2015, pp. 526–532
7. K. Durkota, "Approximate Solutions for Attack Graph Games with Imperfect Information," *Decision and Game Theory for Security*, Springer, 2015, pp. 228–249.
8. T. Sommestad and F. Sandstrom, "An Empirical Test of the Accuracy of an Attack Graph Analysis Tool," *Information and Computer Security*, vol. 23, no. 5, 2015, pp. 516–531.

cn *Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.*

Monte Carlo Continual Resolving for Online Strategy Computation in Imperfect Information Games

Michal Šustr
michal.sustr@aic.fel.cvut.cz

Vojtěch Kovařík
vojta.kovarik@gmail.com

Viliam Lisý
viliam.lisy@fel.cvut.cz

Artificial Intelligence Center, FEE
Czech Technical University
Prague, Czech Republic

ABSTRACT

Online game playing algorithms produce high-quality strategies with a fraction of memory and computation required by their offline alternatives. Continual Resolving (CR) is a recent theoretically sound approach to online game playing that has been used to outperform human professionals in poker. However, parts of the algorithm were specific to poker, which enjoys many properties not shared by other imperfect information games. We present a domain-independent formulation of CR applicable to any two-player zero-sum extensive-form games (EFGs). It works with an abstract resolving algorithm, which can be instantiated by various EFG solvers. We further describe and implement its Monte Carlo variant (MCCR) which uses Monte Carlo Counterfactual Regret Minimization (MCCFR) as a resolver. We prove the correctness of CR and show an $O(T^{-1/2})$-dependence of MCCR's exploitability on the computation time. Furthermore, we present an empirical comparison of MCCR with incremental tree building to Online Outcome Sampling and Information-set MCTS on several domains.

KEYWORDS

counterfactual regret minimization; resolving; imperfect information; Monte Carlo; online play; extensive-form games; Nash equilibrium

ACM Reference Format:
Michal Šustr, Vojtěch Kovařík, and Viliam Lisý. 2019. Monte Carlo Continual Resolving for Online Strategy Computation in Imperfect Information Games. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

1 INTRODUCTION

Strategies for playing games can be pre-computed *offline* for all possible situations, or computed *online* only for the situations that occur in a particular match. The advantage of the offline computation are stronger bounds on the quality of the computed strategy. Therefore, it is preferable if we want to solve a game optimally. On the other hand, online algorithms can produce strong strategies with a fraction of memory and time requirements of the offline approaches. Online game playing algorithms have outperformed humans in Chess [14], Go [27], and no-limit Poker [3, 22].

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada. © 2019 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

While online approaches have always been the method of choice for strong play in perfect information games, it is less clear how to apply them in imperfect information games (IIGs). To find the optimal strategy for a specific situation in an IIG, a player has to reason about the unknown parts of the game state. They depend on the (possibly unobservable) actions of the opponent prior to the situation, which in turn depends on what the optimal decisions are for both players in many other parts of the game. This makes the optimal strategies in distinct parts of the game closely interdependent and makes correct reasoning about the current situation difficult without solving the game as a whole.

Existing online game playing algorithms for imperfect information games either do not provide any guarantees on the quality of the strategy they produce [8, 9, 21], or require the existence of a compact heuristic evaluation function and a significant amount of computation to construct it [4, 22]. Moreover, the algorithms that are theoretically sound were developed primarily for Texas hold'em poker, which has a very particular information structure. After the initial cards are dealt, all of the actions and chance outcomes that follow are perfectly observable. Furthermore, since the players' moves alternate, the number of actions taken by each player is always known. None of this holds in general for games that can be represented as two-player zero-sum extensive-form games (EFGs). In a blind chess [8], we may learn we have lost a piece, but not necessarily which of the opponent's pieces took it. In visibility-based pursuit-evasion [25], we may know the opponent remained hidden, but not in which direction she moved. In phantom games [28], we may learn it is our turn to play, but not how many illegal moves has the opponent attempted. Because of these complications, the previous theoretically sound algorithms for imperfect-information games are no longer directly applicable.

The sole exception is Online Outcome Sampling (OOS) [20]. It is theoretically sound, completely domain independent, and it does not use any pre-computed evaluation function. However, it starts all its samples from the beginning of the game, and it has to keep sampling actions that cannot occur in the match anymore. As a result, its memory requirements grow as more and more actions are taken in the match, and the high variance in its importance sampling corrections slows down the convergence.

We revisit the Continual Resolving algorithm (CR) introduced in [22] for poker and show how it can be generalized in a way that can handle the complications of general two-player zero-sum EFGs. Based on this generic algorithm, we introduce Monte Carlo Continual Resolving (MCCR), which combines MCCFR [18] with

incremental construction of the game tree, similarly to OOS, but replaces its targeted sampling scheme by Continual Resolving. This leads to faster sampling since MCCR starts its samples not from the root, but from the current point in the game. It also decreases the memory requirements by not having to maintain statistics about parts of the game no longer relevant to the current match. Furthermore, it allows evaluating continual resolving in various domains, without the need to construct expensive evaluation functions.

We prove that MCCR's exploitability approaches zero with increasing computational resources and verify this property empirically in multiple domains. We present an extensive experimental comparison of MCCR with OOS, Information-set Monte Carlo Tree Search (IS-MCTS) [9] and MCCFR. We show that MCCR's performance heavily depends on its ability to quickly estimate key statistics close to the root, which is good in some domains, but insufficient in others.

2 BACKGROUND

We now describe the standard notation for IIGs and MCCFR.

2.1 Imperfect Information Games

We focus on two-player zero-sum extensive-form games with imperfect information. Based on [24], game G can be described by

- $\mathcal{H}$ – the set of *histories*, representing sequences of actions.
- $\mathcal{Z}$ – the set of terminal histories (those $z \in \mathcal{H}$ which are not a prefix of any other history). We use $g \sqsubset h$ to denote the fact that g is equal to or a prefix of h.
- $\mathcal{A}(h) := \{a \mid ha \in \mathcal{H}\}$ denotes the set of actions available at a *non-terminal history* $h \in \mathcal{H} \setminus \mathcal{Z}$. The term ha refers to a history, i.e. child of history h by playing a.
- $\mathcal{P} : \mathcal{H} \setminus \mathcal{Z} \to \{1, 2, c\}$ is the *player function* partitioning non-terminal histories into $\mathcal{H}_1$, $\mathcal{H}_2$ and $\mathcal{H}_c$ depending on which player chooses an action at h. Player c is a special player, called "chance" or "nature".
- *The strategy of chance* is a fixed probability distribution σ_c over actions in chance player's histories.
- The *utility function* $u = (u_1, u_2)$ assigns to each terminal history z the rewards $u_1(z), u_2(z) \in \mathbb{R}$ received by players 1 and 2 upon reaching z. We assume that $u_2 = -u_1$.
- The *information-partition* $\mathcal{I} = (\mathcal{I}_1, \mathcal{I}_2)$ captures the imperfect information of G. For each player $i \in \{1, 2\}$, $\mathcal{I}_i$ is a partition of $\mathcal{H}_i$. If $g, h \in \mathcal{H}_i$ belong to the same $I \in \mathcal{I}_i$ then i cannot distinguish between them. Actions available at infoset I are the same as in each history h of I, therefore we overload $\mathcal{A}(I) := \mathcal{A}(h)$. We only consider games with *perfect recall*, where the players always remember their past actions and the information sets visited so far.

A *behavioral strategy* $\sigma_i \in \Sigma_i$ of player i assigns to each $I \in \mathcal{I}_i$ a probability distribution $\sigma(I)$ over available actions $a \in \mathcal{A}(I)$. A *strategy profile* (or simply *strategy*) $\sigma = (\sigma_1, \sigma_2) \in \Sigma_1 \times \Sigma_2$ consists of strategies of players 1 and 2. For a player $i \in \{1, 2\}$, $-i$ will be used to denote the other two actors $\{1, 2, c\} \setminus \{i\}$ in G (for example $\mathcal{H}_{-1} := \mathcal{H}_2 \cup \mathcal{H}_c$) and opp_i denotes i's opponent ($\text{opp}_1 := 2$).

2.2 Nash Equilibria and Counterfactual Values

The *reach probability* of a history $h \in \mathcal{H}$ under σ is defined as $\pi^\sigma(h) = \pi_1^\sigma(h)\pi_2^\sigma(h)\pi_c^\sigma(h)$, where each $\pi_i^\sigma(h)$ is a product of probabilities of the actions taken by player i between the root and h. The reach probabilities $\pi_i^\sigma(h|g)$ and $\pi^\sigma(h|g)$ conditional on being in some $g \sqsubset h$ are defined analogously, except that the products are only taken over the actions on the path between g and h. Finally, $\pi_{-i}^\sigma(\cdot)$ is defined like $\pi^\sigma(\cdot)$, except that in the product $\pi_1^\sigma(\cdot)\pi_2^\sigma(\cdot)\pi_c^\sigma(\cdot)$ the term $\pi_i^\sigma(\cdot)$ is replaced by 1.

The *expected utility* for player i of a strategy profile σ is $u_i(\sigma) = \sum_{z \in \mathcal{Z}} \pi^\sigma(z) u_i(z)$. The profile σ is an *ϵ-Nash equilibrium* (ϵ-NE) if

$$(\forall i \in \{1, 2\}) \; : \; u_i(\sigma) \geq \max_{\sigma_i' \in \Sigma_i} u_i(\sigma_i', \sigma_{\text{opp}_i}) - \epsilon.$$

A Nash equilibrium (NE) is an ϵ-NE with $\epsilon = 0$. It is a standard result that in two-player zero-sum games, all $\sigma^* \in$ NE have the same $u_i(\sigma^*)$ [24]. The *exploitability* $\text{expl}(\sigma)$ of $\sigma \in \Sigma$ is the average of exploitabilities $\text{expl}_i(\sigma)$, $i \in \{1, 2\}$, where

$$\text{expl}_i(\sigma) := u_i(\sigma^*) - \min_{\sigma'_{\text{opp}_i} \in \Sigma_{\text{opp}_i}} u_i(\sigma_i, \sigma'_{\text{opp}_i}).$$

The expected utility conditional on reaching $h \in \mathcal{H}$ is

$$u_i^\sigma(h) = \sum_{h \sqsubset z \in \mathcal{Z}} \pi^\sigma(z|h) u_i(z).$$

An ingenious variant of this concept is the *counterfactual value* (CFV) of a history, defined as $v_i^\sigma(h) := \pi_{-i}^\sigma(h) u_i^\sigma(h)$, and the counterfactual value of taking an action a at h, defined as $v_i^\sigma(h, a) := \pi_{-i}^\sigma(h) u_i^\sigma(ha)$. We set $v_i^\sigma(I) := \sum_{h \in I} v_i^\sigma(h)$ for $I \in \mathcal{I}_i$ and define $v_i^\sigma(I, a)$ analogously. A strategy $\sigma_2^\star \in \Sigma_2$ is a *counterfactual best response* $\text{CBR}(\sigma_1)$ to $\sigma_1 \in \Sigma_1$ if $v_2^{(\sigma_1, \sigma_2^\star)}(I) = \max_{a \in \mathcal{A}(I)} v_2^{(\sigma_1, \sigma_2^\star)}(I, a)$ holds for each $I \in \mathcal{I}_2$ [7].

2.3 Monte Carlo CFR

For a strategy $\sigma \in \Sigma$, $I \in \mathcal{I}_i$ and $a \in \mathcal{A}(I)$, we set the counterfactual regret for not playing a in I under strategy σ to

$$r_i^\sigma(I, a) := v_i^\sigma(I, a) - v_i^\sigma(I). \tag{1}$$

The Counterfactual Regret minimization (CFR) algorithm [29] generates a consecutive sequence of strategies $\sigma^0, \sigma^1, \ldots, \sigma^T$ in such a way that the *immediate counterfactual regret*

$$\bar{R}_{i,\text{imm}}^t(I) := \max_{a \in \mathcal{A}(I)} \bar{R}_{i,\text{imm}}^t(I, a) := \max_{a \in \mathcal{A}(I)} \frac{1}{t} \sum_{t'=1}^{t} r_i^{\sigma^{t'}}(I, a)$$

is minimized for each $I \in \mathcal{I}_i$, $i \in \{1, 2\}$. It does this by using the Regret Matching update rule [1, 12]:

$$\sigma^{t+1}(I, a) := \frac{\max\{\bar{R}_{i,\text{imm}}^t(I, a), 0\}}{\sum_{a' \in \mathcal{A}(I)} \max\{\bar{R}_{i,\text{imm}}^t(I, a'), 0\}}. \tag{2}$$

Since the overall regret is bounded by the sum of immediate counterfactual regrets [29, Theorem 3], this causes the average strategy $\bar{\sigma}^T$ (defined by (3)) to converge to a NE [18, Theorem 1]:

$$\bar{\sigma}^T(I, a) := \frac{\sum_{t=1}^T \pi_i^{\sigma^t}(I) \sigma^t(I, a)}{\sum_{t=1}^T \pi_i^{\sigma^t}(I)} \qquad \text{(where } I \in \mathcal{I}_i\text{)}. \tag{3}$$

In other words, by accumulating immediate cf. regrets at each information set from the strategies $\sigma^0, \ldots, \sigma^t$, we can produce new

strategy σ^{t+1}. However only the average strategy is guaranteed to converge to NE with $O(1/\sqrt{T})$ – the individual regret matching strategies can oscillate. The initial strategy σ^0 is uniform, but in fact any strategy will work. If the sum in the denominator of update rule (2) is zero, $\sigma^{t+1}(I, a)$ is set to be also uniform.

The disadvantage of CFR is the costly need to traverse the whole game tree during each iteration. Monte Carlo CFR [18] works similarly, but only samples a small portion of the game tree each iteration. It calculates sampled variants of CFR's variables, each of which is an unbiased estimate of the original [18, Lemma 1]. We use a particular variant of MCCFR called Outcome Sampling (OS) [18]. OS only samples a single terminal history z at each iteration, using the sampling strategy $\sigma^{t,\epsilon} := (1-\epsilon)\sigma^t + \epsilon \cdot \text{rnd}$, where $\epsilon \in (0, 1]$ controls the exploration and $\text{rnd}(I, a) := \frac{1}{|\mathcal{A}(I)|}$.

This z is then traversed forward (to compute each player's probability $\pi_i^{\sigma^t}(h)$ of playing to reach each prefix of z) and backward (to compute each player's probability $\pi_i^{\sigma^t}(z|h)$ of playing the remaining actions of the history). During the backward traversal, the sampled counterfactual regrets at each visited $I \in \mathcal{I}$ are computed according to (4) and added to $\tilde{R}^T_{i,\text{imm}}(I)$:

$$\tilde{r}_i^{\sigma^t}(I, a) := \begin{cases} w_I \cdot (\pi^{\sigma^t}(z|ha) - \pi^{\sigma^t}(z|h)) & \text{if } ha \sqsubset z \\ w_I \cdot (0 - \pi^{\sigma^t}(z|h)) & \text{otherwise} \end{cases}, \quad (4)$$

where h denotes the prefix of z which is in I and w_I stands for $\frac{1}{\pi^{\sigma^{t,\epsilon}}(z)} \pi_{-i}^{\sigma^t}(z|h) u_i(z)$ [17].

3 DOMAIN-INDEPENDENT FORMULATION OF CONTINUAL RESOLVING

The only domain for which continual resolving has been previously defined and implemented is poker. Poker has several special properties: a) all information sets have a fixed number of histories of the same length, b) public states have the same size and c) only a single player is active in any public state.

There are several complications that occur in more general EFGs: (1) We might be asked to take several turns within a single public state, for example in phantom games. (2) When we are not the acting player, we might be unsure whether it is the opponent's or chance's turn. (3) Finally, both players might be acting within the same public state, for example a secret chance roll determines whether we get to act or not.

In this section, we present an abstract formulation of continual resolving robust enough to handle the complexities of general EFGs. However, we first need to define the concepts like public tree and resolving gadget more carefully.

3.1 Subgames and the Public Tree

To speak about the information available to player i in histories where he doesn't act, we use *augmented information sets*. For player $i \in \{1, 2\}$ and history $h \in \mathcal{H} \setminus \mathcal{Z}$, the i's *observation history* $\vec{O}_i(h)$ in h is the sequence $(I_1, a_1, I_2, a_2, \dots)$ of the information sets visited and actions taken by i on the path to h (incl. $I \ni h$ if $h \in \mathcal{H}_i$). Two histories $g, h \in \mathcal{H} \setminus \mathcal{Z}$ belong to the same *augmented information set* $I \in \mathcal{I}_i^{\text{aug}}$ if $\vec{O}_i(g) = \vec{O}_i(h)$. This is equivalent to the definition from [7], except that our definition makes it clear that $\mathcal{I}_i^{\text{aug}}$ is also defined on $\mathcal{H}_i$ (and coincides there with $\mathcal{I}_i$ because of perfect recall).

Remark 3.1 (Alternatives to $\mathcal{I}^{\text{aug}}$). *$\mathcal{I}^{\text{aug}}$ isn't the only viable way of generalizing information sets. One could alternatively consider some further-unrefineable perfect-recall partition $\mathcal{I}_i^*$ of $\mathcal{H}$ which coincides with $\mathcal{I}_i$ on $\mathcal{H}_i$, and many other variants between the two extremes. We focus only on $\mathcal{I}^{\text{aug}}$, since an in-depth discussion of the general topic would be outside of the scope of this paper.*

We use $\sim$ to denote histories indistinguishable by some player:

$$g \sim h \iff \vec{O}_1(g) = \vec{O}_1(h) \vee \vec{O}_2(g) = \vec{O}_2(h).$$

By $\approx$ we denote the transitive closure of $\sim$. Formally, $g \approx h$ iff

$$(\exists n)\,(\exists h_1, \dots, h_n) : g \sim h_1,\ h_1 \sim h_2,\ \dots,\ h_{n-1} \sim h_n,\ h_n \sim h.$$

If two states do *not* satisfy $g \approx h$, then it is common knowledge that both players can tell them apart.

Definition 3.2 (Public state). *Public partition* is any partition $\mathcal{S}$ of $\mathcal{H} \setminus \mathcal{Z}$ whose elements are closed under $\sim$ and form a tree. An element S of such $\mathcal{S}$ is called a *public state*. The *common knowledge partition* $\mathcal{S}_{\text{ck}}$ is the one consisting of the equivalence classes of $\approx$.

Our definition of $\mathcal{S}$ is a reformulation of the definition of [15] in terms of augmented information sets (which aren't used in [15]). The definition of $\mathcal{S}_{\text{ck}}$ is novel. We endow any $\mathcal{S}$ with the tree structure inherited from $\mathcal{H}$. Clearly, $\mathcal{S}_{\text{ck}}$ is the finest public partition. The concept of a public state is helpful for introducing imperfect-information subgames (which aren't defined in [15]).

Definition 3.3 (Subgame). A *subgame* rooted at a public state S is the set $G(S) := \{h \in \mathcal{H} \mid \exists g \in S : g \sqsubset h\}$.

For comparison, [7] defines a subgame as "a forest of trees, closed under both the descendant relation and membership within $\mathcal{I}_i^{\text{aug}}$ for any player". For any $h \in S \in \mathcal{S}_{\text{ck}}$, the subgame rooted at S is the smallest [7]-subgame containing h. As a result, [7]-subgames are "forests of subgames rooted at common-knowledge public states".

We can see that finer public partitions lead to smaller subgames, which are easier to solve. In this sense, the common-knowledge partition is the "best one". However, finding $\mathcal{S}_{\text{ck}}$ is sometimes non-trivial, which makes the definition of general public states from [15] important. The drawback of this definition is its ambiguity — indeed, it allows for extremes such as grouping the whole $\mathcal{H}$ into a single public state, without giving a practical recipe for arriving at the "intuitively correct" public partition.

3.2 Aggregation and the Upper Frontier

Often, it is useful to aggregate reach probabilities and counterfactual values over (augmented) information sets or public states. In general EFGs, an augmented information set $I \in \mathcal{I}_i^{\text{aug}}$ can be "thick", i.e. it can contain both some $ha \in \mathcal{H}$ and it's parent h. This necessarily happens when we are unsure how many actions were taken by other players between our two successive actions. For such I, we only aggregate over the "upper frontier" $\hat{I} := \{h \in I \mid \nexists g \in I : g \sqsubset h \,\&\, g \neq h\}$ of I [10, 11]: We overload $\pi^\sigma(\cdot)$ as $\pi^\sigma(I) := \sum_{h \in \hat{I}} \pi^\sigma(h)$ and $v_i^\sigma(\cdot)$ as $v_i^\sigma(I) := \sum_{h \in \hat{I}} v_i^\sigma(h)$. We define $\hat{S}$ for $S \in \mathcal{S}$, $\pi_i^\sigma(I)$, $\pi_{-i}^\sigma(I)$ and $v_i^\sigma(I, a)$ analogously. By $\hat{S}(i) := \{I \in \mathcal{I}_i^{\text{aug}} \mid \hat{I} \subseteq \hat{S}\}$ we denote the topmost (augmented) information sets of player i in S.

To the best of our knowledge, the issue of "thick" information sets has only been discussed in the context of non-augmented

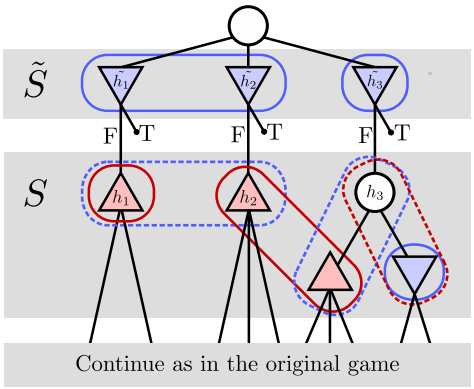


Figure 1: Resolving game $\widetilde{G}(S, \sigma, \tilde{v})$ constructed for player $\triangle$ in a public state S. Player's (augmented) information sets are drawn with solid (resp. dashed) lines of the respective color. The chance node $\bigcirc$ chooses one of $\triangledown$'s histories $\tilde{h}_1, \tilde{h}_2, \tilde{h}_3$, which correspond to the "upper frontier" of S.

information sets in games with imperfect recall [11]. One scenario where thick augmented information sets cause problems is the resolving gadget game, which we discuss next.

3.3 Resolving Gadget Game

We describe a generalization of the resolving gadget game from [7] (cf. [3, 23]) for resolving Player 1's strategy (see Figure 1).

Let $S \in \mathcal{S}$ be a public state to resolve from, $\sigma \in \Sigma$, and let $\tilde{v}(I) \in \mathbb{R}$ for $I \in \hat{S}(i)$ be the required counterfactual values. First, the upper frontier of S is duplicated as $\{\tilde{h} | h \in \hat{S}\} =: \tilde{S}$. Player 2 is the acting player in $\tilde{S}$, and from his point of view, nodes $\tilde{h}$ are partitioned according to $\{\tilde{I} := \{\tilde{h} | h \in \hat{I}\} \mid I \in \hat{S}(2)\}$. In $\tilde{h} \in \tilde{I}$ corresponding to $h \in I$, he can choose between "following" (F) into h and "terminating" (T), which ends the game with utility $\tilde{u}_2(\tilde{h}T) := \tilde{v}(I)\pi^{\sigma}_{-2}(S)/\pi^{\sigma}_{-2}(I)$. From any $h \in \hat{S}$ onward, the game is identical to $G(S)$, except that the utilities are multiplied by a constant: $\tilde{u}_i(z) := u_i(z)\pi^{\sigma}_{-2}(S)$. To turn this into a well-defined game, a root chance node is added and connected to each $h \in \hat{S}$, with transition probabilities $\pi^{\sigma}_{-2}(h)/\pi^{\sigma}_{-2}(S)$.

This game is called the *resolving gadget game* $\widetilde{G}(S, \sigma, \tilde{v})$, or simply $\widetilde{G}(S)$ when there is no risk of confusion, and the variables related to it are denoted by tilde. If $\tilde{\rho} \in \widetilde{\Sigma}$ is a "resolved" strategy in $\widetilde{G}(S)$, we denote the new combined strategy in G as $\sigma^{\text{new}} := \sigma|_{G(S) \leftarrow \tilde{\rho}}$, i.e. play according to strategy $\tilde{\rho}$ in the subgame $G(S)$ and according to σ everywhere else.

The difference between $\widetilde{G}(S, \sigma, \tilde{v})$ and the original version of [7] is that our modification only duplicates the upper frontier $\hat{S}$ and uses normalization constant $\sum_{\hat{S}} \pi^{\sigma}_{-2}(h)$ (rather than $\sum_{S} \pi^{\sigma}_{-2}(h)$) and estimates $\tilde{v}(I) = \sum_{\hat{I}} \tilde{v}(h)$ (rather than $\sum_{I} \tilde{v}(h)$). This enables $\widetilde{G}(S, \sigma, \tilde{v})$ to handle domains with thick information sets and public states. While tedious, it is straightforward to check that $\widetilde{G}(S, \sigma, \tilde{v})$ has all the properties proved in [6, 7, 22]. Without our modification, the resolving games would either sometimes be ill-defined, or wouldn't have the desired properties.

Input : Information set $I \in \mathcal{I}_1$
Output: An action $a \in \mathcal{A}(I)$

1 S $\leftarrow$ the public state which contains I;
2 **if** S $\notin$ KPS **then**
3 $\quad \widetilde{G}(S) \leftarrow$ `BuildResolvingGame(S,D(S))`;
4 $\quad$ KPS $\leftarrow$ KPS $\cup$ S;
5 $\quad$ NPS $\leftarrow$ all $S' \in \mathcal{S}$ where CR acts for the first time after leaving KPS;
6 $\quad \tilde{\rho}, \tilde{\text{D}} \leftarrow$ `Resolve`$(\widetilde{G}(S)$,NPS);
7 $\quad \sigma_1|_{S'} \leftarrow \tilde{\rho}|_{S'}$;
8 $\quad$ D $\leftarrow$ calculate data for NPS based on D, σ_1 and $\tilde{\text{D}}$;
9 **end**
10 **return** a $\sim \sigma_1(I)$

Algorithm 1: Function `Play` of Continual Resolving

The following properties are helpful to get an intuitive understanding of gadget games. Their more general versions and proofs (resp. references for proofs) are listed in the appendix.

Lemma 3.4 (Gadget game preserves opponent's values). *For each $I \in \mathcal{I}_2^{\text{aug}}$ with $I \subset G(S)$, we have $v_2^{\sigma^{\text{new}}}(I) = \tilde{v}_2^{\tilde{\rho}}(I)$.*

Note that the conclusion does *not* hold for counterfactual values of the (resolving) player 1! (This can be easily verified on a simple specific example such as Matching Pennies.)

Lemma 3.5 (Optimal resolving). *If σ and $\tilde{\rho}$ are both Nash equilibria and $\tilde{v}(I) = v_2^{\sigma}(I)$ for each $I \in \hat{S}(2)$, then σ_1^{new} is not exploitable.*

3.4 Continual Resolving

Domain-independent continual resolving closely follows the structure of continual resolving for poker [22], but uses a generalized resolving gadget and handles situations which do not arise in poker, such as multiple moves in one public state. We explain it from the perspective of Player 1. The abstract CR keeps track of strategy σ_1 it has computed in previous moves. Whenever it gets to a public state S, where σ_1 has not been computed, it resolves the subgame $G(S)$. As a by-product of this resolving, it estimates opponent's counterfactual values $v_2^{\sigma_1, \text{CBR}(\sigma_1)}$ for all public states that might come next, allowing it to keep resolving as the game progresses.

CR repetitively calls a `Play` function which takes the current information set $I \in \mathcal{I}_1$ as the input and returns an action $a \in \mathcal{A}(I)$ for Player 1 to play. It maintains the following variables:

- $S \in \mathcal{S}$ ... the current public state,
- KPS $\subset \mathcal{S}$ ... the public states where strategy is known,
- σ_1 ... a strategy defined for every $I \in \mathcal{I}_1$ in KPS,
- NPS $\subset \mathcal{S}$ ... the public states where CR may resolve next,
- $D(S')$ for $S' \in$ NPS ... data allowing resolving at S', such as the estimates of opponent's counterfactual values.

The pseudo-code for CR is described in Algorithm 1. If the current public state belongs to KPS, then the strategy $\sigma_1(I)$ is defined, and we sample action a from it. Otherwise, we should have the data necessary to build some resolving game $\widetilde{G}(S)$ (line 3). We then determine the public states NPS where we might need to resolve next (line 5). We solve $\widetilde{G}(S)$ via some resolving method which also computes the data necessary to resolve from any $S' \in$ NPS (line

6). Finally, we save the resolved strategy in S and update the data needed for resolving (line 7-9). To initialize the variables before the first resolving, we set KPS and σ_1 to $\emptyset$, find appropriate NPS, and start solving the game from the root using the same solver as `Play`, i.e. $\_, D \leftarrow$ `Resolve`(G, NPS).

We now consider CR variants that use the gadget game from Section 3.3 and data of the form $D = (r_1, \tilde{v})$, where $r_1(S') = (\pi_1^{\sigma_1}(h))_{S'}$ is CR's range and $\tilde{v}(S') = (\tilde{v}(J))_J$ estimates opponent's counterfactual value at each $J \in S'(2)$. We shall use the following notation: S_n is the n-th public state from which CR resolves; $\tilde{\rho}_n$ is the corresponding strategy in $\widetilde{G}(S_n)$; σ_1^n is CR's strategy after n-th resolving, defined on KPS_n; the optimal extension of σ_1^n is

$$\sigma_1^{*n} := \text{argmin}_{\nu_1 \in \Sigma_1} \text{expl}_1 \left(\sigma_1^n|_{\text{KPS}_n} \cup \nu_1|_{\mathcal{S} \setminus \text{KPS}_n} \right).$$

Lemmata 24 and 25 of [22] (summarized into Lemma A.5 in our Appendix A) give the following generalization of [22, Theorem S1]:

Theorem 3.6 (Continual resolving bound). *Suppose that CR uses $D = (r_1, \tilde{v})$ and $\widetilde{G}(S, \sigma_1, \tilde{v})$. Then the exploitability of its strategy is bounded by $\text{expl}_1(\sigma_1) \leq \epsilon_0^{\tilde{v}} + \epsilon_1^R + \epsilon_1^{\tilde{v}} + \cdots + \epsilon_{N-1}^{\tilde{v}} + \epsilon_N^R$, where N is the number of resolving steps and $\epsilon_n^R := \widetilde{\text{expl}}_1(\tilde{\rho}_n)$, $\epsilon_n^{\tilde{v}} := \sum_{J \in \hat{S}_{n+1}(2)} \left| \tilde{v}(J) - v_2^{\sigma_1^{*n}, CBR}(J) \right|$ are the exploitability (in $\widetilde{G}(S_n)$) and value estimation error made by the n-th resolver (resp. initialization for $n = 0$).*

The DeepStack algorithm from [22] is a poker-specific instance of the CR variant described in the above paragraph. Its resolver is a modification of CFR with neural network heuristics and sparse look-ahead trees. We make CR domain-independent and allowing for different resolving games (`BuildResolvingGame`), algorithms (`Resolve`), and schemes (by changing line 5).

4 MONTE CARLO CONTINUAL RESOLVING

Monte Carlo Continual Resolving is a specific instance of CR which uses Outcome Sampling MCCFR for game (re)solving. Its data are of the form $D = (r_1, \tilde{v})$ described above and it resolves using the gadget game from Section 3.3. We first present an abstract version of the algorithms that we formally analyze, and then add improvements that make it practical. To simplify the theoretical analysis, we assume MCCFR computes the exact counterfactual value of resulting average strategy $\bar{\sigma}^T$ for further resolving. (We later discuss more realistic alternatives.) The following theorem shows that MCCR's exploitability converges to 0 at the rate of $O(T^{-1/2})$.

Theorem 4.1 (MCCR bound). *With probability at least $(1-p)^{N+1}$, the exploitability of strategy σ computed by MCCR satisfies*

$$\text{expl}_i(\sigma) \leq \left(\sqrt{2}/\sqrt{p} + 1 \right) |\mathcal{I}_i| \frac{\Delta_{u,i}\sqrt{A_i}}{\delta} \left(\frac{2}{\sqrt{T_0}} + \frac{2N-1}{\sqrt{T_R}} \right),$$

where T_0 and T_R are the numbers of MCCR's iterations in pre-play and each resolving, N is the required number of resolvings, $\delta = \min_{z,t} q_t(z)$ where $q_t(z)$ is the probability of sampling $z \in \mathcal{Z}$ at iteration t, $\Delta_{u,i} = \max_{z,z'} |u_i(z) - u_i(z')|$ and $A_i = \max_{I \in \mathcal{I}_i} |\mathcal{A}(I)|$.

The proof is presented in the appendix. Essentially, it inductively combines the OS bound (Lemma A.1) with the guarantees available for resolving games in order to compute the overall exploitability bound.[1] For specific domains, a much tighter bound can be obtained by going through our proof in more detail and noting that the size of subgames decreases exponentially as the game progresses (whereas the proof assumes that it remains constant). In effect, this would replace the N in the bound above by a small constant.

4.1 Practical Modifications

Above, we describe an abstract version of MCCR optimized for clarity and ease of subsequent theoretical analysis. We now describe the version of MCCR that we implemented in practice. The code used for the experiments is available online at https://github.com/aicenter/gtlibrary-java/tree/mccr.

4.1.1 Incremental Tree-Building. A massive reduction in the memory requirements can be achieved by building the game tree incrementally, similarly to Monte Carlo Tree Search (MCTS) [5]. We start with a tree that only contains the root. When an information set is reached that is not in memory, it is added to it and a playout policy (e.g., uniformly random) is used for the remainder of the sample. In playout, information sets are not added to memory. Only the regrets in information sets stored in the memory are updated.

4.1.2 Counterfactual Value Estimation. Since the computation of the exact counterfactual values of the average strategy needed by $\widetilde{G}(S, \sigma, \cdot)$ requires the traversal of the whole game tree, we have to work with their estimates instead. To this end, our MCCFR additionally computes the opponent's *sampled counterfactual values*

$$\tilde{v}_2^{\sigma^t}(I) := \frac{1}{\pi^{\sigma^t,\epsilon}(z)} \pi_{-2}^{\sigma^t}(h) \pi^{\sigma^t}(z|h) u_2(z).$$

It is not possible to compute the exact counterfactual value of the average strategy just from the values of the current strategies. Once the T iterations are complete, the standard way of estimating the counterfactual values of $\bar{\sigma}^T$ is using *arithmetic averages*

$$\tilde{v}(I) := \frac{1}{T} \sum \tilde{v}_2^{\sigma^t}(I). \tag{5}$$

However, we have observed better results with *weighted averages*

$$\tilde{v}(h) := \sum_t \tilde{\pi}^{\sigma^t}(h)\, v_2^{\sigma^t}(h) \;/\; \sum_t \tilde{\pi}^{\sigma^t}(h). \tag{6}$$

The stability and accuracy of these estimates is experimentally evaluated in Section 5 and further analyzed in Appendix B. We also propose an unbiased estimate of the exact values computed from the already executed samples, but its variance makes it impractical.

4.1.3 Root Distribution of Gadgets. As in [22], we use the information about opponent's strategy from previous resolving when constructing the gadget game. Rather than being proportional to $\pi_{-2}(h)$, the root probabilities are proportional to $\pi_{-2}(h)(\pi_2(h) + \epsilon)$. This modification is sound as long as $\epsilon > 0$.

[1] Note that Theorem 4.1 isn't a straightforward corollary of Theorem 3.6, since calculating the numbers $\epsilon_n^{\tilde{v}}$ does require non-trivial work. In particular, $\bar{\sigma}^T$ from the n-th resolving isn't the same as σ_1^{n*}, $CBR(\sigma_1^{n*})$ and the simplifying assumption about $\tilde{v}$ is *not* equivalent to assuming that $\epsilon_n^{\tilde{v}} = 0$.

4.1.4 Custom Sampling Scheme. To improve the efficiency of resolving by MCCFR, we use a custom sampling scheme which differs from OS in two aspects. First, we modify the above sampling scheme such that with probability 90% we sample a history that belongs to the current information set I. This allows us to focus on the most relevant part of the game. Second, whenever $\tilde{h} \in \tilde{S}$ is visited by MCCFR, we sample both actions (T and F). This increases the transparency of the algorithm, since all iterations now "do a similar amount of work" (rather than some terminating immediately). These modifications are theoretically sound, since the resulting sampling scheme still satisfies the assumptions of the general MCCFR bound from [17].

4.1.5 Keeping the Data between Successive Resolvings. Both in pre-play and subsequent resolvings, MCCFR operates on successively smaller and smaller subsets of the game tree. In particular, we don't need to start each resolving from scratch, but we can re-use the previous computations. To do this, we initialize each resolving MCCFR with the MCCFR variables (regrets, average strategy and the corresponding value estimates) from the previous resolving (resp. pre-play). In practice this is accomplished by simply not resetting the data from the previous MCCFR. While not being backed up by theory, this approach worked better in most practical scenarios, and we believe it can be made correct with the use of warm-starting [2] of the resolving gadget.

5 EXPERIMENTAL EVALUATION

After brief introduction of competing methods and explaining the used methodology, we focus on evaluating the alternative methods to estimate the counterfactual values required for resolving during MCCFR. Next, we evaluate how quickly and reliably these values can be estimated in different domains, since these values are crucial for good performance of MCCR. Finally, we compare exploitability and head-to-head performance to competing methods.

5.1 Competing Methods

Information-Set Monte Carlo Tree Search. IS-MCTS [19] runs MCTS samples as in a perfect information game, but computes statistics for the whole information set and not individual states. When initiated from a non-empty match history, it starts samples uniformly from the states in the current information set. We use two selection functions: Upper Confidence bound applied to Trees (UCT) [16] and Regret Matching (RM) [13]. We use the same settings as in [20]: UCT constant 2x the maximal game outcome, and RM with exploration 0.2. In the following, we refer to IS-MCTS with the corresponding selection function by only `UCT` or `RM`.

Online Outcome Sampling. OOS [20] is an online search variant of MCCFR. MCCFR samples from the root of the tree and needs to pre-build the whole game tree. OOS has two primary distinctions from MCCFR: it builds its search tree incrementally and it can bias samples with some probability to any specific parts of the game tree. This is used to target the information sets (OOS-IST) or the public states (OOS-PST) where the players act during a match.

We do not run OOS-PST on domain of IIGS, due to non-trivial biasing of sampling towards current public state.

We further compare to MCCFR with incremental tree building and the random player denoted RND.

5.2 Computing Exploitability

Since the online game playing algorithms do not compute the strategy for the whole game, evaluating exploitability of the strategies they play is more complicated. One approach, called brute-force in [20], suggest ensuring that the online algorithm is executed in each information set in the game and combining the computed strategies. If the thinking time of the algorithm per move is t, it requires $O(t|\mathcal{I}|)$ time to compute one combined strategy and multiple runs are required to achieve statistical significance for randomized algorithms. While this is prohibitively expensive even for the smaller games used in our evaluation, computing the strategy for each public state, instead of each information set is already feasible. We use this approach, however, it means we have to disable the additional targeting of the current information set in the resolving gadget proposed in Section 4.1.4.

There are two options how to deal with the variance in the combined strategies in different runs of the algorithm in order to compute the exploitability of the real strategy realized by the algorithm. The pessimistic option is to compute the exploitability of each combined strategy and average the results. This assumes the opponent knows the random numbers used by the algorithm for sampling in each resolving. A more realistic option is to average the combined strategies from different runs into an *expected strategy* $\bar{\tilde{\sigma}}$ and compute its exploitability. We use the latter.

5.3 Domains

For direct comparison with prior work, we use same domains as in [20] with parametrizations noted in parentheses: Imperfect Information Goofspiel `IIGS(N)`, Liar's Dice `LD(D1,D2,F)` and Generic Poker `GP(T,C,R,B)`. We add Phantom Tic-Tac-Toe `PTTT` to also have a domain with thick public states, and use Biased Rock Paper Scissors `B-RPS` for small experiments. The detailed rules are in Appendix C with the sizes of the domains in Table 2. We use small and large variants of the domains based on their parametrization. Note that large variants are about 10^4 up to 10^{15} times larger than the smaller ones.

5.4 Results

5.4.1 Averaging of Sampled CFVs. As mentioned in Section 4.1.2, computing the exact counterfactual values of the average strategy $\bar{\sigma}^T$ is often prohibitive, and we replace it by using the arithmetic or weighted averages instead. To compare the two approaches, we run MCCFR on the B-RPS domain (which only has a single NE $\sigma*$ to which $\bar{\sigma}^T$ converges) and measure the distance $\Delta v(t)$ between the estimates and the correct action-values $v_1^{\sigma*}(\text{root}, a)$. In Figure 2 (left), the weighted averages converge to the correct values with increasing number of samples, while the arithmetic averages eventually start diverging. The weighted averages are more accurate (see Appendix, Figure 4 for comparison on each domain) and we will use them exclusively in following experiments.

5.4.2 Stability of CFVs. To find a nearly optimal strategy when resolving, MCCR first needs CFVs of such a strategy (Lemma A.5). However, the MCCFR resolver typically won't have enough time to find such $\bar{\sigma}^T$. If MCCR is to work, the CFVs computed by MCCFR

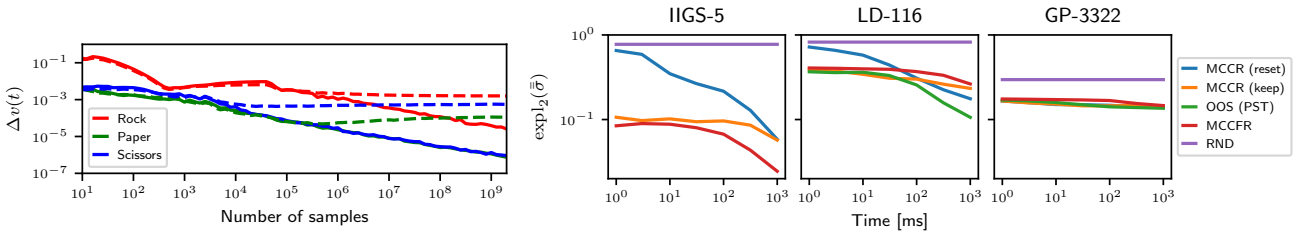


Figure 2: Left – Estimation error of the arithmetic (dashed lines) and weighted averages (solid lines) of action values in B-RPS. Right – Exploitability of $\bar{\tilde{\sigma}}$ as a function of the resolving time. All algorithms have pre-play of 300ms.

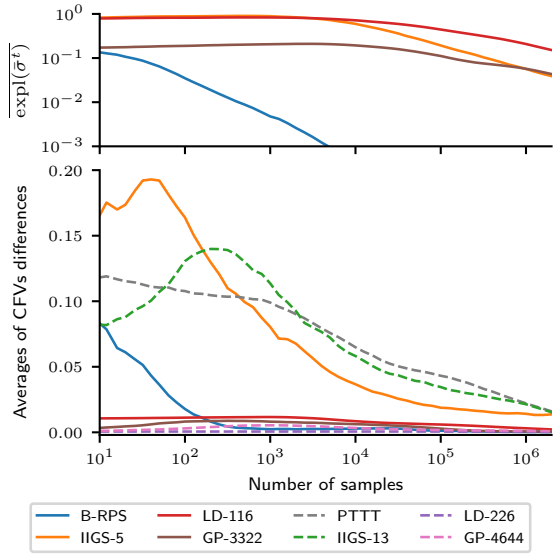


Figure 3: A comparison of exploitability (top) with "CFV instability" (bottom) in different domains. For $t = 10^7$, the differences are 0 by definition.

need to be close to those of an approximate equilibrium CFVs even though they correspond to an exploitable strategy.

We run MCCFR in the root and focus on CFVs in the public states where player 1 acts for the 2nd time (the gadget isn't needed for the first action, and thus neither are CFVs):

$$\Omega := \{J \in \mathcal{I}_2^{\text{aug}} \mid \exists S' \in \mathcal{S}, h \in S' : J \subset S' \& \text{ pl. 1 played once in } h\}.$$

Since there are multiple equilibria MCCFR might converge to, we cannot measure the convergence exactly. Instead, we measure the "instability" of CFV estimates by saving $\tilde{v}_2^t(J)$ for $t \leq T$, tracking how $\Delta_t(J) := |\tilde{v}_2^t(J) - \tilde{v}_2^T(J)|$ changes over time, and aggregating it into $\frac{1}{|\Omega|}\sum_J \Delta_t(J)$. We repeat this experiment with 100 different MCCFR seeds and measure the expectation of the aggregates and, for the small domains, the expected exploitability of $\bar{\sigma}^t$. If the resulting "instability" is close to zero after 10^5 samples (our typical time budget), we consider CFVs to be sufficiently stable.

Figure 3 confirms that in small domains (LD, GP), CFVs stabilize long before the exploitability of the strategy gets low. The error still decreases in larger games (GS, PTTT), but at a slow rate.

5.4.3 Comparison of Exploitability with Other Algorithms. We compare the exploitability of MCCR to OOS-PST and MCCFR, and include random player for reference. We do not include OOS-IST, whose exploitability is comparable to that of OOS-PST [20]. For an evaluation of IS-MCTS's exploitability (which is very high, with the exception of poker) we refer the reader to [19, 20].

Figure 2 (right) confirms that for all algorithms, the exploitability decreases with the increased time per move. MCCR is better than MCCFR on LD and worse on IIGS. The keep variant of MCCR is initially less exploitable than the reset variant, but improves slower. This suggests the keep variant could be improved.

5.4.4 Influence of the Exploration Rate. One of MCCR's parameters is the exploration rate ϵ of its MCCFR resolver. When measuring the exploitability of MCCR we observed no noteworthy influence of ϵ (for $\epsilon = 0.2, 0.4, 0.6, 0.8$, across all of the evaluated domains).

5.4.5 Head-to-head Performance. For each pair of algorithms, thousands of matches have been played on each domain, alternating positions of player 1 and 2. In the smaller (larger) domains, players have 0.3*s* (5s) of pre-play computation and 0.1*s* (1s) per move. Table 1 summarizes the results as percentages of the maximum domain payoff.

Note that the results of the matches are not necessarily transitive, since they are not representative of the algorithms' exploitability. When computationally tractable, the previous experiment 5.4.3 is therefore a much better comparison of an algorithm's performance.

Both variants of MCCR significantly outperform the random opponent in all games. With the exception of PTTT, they are also at least as good as the MCCFR "baseline". This is because public states in PTTT represent the number of moves made, which results in a non-branching public tree and resolving games occupy the entire level as in the original game. MCCR is better than OOS-PST in LD and GP, and better than OOS-IST in large IIGS. MCCR is worse than IS-MCTS on all games with the exception of small LD. However, this does not necessarily mean that MCCR's exploitability is higher than for IS-MCTS in the larger domains, MCCR only fails to find the strategy that would exploit IS-MCTS.

6 CONCLUSION

We propose a generalization of Continual Resolving from poker [22] to other extensive-form games. We show that the structure of the public tree may be more complex in general, and propose an extended version of the resolving gadget necessary to handle this

IIGS-5	MCCR (reset)	MCCFR	OOS-PST	OOS-IST	RM	UCT	RND
MCCR (keep)	0.4 ± 1.2	0.6 ± 1.3	–	-2.6 ± 1.3	-2.8 ± 1.3	-6.5 ± 1.2	51.2 ± 1.2
MCCR (reset)		-0.8 ± 1.2	–	-2.5 ± 1.2	-5.5 ± 1.2	-8.1 ± 1.2	49.1 ± 1.2
MCCFR			–	-1.3 ± 1.3	-2.7 ± 1.3	-5.6 ± 1.2	48.5 ± 1.2
OOS-PST				–	–	–	–
OOS-IST					-2.0 ± 1.3	-5.2 ± 1.2	54.5 ± 1.1
RM						-16.6 ± 1.2	67.8 ± 1.0
UCT							70.0 ± 1.0

IIGS-13	MCCR (reset)	MCCFR	OOS-PST	OOS-IST	RM	UCT	RND
MCCR (keep)	-18.8 ± 5.6	12.8 ± 5.7	–	-7.3 ± 5.7	-56.4 ± 4.7	-69.1 ± 4.1	43.9 ± 5.1
MCCR (reset)		24.4 ± 5.5	–	4.9 ± 2.6	-35.6 ± 5.3	-56.0 ± 4.7	55.6 ± 4.7
MCCFR			–	-22.8 ± 5.6	-59.9 ± 4.6	-75.1 ± 3.7	37.8 ± 5.3
OOS-PST				–	–	–	–
OOS-IST					-44.4 ± 5.1	-61.2 ± 4.5	58.2 ± 4.6
RM						-22.8 ± 5.6	82.3 ± 3.2
UCT							91.2 ± 2.3

LD-116	MCCR (reset)	MCCFR	OOS-PST	OOS-IST	RM	UCT	RND
MCCR (keep)	-1.4 ± 2.0	9.7 ± 2.0	5.2 ± 2.0	-4.1 ± 2.0	2.6 ± 1.0	5.6 ± 2.0	60.9 ± 1.6
MCCR (reset)		11.6 ± 2.0	10.1 ± 2.0	-3.9 ± 2.0	-5.5 ± 2.0	5.3 ± 2.0	60.5 ± 1.6
MCCFR			-6.8 ± 2.0	-8.7 ± 2.0	-4.7 ± 2.0	1.9 ± 1.0	54.0 ± 1.7
OOS-PST				-4.3 ± 2.0	-3.0 ± 2.0	6.5 ± 2.0	60.3 ± 1.6
OOS-IST					-1.7 ± 1.0	5.2 ± 2.0	64.8 ± 1.6
RM						5.2 ± 2.0	66.1 ± 1.5
UCT							65.4 ± 1.5

LD-226	MCCR (reset)	MCCFR	OOS-PST	OOS-IST	RM	UCT	RND
MCCR (keep)	6.2 ± 5.4	45.3 ± 5.7	46.1 ± 5.7	-22.7 ± 5.9	-33.6 ± 5.7	-33.4 ± 5.7	76.2 ± 4.2
MCCR (reset)		37.8 ± 5.4	44.2 ± 5.7	-31.2 ± 5.7	-39.8 ± 5.4	-44.8 ± 5.2	81.6 ± 4.5
MCCFR			-5.4 ± 4.6	-55.7 ± 4.5	-49.3 ± 4.6	-47.8 ± 5.1	45.8 ± 5.2
OOS-PST				-53.6 ± 5.3	-51.5 ± 4.9	-46.4 ± 5.1	49.6 ± 4.1
OOS-IST					-12.0 ± 5.6	-22.5 ± 5.6	83.8 ± 3.8
RM						-11.6 ± 5.7	79.7 ± 3.5
UCT							75.4 ± 3.8

GP-3322	MCCR (reset)	MCCFR	OOS-PST	OOS-IST	RM	UCT	RND
MCCR (keep)	0.2 ± 0.4	2.2 ± 0.5	1.7 ± 0.5	0.4 ± 0.2	-0.5 ± 0.4	-1.5 ± 0.4	5.9 ± 0.5
MCCR (reset)		0.7 ± 0.4	0.4 ± 0.2	-0.3 ± 0.2	-0.5 ± 0.4	-1.0 ± 0.3	5.5 ± 0.4
MCCFR			-1.9 ± 0.6	-2.7 ± 0.6	-3.6 ± 0.5	-3.3 ± 0.5	6.0 ± 0.6
OOS-PST				-1.0 ± 0.9	-2.1 ± 0.5	-2.8 ± 0.4	7.4 ± 0.6
OOS-IST					-1.3 ± 0.5	-2.1 ± 0.4	7.4 ± 0.6
RM						-1.2 ± 0.4	7.8 ± 0.5
UCT							6.3 ± 0.4

GP-4644	MCCR (reset)	MCCFR	OOS-PST	OOS-IST	RM	UCT	RND
MCCR (keep)	1.6 ± 1.2	7.3 ± 2.6	14.2 ± 2.5	-3.4 ± 2.3	-4.1 ± 1.8	-6.9 ± 1.5	19.3 ± 2.6
MCCR (reset)		9.5 ± 2.0	11.9 ± 2.0	-3.5 ± 1.8	-3.0 ± 1.5	-2.5 ± 1.3	15.8 ± 2.2
MCCFR			-8.7 ± 3.1	-13.3 ± 2.9	-9.6 ± 2.3	-6.8 ± 1.9	12.0 ± 3.0
OOS-PST				-8.1 ± 3.0	-8.9 ± 2.4	-5.0 ± 2.0	11.8 ± 3.1
OOS-IST					-2.1 ± 1.8	-1.6 ± 1.2	20.4 ± 2.9
RM						-0.3 ± 1.1	20.5 ± 2.3
UCT							17.6 ± 2.0

PTTT	MCCR (reset)	MCCFR	OOS-PST	OOS-IST	RM	UCT	RND
MCCR (keep)	17.7 ± 3.8	-1.1 ± 0.9	-1.8 ± 1.6	-5.0 ± 3.7	-6.9 ± 3.7	-6.2 ± 3.7	25.5 ± 3.8
MCCR (reset)		-6.2 ± 3.8	-9.8 ± 3.7	-14.6 ± 3.6	-20.9 ± 3.6	-14.3 ± 3.7	21.6 ± 3.7
MCCFR			0.1 ± 3.7	-2.1 ± 1.5	-5.2 ± 3.7	-4.0 ± 3.6	27.9 ± 3.7
OOS-PST				-5.6 ± 3.7	-5.7 ± 3.7	-5.2 ± 3.7	29.4 ± 3.7
OOS-IST					-3.5 ± 3.2	-3.9 ± 3.6	35.1 ± 3.6
RM						5.6 ± 3.6	51.3 ± 3.3
UCT							52.6 ± 3.3

Table 1: Head-to-head performance. Positive numbers mean that the row algorithm is winning against the column algorithm by the given percentage of the maximum payoff in the domain. Gray numbers indicate the winner isn't statistically significant.

complexity. Furthermore, both players may play in the same public state (possibly multiple times), and we extend the definition of Continual Resolving to allow this case as well. We present a completely domain-independent version of the algorithm that can be applied to any EFG, is sufficiently robust to use variable resolving schemes, and can be coupled with different resolving games and algorithms (including classical CFR, depth-limited search utilizing neural networks, or other domain-specific heuristics). We show that the existing theory naturally translates to this generalized case.

We further introduce Monte Carlo CR as a specific instance of this abstract algorithm that uses MCCFR as a resolver. It allows deploying continual resolving on any domain, without the need for expensive construction of evaluation functions. MCCR is theoretically sound as demonstrated by Theorem 4.1, constitutes an improvement over MCCFR in the online setting in terms head-to-head performance, and doesn't have the restrictive memory requirements of OOS. The experimental evaluation shows that MCCR is very sensitive to the quality of its counterfactual value estimates. With good estimates, its worst-case performance (i.e. exploitability) improves faster than that of OOS. In head-to-head matches MCCR plays similarly to OOS, but it only outperforms IS-MCTS in one of the smaller tested domains. Note however that the lack of theoretical guarantees of IS-MCTS often translates into severe exploitability in practice [20], and this cannot be alleviated by increasing IS-MCTS's computational resources [19]. In domains where MCCR's counterfactual value estimates are less precise, its exploitability still converges to zero, but at a slower rate than OOS, and its head-to-head performance is noticeably weaker than that of both OOS and IS-MCTS.

In the future work, the quality of MCCR's estimates might be improved by variance reduction [26], exploring ways of improving these estimates over the course of the game, or by finding an alternative source from which they can be obtained. We also plan to test the hypothesis that there are classes of games where MCCR performs much better than the competing algorithms (in particular, we suspect this might be true for small variants of turn-based computer games such as Heroes of Might & Magic or Civilization).

ACKNOWLEDGMENTS

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the program "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated. This work was supported by Czech science foundation grant no. 18-27483Y.

REFERENCES

[1] David Blackwell and others. 1956. An analog of the minimax theorem for vector payoffs. *Pacific J. Math.* 6, 1 (1956), 1–8.

[2] Noam Brown and Tuomas Sandholm. 2016. Strategy-Based Warm Starting for Regret Minimization in Games.. In *AAAI*. 432–438.

[3] Noam Brown and Tuomas Sandholm. 2017. Safe and nested subgame solving for imperfect-information games. In *Advances in Neural Information Processing Systems*. 689–699.

[4] Noam Brown, Tuomas Sandholm, and Brandon Amos. 2018. Depth-Limited Solving for Imperfect-Information Games. *arXiv preprint arXiv:1805.08195* (2018).

[5] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4, 1 (2012), 1–43.

[6] Neil Burch. 2017. *Time and Space: Why Imperfect Information Games are Hard*. Ph.D. Dissertation. University of Alberta.

[7] Neil Burch, Michael Johanson, and Michael Bowling. 2014. Solving Imperfect Information Games Using Decomposition.. In *AAAI*. 602–608.

[8] Paolo Ciancarini and Gian Piero Favini. 2010. Monte Carlo tree search in Kriegspiel. *Artificial Intelligence* 174 (July 2010), 670–684. Issue 11.

[9] Peter I Cowling, Edward J Powley, and Daniel Whitehouse. 2012. Information set monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 2 (2012), 120–143.

[10] Joseph Y Halpern. 1997. On ambiguities in the interpretation of game trees. *Games and Economic Behavior* 20, 1 (1997), 66–96.

[11] Joseph Y Halpern and Rafael Pass. 2016. Sequential Equilibrium in Games of Imperfect Recall.. In *KR*. 278–287.

[12] Sergiu Hart and Andreu Mas-Colell. 2000. A simple adaptive procedure leading to correlated equilibrium. *Econometrica* 68, 5 (2000), 1127–1150.

[13] Sergiu Hart and Andreu Mas-Colell. 2001. A reinforcement procedure leading to correlated equilibrium. In *Economics Essays*. Springer, 181–200.

[14] Feng-Hsiung Hsu. 2006. *Behind Deep Blue: Building the Computer that Defeated the World Chess Championship*. Princeton University Press.

[15] Michael Johanson, Kevin Waugh, Michael Bowling, and Martin Zinkevich. 2011. Accelerating best response calculation in large extensive games. In *IJCAI*, Vol. 11. 258–265.

[16] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*. Springer, 282–293.

[17] Marc Lanctot. 2013. *Monte Carlo sampling and regret minimization for equilibrium computation and decision-making in large extensive form games*. Ph.D. Dissertation. University of Alberta.

[18] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. 2009. Monte Carlo sampling for regret minimization in extensive games. In *Advances in neural information processing systems*. 1078–1086.

[19] Viliam Lisy. 2014. Alternative selection functions for Information Set Monte Carlo tree search. *Acta Polytechnica* 54, 5 (2014), 333–340.

[20] Viliam Lisý, Marc Lanctot, and Michael Bowling. 2015. Online monte carlo counterfactual regret minimization for search in imperfect information games. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 27–36.

[21] Jeffrey Long, Nathan R Sturtevant, Michael Buro, and Timothy Furtak. 2010. Understanding the success of perfect information monte carlo sampling in game tree search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. AAAI Press, 134–140.

[22] Matej 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 heads-up no-limit poker. *Science* 356, 6337 (2017), 508–513.

[23] Matej Moravcik, Martin Schmid, Karel Ha, Milan Hladik, and Stephen J Gaukrodger. 2016. Refining Subgames in Large Imperfect Information Games.. In *AAAI*. 572–578.

[24] Martin J Osborne and Ariel Rubinstein. 1994. *A course in game theory*. MIT press.

[25] Eric Raboin, Dana Nau, Ugur Kuter, Satyandra K Gupta, and Petr Svec. 2010. Strategy generation in multi-agent imperfect-information pursuit games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 947–954.

[26] Martin Schmid, Neil Burch, Marc Lanctot, Matej Moravcik, Rudolf Kadlec, and Michael Bowling. 2018. Variance Reduction in Monte Carlo Counterfactual Regret Minimization (VR-MCCFR) for Extensive Form Games using Baselines. *arXiv preprint arXiv:1809.03057* (2018).

[27] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, and others. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[28] Fabien Teytaud and Olivier Teytaud. 2011. Lemmas on partial observation, with application to phantom games. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*. IEEE, 243–249.

[29] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. 2008. Regret minimization in games with incomplete information. In *Advances in neural information processing systems*. 1729–1736.

Online Monte Carlo Counterfactual Regret Minimization for Search in Imperfect Information Games

Viliam Lisý
Dept. Computer Science, FEE
Czech Technical University
in Prague, Czech Republic
lisy@agents.fel.cvut.cz

Marc Lanctot*
Dept. Knowledge Engineering
Maastricht University
The Netherlands
marc.lanctot@gmail.com

Michael Bowling
Dept. Computing Science
University of Alberta,
Edmonton, Canada
bowling@cs.ualberta.ca

ABSTRACT

Online search in games has been a core interest of artificial intelligence. Search in imperfect information games (*e.g.*, Poker, Bridge, Skat) is particularly challenging due to the complexities introduced by hidden information. In this paper, we present Online Outcome Sampling, an online search variant of Monte Carlo Counterfactual Regret Minimization, which preserves its convergence to Nash equilibrium. We show that OOS can overcome the problem of non-locality encountered by previous search algorithms and perform well against its worst-case opponents. We show that exploitability of the strategies played by OOS decreases as the amount of search time increases, and that preexisting Information Set Monte Carlo tree search (ISMCTS) can get more exploitable over time. In head-to-head play, OOS outperforms ISMCTS in games where non-locality plays a significant role, given a sufficient computation time per move.

Categories and Subject Descriptors

I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems—*Games*

General Terms

Algorithms

Keywords

Imperfect information games; online search; Nash equilibrium; Monte Carlo tree search; regret minimization

1. INTRODUCTION

Algorithms for creating strong strategies in games have been a core interest of artificial intelligence research. Strong strategies are useful not only as research benchmarks and opponents in digital entertainment, but they are becoming increasingly important also in real world applications, such as critical infrastructure security [36]. Because these games are often played against unknown opponents, a good strategy should be sufficiently robust against any possible counterstrategy used by the adversary. In strictly competitive games, the strategy that is optimal with respect to this criterion is the Nash equilibrium. It is guaranteed to provide the maximal possible expected value against any strategy of the opponent. For example, an equilibrium defense strategy minimizes the maximal damage caused by any attack.

When preparation time is abundant and the exact problem definition is known in advance, an equilibrium strategy for a smaller abstract game can be pre-computed and then used during game play. This *offline approach* has been remarkably successful in Computer Poker [2, 12, 32, 31, 21]. However, the preparation time is often very limited. The model of the game may become known only shortly before acting is necessary, such as in general game-playing, search or pursuit in a previously unknown environment, and general-purpose robotics. In these circumstances, creating and solving an abstract representation of the problem may not be possible. In these cases, agents must *decide online*: make initial decisions quickly and then spend effort to improve their strategy while the interaction is taking place.

In this paper, we propose a search algorithm that can compute an approximate Nash equilibrium (NE) strategy in two-player zero-sum games *online*, in a previously unknown game, and with a very limited computation time and space. Online Outcome Sampling (OOS) is a simulation-based algorithm based on Monte Carlo Counterfactual Regret Minimization (MCCFR) [24]. OOS makes two novel modifications to MCCFR. First, OOS builds its search tree incrementally, like Monte Carlo tree search (MCTS) [7, 3, 22]. Second, when performing additional computation after some moves have been played in the match, the following samples are targeted primarily to the parts of the game tree, which are more relevant to the current situation in the game. We show that OOS is *consistent*, *i.e.* it is guaranteed to converge to an equilibrium strategy as search time increases. To the best of our knowledge, this is not the case for any existing online game playing algorithm, all of which suffer from the problem of *non-locality* in these games.

We compare convergence and head-to-head performance of OOS to Information Set MCTS [8, 37, 25] in three fundamentally different games. We develop a novel methodology for evaluating convergence of online algorithms that do not produce a complete strategy. The results show that OOS, unlike ISMCTS, consistently converges close to NE strat-

*This author has a new affiliation: Google DeepMind, London, United Kingdom.

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*
Copyright © 2015, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

egy in all three games. In head-to-head performance, with sufficient time, OOS either ties or outperforms ISMCTS.

2. BACKGROUND AND RELATED WORK

Most imperfect information search algorithms are built upon perfect information search algorithms, such as minimax or Monte Carlo tree search (MCTS). One of the first popular applications of imperfect information search was Ginsberg's Bridge-playing program, GIB [15, 16]. In GIB, perfect information search is performed on a *determinized* instance of the game: one where all players can see usually hidden information. This approach has also performed well in other games like Scrabble [34], Hearts [35], and Skat [5].

Several problems have been identified with techniques based on determinization, such as Perfect Information Monte Carlo (PIMC) [28]. Two commonly reported problems are *strategy fusion* and *non-locality* [9]. Strategy fusion occurs when distinct actions are recommended in states that the player is not able to distinguish due to the imperfect information. Strategy fusion can be overcome by imposing the proper information constraints during search [9, 6, 30, 26, 8].

Non-locality occurs due to optimal payoffs not being recursively defined over subgames as in perfect information games. As a result, guarantees normally provided by search algorithms built on subgame decomposition no longer hold. Previous work has been done for accelerating offline equilibrium computation by solving subgames in end-game situations [13, 14, 11]. While these techniques tend to help in practice, non-locality can prevent them from producing equilibrium strategies; we discuss this further in Section 2.2.

The last problem is the need for randomized strategies in imperfect information games. In many games, any deterministic strategy predictable by the opponent can lead to arbitrarily large losses. To overcome this problem, MCTS algorithms sometimes randomize by sampling actions from its empirical frequency counts. Evidence suggests that the resulting strategies will not converge to the optimal solution even in very small imperfect information games like Biased Rock-Paper-Scissors and Kuhn poker [33, 30]. Our results show that, the empirical frequencies can *diverge away* from the equilibrium strategies. A little more promising are the MCTS approaches that use alternative selection functions, such as Exp3 or Regret Matching [25]. Variants of MCTS with these selection functions have been shown both formally and empirically [27, 23] to converge to Nash equilibrium in games with simultaneous moves, which are the simplest subclass of imperfect information games. However, we show that even these selection functions cannot help to overcome the problem of non-locality.

2.1 Extensive-Form Games

We focus on two-player zero-sum extensive form games and base our notation on [29]. These games model sequential decision making **players** denoted $i \in N = \{1, 2\}$. In turn, players choose **actions** leading to sequences called **histories** $h \in H$. A history $z \in Z$, where $Z \subseteq H$, is called a **terminal history** and represents a full game from start to end. At each terminal history z there is a payoff $u_i(z)$ to each player i. At each nonterminal history h, there is a single current player to act, determined by $P : H \backslash Z \to N \cup \{c\}$ where c is a special player called **chance** that plays with a fixed stochastic strategy. For example, chance is used to represent rolls of dice and card draws. The game starts in the empty history $\emptyset$, and at each step, given the current history h, the current player chooses an action $a \in A(h)$ leading to successor history $h' = ha$; in this case we call h a **prefix** of h' and denote this relationship by $h \sqsubset h'$. Also, for all $h, h', h'' \in H$, if $h \sqsubset h'$ and $h' \sqsubset h''$ then $h \sqsubset h''$, and $h \sqsubset h$. Sets H and $A(h)$ are finite and histories have finite length.

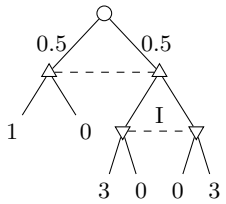


Figure 1: An extensive-form game demonstrating the problem of non-locality with maximizing △, minimizing ▽ and chance ◯ players.

Set $\mathcal{I}_i$ is an partition over $H_i = \{h \mid P(h) = i\}$ where each part is called an **information set**. Intuitively, an information set $I \in \mathcal{I}_i$ of player i represents a state of the game with respect to what player i knows. Formally, I is a set of histories that a player cannot tell apart (due to information hidden from that player). For all $h, h' \in I$, $A(h) = A(h')$ and $P(h) = P(h')$; hence, we extend the definition to $A(I)$, $P(I)$, and denote $I(h)$ the information set containing h.

A **behavioral strategy** for player i is a function mapping each information set $I \in \mathcal{I}_i$ to a probability distribution over the actions $A(I)$, denoted by $\sigma_i(I)$. For a profile σ, we denote the probability of reaching a terminal history z under σ as $\pi^\sigma(z) = \prod_{i \in N \cup c} \pi_i(z)$, where each $\pi_i(z) = \prod_{ha \sqsubset z, P(h)=i} \sigma_i(I(h), a)$ is a product of probabilities of the actions taken by player i along z. We use $\pi_i^\sigma(h, z)$ and $\pi^\sigma(h, z)$ for $h \sqsubset z$ to refer to the product of only the probabilities of actions along the sequence from the end of h to the end of z. Define Σ_i to be the set of behavioral strategies for player i. As is convention, $-i$ refers to player i's opponent.

An **ϵ-Nash equilibrium**, σ, is a set of σ_i for $i \in N$ such that the benefit of switching to some alternative σ_i' is limited by ϵ, i.e., $\forall i \in N : \max_{\sigma_i' \in \Sigma_i} u_i(\sigma_i', \sigma_{-i}) - u_i(\sigma) \leq \epsilon$. When $\epsilon = 0$, the profile is called a Nash equilibrium. When $|N| = 2$ and $u_1(z) + u_2(z) = k$ for all $z \in Z$, then the game is a zero-sum game. In these games, different equilibrium strategies result in the same expected payoff against any arbitrary opponent equilibrium strategy and at least the same payoff for any opponent strategy. The **exploitability** of a profile σ is the sum of strategies' distances from an equilibrium, $\epsilon_\sigma = \max_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \sigma_2) + \max_{\sigma_2' \in \Sigma_1} u_2(\sigma_1, \sigma_2')$.

In a **match** (online game), each player is allowed little or no preparation time before playing (preventing the offline computation of approximate equilibria solutions). The current **match history**, $\mathtt{m} \in H$, is initially the empty history $\emptyset$ representing the start of the match. Each turn, player $P(\mathtt{m})$ is given t time units to decide on a **match action** $\mathtt{a} \in A(\mathtt{m})$ and the match history then changes from $\mathtt{m}$ to $\mathtt{ma}$. There is a referee who knows $\mathtt{m}$, samples chance outcomes from $\sigma_c(\mathtt{m})$, and reveals $I(\mathtt{m})$ to player $P(\mathtt{m})$. The match terminates when $\mathtt{m} \in Z$, giving each player i a payoff of $u_i(\mathtt{m})$.

2.2 The Problem of Non-Locality

We demonstrate the problem of non-locality on the game in Figure 1. It starts with a decision of chance ◯ which leads to two nodes that cannot be distinguished by the maximiz-

ing player $\triangle$. On the left, the game ends after his actions. On the right, both his actions lead to information set I of the minimizing player $\triangledown$, whose utility is minus the depicted value. The optimal (Nash equilibrium) strategy in I depends on the value of the leftmost leaf, even though this outcome of the game cannot be reached once the game entered information set I. Because $\triangle$ does not know at the time of his move if $\triangledown$ will play, he needs to randomize its decision so that $\triangledown$ cannot easily make the move that is more harmful for $\triangle$. However, he would prefer to play left for the case that $\triangledown$ will not play. Player $\triangledown$ knows that $\triangle$ has this dilemma and tries to exploit it as much as possible. Uniform strategy at I clearly motivates $\triangle$ to play left, leading to expected utility $0.5 \cdot 1 + 0.5 \cdot 1.5 = 1.25$, instead of $0.5 \cdot 0 + 0.5 \cdot 1.5 = 0.75$. If $\triangledown$ plays right a little more often, $\triangle$ is still motivated to play left and $\triangledown$ improves his payoff. This holds until $\triangledown$ plays (left,right) with probabilities $(\frac{1}{3}, \frac{2}{3})$, which results to expected reward of 1 for both actions of $\triangle$. If $\triangle$ plays $(\frac{1}{2}, \frac{1}{2})$, $\triangledown$ cannot lower the utility anymore and the pair of strategies is an equilibrium. If the leftmost leaf was 2 instead of 1, the optimal strategy in I would be $(\frac{1}{6}, \frac{5}{6})$.

Current approaches, after reaching the information set I, will repeatedly sample and search one of subtrees in I, aggregating the information collected to make a final recommendation. However, even if the information structure is kept intact, such as in ISMCTS [8], the problem still occurs. If subtrees of I are sampled equally often, a searching player will not have any preference between left and right and will recommend $(\frac{1}{2}, \frac{1}{2})$, which is suboptimal. Moreover, the uniform probability of being in each state of I corresponds to the distribution over the states given the optimal play, so the problem occurs even if subtrees are sampled from the correct belief distribution. Note that this is a simple example; in larger games, this problem could occur over much longer paths or many times in different parts of the game. The analysis in [28] suggests that the effect may be critical in games with low *disambiguation factor*, where private information is slowly (or never) revealed throughout a match.

To the best of our knowledge, OOS is the first online search algorithm that solves this problem. It does it by starting each sample from the root of the game. If the computed strategy tends to come closer to the uniform strategy in I, the updates in the maximizing player's information set will modify the strategy to choose left more often. The following samples will reach I more often at the left state and consequently modify the strategy in I in the right direction.

2.3 Offline Equilibrium Approximation

There are multiple algorithms for computing approximate equilibrium strategies offline [32]. We focus on a popular choice among Poker researchers due to its sampling variants.

Counterfactual Regret (CFR) is a notion of regret at the information set level for extensive-form games [38]. The CFR algorithm iteratively learns strategies in self-play, converging to an equilibrium. The **counterfactual value** of reaching information set I is the expected payoff given that player i played to reach I, the opponents played σ_{-i} and both players played σ after I was reached:

$$v_i(I, \sigma) = \sum_{(h,z) \in Z_I} \pi^{\sigma}_{-i}(h) \pi^{\sigma}(h, z) u_i(z), \tag{1}$$

where $Z_I = \{(h, z) \mid z \in Z, h \in I, h \sqsubset z\}$. Suppose, at time t, player i plays with strategy σ^t_i. Define $\sigma^t_{I \to a}$ as identical to σ^t except at I action a is taken with probability 1. The counterfactual regret of not taking $a \in A(I)$ at time t is $r^t(I, a) = v_i(I, \sigma^t_{I \to a}) - v_i(I, \sigma^t)$. The algorithm maintains the cumulative regret $R^T(I, a) = \sum_{t=1}^{T} r^t(I, a)$, for every action at every information set. Then, the distribution at each information set for the next iteration $\sigma^{T+1}(I)$ is obtained using regret-matching [17]. The distribution is proportional to the positive portion of the individual actions' regret:

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

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

$$\bar{\sigma}^T(I, a) = \frac{\sum_{t=1}^{T} \pi^{\sigma^t}_i(I) \sigma^t(I, a)}{\sum_{t=1} \pi^{\sigma^t}_i(I)}, \tag{2}$$

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 [38], and hence 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 [24]. In the **outcome sampling** (OS) variant of the algorithm, a single terminal history $z \in Z$ is sampled in each iteration. The algorithm updates the regret in the information sets visited along z using the **sampled counterfactual value**,

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

where $q(z)$ is the probability of sampling z. If every $z \in Z$ has 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 values on the sampled information sets values also eventually converges to the approximate equilibrium of the game with high probability.

3. ONLINE OUTCOME SAMPLING

When outcome sampling is used in the offline setting, data structures for all information sets are allocated before the first iteration. In each iteration, all sampled information sets get updated. We make two essential modifications to adapt outcome sampling to the online setting.

Incremental Game Tree Building. Before the match begins, only the very first (root) information set is added to memory. In each iteration, a single information set (at most) is added to memory. In particular, when an information set is reached that is not in memory, it is added to memory and then a default playout policy (*e.g.*, uniform random) takes over for the remainder of the simulation. Along the playout portion (tail) of the simulation, information sets are not added to memory nor updated. Along the tree portion (head), information sets are updated as normal. This way, only the relevant information sets are stored in the memory.

In-Match Search Targeting. Suppose several moves have been played since the start of the match leading to `m`. Plain outcome sampling would continue to sample from the root of the game (not the current match history `m`), rarely reaching the region of the game space that the match has headed toward. Hence, the second modification we propose

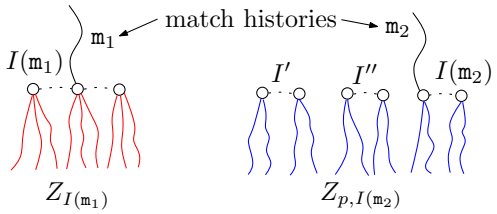


Figure 2: Example subgames targeted by IST (left) and PST (right). All histories $\{ h \mid h \in I' \cup I'' \cup I(\mathtt{m}_2) \}$ share a common public action sequence ($p(\mathtt{m}_2)$).

is directing the search towards the histories that are more likely to occur during the match currently played. Note that the complete history is typically unknown to the players, who know only their information sets. Unlike in ISMCTS, OOS always runs samples from the root of the game tree, even with non-empty match history. We now describe two specific targeting methods demonstrated in Figure 2.

3.1 Information Set Targeting (IST)

Suppose the match history is $\mathtt{m}$. IST samples histories reaching the current information set ($I(\mathtt{m})$), i.e., $(h, z) \in Z_{I(\mathtt{m})}$, with higher probability than other histories. The intuition is that these histories are particularly relevant since the searching player *knows* that one of these z will describe the match at its completion. However, focusing *only* on these histories may cause problems because of the non-locality and the convergence guarantees would be lost.

Consider again the game in Figure 1. If the minimizing player knows it is in the information set I and focuses all its search to this information set, she computes the suboptimal uniform strategy. Any fixed non-zero probability of sampling the left chance action eventually solves the problem. The regrets are multiplied by the reciprocal of the sampling probability; hence, they influence the strategy in the information set proportionally stronger if the samples are rare.

Note that previous methods, such as PIMC and ISMCTS, *never* update any information sets, which are not reachable from $I(\mathtt{m})$. In contrast, in IST *all* information sets in memory have positive probability of being reached and updated.

3.2 Public Subgame Targeting (PST)

IST may "reveal" information about the current information set available to only one of the players affecting the computed strategy (this vanishes in the limit due to consistency but could have a short-term effect). PST only uses information available to both players to target the search to prevent this effect. A **public action** is an action in the "public tree" defined in [20] and commonly used in offline CFR. Informally, an action is said to be public if it is observable by all players (e.g., bids in Liar's Dice and bets in Poker are public). Formally, an action a is public, iff $\forall i, \forall I \in \mathcal{I}_i, \forall h_1, h_2 \in I : a \in h_1 \Leftrightarrow a \in h_2$. For example, the extensive-form version of Rock, Paper, Scissors has two information sets $I_1 = \emptyset$ and $I_2 = \{r, p, s\}$; it has no public actions, because each history in I_2 contains a single unique action (the unobserved ones taken by the first player).

Given a history h, let $p(h)$ be the sequence of public actions along h in the same order that they were taken in h. Define the **public subgame** induced by I to be the one whose terminal history set is

$$Z_{p,I(h)} = \{(h', z) \mid z \in Z, h' \in H, p(h') = p(h), h' \sqsubset z\}.$$

With match history $\mathtt{m}$, PST samples $z \in Z_{p,I(\mathtt{m})}$ with higher probability than terminal histories outside this set.

In a game of poker, suppose the chance decides on the private cards of the players, the first player bets one chip and the second player calls. At this point, the public actions are: bet one chip and call. The public subgame $Z_{p,I(h)}$ contains every terminal history (including every combination of private chance outcomes for all players) whose first two public actions are: bet one chip and call.

```
1  OOS(h, π_i, π_{-i}, s_1, s_2, i):
2    if h ∈ Z then
3      return (1, δs_1 + (1 − δ)s_2, u_i(z))
4    else if P(h) = c then
5      Sample an outcome a and let ρ_1, ρ_2 be its
       probability in targeted and untargeted setting
6      (x, l, u) ← OOS(ha, π_i, ρ_2 π_{-i}, ρ_1 s_1, ρ_2 s_2, i)
7      return (ρ_2 x, l, u)
8    I ← getInfoset(h, P(h))
9    Let (a, s'_1, s'_2) ← Sample(h, I, i, ε)
10   if I is not in memory then
11     Add I to memory
12     σ(I) ← Unif(A(I))
13     (x, l, u) ← Playout(ha, (δs_1 + (1−δ)s_2) / |A(I)|)
14   else
15     σ(I) ← RegretMatching(r_I)
16     π'_{P(h)} ← σ(I, a) π_{P(h)}
17     π'_{−P(h)} ← π_{−P(h)}
18     (x, l, u) ← OOS(ha, π'_i, π'_{−i}, s'_1, s'_2, i)
19   c ← x
20   x ← xσ(I, a)
21   for a' ∈ A(I) do
22     if P(h) = i then
23       W ← uπ_{−i} / l
24       if a' = a then
25         r_I[a'] ← r_I[a'] + (c − x)W
26       else
27         r_I[a'] ← r_I[a'] − xW
28     else
29       s_I[a'] ← s_I[a'] + 1/(δs_1 + (1−δ)s_2) π_{−i} σ(I, a')
30   return (x, l, u)
```

Algorithm 1: Online Outcome Sampling.

3.3 Algorithm

The algorithm iteratively samples a single trajectory from the root $\emptyset$ to a terminal history. At each information set in memory, I, it maintains two tables: r_I stores cumulative regret for each action $a \in A(I)$, and s_I stores the cumulative average strategy probability of each action.

Depending on the targeting method (IST or PST), Z_{sub} is one of $Z_{I(\mathtt{m})}$ or $Z_{p,I(\mathtt{m})}$. The pseudo-code is presented as Algorithm 1. Each iteration is represented by two calls of OOS where the update player $i \in \{1, 2\}$ is alternated. Before each iteration, a *scenario* is decided: with probability δ the iteration targets the subgame and chooses $z \in Z_{sub}$ and with probability $(1 - \delta)$ the usual OS sampling determines

$z \in Z$. As a result, OOS with $\delta = 0$ becomes MCCFR with incremental tree building.

The first parameter of OOS is the current history. The next two are strategy's reach probabilities for the update player i and the opponent. The third and fourth parameters are initially the overall probabilities that the current sample is generated, one for each scenario: first the targeted and then the untargeted. With non-empty match history, these two parameters include an additional weighting factor w_T explained later. The last parameter is the update player. Initial calls have the form $\text{OOS}(\emptyset, 1, 1, 1/w_T, 1/w_T, i)$. For the return values, x is a suffix/tail reach probability for both players, l is the root-to-leaf sample probability, and u is the payoff of the trajectory in view of the update player.

The ϵ-on-policy sampling distribution used at each information set is defined as

$$\Phi(I,i) = \begin{cases} \epsilon \cdot \text{Unif}(A(I)) + (1-\epsilon)\sigma_i(I) & \text{if } P(I) = i \\ \sigma_i(I) & \text{otherwise,} \end{cases}$$

and denote $\Phi(I,i,a)$ the probability of sampling $a \in A(I)$.

The sampling at chance's choices on line 5 depends on the method and the scenario being used. For example, when using information set targeting, the outcome that is sampled must be consistent with match history.

A critical part of the algorithm is the action chosen and sample reach updates on line 9. If I is not in memory, then an action is sampled uniformly. Otherwise, in the targeted scenario, the current history h is always in the targeted part of the game and an action from $\{a \mid \exists z \in Z\ (ha, z) \in Z_{sub}\}$ is selected using the distribution $\Phi(I(h), i)$ normalized to one on this subset of actions. If we define $sum = \sum_{(ha,z)\in Z_{sub}} \Phi(I,i,a)$ then $s'_1 = s_1\Phi(I,i,a)/sum$. In the untargeted scenario, any action $a \sim \Phi(I,i)$ can be sampled. If the action is not leaving the targeted part of the game (i.e., $(ha,z) \in Z_{sub}$) then $s'_1 = s_1\Phi(I,i,a)/sum$ otherwise $s'_1 = 0$. In all cases $s'_2 = \Phi(I,i,a)s_2$.

These sample reach probabilities are combined into one at a terminal history on line 3, start of the playout on line 13, and when updating the average strategy on line 29.

The playout on line 13 samples to the end of the game with some playout policy at each step; we use uniform random, but in general one could use an informed policy based on domain knowledge as well. Unlike MCTS, the playout policy in OOS must compute l when reaching a terminal and update the tail probability x when returning as done on line 20. Lines 16 and 17 modify $P(h)$'s reach probability by multiplying it by $\sigma(I,a)$, keeping the other value the same. Lines 19 to 24 contain the usual outcome sampling updates. Note that regrets are updated at the update player histories, while average strategy tables at opponent histories.

Now we explain the role of the weighting factor w_T. Note that on lines 23 and 29, the regret and strategy updates are multiplied by the reciprocal of the probability of sampling the sample leaf and the current state. Consider the updates in the current information set of the game $I_{\mathtt{m}}$. In the initial samples of the algorithm with empty match history, this information set was on average sampled with a very low probability s_0. Let's say that out of 10^6 samples started from the root, 1000 samples reached this particular information set. In that case, $s_0 = 0.001$ and the regret updates caused by each of these samples were multiplied by $\frac{1}{s_0} = 1000$. Now the game has actually reached the history $\mathtt{m}$ and due to targeting, half of the next 10^6 samples reach $I(\mathtt{m})$. It means that $s_{\mathtt{m}} = 0.5$ and the regrets will be multiplied only by 2. As a results, the updates of the first (generally less precise) 1000 samples are all together weighted the same as the 5×10^5 later samples, which makes it almost impossible to compensate for the initial errors. In order to prevent this effect, we add the weighting factor w_T to compensate the change of targeting and make each of the samples have similar weight at $I(\mathtt{m})$. In our example, $w_T = \frac{s_{\mathtt{m}}}{s_0}$. More formally, when running the iterations from match history $\mathtt{m}$, we define the weighting factor as the probability of reaching $I(\mathtt{m})$ without any targeting divided by the probability of reaching $I(\mathtt{m})$ with the current targeting, assuming the players play according to the current mean strategy profile $\bar{\pi}$:

$$\frac{1}{w_T(\mathtt{m})} = (1-\delta) + \delta\frac{\sum_{(h,z)\in I(\mathtt{m})} \bar{\pi}(h)}{\sum_{z\in Z_{sub(\mathtt{m})}} \bar{\pi}(z)}. \tag{3}$$

OOS would quickly stop improving a strategy in information sets that are below an irrational move of the opponent ($\pi_{-i} = 0$). This cannot be avoided even if many samples are targeted to the information set. Therefore, we use a more explorative regret matching, $\sigma_\gamma^{T+1}(I,a) = \gamma/|A(I)| + (1-\gamma)\sigma^{T+1}(I,a)$, with $\gamma = 0.01$. This affects the worst-case regret bound at most by γ (due to the linearity of the expected utility), but allows for much better play.

3.3.1 Consistency

THEOREM 1. *Let $\bar{\sigma}^t_m(\delta, \mathtt{m})$ be a strategy produced by OOS with scheme IST or PST using $\delta < 1$ started from $\mathtt{m}$ run for t iterations, with exploration $\epsilon > 0$. For any $p \in (0,1], \varepsilon > 0$ there exists $t < \infty$ such that with probability $1-p$ the strategy $\bar{\sigma}^t_m(\delta, \mathtt{m})$ is a ε-equilibrium strategy.*

PROOF. (Sketch) Each terminal history has nonzero probability of being sampled, eventually every information set will be contained in memory. The algorithm then becomes MCCFR with a non-uniform sampling scheme. By [24, Theorem 5] OOS minimizes regret with high probability. The additional weighting of all updates of the algorithm by the same constant has no influence on the computed average strategy and the strategies computed by regret matching, because it uses only relative proportions of the regrets. □

4. EMPIRICAL EVALUATION

We now compare the exploitability and head-to-head performance of OOS and ISMCTS.

4.1 Information Set Monte Carlo Tree Search

We have implemented ISMCTS in the same framework with similar data representations as OOS. ISMCTS runs MCTS samples as in a perfect information game, but uses common statistics computed for the whole information set and not individual states in the selection phase. When initiated from a non-empty match history, it starts samples uniformly from the states in the current information set. We use two selection functions in our experiments. First, the commonly used UCT as suggested in [8]. We use two times the maximal game outcome as the exploration parameter C. In matches, we select the action with the highest number of iterations. For evaluating convergence, this would often have the maximum possible exploitability in the evaluated games.

Therefore, we use the distribution given by the empirical frequencies for this part of evaluation. Second, we use Regret Matching (RM) as the selection strategy, because it has been shown to empirically perform better in many games [25] and because it is based on the same principles that are behind CFR. For RM, we use exploration 0.2, the mean strategy for evaluating exploitability and samples from this strategy in head to head matches. In the following, we refer to ISMCTS with the corresponding selection function by only UCT/RM.

4.2 Evaluating Performance

In games like Poker and Liar's Dice, it is often critical to play in such a way that the opponent cannot easily infer the private information. This explains partly the success of CFR-based methods in the offline setting. In the online setting, since the tree is built incrementally, only partial strategies are produced. We are unaware of any methods for assessing the worst-case exploitability of strategies produced by an online search algorithm.

There is a brute-force approach that runs the search at each information set and computes exploitability of a strategy composed from strategies computed for these information sets. However, it requires $O(t|I|)$ seconds for a single run and multiple runs are required for statistically significant results. Instead, we propose an approximate multi-match **aggregate method**. It creates a global (accumulating) strategy data structure for each player. Then, it runs a fixed number of matches (500 in our experiments) of the search algorithm against a random opponent. In each information set reached in the match, the information computed (visit counts in ISMCTS, s_I in OOS) is added into the global strategy of the player who searched. Besides the information in the current information set, the information computed in all successor information sets is also added with a weight proportional to the number of visits of the information set. If an information set is never reached directly, this gives at least some estimate of how the strategy would behave there. If it is, the number of samples in the information set outweights the less precise estimates from the cases where it is not. In the information sets not added to the global strategy, a fixed random action is selected. The result is the exploitability of the global strategy.

4.3 Domains

Imperfect Information Goofspiel In II-GS(N), each player is given a private hand of bid cards with values 0 to $N-1$. A different deck of N point cards is placed face up in a stack. On their turn, each player bids for the top point card by secretly choosing a single card in their hand. The highest bidder gets the point card and adds the point total to their score, discarding the points in the case of a tie. This is repeated N times and the player with the highest score wins. In II-Goofspiel, the players only discover who won or lost a bid, but not the bid cards played. Also, we assume the point-stack is strictly increasing: 0, 1, $\dots N-1$. This way the game does not have non-trivial chance nodes, all actions are private and information sets have various sizes.

Liar's Dice LD(D_1,D_2), also known as Dudo, Perudo, and Bluff is a dice-bidding game. Each die has faces ⚀ to ⚅ and a star $\star$. Each player i rolls D_i of these dice without showing them to their opponent. Each round, players alternate by bidding on the outcome of all dice in play until one player "calls liar", *i.e.* claims that their opponent's latest bid does not hold. If the bid holds, the calling player loses; otherwise, she wins. A bid consists of a quantity of dice and a face value. A face of $\star$ is considered wild and counts as matching any other face. To bid, the player must increase either the quantity or face value of the current bid (or both). All actions in this game are public. The only hidden information is caused by chance at the beginning of the game. Therefore, the size of all information sets is identical.

Generic Poker GP(T, C, R, B) is a simplified poker game inspired by Leduc Hold'em. First, both players are required to put one chip in the pot. Next, chance deals a single private card to each player, and the betting round begins. A player can either *fold* (the opponent wins the pot), *check* (let the opponent make the next move), *bet* (add some amount of chips, as first in the round), *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, chance deals one public card on the table, and a 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, or (2) none of the players' cards matches the table card and her private card is higher than the private card of the opponent. If no player wins, the game is a draw and the pot is split. The parameters of the game are the number of types of the cards (T), the number of cards of each type (C), the maximum length of sequence of raises in a betting round (R), and the number of different sizes of bets (i.e., amount of chips added to the pot) for *bet*/*raise* actions. This game is similar to Liar's Dice in having only public actions. However, it includes additional chance nodes later in the game, which reveal part of the information not available before. Moreover, it has integer results and not just win/draw/loss.

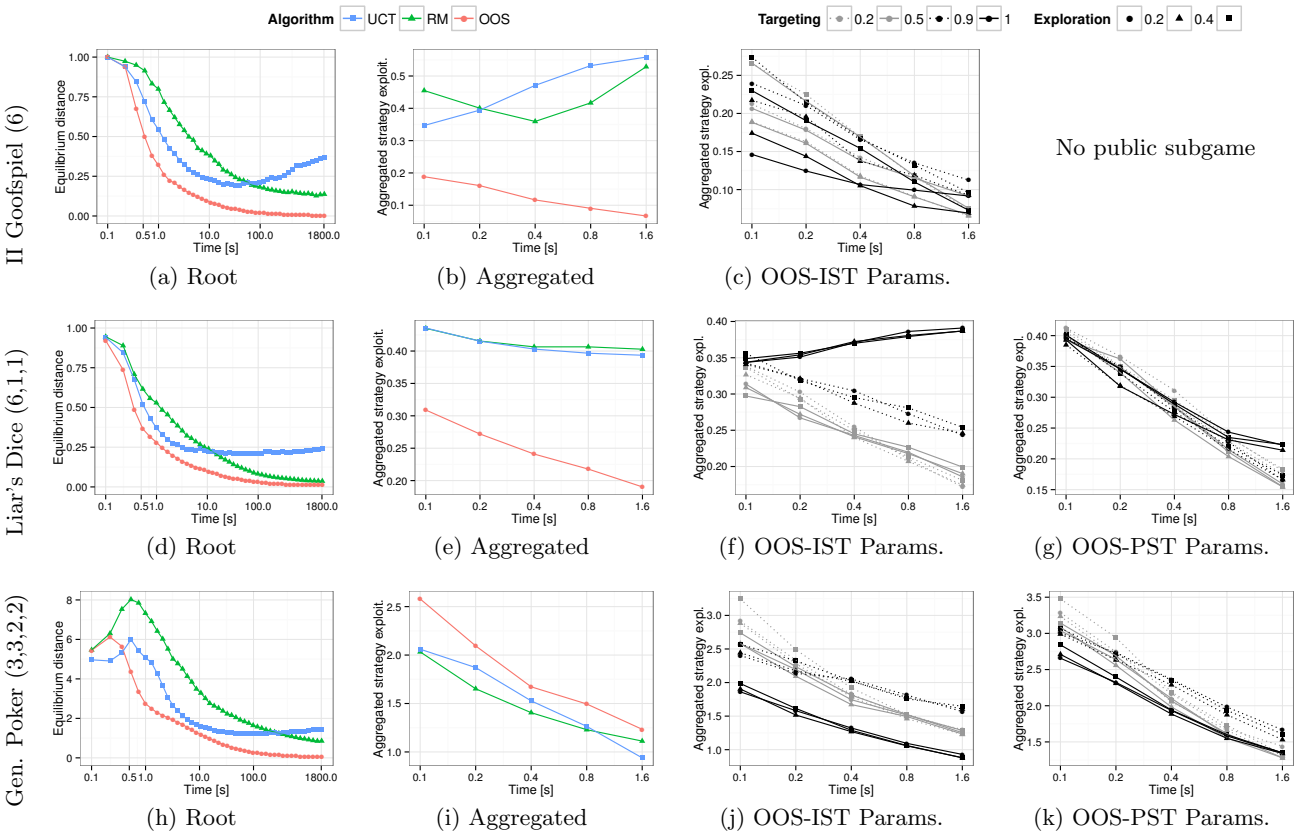
4.4 Results

We first focus on LD(1,1), II-GS(6), and GP(3,3,2,2). While these games are considered small by search algorithm standards, it is still possible to compute the exact worst case counterstrategies (i.e., best response) to measure exploitability. After analyzing the convergence, we proceed to performance in head-to-head matches also in larger variants of the games.

4.4.1 Convergence at the root

First we evaluate the speed of convergence of the proposed algorithm when run in the root of the games. The strategy in the information sets, which are not included in the tree yet, is assumed to be a fixed random action identical for all algorithms. The impact of the exploration parameters of the algorithms is not very strong here and we set it to 0.6 for OOS. The results are in Figures 3(a,d,h). OOS clearly converges the fastest in all games, eventually converging to the exact equilibrium. The exact convergence is not surprising, because the algorithm run from the root is MCCFR, only with incremental tree building.

In all three games, the empirical action frequencies in UCT first converge quite quickly towards less exploitable strategies, but they always start to become more exploitable at some point. This does not happen with RM, but its initial convergence is slower.



(a) Root (b) Aggregated (c) OOS-IST Params.

(d) Root (e) Aggregated (f) OOS-IST Params. (g) OOS-PST Params.

(h) Root (i) Aggregated (j) OOS-IST Params. (k) OOS-PST Params.

Figure 3: Exploitability ISMCTS with various selection functions and OOS with various parameter settings in the smaller game variants (rows).

4.4.2 *Aggregated strategy exploitability*

Figures 3(b,e,i) present the exploitability of the strategies computed by different algorithms estimated by the aggregate method. The OOS in this graphs is run with IST, $\gamma = 0.4$ for exploration and $\delta = 0.5$ for targeting. The x-axis denotes the amount of computation per move. In Goofspiel 3(b), both ISMCTS variants produce a more exploitable strategy with more computation time per move. For any of the evaluated time settings, OOS is substantially less exploitable. In Liar's Dice 3(e), ISMCTS does not seem to get worse with additional computation time, but it improves marginally with more than 0.4 seconds per move. OOS steadily approaches the equilibrium. In Poker 3(i), the exploitability is lower for ISMCTS variants than for OOS with the default settings. With the limited computation time, having more iterations to avoid bad moves seems to be more important than guessing and hiding the private information of the players.

The remaining plots in Figures 3 present how the exploitability of the strategies produced by OOS depends on the selected targeting scheme and parameters: the amount of targeting shown by color and the amount of exploration shown by point shapes. Figures 3(c,f,j) show the information set targeting and 3(g,k) show the public subgame targeting. Overall, OOS produces strategies closer to the equilibrium as the computation time per move increases. The only exception is IST with full targeting ($\delta = 1$) in Liar's Dice (Figure 3(f)). This confirms that sampling the histories that will certainly not occur in the game anymore is necessary to converge to the optimal strategy in LD. In the other two games, full targeting produces the least exploitable strategies with very short computation times. This indicates that sophisticated modeling of hidden information can be counterproductive with very short computation times. With more time per move, weaker targeting performs as well as the stronger targeting or even better in case of PST.

4.4.3 *Head-to-head performance*

After we confirmed that we achieved the goal of creating an online game playing algorithm that converges close to Nash equilibrium, we evaluate its game playing performance in head-to-head matches with ISMCTS. For each game, we focus on two sizes: the smaller variant, for which we analyzed the exploitability in the previous section and a substantially larger variant that is closer to the size of the domains where online search algorithms are typically applied. The largest domain is II-GS(13) with approximately 3.8×10^{19} terminal histories.

The results are summarized in Figure 4. For Goofspiel and Liar's Dice, we present the win-rate, where a tie is considered half win half loss. II-GS is perfectly balanced for both players. In LD, the first player has a slight disadvantage corresponding to approximately 1% in the win rate and we do not compensate for that. Poker is not balanced. If both players play the optimal strategy, the first player loses 0.11 chips on average. This value is practically the

0.1s	OOS	UCT	RM	RND
OOS	49.3(2.7)	50.0(1.3)	50.5(1.3)	83.1(1.5)
UCT	51.2(2.2)	62.9(2.6)	62.7(2.5)	84.0(2.1)
RM	52.3(1.4)	70.6(1.7)	73.1(2.1)	87.8(1.4)
RND	17.1(2.2)	15.7(2.2)	10.3(1.8)	49.7(3.0)

(a) II Goofspiel (6)

0.1s	OOS	UCT	RM	RND
OOS	48.5(2.0)	51.7(2.0)	56.8(2.8)	83.3(1.5)
UCT	45.7(2.0)	49.7(2.0)	52.9(2.8)	87.2(1.3)
RM	47.9(2.8)	49.0(2.8)	54.3(2.8)	84.8(2.0)
RND	16.9(1.5)	19.5(1.6)	18.1(2.2)	47.7(2.0)

(b) Liar's Dice (1,1)

0.1s	OOS	UCT	RM	RND
OOS	0.25(0.53)	-0.45(0.10)	-0.40(0.10)	1.85(0.16)
UCT	0.53(0.10)	0.21(0.15)	0.20(0.15)	1.35(0.36)
RM	0.39(0.13)	0.21(0.40)	-0.12(0.32)	1.75(0.44)
RND	-2.17(0.20)	-2.22(0.54)	-3.24(0.55)	1.07(0.61)

(c) Generic Poker (3,3,2,2)

1s	OOS	UCT	RM	RND
OOS	49.2(2.5)	28.3(3.9)	35.1(4.2)	83.0(3.2)
UCT	69.2(4.0)	70.0(2.8)	61.0(3.0)	90.6(1.8)
RM	67.5(4.0)	73.8(2.7)	67.5(2.9)	92.8(1.5)
RND	19.6(3.4)	6.2(1.5)	4.9(1.3)	49.0(3.0)

(d) II Goofspiel (13)

5s	OOS	UCT	RM	RND
OOS	49.9(3.7)	55.4(2.3)	53.3(3.7)	91.3(1.7)
UCT	49.8(2.2)	56.2(3.1)	49.8(3.1)	93.7(1.5)
RM	53.3(3.5)	51.7(4.0)	51.0(4.0)	88.7(2.0)
RND	7.3(1.6)	12.9(2.1)	12.7(2.1)	47.3(3.0)

(e) Liar's Dice (2,2)

1s	OOS	UCT	RM	RND
OOS	-0.54(0.47)	-1.79(0.27)	-2.19(0.32)	8.36(0.54)
UCT	-0.07(0.35)	0.22(0.10)	-0.76(0.18)	9.48(0.40)
RM	-0.12(0.38)	-0.28(0.17)	-0.97(0.22)	9.09(0.44)
RND	-3.25(0.43)	-3.91(0.27)	-4.12(0.31)	2.51(0.51)

(f) Generic Poker (4,6,4,4)

Figure 4: Head-to-head matches in small (a-c) and large (d-f) game variants. The top left corner indicates the computation time per move and the brackets show 95% confidence intervals. OOS run with IST for II-GS and PST for the other games, targeting $\delta = 0.9$ and exploration $\epsilon = 0.4$; UCT with $C = 2 \cdot \max$; RM with $\gamma = 0.2$.

same for both evaluated sizes of the game. For clarity, we present the difference between this game value and the mean reward obtained by the players in the matches. A value 0.1 in the tables means that the row player on average wins 0.1 more than the game value from the column player per match. Each entry in the tables presents the result of at least 500 mutual matches between two algorithms from the perspective of the one in the row header. The values in the brackets are the size of half of the 95% confidence intervals centered in the presented mean. If it is not specified otherwise, OOS was run with IST for II-Goofspiel and PST for the other games, targeting $\delta = 0.9$ and exploration $\epsilon = 0.4$.

Figure 4(a) presents the results for the smaller Goofspiel and 0.1 second of computation per move. Even though the game is balanced, the results of ISMCTS are strongly biased towards the first player. The second player has generally larger information sets and needs to better evaluate the hidden information to choose a good move. OOS does not have this problem in the smaller game. It is the only algorithm that managed to prevent ISMCTS from exploiting it. In both positions, the algorithm won approximately 50% of matches, which corresponds to the value of the game. On the other hand, the first line in the table shows that OOS did not exploit the weaker play of ISMCTS, even though it is possible. This is a common disadvantage of Nash equilibrium play. One simple way to overcome this would be to trade a bit of exploitability to gain some exploitation; computing a restricted Nash response using MCRNR [30] (a minor modification of MCCFR) allows the best possible trade-off for a given importance between exploitability and exploitation. In the large variant of Goofspiel (see Figure 4(d)) with one second per move, the situation is different. OOS does not manage to converge sufficiently close to the equilibrium and ISMCTS exploits it from both positions to a similar extent. Increasing the computation time to 5 seconds per move helps OOS to win 35% matches against UCT, but it would likely need a substantially longer time to reach the equilibrium.

The results on Liar's Dice are even more promising for OOS. In the smaller game with 0.1 second per move (Figure 4(b)), OOS statistically significantly wins over both variants of ISMCTS from at least one position. This is not the case for the larger game and 1 second per move, where OOS already wins only 45% and 40% of matches against UCT and RM from the first position and loses 69% and 66% from the second position. However, with 5 seconds and the exploration parameter set to $\epsilon = 0.8$ to balance the need for exploring a large number of actions in this game, OOS again wins over UCT and ties with RM (Figure 4(e)).

As indicated by the exploitability results, OOS performs the worst against ISMCTS in the Poker domain. In the smaller variant (Figure 4(c)), it is losing from both positions. The result of OOS from the first position against UCT improves to -0.22 with 5 seconds per move, but even more time would be required to tie. From the first (generally more difficult) position, the situation is similar also in the larger Poker (Figure 4(f)). It is not the case for the second position, where OOS is able to tie both ISMCTS variants. This could be because Poker games have very high variance.

5. CONCLUSION

We have introduced Online Outcome Sampling, the first online game playing algorithm for imperfect information games that is guaranteed to produce approximate equilibrium strategies as search time per move increases. In order to achieve this, we do not modify a perfect information search algorithm to avoid the main problems caused by imperfect information, as in previous work. We instead start from an offline equilibrium computation method and adapt it to make the most of the limited time it has by using the information about the current position in the match. Besides incremental tree building, we propose two methods for targeting relevant parts of the game based on the current match history. In empirical evaluation, we show that a player using ISMCTS can suffer almost arbitrarily large loss when the opponent knows the algorithm she uses. The expected worst-case loss of using OOS decreases close to its minimum with additional search time. Furthermore, even with these guarantees, we can achieve comparable game playing performance to ISMCTS in mutual matches.

In future work, we hope to investigate more and larger games and the effect of informed playout policies. We hope to compare practical performance against other baseline algorithms such as PIMC [28], MMCTS [1], IIMC [10], and Smooth UCT [18]. Finally, one interesting question is whether the subgame decomposition ideas of [4, 19] could be adapted to the online search setting.

6. ACKNOWLEDGMENTS

This work is partially funded by the Czech Science Foundation, grant P202/12/2054 and 15-23235S; and by the Netherlands Organisation for Scientific Research (NWO) in the framework of the project Go4Nature, grant 612.000.938.

REFERENCES

[1] D. Auger. Multiple tree for partially observable Monte-Carlo tree search. In *Applications of Evolutionary Computation (EvoApplications 2011), Part I*, volume 6624 of *LNCS*, pages 53–62, 2011.

[2] M. Bowling, N. Burch, M. Johanson, and O. Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149, 2015.

[3] C. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012.

[4] N. Burch, M. Johanson, and M. Bowling. Solving imperfect information games using decomposition. In *28th AAAI Conference on Artificial Intelligence*, 2014.

[5] M. Buro, J. Long, T. Furtak, and N. Sturtevant. Improving state evaluation, inference, and search in trick-based card games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1407–1413, 2009.

[6] P. Ciancarini and G. Favini. Monte Carlo tree search in Kriegspiel. *Artificial Intelligence*, 174(11):670–684, 2010.

[7] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Proceedings of the 5th international conference on Computers and games*, CG'06, pages 72–83. Springer-Verlag, 2007.

[8] P. I. Cowling, E. J. Powley, and D. Whitehouse. Information set Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):120–143, December 2012.

[9] I. Frank, D. Basin, and H. Matsubara. Finding optimal strategies for imperfect information games. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 500–507, 1998.

[10] T. Furtak and M. Buro. Recursive Monte Carlo search for imperfect information games. In *IEEE Conference on Computational Intelligence in Games (CIG 2013)*, 2013.

[11] S. Ganzfried and T. Sandholm. Improving performance in imperfect-information games with large state and action spaces by solving endgames. In *AAAI Workshop on Computer Poker and Incomplete Information*, 2013.

[12] A. Gilpin. *Algorithms for Abstracting and Solving Imperfect Information Games*. PhD thesis, Carnegie Mellon University, 2009.

[13] A. Gilpin and T. Sandholm. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence*, pages 1453–1454, 2006.

[14] A. Gilpin and T. Sandholm. Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em poker. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2007.

[15] M. Ginsberg. Partition search. In *Proceedings of the Thirteenth National Conference on Artitificial Intelligence (AAAI)*, pages 228–233, 1996.

[16] M. Ginsberg. GIB: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, 14:303–358, 2001.

[17] S. Hart and A. Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

[18] J. Heinrich and D. Silver. Self-play Monte-Carlo tree search in computer poker. In *Proceedings of the AAAI Workshop on Computer Poker and Perfect Information*, 2014.

[19] E. Jackson. A time and space efficient algorithm for approximately solving large imperfect information games. In *Proceedings of the AAAI Workshop on Computer Poker and Perfect Information*, 2014.

[20] M. Johanson, N. Bard, M. Lanctot, R. Gibson, and M. Bowling. Efficient Nash equilibrium approximation through Monte Carlo counterfactual regret minimization. In *Proceedings of the Eleventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2012.

[21] M. Johanson, N. Burch, R. Valenzano, and M. Bowling. Evaluating state-space abstractions in extensive-form games. In *Proceedings of the 12th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2013.

[22] L. Kocsis and C. Szepesvári. Bandit-based Monte Carlo planning. In *15th European Conference on Machine Learning*, volume 4212 of *LNCS*, pages 282–293, 2006.

[23] M. Lanctot, V. Lisý, and M. H. Winands. Monte Carlo tree search in simultaneous move games with applications to Goofspiel. In *Computer Games*, volume 408 of *Communications in Computer and Information Science*, pages 28–43. Springer, 2014.

[24] M. Lanctot, K. Waugh, M. Bowling, and M. Zinkevich. Sampling for regret minimization in extensive games. In *Advances in Neural Information Processing Systems (NIPS 2009)*, pages 1078–1086, 2009.

[25] V. Lisy. Alternative selection functions for information set Monte Carlo tree search. *Acta Polytechnica: Journal of Advanced Engineering*, 54(5):333–340, 2014.

[26] V. Lisy, B. Bosansky, and M. Pechoucek. Anytime algorithms for multi-agent visibility-based pursuit-evasion games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1301–1302, 2012.

[27] V. Lisy, V. Kovarik, M. Lanctot, and B. Bosansky. Convergence of Monte Carlo tree search in simultaneous move games. In *Advances in Neural Information Processing Systems 26*, pages 2112–2120, 2013.

[28] J. Long, N. R. Sturtevant, M. Buro, and T. Furtak. Understanding the success of perfect information Monte Carlo sampling in game tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 134–140, 2010.

[29] M. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[30] M. Ponsen, S. de Jong, and M. Lanctot. Computing approximate Nash equilibria and robust best-responses using sampling. *Journal of Artificial Intelligence Research*, 42:575–605, 2011.

[31] J. Rubin and I. Watson. Computer poker: A review. *Artificial Intelligence*, 175(5–6):958–987, 2011.

[32] T. Sandholm. The state of solving large incomplete-information games, and application to poker. *AI Magazine*, 31(4):13–32, 2010.

[33] M. Shafiei, N. R. Sturtevant, and J. Schaeffer. Comparing UCT versus CFR in simultaneous games. In *IJCAI Workshop on General Game-Playing (GIGA)*, pages 75–82, 2009.

[34] B. Sheppard. World-championship-caliber scrabble. *Artificial Intelligence*, 134:241–275, 2002.

[35] N. R. Sturtevant. An analysis of UCT in multi-player games. *ICGA Journal*, 31(4):195–208, 2008.

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

[37] D. Whitehouse, P. Cowling, E. Powley, and J. Rollason. Integrating Monte Carlo tree search with knowledge-based methods to create engaging play in a commercial mobile game. In *9th Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 100–106, 2013.

[38] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, 2008.

Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)

An Algorithm for Constructing and Solving Imperfect Recall Abstractions of Large Extensive-Form Games

Jiří Čermák, Branislav Bošanský, Viliam Lisý
Department of Computer Science, Czech Technical University in Prague
{cermak, bosansky, lisy}@agents.fel.cvut.cz

Abstract

We solve large two-player zero-sum extensive-form games with perfect recall. We propose a new algorithm based on fictitious play that significantly reduces memory requirements for storing average strategies. The key feature is exploiting imperfect recall abstractions while preserving the convergence rate and guarantees of fictitious play applied directly to the perfect recall game. The algorithm creates a coarse imperfect recall abstraction of the perfect recall game and automatically refines its information set structure only where the imperfect recall might cause problems. Experimental evaluation shows that our novel algorithm is able to solve a simplified poker game with $7 \cdot 10^5$ information sets using an abstracted game with only 1.8% of information sets of the original game. Additional experiments on poker and randomly generated games suggest that the relative size of the abstraction decreases as the size of the solved games increases.

1 Introduction

Dynamic games with a finite number of moves can be modeled as extensive-form games (EFGs) that are general enough to represent scenarios with stochastic events and imperfect information. EFGs can model recreational games, such as poker, as well as real-world situations in physical security [Lisý *et al.*, 2016], auctions, or medicine [Chen and Bowling, 2012]. 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 solving EFGs. First, the online (or game-playing) algorithms which given the observations of the game state compute the action to be played. Second, the 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 [Bošanský *et al.*, 2016]. One exception is the recently introduced continual resolving algorithm used in DeepStack [Moravčík *et al.*, 2017], which provides less exploitable strategies than existing offline algorithms in heads-up no-limit Texas Hold'em, an imperfect information game with 10^{160} decision points. The main caveat is that DeepStack exploits the specific structure of poker where all actions are observable, and the generalization to other games is not straightforward. We thus focus on offline algorithms.

Most of the existing offline algorithms [von Stengel, 1996; Zinkevich *et al.*, 2008] require players to remember all the information gained during the game – a property denoted as a *perfect recall*. The main disadvantage of perfect recall is that it causes the size of strategies (a randomized selection of an action in each information set) to grow exponentially with the number of moves. Therefore, a popular approach is to use *abstractions* [Gilpin *et al.*, 2007] – create an abstracted game by merging information sets to reduce the size of the strategy representation, solve the abstracted game, and translate the strategies back to the original game. The majority of existing algorithms create perfect recall abstractions, where the requirement of perfect memory severely limits possible reductions in the size of strategies of the abstracted game, as it still grows exponentially with increasing number of moves in the abstracted game (e.g., see [Gilpin and Sandholm, 2007; Kroer and Sandholm, 2014; Brown and Sandholm, 2015]). Additionally, finding optimal perfect recall abstractions is computationally hard [Kroer and Sandholm, 2014].

A limited amount of work relaxes the perfect recall restriction in abstractions. Very specific imperfect recall abstractions that allow using perfect recall solution techniques are *(skew) well-formed games* [Lanctot *et al.*, 2012; Kroer and Sandholm, 2016] and *normal-form games with sequential strategies* [Bošanský *et al.*, 2015; Lisý *et al.*, 2016]. Skew well-formed games only merge information sets, which satisfy strict restrictions on the structure of the game tree above and below them, such that for all possible strategies of the opponent, a strategy which is optimal in one of the information sets must have bounded error in the other merged sets. Players cannot observe actions of the opponent at all in normal-form games with sequential strategies. These restrictions prevent us from creating sufficiently small and useful abstracted games and thus fully exploit the possibilities of imperfect recall. Existing methods for using imperfect recall abstractions without severe limitations cannot provide any guarantees of the quality of computed strategies [Waugh *et al.*, 2009], or assume that the abstraction is given and require computationally complex algorithms to solve it [Čermák *et al.*, 2017].

We address these issues in this paper and demonstrate the possible space savings achievable by automated imperfect recall abstractions. We create a novel domain-independent algorithm FPIRA, which starts with creating a coarse imperfect recall abstraction of the given perfect recall game. FPIRA then uses the fictitious play framework to simultaneously solve and refine the imperfect recall abstraction, to guarantee the convergence to Nash equilibrium of the original perfect recall game. FPIRA is conceptually similar to Double Oracle algorithm (DO) [Bošanský *et al.*, 2014] since it creates a smaller version of the original game and repeatedly refines it until the optimal solution of the original game is found. FPIRA, however, uses a game with imperfect recall during the computation, while DO uses a smaller perfect recall game. Hence, FPIRA exploits completely different type of sparseness than DO. The experimental evaluation shows that FPIRA can solve modification of Leduc hold'em with $7 \cdot 10^5$ information sets using an abstracted game with 1.8% of information sets of the original game. Furthermore, experiments on other simplified pokers and random games suggest that the relative size of the needed abstractions significantly decreases as the size of the solved games increases.

2 Extensive-Form Games

We first describe necessary technical introduction to extensive-form games (EFGs). A two player EFG G is a tuple $\{\mathcal{P}, \mathcal{H}, \mathcal{Z}, P, u, \mathcal{I}, A\}$. $\mathcal{P} = \{1, 2\}$ denotes the set of players. We use i to refer to a player and $-i$ to refer to the opponent of i. Set $\mathcal{H}$ contains all the states of the game. $P : \mathcal{H} \to \mathcal{P} \cup \{c\}$ is the function associating a player from $\mathcal{P}$ or nature c with every $h \in \mathcal{H}$. Nature c represents the stochastic environment of the game. $\mathcal{Z} \subseteq \mathcal{H}$ is a set of terminal states. $u_i(z)$ is a utility function assigning to each leaf the value of preference for player i; $u_i : \mathcal{Z} \to \mathbb{R}$. For zero-sum games it holds that $u_i(z) = -u_{-i}(z), \forall z \in \mathcal{Z}$. The imperfect information is defined using the information sets. $\mathcal{I}_i$ is a partitioning of all $\{h \in \mathcal{H} : P(h) = i\}$ into these information sets. All states h contained in one information set $I_i \in \mathcal{I}_i$ are indistinguishable to player i. The set of available actions $A(h)$ is the same $\forall h \in I_i$. We overload the notation and use $A(I_i)$ as actions available in I_i. A *sequence* σ_i is a list of actions of player i ordered by their occurrence on the path from the root of the game tree to some node. By $seq_i(h)$ we denote the sequence of player i leading to the state h. We overload the notation and use $seq_i(I)$ as a set of sequences of player i leading to the information set I. A game has *perfect recall* iff $\forall i \in \mathcal{P}\ \forall I_i \in \mathcal{I}_i$, for all the states $h, h' \in I_i$ 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*.

Definition 1. *By the* coarsest perfect recall refinement *of an imperfect recall game G we define a perfect recall game G' where we split the imperfect recall information sets to largest subsets satisfying the perfect recall assumption.*

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 $\forall I_i \in \mathcal{I}_i$ a member of $A(I_i)$. $\mathcal{S}_i$ 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 $A(I_i)$ for each I_i. $\mathcal{B}_i$ is a set of all behavioral strategies for i, $\mathcal{B}_i^p \subseteq \mathcal{B}_i$ denotes the set of deterministic behavioral strategies for i. A *strategy profile* is a set of strategies, one strategy for each player. 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 their expressive power can differ in imperfect recall games [Kuhn, 1953]. Moreover, the size of these representations differs significantly. Mixed strategies of player i use probability distribution over $\mathcal{S}_i$, where $|\mathcal{S}_i| \in \mathcal{O}(e^{|\mathcal{Z}|})$. Behavioral strategies create probability distribution over the set of actions (its size is proportional to the number of information sets, which can be exponentially smaller than $|\mathcal{Z}|$). Hence we need to use behavioral strategies if we want to exploit the space savings caused by the reduced number of information sets due to some information abstraction.

A *best response* of player i against b_{-i} is a strategy $b_i^{BR} \in BR(b_{-i})$, where $u_i(b_i^{BR}, b_{-i}) \geq u_i(b'_i, b_{-i})$ for all $b'_i \in \mathcal{B}_i$ ($BR(b_{-i})$ denotes a set of all best responses to b_{-i}).

Definition 2. *We say that b_i and b'_i are* realization equivalent *if for any b_{-i}, $\forall z \in \mathcal{Z}\ \pi^b(z) = \pi^{b'}(z)$, where $b = (b_i, b_{-i})$ and $b' = (b'_i, b_{-i})$ and $\pi^b(z)$ stands for the probability that z is reached when playing accordint to b.*

The concept of realization equivalence can be applied also to different strategy representations. Finally, we define the Nash equilibrium in behavioral strategies.

Definition 3. *We say that strategy profile $b = \{b_i, b_{-i}\}$ is a Nash equilibrium (NE) in behavioral strategies iff $\forall i \in \mathcal{P}\ \forall b_i^p \in \mathcal{B}_i^p : u_i(b_i, b_{-i}) \geq u_i(b_i^p, b_{-i})$.*

3 Fictitious Play

Fictitious play (FP) is an iterative algorithm defined on normal-form games [Brown, 1949]. It keeps track of average strategies of both players $\bar{m}_i^T, \bar{m}_{-i}^T$. Players take turn updating their average strategy. In iteration T, player i computes $s_i^T \in BR(\bar{m}_{-i}^{T-1})$. He then updates his average strategy $\bar{m}_i^T = \frac{T_i - 1}{T_i} \bar{m}_i^{T-1} + \frac{1}{T_i} s_i^T$ (T_i is the number of updates performed by i plus 1). In two-player zero-sum games $\bar{m}_i^T, \bar{m}_{-i}^T$ converge to a NE [Robinson, 1951]. There is a long-standing conjecture [Karlin, 2003; Daskalakis and Pan, 2014] that the convergence rate of FP is $O(T^{-\frac{1}{2}})$, the same order as the convergence rate of Counterfactual Regret Minimization (CFR) [Zinkevich *et al.*, 2008] (though the empirical convergence of CFR tends to be better).

When applying FP to behavioral strategies in perfect recall zero-sum EFG G', one must update the average behavioral strategy $\bar{b}_i^t$ such that it is realization equivalent to $\bar{m}_i^t$ obtained when solving the normal form game corresponding to G' for all t and all $i \in \mathcal{P}$ to keep the convergence guarantees. To update the behavioral strategy in such a way we use the following Lemma [Heinrich *et al.*, 2015].

Lemma 1. *Let b_i, b'_i be two behavioral strategies and m_i, m'_i two mixed strategies realization equivalent to b_i, b'_i, and $\lambda_1, \lambda_2 \in (0,1)$, $\lambda_1 + \lambda_2 = 1$. Then $\forall I \in \mathcal{I}_i$*

$$b''_i(I) = b_i(I) + \frac{\lambda_2 \pi_i^{b'_i}(I)}{\lambda_1 \pi_i^{b_i}(I) + \lambda_2 \pi_i^{b'_i}(I)}(b'_i(I) - b_i(I)),$$

where $\pi_i^{b_i}(I)$ is the probability that I is visited when playing b_i, defines a behavioral strategy b''_i realization equivalent to the mixed strategy $m''_i = \lambda_1 m_i + \lambda_2 m'_i$.

4 The FPIRA Algorithm

Let us now describe the main algorithm (denoted as FPIRA, Fictitious Play for Imperfect Recall Abstractions) presented in this paper and prove its convergence in two-player zero-sum EFGs. We give a high-level idea behind FPIRA, and we provide a pseudocode with the description of all steps.

Given a perfect recall game G', FPIRA creates a coarse imperfect recall abstraction G of G'. The algorithm then follows the FP procedure. It keeps track of average strategies of both players in the information set structure of G and updates the strategies in every iteration based on the best responses to the average strategies computed directly in G'. To ensure the convergence to Nash equilibrium of G', FPIRA refines the information set structure of G when needed to make sure that the strategy update does not lead to more exploitable average strategies in the following iterations compared to the strategy update made directly in G'.

Algorithm 1: FPIRA algorithm

input : G', T
output : $\bar{b}_i^T, \bar{b}_{-i}^T, G^T$

1 $G^1 \leftarrow$ `BuildAbstraction`(G')
2 $\bar{b}_1^0 \leftarrow$ `PureStrat`$(G^1), \bar{b}_2^0 \leftarrow$ `PureStrat`(G^1)
3 **for** $t \in \{1, ..., T\}$ **do**
4 $\quad i \leftarrow$ `ActingPlayer`(t)
5 $\quad b_i^t \leftarrow BR(G', \bar{b}_{-i}^{t-1})$
6 $\quad G^t \leftarrow$ `RefineForBR`(G^t, b_i^t)
7 $\quad \hat{b}_i^t \leftarrow$ `UpdateStrategy`$(G^t, \bar{b}_i^{t-1}, b_i^t)$
8 $\quad \widetilde{b}_i^t \leftarrow$ `UpdateStrategy`$(G', \bar{b}_i^{t-1}, b_i^t)$
9 $\quad$ **if** `ComputeDelta`$(G', \hat{b}_i^t, \widetilde{b}_i^t) > 0$ **then**
10 $\quad\quad G^{t+1} \leftarrow$ `Refine`$(G^t), \bar{b}_i^t \leftarrow \widetilde{b}_i^t$
11 $\quad$ **else**
12 $\quad\quad G^{t+1} \leftarrow G^t, \bar{b}_i^t \leftarrow \hat{b}_i^t$

In Algorithm 1 we present the pseudocode of FPIRA. FPIRA is given the original perfect recall game $G' = \{\mathcal{P}, \mathcal{H}, \mathcal{Z}, P, u, \mathcal{I}', A'\}$ and a number of iterations to perform T. FPIRA first creates a coarse imperfect recall abstraction $G^1 = \{\mathcal{P}, \mathcal{H}, \mathcal{Z}, P, u, \mathcal{I}^1, A^1\}$ of G' (line 1) as described in Section 4.1. Next, it initializes the strategies of both players to an arbitrary pure strategy in G^1 (line 2). FPIRA then performs T iterations. In every iteration it updates the average strategy of one of the players and if needed the information set structure of the abstraction (the game used in iteration t is denoted as $G^t = \{\mathcal{P}, \mathcal{H}, \mathcal{Z}, P, u, \mathcal{I}^t, A^t\}$). In every iteration player i first computes the best response b_i^t to $\bar{b}_{-i}^{t-1}$ in G' (line 5). Since b_i^t is computed in G', FPIRA first needs to make sure that the structure of information sets in G^t allows b_i^t to be played. If not, G^t is updated as described in Section 4.2, Case 1 (line 6). Next, FPIRA computes $\hat{b}_i^t$ as the strategy resulting from the update in abstracted G^t (line 7) and $\widetilde{b}_i^t$ as the strategy resulting from the update in original G' (line 8). FPIRA then checks whether the update in G^t changes the expected values of the pure strategies of the opponent compared to the update in G' using $\hat{b}_i^t$ and $\widetilde{b}_i^t$ (line 9, Section 4.2 Case 2). If yes, FPIRA refines the information set structure of G^t, creating G^{t+1} such that no error in expected values of pure strategies of the opponent is created (Section 4.2 Case 2), sets $\bar{b}_i^t = \widetilde{b}_i^t$ (line 10), and continues using G^{t+1}. If there is no need to update the structure of G^t, FPIRA sets $G^{t+1} = G^t, \bar{b}_i^t = \hat{b}_i^t$ and continues with the next iteration.

4.1 Creating the Initial Abstraction

FPIRA creates G^1 (line 1 in Algorithm 1) as a coarse imperfect recall abstraction of G' by merging possible information sets, such that the coarsest perfect recall refinement of G^1 is G'. To achieve this, FPIRA needs to make sure that when merging a set of information sets $\mathcal{J} \subseteq \mathcal{I}_i$ there are no two distinct $I, I' \in \mathcal{J}$ which, when merged, create a perfect recall information set. This is required since, as discussed in the next section, the algorithm splits only imperfect recall information sets. If FPIRA joined information sets not resulting in the imperfect recall information set, it would end up solving a different game.

More formally, we use the following algorithm to build G^1. For all i create $\mathcal{K}_i$ as a set of disjoint subsets of information sets $\mathcal{I}'_i$ of G', such that $\bigcup_{\mathcal{K}' \in \mathcal{K}_i} \mathcal{K}' = \mathcal{I}'_i$, and

$$\forall \mathcal{K}' \in \mathcal{K}_i, \forall I, I' \in \mathcal{K}' : |A'(I)| = |A'(I')| \wedge |seq_i(I)| = |seq_i(I')|.$$

In other words, all the information sets in $\mathcal{K}'$ must have the same number of actions available and the same length of the sequence of i leading to them. Every $\mathcal{K}' \in \mathcal{K}_i$ gives us candidates for merging. However, as discussed above, we need to make sure that we never merge any pair of information sets, which would result in a perfect recall information set after the merge. Hence, we further split every $\mathcal{K}' \in \mathcal{K}_i$ to the smallest possible set $\mathcal{J} = \{\mathcal{J}^1, ..., \mathcal{J}^k\}$, such that

$$\forall \mathcal{J}^j \in \mathcal{J}, \forall I, I'' \in \mathcal{J}^j : seq_i(I) \neq seq_i(I').$$

I and I' are information sets of G', hence both $seq_i(I)$ and $seq_i(I')$ are singletons. $\mathcal{J}$ needs not be unique, in our implementation we choose randomly between possible $\mathcal{J}$. Finally, every information set in the information set structure $\mathcal{I}^1$ of G^1 corresponds to $\mathcal{J}^j$ from the union of all $\mathcal{J}$ for all players.

4.2 Updating G^t

There are two reasons for splitting some $I \in \mathcal{I}_i^t$ in iteration t (we assume player i computes the best response in t): (1) the best response computed in G' prescribes more than one action in I or (2) I causes expected values of some pure strategy of $-i$ to be different after the average strategy update of i compared to what would happen when updating in G'.

To formally describe the splitting rules, let us first define mappings $\Phi_t : \mathcal{I}' \to \mathcal{I}^t$, which for $I \in \mathcal{I}'$ returns the information set containing I in G^t and $\Phi_t^{-1} : \mathcal{I}^t \to \wp(\mathcal{I}')$, the inverse of Φ_t. By $\Xi_t : A' \to A^t$ and $\Xi_t^{-1} : A^t \to \wp(A')$ we denote a mapping of actions from G' to G^t and vice versa.

Case 1: FPIRA checks in every iteration t if there exists $I \in \mathcal{I}_i^t$ where the best response b_i^t prescibes more than 1 action. If yes, FPIRA splits I to a set of information sets $\hat{\mathcal{I}}$ and inforation set I'', such that $\forall \hat{I}_a \in \hat{\mathcal{I}}$, $\hat{I}_a$ is a unification of all $I' \in \Phi_t^{-1}(I)$ where $b_i^t(I', a') = 1$ for $\Xi_t(a') = a$ and $I'' = \{h \in I | \forall \hat{I} \in \hat{\mathcal{I}} : h \notin \hat{I}\}$ (line 6 in Algorithm 1).

Case 2: The algotrithm first constructs the average behavioral strategy $\hat{b}_i^t$ in G^t (line 7). This is done according to Lemma 1 from $\bar{b}_i^{t-1}$ with weight $\frac{t_i - 1}{t_i}$ and b_i^t with weight $\frac{1}{t_i}$, where t_i is the number of updates performed by i so far, plus 1 for the initial strategy (b_i^t is used with mappings Φ_t and Ξ_t). Next, FPIRA constructs $\widetilde{b}_i^t$ (line 8) in the same way in the information set structure of G' ($\bar{b}_i^{t-1}$ is used with mappings Φ_t^{-1} and Ξ_t^{-1}). FPIRA then computes

$$\Delta_i^t = \max_{b_{-i} \in \mathcal{B}_{-i}^p} |u_{-i}(\widetilde{b}_i^t, b_{-i}) - u_{-i}(\hat{b}_i^t, b_{-i})|,$$

as described below (line 9). If $\Delta_i^t = 0$, none of the pure strategies of $-i$ changed its expected value compared to the update in G'. In this case, FPIRA sets $G^{t+1} = G^t$, $\bar{b}_i^t = \hat{b}_i^t$ (line 12). If $\Delta_i^t > 0$, the expected value of some pure strategy of $-i$ changed when updating the strategy in G^t, compared to the expected value it would get against the strategy updated in G'. FPIRA then creates G^{t+1} in the following way. Every imperfect recall information set $I \in \mathcal{I}_i^t$ which is visited when playing b_i^t is split to a set of information sets $\hat{\mathcal{I}} \subseteq \Phi_t^{-1}(I)$ and an information set I'', such that $\hat{\mathcal{I}}$ contains all the $I' \in \Phi_t^{-1}(I)$ which can be visited when playing b_i^t, I'' contains the rest of $h \in I$. The average strategy in all $I' \in \hat{\mathcal{I}} \cup \{I''\}$ before the strategy update is set to the strategy previously played in I. More formally, $\forall I' \in \hat{\mathcal{I}} \cup \{I''\}$ the strategy is set to

$$\bar{b}_i^{t-1}(I', a) = \bar{b}_i^{t-1}(\Phi_t(I'), \Xi_t(a)), \forall a \in A'(I').$$

The strategy resulting from update in G' is a valid strategy in G^{t+1} after such update, hence $\bar{b}_i^t = \widetilde{b}_i^t$. Notice that G' is still the coarsest perfect recall refinement of G^{t+1}, additionally by setting $\bar{b}_i^t = \widetilde{b}_i^t$, we made sure that $\Delta_i^t = 0$ since the update is now equal to the update that would occur in G'. This, as we will show in Section 4.3, is sufficient to guarantee the convergence of $\bar{b}_i^t, \bar{b}_{-i}^t$ to Nash equilibrium of G'.

Computing Δ_i^t. Given $\hat{b}_i^t$ and $\widetilde{b}_i^t$, Δ_i^t can be computed as

$$\Delta_i^t = \max_{b_{-i} \in \mathcal{B}_{-i}^p} \sum_{z \in \mathcal{Z}} |\pi_{-i}^{b_{-i}}(z) \left[\pi_i^{\widetilde{b}_i^t}(z) - \pi_i^{\hat{b}_i^t}(z) \right] u_{-i}(z)|.$$

Δ_i^t can be computed in $O(|\mathcal{Z}|)$ as a standard best response tree traversal.

Example 1. *Let us demonstrate several iterations of* FPIRA *algorithm. Consider the perfect recall game from Figure 1 (a) as G' and the imperfect recall game from Figure 1 (b) as G^1. The function Ξ_1 is $\Xi_1(t) = \Xi_1(v) = c, \Xi_1(u) = \Xi_1(w) = d$, identity otherwise. Note that when we apply strategies from G' to G^t and vice versa, we assume that it is done with respect to Ξ_t and Ξ_t^{-1}. Lets assume that* FPIRA *first initializes the strategies to $\bar{b}_1^0(b) = \bar{b}_1^0(d) = 1, \bar{b}_2^0(e) = 1$.*

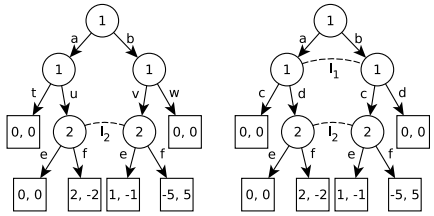


Figure 1: (a) G' for demonstration of FPIRA iterations (b) G^1 for demonstration of FPIRA.

Iteration 1: *The player 1 starts in iteration 1.* FPIRA *computes $b_1^1 \in BR(\bar{b}_2^0)$ in G', resulting in $b_1^1(b) = b_1^1(v) = 1$. Next,* FPIRA *checks whether b_1^1 is playable in G^1. Since there is no information set in G^1 for which b_1^1 assigns more than one action, we do not need to update G^1 in any way. We follow by computing $\hat{b}_1^1$ and $\widetilde{b}_1^1$ according to Lemma 1 with $\lambda_1 = \lambda_2 = 0.5$. In this case $\hat{b}_1^1(b) = \widetilde{b}_1^1(b) = 1, \hat{b}_1^1(c) = \widetilde{b}_1^1(v) = 0.5$. Since $\hat{b}_1^1$ and $\widetilde{b}_1^1$ are equal, w.r.t. Ξ_1, we know that $\Delta_i = 0$. Hence we let $G^2 = G^1$, $\bar{b}_1^1 = \hat{b}_1^1$ and $\Xi_2 = \Xi_1$.*

Iteration 2: *Player 2 continues in iteration 2. Notice that player 2 does not have imperfect recall, hence there is no need to discuss this iteration in such detail.* FPIRA *computes the best response to $\bar{b}_1^1$, resulting in $b_2^2(f) = 1$. The algorithm then computes $\hat{b}_2^2$ and $\widetilde{b}_2^2$, resulting in $\hat{b}_2^2(e) = \widetilde{b}_2^2(e) = 0.5$. Hence, we let $G^3 = G^2$, $\bar{b}_2^2 = \hat{b}_2^2$ and $\Xi_3 = \Xi_2$.*

Iteration 3: *The best response in this iteration is $b_1^3(a) = b_1^3(u) = 1$, which is again playable in G^3, hence we do not need to update G^3 at this point.* FPIRA *computes $\hat{b}_1^3$ resulting in $\hat{b}_1^3(a) = \frac{1}{3}, \hat{b}_1^3(d) = \frac{2}{3}$, $\widetilde{b}_1^3$ is, on the other hand, $\widetilde{b}_1^3(a) = \frac{1}{3}$, $\widetilde{b}_1^3(t) = 1, \widetilde{b}_1^3(w) = 0.5$ (both according to Lemma 1 with $\lambda_1 = \frac{2}{3}, \lambda_2 = \frac{1}{3}$). In this case, $\Delta_1^3 = 1$ since by playing f player 2 gets $\frac{2}{3}$ against $\hat{b}_1^3$ compared to $\frac{5}{3}$ against $\widetilde{b}_1^3$. Hence, the algorithm splits all imperfect recall information sets reachable when playing b_1^3, in this case I_1, as described in Section 4.2, Case 2, resulting in G'. Therefore, $G^4 = G'$, $\bar{b}_1^3 = \widetilde{b}_1^3$ and Ξ_4 is set to identity.*

4.3 Theoretical Properties

We show that the convergence quarantees of FP in two-player zero-sum perfect recall game G' [Heinrich *et al.*, 2015] directly apply to FPIRA solving G'.

Theorem 1. *The exploitability of $\bar{b}_i^t$ computed by* FPIRA *applied to perfect recall two-player zero-sum EFG G' is exactly equal to the exploitability of $\bar{b'}_i^t$, computed by FP applied to G' in all iterations t and for all i.*

Proof. Assume that the initial strategies $\bar{b}_1^0, \bar{b}_2^0$ in FPIRA and initial strategies $\bar{b'}_1^0, \bar{b'}_2^0$ in the FP are realization equivalent, additionally assume that the same tie breaking rules are used

when more than one best response is available. We prove the Theorem by induction. If

$$\forall b_{-i} \in \mathcal{B}^p_{-i} : u_{-i}(\bar{b}^t_i, b_{-i}) = u_{-i}(\bar{b'}^t_i, b_{-i}), \quad (1)$$

$$\forall b_i \in \mathcal{B}^p_i : u_i(b_i, \bar{b}^t_{-i}) = u_i(b_i, \bar{b'}^t_{-i}), \quad (2)$$

where $\mathcal{B}^p$ is the set of pure behavioral strategies in G', then

$$b_{-i} \in \mathcal{B}^p_{-i} : u_{-i}(\bar{b}^{t+1}_i, b_{-i}) = u_{-i}(\bar{b'}^{t+1}_i, b_{-i}).$$

First, the initial step trivially holds from the initialization of strategies. Now let us show that the induction step holds. Let b^t_i be the best response chosen in iteration t in FPIRA and b'^t_i be the best response chosen in t in FP. From (2) and the use of the same tie breaking rule we know that $b^t_i = b'^t_i$. From Lemma 1 we know that

$$\forall b_{-i} \in \mathcal{B}^p_{-i} : u_{-i}(\bar{b'}^{t+1}_i, b_{-i}) = \frac{t_i}{t_i+1} u_{-i}(\bar{b'}^t_i, b_{-i}) + \frac{1}{t_i+1} u_{-i}(b'^t_i, b_{-i}),$$

However, same holds also for $\bar{b}^{t+1}_i$ since FPIRA creates G^{t+1} from G^t so that $\Delta^t_i = 0$. Hence

$$\forall b_{-i} \in \mathcal{B}^p_{-i} : u_{-i}(\bar{b}^{t+1}_i, b_{-i}) = \frac{t_i}{t_i+1} u_{-i}(\bar{b}^t_i, b_{-i}) + \frac{1}{t_i+1} u_{-i}(b^t_i, b_{-i}).$$

From (1) and from the equality $b^t_i = b'^t_i$ follows that

$$\forall b_{-i} \in \mathcal{B}^p_{-i} u_{-i}(\bar{b}^{t+1}_i, b_{-i}) = u_{-i}(\bar{b'}^{t+1}_i, b_{-i}),$$

and therefore also

$$\max_{b_{-i} \in \mathcal{B}^p_{-i}} u_{-i}(\bar{b}^{t+1}_i, b_{-i}) = \max_{b_{-i} \in \mathcal{B}^p_{-i}} u_{-i}(\bar{b'}^{t+1}_i, b_{-i}). \quad \square$$

4.4 Memory and Time Efficiency

FPIRA needs to store the average behavioral strategy for every action in every information set of the solved game, hence storing the average strategy in G^t instead of G' results in significant memory savings directly proportional to the decrease of information set count. Additionally, when the algorithm computes $\widetilde{b}^t_i$, it can temporarily refine the information set structure of G^t only in the parts of the tree that can be visited when playing the pure best response b^t_i according to $\mathcal{I}'_i$ to avoid representing and storing G'. Moreover, one typically does not have to store and traverse the whole game tree when computing a best response. When storing the b^t_i, we do not store behavior in the parts of the game unreachable due to actions of i. For this reason, there are typically large parts of the game tree omitted, since i plays only 1 action in his information sets. Hence, the best response computation does not prevent us from solving large domains with excessive memory requirements (we provide results showing that the best responses are small in Section 5). Finally, efficient domain-specific implementations of best response (e.g., on poker [Johanson *et al.*, 2011]) can be employed to further reduce the memory and time requirements. The iteration of FPIRA takes approximately twice the time needed to perform one iteration of FP in G', as it now consists of the standard best response computation in G', the modified best response computation to obtain Δ^t_i and two updates of average behavioral strategies (which are faster than the update in G' since the average strategy is smaller).

5 Experiments

We introduce the domains used for experimental evaluation of FPIRA. We follow by the discussion of the convergence and the size of abstractions needed to solve these domains.

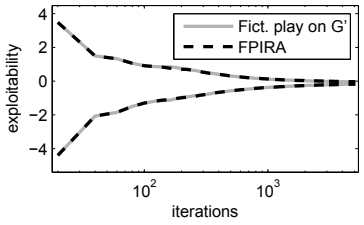
Leduc Hold'em. Leduc Hold'em is a two-player poker, which is used as a common benchmark in imperfect-information game solving because it is small enough to be solved but still strategically complex. There is a deck of cards with a given number of card types and a given number of cards per type (in standard Leduc hold'em there are 3 types of cards, 2 cards for each type). There are two rounds. In the first round, each player places an ante of 1 chip in the pot and receives a single private card. A round of betting follows. Every player can bet from a limited set of allowed values or check. After a bet, the player can raise, again choosing the value from a limited set, call or forfeit the game by folding. The number of consecutive raises is limited. A public shared card is dealt after one of the players calls or after both players check. Another round of betting takes place with identical rules. The player with the highest pair wins. If none of the players has a pair, the player with the highest card wins.

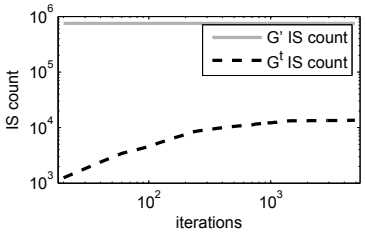
Random Games. Since there is no standardized collection of benchmark EFGs, we use randomly generated games to obtain statistically significant results. We randomly generate a perfect recall game with varying depth and branching factor 5. To control the information set structure, we use observations assigned to actions – for player i, nodes h with the same observations generated by all actions in history belong to the same information set. Every action generates a temporary utility which is randomly generated in the interval $(-2, 2)$. The utility for player 1 in every leaf is then computed as the sum of the temporary utilities of actions leading from the root of the game to the leaf. In this way, we create more realistic games, with the notion of good and bad moves.

5.1 Results

In all the presented results, FPIRA was terminated when the difference of the expected values of best responses against the average strategies was below 10^{-2}.

Leduc Hold'em. In Figure 2 (a) we show the exploitability of the average strategies computed by the FP and FPIRA (y-axis) as a function of iterations (log x-axis) on Leduc Hold'em with 4 card types, 3 for each type, 2 possible bet and raise values and 4 consecutive raises allowed. The observed identical convergence rate of FPIRA and the FP is a direct consequence of Theorem 1. Both algorithms needed $\sim 5{\cdot}10^3$ iterations to converge to the gap 10^{-2} in the expected values of best responses against the average strategies. In Figure 2 (b) we show the information set count of G^t and G' (log y-axis) as a function of iterations (log x-axis) in the same setting. As expected, the highest increase in the information set count of G^t is at the beginning of the algorithm since the information set structure is extremely coarse and the strategies vary significantly between iterations. In the later stages of the convergence, we observe almost no changes in the information set structure. Additionally, in Figure 2 (c) we present relative information set counts for initial and final abstraction, relative size of the largest best response which needed to be





	G^1 size	G^∞ size	BR size	G' size
b. 2, r. 2, c.r. 1	4.4%	31.1%	10.7%	8216
b. 3, r. 3, c.r. 1	1.8%	22.7%	4.0%	30200
b. 4, r. 4, c.r. 1	0.9%	15.9%	2.1%	81016
b. 5, r. 5, c.r. 1	0.6%	8.5%	0.9%	179096
b. 2, r. 2, c.r. 2	1.4%	16.6%	3.0%	40600
b. 2, r. 2, c.r. 3	0.4%	5.6%	0.7%	179096
b. 2, r. 2, c.r. 4	0.1%	1.8%	0.2%	751000

Figure 2: (a) The exploitability of the average strategies on Leduc Hold'em with 4 card types, 3 for each, 2 bet and raise values and 4 consecutive raises allowed. (b) The information set count of G^t and G' in the same setting. (c) Relative information set counts of G^1 and final abstraction G^∞, relative size of the largest best response which needed to be stored, and total information set count of G' for Leduc holdem with deck with 4 types of cards, 3 cards for each, for increasing number of bets and raises (rows 1 to 4) and increasing consecutive raise count (row 1, 5 - 7)

stored and total information set count of the original perfect recall game in poker domains, first when increasing number of bets and raises and when increasing number of allowed consecutive raises. The size of both abstractions decreases in both parameters, hence the sizes of the average strategies stored in FPIRA decrease proportionally. Besides storing the average strategy in the current abstraction, FPIRA also stores the best response computed in G' before updating the average strategy. To show the maximal memory needed to store these best responses we provide the number of information sets of G' for which there is an action prescribed in the largest best response during the run of FPIRA. These results show that the sizes of the best responses do not threaten the memory efficiency of FPIRA. This leads to significant memory savings in large domains with only approximately twice as much time needed per iteration compared to the FP applied directly to the perfect recall game. Additionally, the results suggest that further scaling of the solved domains will significantly reduce the relative size of the abstractions. This observation is further supported by the fact that the relative support size of Nash equilibria in poker-like domains decreases as the game size increases [Schmid *et al.*, 2014].

Random games. To show the performance of FPIRA in games with varying structure, we performed additional experiments on 50 instances of random games for depth 7 and 30 instances for depth 8. FPIRA was able to solve the instances using abstractions which had on average only $4.9\% \pm 0.7\%$ of the information sets of the original game for depth 7 and $1.2\% \pm 0.3\%$ for depth 8. The perfect recall instances had on average $2.6 \cdot 10^4 \pm 4 \cdot 10^3$ and $1.1 \cdot 10^5 \pm 3 \cdot 10^4$ information sets for depth 7 and 8 respectively. Notice that the sizes of abstractions needed to solve the random games are significantly smaller than in the case of poker with similar information set counts. This is caused by the moves of nature at the start of poker, which cause large parts of the game tree to be visited when playing according to one pure strategy. Hence when computing Δ_i^t, there is a significantly higher number of pure strategies of $-i$ that we need to check, and therefore more information set splits, compared to games with no nature.

6 Conclusion

We present the first algorithm that automatically creates imperfect recall abstractions and uses them to solve the original extensive-form game with perfect recall. While the imperfect recall abstractions of perfect recall games can significantly reduce the space necessary for storing strategies, their use has been rather limited. Previous works use either very restricted subclasses of imperfect recall abstractions [Lanctot *et al.*, 2012; Kroer and Sandholm, 2016; Bošanský *et al.*, 2015], heuristic approaches [Waugh *et al.*, 2009], or only focus on solving the given imperfect recall game [Čermák *et al.*, 2017].

Our main contribution is the memory efficient algorithm FPIRA that provides an automatically created imperfect recall abstraction that is sufficient for a Nash equilibrium strategy in the original perfect recall game. As a consequence, a strategy from the abstracted game can be used directly in the original game without the need to use translation techniques, nor is the quality of such strategy affected by choice of the abstraction. The FPIRA algorithm is based on fictitious play (FP) and provides the same guarantees of the convergence as if FP were applied directly to the original perfect recall game. The experimental evaluation shows that we are able to solve modification of Leduc hold'em with $7 \cdot 10^5$ information sets using an abstracted game with 1.8% of information sets of the original game. Furthermore, the experiments on different versions of Leduc hold'em and random games suggest that the relative size of the needed abstractions significantly decreases as the size of the solved games increases.

A natural step for future work is to generalize FPIRA to other learning algorithms – e.g., Counterfactual Regret Minimization (CFR) [Zinkevich *et al.*, 2008] that is used for solving large extensive-form games with perfect recall. However, this generalization is not straightforward since the updates, and the identification of whether an information set should be split is significantly more challenging in that case.

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. 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".

References

[Bošanský *et al.*, 2015] Branislav Bošanský, Albert Xin Jiang, Milind Tambe, and Christopher Kiekintveld. Combining compact representation and incremental generation in large games with sequential strategies. In *AAAI*, pages 812–818, 2015.

[Bošanský *et al.*, 2016] Branislav Bošanský, Viliam Lisý, Marc Lanctot, Jiří Čermák, and Mark H.M. Winands. Algorithms for Computing Strategies in Two-Player Simultaneous Move Games. *Artificial Intelligence*, 237:1–40, 2016.

[Bošanský *et al.*, 2014] Branislav Bošanský, Christopher Kiekintveld, Viliam Lisý, and Michal Pěchouček. An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information. *Journal of Artificial Intelligence Research*, 51:829–866, 2014.

[Brown and Sandholm, 2015] Noam Brown and Tuomas W Sandholm. Simultaneous abstraction and equilibrium finding in games. In *AAAI*, 2015.

[Brown, 1949] George W Brown. Some notes on computation of games solutions. Technical report, DTIC Document, 1949.

[Čermák *et al.*, 2017] Jiři Čermák, Branislav Bošanskỳ, and Michal Pěchouček. Combining incremental strategy generation and branch and bound search for computing maxmin strategies in imperfect recall games. In *AAMAS*, pages 902–910, 2017.

[Chen and Bowling, 2012] Katherine Chen and Michael Bowling. Tractable objectives for robust policy optimization. In *NIPS*, pages 2078–2086, 2012.

[Daskalakis and Pan, 2014] Constantinos Daskalakis and Qinxuan Pan. A counter-example to karlin's strong conjecture for fictitious play. In *Annual Symposium on Foundations of Computer Science*, pages 11–20. IEEE, 2014.

[Gilpin and Sandholm, 2007] Andrew Gilpin and Tuomas Sandholm. Lossless Abstraction of Imperfect Information Games. *Journal of the ACM*, 54(5), 2007.

[Gilpin *et al.*, 2007] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of texas hold'em poker. In *AAAI*, pages 50–57, 2007.

[Heinrich *et al.*, 2015] Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *ICML*, pages 805–813, 2015.

[Johanson *et al.*, 2011] Michael Johanson, Kevin Waugh, Michael Bowling, and Martin Zinkevich. Accelerating best response calculation in large extensive games. In *IJCAI*, volume 11, pages 258–265, 2011.

[Karlin, 2003] Samuel Karlin. *Mathematical methods and theory in games, programming, and economics*, volume 2. Courier Corporation, 2003.

[Kroer and Sandholm, 2014] Christian Kroer and Tuomas Sandholm. Extensive-Form Game Abstraction with Bounds. In *EC*, pages 621–638. ACM, 2014.

[Kroer and Sandholm, 2016] Christian Kroer and Tuomas Sandholm. Imperfect-Recall Abstractions with Bounds in Games. In *EC*, pages 459–476. ACM, 2016.

[Kuhn, 1953] Harold W. Kuhn. Extensive Games and the Problem of Information. *Contributions to the Theory of Games*, II:193–216, 1953.

[Lanctot *et al.*, 2012] Marc Lanctot, Richard Gibson, Neil Burch, Martin Zinkevich, and Michael Bowling. No-Regret Learning in Extensive-Form Games with Imperfect Recall. In *ICML*, pages 1–21, 2012.

[Lisý *et al.*, 2016] Viliam Lisý, Trevor Davis, and Michael Bowling. Counterfactual Regret Minimization in Sequential Security Games. In *AAAI*, 2016.

[Moravčík *et al.*, 2017] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 2017.

[Robinson, 1951] Julia Robinson. An iterative method of solving a game. *Annals of mathematics*, pages 296–301, 1951.

[Schmid *et al.*, 2014] Martin Schmid, Matej Moravcik, and Milan Hladik. Bounding the support size in extensive form games with imperfect information. In *AAAI*, pages 784–790, 2014.

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

[Waugh *et al.*, 2009] Kevin Waugh, Martin Zinkevich, Michael Johanson, Morgan Kan, David Schnizlein, and Michael H Bowling. A Practical Use of Imperfect Recall. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*, 2009.

[Zinkevich *et al.*, 2008] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret Minimization in Games with Incomplete Information. In *NIPS*, pages 1729–1736, 2008.

Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Counterfactual Regret Minimization in Sequential Security Games

Viliam Lisý and **Trevor Davis** and **Michael Bowling**
Department of Computing Science
University of Alberta, Edmonton, AB, Canada T6G 2E8
{lisy,trdavis1,bowling}@ualberta.ca

Abstract

Many real world security problems can be modelled as finite zero-sum games with structured sequential strategies and limited interactions between the players. An abstract class of games unifying these models are the normal-form games with sequential strategies (NFGSS). We show that all games from this class can be modelled as well-formed imperfect-recall extensive-form games and consequently can be solved by counterfactual regret minimization. We propose an adaptation of the CFR$^+$ algorithm for NFGSS and compare its performance to the standard methods based on linear programming and incremental game generation. We validate our approach on two security-inspired domains. We show that with a negligible loss in precision, CFR$^+$ can compute a Nash equilibrium with five times less computation than its competitors.

Game theory has been recently used to model many real world security problems, such as protecting airports (Pita et al. 2008) or airplanes (Tsai et al. 2009) from terrorist attacks, preventing fare evaders form misusing public transport (Yin et al. 2012), preventing attacks in computer networks (Durkota et al. 2015), or protecting wildlife from poachers (Fang, Stone, and Tambe 2015). Many of these security problems are sequential in nature. Rather than a single monolithic action, the players' strategies are formed by sequences of smaller individual decisions. For example, the ticket inspectors make a sequence of decisions about where to check tickets and which train to take; a network administrator protects the network against a sequence of actions an attacker uses to penetrate deeper into the network.

Sequential decision making in games has been extensively studied from various perspectives. Recent years have brought significant progress in solving massive imperfect-information extensive-form games with a focus on the game of poker. Counterfactual regret minimization (Zinkevich et al. 2008) is the family of algorithms that has facilitated much of this progress, with a recent incarnation (Tammelin et al. 2015) essentially solving for the first time a variant of poker commonly played by people (Bowling et al. 2015). However, there has not been any transfer of these results to research on real world security problems.

Copyright © 2016, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

We focus on an abstract class of sequential games that can model many sequential security games, such as games taking place in physical space that can be discretized as a graph. This class of games is called normal-form games with sequential strategies (NFGSS) (Bosansky et al. 2015) and it includes, for example, existing game theoretic models of ticket inspection (Jiang et al. 2013), border patrolling (Bosansky et al. 2015), and securing road networks (Jain et al. 2011).

In this work we formally prove that any NFGSS can be modelled as a slightly generalized chance-relaxed skew well-formed imperfect-recall game (CRSWF) (Lanctot et al. 2012; Kroer and Sandholm 2014), a subclass of extensive-form games with imperfect recall in which counterfactual regret minimization is guaranteed to converge to the optimal strategy. We then show how to adapt the recent variant of the algorithm, CFR$^+$, directly to NFGSS and present experimental validation on two distinct domains modelling search games and ticket inspection. We show that CFR$^+$ is applicable and efficient in domains with imperfect recall that are substantially different from poker. Moreover, if we are willing to sacrifice a negligible degree of approximation, CFR$^+$ can find a solution substantially faster than methods traditionally used in research on security games, such as formulating the game as a linear program (LP) and incrementally building the game model by double oracle methods.

Game Model

NFGSS is a class of two-player zero-sum sequential games in which each player i's strategy space has the structure of a finite directed acyclic Markov decision process (MDP) with the set of states S_i, set of actions A_i and a stochastic transition function $T : S_i \times A_i \to \Delta(S_i)$. The transition function defines the probability of reaching next states after an action is executed in the current state. We denote by $\Delta(\cdot)$ a set of probability distributions over a set; $A(s_i) \subseteq A_i$ the actions applicable in state s_i; and s_i^0 the initial state of the MDP.

A player's strategy is a probability distribution over actions applicable in each state. We denote $\delta_i(s_i, a_i)$ to be the probability that a player following strategy δ_i reaches state s_i and then executes action a_i. An important restriction in this class of games is that the utility is defined in terms of *marginal utilities* for simultaneously executed state-action pairs $(s_i, a_i) \in S_i \times A_i$ by the players. The first player's

(negative of second player's) expected utility has the form:

$$u(\delta_1, \delta_2) = \sum_{S_1 \times A_1} \sum_{S_2 \times A_2} \delta_1(s_1, a_1)\delta_2(s_2, a_2)U((s_1, a_1), (s_2, a_2))$$

for some $U : S_1 \times A_1 \times S_2 \times A_2 \to \mathbb{R}$.

Background

In this paper, we show that the games from NFGSS can be transformed to a previously studied subclass of extensive-form games with imperfect recall. This section defines the concepts required to understand this transformation.

Extensive-form games

Two-player extensive-form games model sequential decision making of *players* denoted $i \in N = \{1, 2\}$. In turn, players choose *actions* leading to sequences called *histories* $h \in H$. A history $z \in Z$, where $Z \subseteq H$, is called a *terminal history* if it represents a full game from start to end. At each terminal history z there is a payoff $u_i(z)$ to each player i. At each nonterminal history h, there is a single current player to act, determined by $P : H \backslash Z \to N \cup \{c\}$ where c is a special player called *chance* that plays with a fixed known stochastic strategy. For example, chance is used to represent rolls of dice and card draws. The game starts in the empty history $\emptyset$, and at each step, given the current history h, the current player chooses an action $a \in A(h)$ leading to successor history $h' = ha$. We call h a *prefix* of h', denoted $h \sqsubseteq h'$, and generalize this relation to its transitive and reflexive closure.

Set $\mathcal{I}_i$ is a partition over $H_i = \{h \in H : P(h) = i\}$ where each part is called an *information set*. An information set $I \in \mathcal{I}_i$ of player i is a set of histories that a player cannot tell apart (due to information hidden from that player). For all $h, h' \in I$, $A(h) = A(h')$ and $P(h) = P(h')$; hence, we extend the definition to $A(I)$, $P(I)$, and denote $I(h)$ the information set containing h. Let $X(h)$ be the set of information set and action pairs on the path from the root of the game to history $h \in H$; $X_i(h) \subseteq X(h)$ be only the pairs where player i chooses an action; and $X_i(h, z)$ only the pairs i chooses on the path from h to z. A game has *perfect recall*, if players remember all actions they took: $\forall i \in N\ \forall h \in H_i\ \forall h' \in I(h)\ X_i(h') = X_i(h)$. If this condition does not hold, the game has *imperfect recall*. In this paper we don't consider a form of imperfect recall often called absent mindedness, in which a history and a prefix of that history may be contained in the same information set.

A *behavioral strategy* for player i is a function mapping each information set $I \in \mathcal{I}_i$ to a probability distribution over the actions $A(I)$, denoted by $\sigma_i(I)$. For a *profile* $\sigma = (\sigma_1, \sigma_2)$, we denote the probability of reaching a terminal history z under σ as $\pi^\sigma(z) = \prod_{i \in N \cup \{c\}} \pi_i^\sigma(z)$, where each $\pi_i^\sigma(z) = \prod_{ha \sqsubseteq z, P(h)=i} \sigma_i(I(h), a)$ is a product of probabilities of the actions taken by player i along z. We use $\pi_i^\sigma(h, z)$ and $\pi^\sigma(h, z)$ for $h \sqsubseteq z$ to refer to the product of only the probabilities of actions along the sequence from the end of h to the end of z. We define Σ_i to be the set of behavioral strategies for player i and extend the utility function to strategy profiles as $u_i(\sigma) = \sum_{z \in Z} \pi^\sigma(z) u_i(z)$. By convention, $-i$ refers to player i's opponent and chance player, or just the opponent if the context does not admit chance.

An *ϵ-Nash equilibrium*, σ, is a strategy profile such that the benefit of switching to some alternative σ_i' is limited by ϵ, i.e., $\forall i \in N : \max_{\sigma_i' \in \Sigma_i} u_i(\sigma_i', \sigma_{-i}) - u_i(\sigma) \leq \epsilon$. When $\epsilon = 0$, the profile is called a Nash equilibrium. We focus on zero-sum games, where $u_2(z) = -u_1(z)$ and define *precision* of a profile σ as the sum of strategies' distances from an equilibrium, $\epsilon_\sigma = \max_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \sigma_2) + \max_{\sigma_2' \in \Sigma_2} u_2(\sigma_1, \sigma_2')$.

Counterfactual Regret Minimization

Counterfactual Regret is a notion of regret at the information set level for extensive-form games (Zinkevich et al. 2008). The Counterfactual Regret minimization algorithms iteratively learn strategies in self-play, converging to an equilibrium. The *counterfactual value* of 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:

$$v_i(I, \sigma) = \sum_{z \in Z_I} \pi_{-i}^\sigma(z[I]) \pi^\sigma(z[I], z) u_i(z), \qquad (1)$$

where $Z_I = \{z \in Z : \exists h \in I, h \sqsubseteq z\}$, $z[I] = h$ such that $h \sqsubseteq z, h \in I$, and $\pi_{-i}^\sigma(h)$ is the reach probability due to the opponent and chance. Define $\sigma_{I \to a}^t$ to be a strategy identical to σ^t except at I action a is taken with probability 1. The counterfactual regret of not taking $a \in A(I)$ at time t is $r^t(I, a) = v_i(I, \sigma_{I \to a}^t) - v_i(I, \sigma^t)$. We use the most recent CFR$^+$ (Tammelin et al. 2015) algorithm to minimize these regrets. This algorithm maintains values $Q^t(I, a) = \max(0, Q^{t-1}(I, a) + r^t(I, a))$ with $Q^0(I, a) = 0$ for every action at every information set. The strategy for the next iteration $\sigma^{t+1}(I)$ is proportional to the maintained values:

$$\sigma^{T+1}(I, a) = \begin{cases} Q^T(I, a)/Q_{sum}^T(I) & \text{if } Q_{sum}^T(I) > 0 \\ 1/|A(I)| & \text{otherwise,} \end{cases} \qquad (2)$$

where $Q_{sum}^T(I) = \sum_{a' \in A(I)} Q^T(I, a')$. Furthermore, the algorithm maintains the weighted average strategy profile:

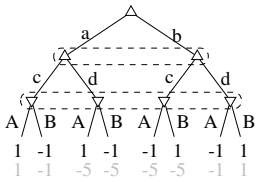
$$\bar{\sigma}^T(I, a) = \frac{\sum_{t=1}^T t \cdot \pi_i^{\sigma^t}(I) \sigma^t(I, a)}{\sum_{t=1}^T t \cdot \pi_i^{\sigma^t}(I)}, \qquad (3)$$

where $\pi_i^{\sigma^t}(I) = \sum_{h \in I} \pi_i^{\sigma^t}(h)$. The combination of the counterfactual regret minimizers in individual information sets also minimizes the overall average regret, and hence the average profile is an ϵ-equilibrium, with $\epsilon \to 0$ as $T \to \infty$.

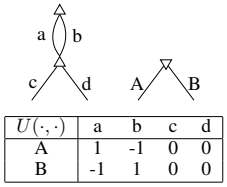
Well-formed Imperfect-Recall Games

Imperfect recall in general introduces complications in finding optimal solutions for games and the problem often becomes NP-hard (Koller and Megiddo 1992). Therefore Lanctot at al. (2012) introduce a subclass of imperfect-recall games which does not suffer from these problems. This subclass was further extended in (Kroer and Sandholm 2014).

For an imperfect-recall game Γ, we define its *perfect-recall refinement* Γ', which is the exact same game, but the information sets $\mathcal{I}_i$ are further partitioned to $\mathcal{I}_i'$ by splitting the histories that would violate the perfect-recall condition



(a) Imperfect-recall EFG



$U(\cdot,\cdot)$	a	b	c	d
A	1	-1	0	0
B	-1	1	0	0

(b) Equivalent NFGSS

Figure 1: Example of an imperfect-recall game and an equivalent NFGSS. With the grey utilities, the game cannot be represented as NFGSS.

to separate information sets. It means that histories h_1 and h_2 are in the same information set in the refinement if and only if they are in the same information set in the imperfect-recall game and $X_i(h_1) = X_i(h_2)$. We denote $D(I) \subset \mathcal{I}'_i$ the set of information sets $I \in \mathcal{I}_i$ is divided into.

Following (Kroer and Sandholm 2014), an extensive-form game with imperfect recall Γ is a *chance-relaxed skew well-formed* game with respect to its perfect-recall refinement Γ', if for all $i \in N, \bar{I} \in \mathcal{I}_i, I, J \in D(\bar{I})$, there exists a bijection $\phi : Z_I \to Z_J$, such that for all $z \in Z_I$:

1. The actions of the opponent and chance on the paths to the leaves mapped to each other are the same ($X_{-i}(z) = X_{-i}(\phi(z))$) or span the whole information sets on the same level if they differ ($\forall(\tilde{I}, a) \in X_{-i}(z) \setminus X_{-i}(\phi(z)), \forall z' \in Z_I : (\tilde{I}, a) \in X_{-i}(z')$). Since ϕ is a bijection, the roles of I and J are interchangeable.
2. The actions of player i after reaching $\bar{I}$ on paths to leaves which are mapped to each other are the same: $X_i(z[I], z) = X_i(\phi(z)[J], \phi(z))$.

Kroer and Sandholm (2014) show that in well-formed games, minimizing counterfactual regret in individual information sets also minimizes regret in its perfect-recall refinement. The relation between the regrets can be expressed in terms of the following error terms:

3. $|u_i(z) - \alpha_{I,J} u_i(\phi(z))| \leq \epsilon^R_{I,J}(z)$ for some fixed $\alpha_{I,J} \in \mathbb{R}$ is the reward error at z with respect to I, J;
4. $|\pi_0(z[I], z) - \pi_0(\phi(z)[J], \phi(z))| \leq \epsilon^0_{I,J}(z)$ is the leaf probability error at z with respect to I, J;
5. $\left| \frac{\pi_0(z[I])}{\pi_0(I)} - \frac{\pi_0(\phi(z)[J])}{\pi_0(J)} \right| \leq \epsilon^D_{I,J}(z[I])$ is the distribution error of $z[I]$.

For the clarity of exposition, we require all these errors to be 0. In that case, a direct consequence of Theorem 1 in (Kroer and Sandholm 2014) is that running a counterfactual regret minimization algorithm in the well-formed imperfect-recall game converges to a Nash equilibrium in the refinement.

NFGSS as CRSWF Game

This section proves that any NFGSS can also be represented as a well-formed imperfect-recall game. We start with a simple example showing the relation between the two classes of games and why the standard issues with imperfect recall do not occur in NFGSS. Figure 1a with the black utility values presents an imperfect-recall extensive-form game, which is equivalent to the NFGSS in Figure 1b. Even though the maximizing player $\triangle$ forgets whether she played a or b, her following decision does not influence the utilities. Therefore, it is a CRSWF game and it can be represented as an NFGSS. The same game structure with the grey utilities is a typical example of a problematic imperfect-recall game. It does not have a Nash equilibrium in behavioral strategies (Wichardt 2008), it has a non-zero reward error for any bijection between histories below a and b, and it cannot be represented as an NFGSS.

Before we state the main theorem, we need to extend the definition of the reward error in CRSWF games to allow forgetting the rewards accumulated in the past if they do not influence future observations, actions, or rewards.

Proposition 1 *If we define the reward error as*

$$|u_i(z) - \beta_I - \alpha_{I,J}\left(u_i(\phi(z)) - \beta_J\right)| \leq \epsilon^R_{I,J}(z)$$

for some fixed $\alpha_{I,J}, \beta_I, \beta_J \in \mathbb{R}$, then Theorem 1 from (Kroer and Sandholm 2014) still holds. Consequently, converging to zero counterfactual regret in all information sets translates to converging to a Nash equilibrium in any perfect recall refinement of a game in case of zero error terms.

The complete statement and the proof of this proposition are included in the appendix available online. The main idea of the proof is that for any information set $\bar{I}$ in the imperfect-recall game, we can create a modified game with utilities

$$u^{\bar{I}}_i(z) = \begin{cases} u_i(z) - \beta_I & \text{if } z \in Z_I, I \in D(\bar{I}) \\ u_i(z) & \text{otherwise.} \end{cases} \quad (4)$$

In this game, we can use Proposition 1 from (Kroer and Sandholm 2014) to bound regret in $\bar{I}$ with the original definition of the reward error. Finally, we show that the regrets in the modified games are exactly the same as the regrets in the original game, since the shifts β_I cancel out.

For an NFGSS G=$(S_1, A_1, S_2, A_2, T, U)$, a *corresponding* extensive-form game $\Gamma(G) = (N, \mathcal{A}, H, Z, \mathcal{I}, u)$ is:

$\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_c$ The set of player's actions in the EFG is the set of state-action pairs from his MDP: $\mathcal{A}_i = \{(s, a) : s \in S_i, a \in A(s)\}$. Execution of each state-action pair (s, a) leads to a chance node with actions representing uncertainty in the transition using actions from $\mathcal{A}_c = \{(s, a, s') : s, s' \in S_i, a \in A(s), T(s, a, s') > 0\}$. The chance probabilities of these actions are naturally derived from T.

H A history $h \in H$ is a sequence of actions in which each player's action is followed by a corresponding chance action as defined above. The ordering of the actions of different players can be arbitrary. We assume that each history starts with actions of the first player until she reaches a terminal state of her MDP. The actions of the second player follow afterwards. For convenience, we refer to histories in these corresponding games as a pair (h_{1+c}, h_{2+c}) with each player's actions and consequent chance actions and do not deal with action ordering among different players explicitly.

Z Let $ls_i : H \to S_i$ be a function returning the MDP state of player i after last chance action consequent to her action in a history. The terminal histories are the histories in which both players reached the terminal state of their MDP: $Z = \{h \in H : A(ls_1(h)) = A(ls_2(h)) = \emptyset\}$.

$\mathcal{I}$ Each information set corresponds to a node in a player's MDP. Histories h^1, h^2 in which player i takes action ($P(h^1) = P(h^2) = i$), belong to the same information set ($I(h^1) = I(h^2)$) if and only if they end by the same state for the player: $ls_i(h^1) = ls_i(h^2)$.

u The utility of player 1 in a terminal history z is the sum of marginal utilities for all pairs of actions in the history $u_1(z) = \sum_{(s_1,a_1)\in z_1} \sum_{(s_2,a_2)\in z_2} U((s_1, a_1), (s_2, a_2))$.

Theorem 2 *Let G be an NFGSS, then the corresponding extensive-form game $\Gamma(G)$ is a chance-relaxed skew well-formed imperfect-recall game with zero errors.*

Proof. Consider $\Gamma' = (N, \mathcal{A}, H, Z, \mathcal{I}', u)$, a perfect-recall refinement of $\Gamma(G)$, where each player's information sets are subdivided based on unique histories of her and subsequent chance actions (i.e., a path in the MDP). To show that Γ is well-formed with respect to Γ', we define for each $I, J \in D(\bar{I}), \bar{I} \in \mathcal{I}$ the bijection $\phi : Z_I \to Z_J$ to map to each other the terminal histories passing through I and J with the same opponent's actions and the same player $i = P(I)$'s actions after I or J is reached.

$$\forall z = (z_{i+c}, z_{-i+c}) \in Z_I \; \phi(z) = (h(J)_{i+c} z[I-]_{i+c}, z_{-i+c})$$

where $h(J)_{i+c}$ denotes the unique player $P(J)$'s and consequent chance history leading to J in the refinement Γ' and $z[I-]$ denotes the suffix of history z after reaching information set I. Subscript $i+c$ added to any (partial) history refers to only actions of player i and the consequent chance actions. The remaining actions are referred to by $-i+c$. This bijection is well defined, because the possible future sequences of actions of player i are the same after reaching the same node in his MDP using different paths.

We follow by checking if all conditions required for a well-formed imperfect-recall game are satisfied with ϕ.

1) The actions of the opponent and their consequent chance actions are exactly the same in the mapping. The only complication could be the player i's chance actions, which generally differ for histories in I and J. However, the outcome of the chance nodes is always known to the player before he selects a next action. Therefore, in the perfect-recall refinement, all histories in each information set contain the exact same chance action on each level.

2) The second condition is trivially satisfied directly from the definition of ϕ.

3) The original definition of CRSWF games is not sufficient to guarantee zero reward error for all histories. Consider an example with the structure exactly the same as in Figure 1, but the marginal utilities $U(A, b) = 5$, $U(A, c) = 1$ and zero for all other action pairs. Let the information sets in the refinement be $I = \{a\}$ and $J = \{b\}$. Based on the definition of ϕ and u above:

$$\phi(acA) = bcA, \quad \phi(adA) = bdA$$

$$u(acA) = 1,\ u(bcA) = 6,\ u(adA) = 0,\ u(bdA) = 5.$$

There is no $\alpha_{I,J}$, such that $|1 - \alpha_{I,J}6| = 0 = |0 - \alpha_{I,J}5|$. However, this game can still be solved by counterfactual regret minimization thanks to Proposition 1. Past rewards can be forgotten and the future rewards are exactly the same.

4) The histories below information sets I and J are formed by the exact same chance actions. Therefore, the leaf probability error is zero.

5) We established above that all histories in any information set I in the refinement include the exact same chance actions of the player $i = P(I)$ on the way to I; therefore $\pi_0(z[I]_{i+c})/\pi_0(I) = 1/|I|$. There exist a bijection between I and J; hence, $|I| = |J|$. The distribution error is

$$\left| \frac{\pi_0(z[I]_{i+c})\pi_0(z[I]_{-i+c})}{\pi_0(I)} - \frac{\pi_0(\phi(z)[J]_{i+c})\pi_0(\phi(z)[J]_{-i+c})}{\pi_0(J)} \right| =$$

$$= \frac{1}{|I|} |\pi_0(z[I]_{-i+c}) - \pi_0(\phi(z)[J]_{-i+c})| = 0. \quad (5)$$

The last equality holds because the opponent's chance actions are copied in ϕ. ■

There are two main consequences of this transformation. First, we can run counterfactual regret minimization on the EFGs corresponding to NFGSS and we are guaranteed that these algorithms converge to a well-defined Nash equilibrium solution. Second, since the computed equilibrium is an equilibrium in any perfect-recall refinement, it proves that the players gain no benefit from remembering the path that they take to individual states of an NFGSS.

CFR$^+$ for NFGSS

Previous works using CFR in imperfect-recall games use random sampling to update counterfactual regrets (Waugh et al. 2009). For NFGSS, we derive a more efficient version of the algorithm which does not require sampling.

The algorithm stores for each action-value pair the special form of regret Q defined in CFR$^+$ and the mean strategy $\bar{\sigma}$; and for each MDP state the probability $p(s)$ of reaching the state under the current strategy. The current strategies σ can be stored or always computed from Q to save memory. The pseudocode is presented in Figure 2. In the main part of the algorithm denoted by NFGSS-CFR$^+$, each iteration of the algorithm consists of separate updates for individual players (lines 4-6). First, the current strategy of the opponent is used to compute reach probabilities for all states and update the mean strategy in his MDP (line 5). The reach probabilities are used to compute the regrets Q and new strategies for all action-value pairs of the player (line 6). Finally, we need to normalize the mean strategies, since they are stored as sums.

The update of the probabilities on lines 1-6 of the second function in Figure 2 is straightforward. Since the MDPs are assumed to be acyclic, we can store the states in topological order and easily traverse them from the root to leaves, always having the predecessors resolved before reaching the successors. Once a new probability is computed, the mean strategy can be updated immediately on line 7. For updating regrets and strategies in the third function, we process the state in the reverse order. For each state, we compute the expected value of the game after it reaches the state $v(s_i)$ and the expected value of playing each action available in

NFGSS-CFR$^+$

1: $\forall i \in N\ \sigma_i :=$ uniform strategy
2: $\forall i \in N\ Q_i := 0, \bar{\sigma}_i = 0$
3: **for** iteration $t \in (1, 2, \dots)$ **do**
4: **for** $i \in N$ **do**
5: UpdateStateProbabilitiesMeanStrategies$(-i, t)$
6: UpdateActionRegretsCurStrategies(i)
7: $\forall i \in N$ Normalize$(\bar{\sigma}_i)$

UpdateStateProbabilitiesMeanStrategies(i, t)

1: $\forall s_i \in S_i\ p(s_i) = 0$
2: $p(s_i^0) := 1$
3: **for** $s_i \in S_i$ in topological order **do**
4: **for** $a_i \in A(s_i)$ **do**
5: **for** $s' \in T(s_i, a_i)$ **do**
6: $p(s') \mathrel{+}= p(s_i)\sigma(s_i, a_i)T(s_i, a_i, s')$
7: $\bar{\sigma}_i(s_i, a_i) \mathrel{+}= tp(s_i)\sigma_i(s_i, a_i)$

UpdateActionRegretsCurStrategies(i)

1: $\forall s_i \in S_i\ v(s_i) = 0$
2: **for** $s_i \in S_i$ in reverse topological order **do**
3: **for** $a_i \in A(s_i)$ **do**
4: $v(a_i) := 0$
5: **for** $s' \in T(s_i, a_i)$ **do**
6: $v(a_i) \mathrel{+}= T(s_i, a_i, s')v(s')$
7: **for** $s_{-i}, a_{-i} \in S_{-i} \times A_{-i}$ **do**
8: $v(a_i) \mathrel{+}= p(s_{-i})\sigma(s_{-i}, a_{-i})U_i(s_i, a_i, s_{-i}, a_{-i})$
9: $v(s_i) \mathrel{+}= \sigma(s_i, a_i)v(a_i)$
10: **for** $a_i \in A(s_i)$ **do**
11: $Q(s_i, a_i) := \max(0, Q(s_i, a_i) + v(a_i) - v(s_i))$
12: $\sigma_i(s_i) :=$ RegretMatching$(Q(s_i))$

Figure 2: Adaptation of CFR$^+$ for NFGSS

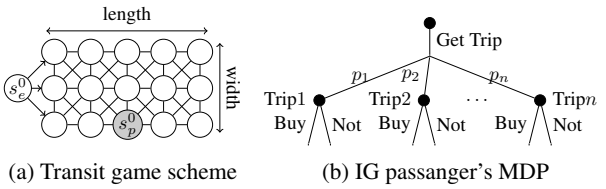
the state (lines 4-9). The value of the action is the sum of the expected value of the successors (lines 5-6) and the marginal utility of executing the action, with respect to the probabilities that the opponent executes her actions (lines 7-8). We implement this step efficiently using sparse representation of U. The expected value of the state is the sum of the expected values of executing individual actions weighted by their probability in the current strategy (line 9). The regrets are updated as prescribed by CFR$^+$ (lines 10-11) and the new strategy is computed by regret matching (Equation 2).

Proposition 3 *The NFGSS-CFR$^+$ algorithm converges to an ϵ-Nash equilibrium of the game after*

$$T \leq \max_{i \in N} \frac{\Delta^2 |S_i|^2 |A_i|}{\epsilon^2}$$

iterations, where Δ is the overall range of (not marginal) utility values in the game.

This proposition holds for CFR (and therefore also CFR$^+$) in CRSWF extensive-form games with zero error terms (Lanctot et al. 2012). We only have to show that NFGSS-CFR$^+$ performs exactly the same updates as it would perform in the equivalent EFG defined above. Focus on a specific information set I in iteration t and assume all updates in previous iterations were equivalent. On iteration t, the current strategy is the same and therefore, the values $v(a_i)$ and $v(s_i)$ computed on lines 3-9 of the algorithm represent the same counterfactual values as the values computed on the equivalent EFG based on Equation 1. Since the update is defined only in terms of counterfactual values and the current strategy (in case of mean strategy), all updates are exactly the same.



(a) Transit game scheme (b) IG passanger's MDP

Figure 3: Evaluation domain schemas.

Experimental Evaluation

Transit game (TG) is the game used for evaluation in (Bosansky et al. 2015). For fair comparison, we have implemented CFR$^+$ within their publicly available code base. The game is a search game in Euclidean space discretized as an eight-connected rectangular grid, such as in Figure 3a. The evader attempts to cross this grid from left to right without meeting the patroller. Each move incurs a small penalty (0.02) to the evader. The patroller starts in his base marked in grey and moves on the graph along the edges (or stays at its current node) for d time steps. If he is not back at the base at the end of the game, he suffers a penalty (20). The movement of each player fails with probability 0.1, which causes the player to stay at its previous position. Every time the players meet, the evader looses one point of utility to the patroller. The authors in (Bosansky et al. 2015) approximate this game as zero-sum; hence, all the penalties suffered by one player are considered to be gains of the other player. In the evaluation below, we consider the transit game of size w to be played on $w \times 2w$ grid for $d = 2w + 4$ time steps.

Ticket inspection game (IG) is based on (Jiang et al. 2013). It models scheduling a three hours long shift of ticket inspectors on a single train line. Jiang at al. use real world data from the LA metro system and schedule for multiple patrols. We consider a single patrol and generate synthetic data to create problem instances of varying size. The data include (1) the train schedules, which are in both directions generated non-uniformly, starting with approximately 5 times longer intervals at the beginning of the shift than in the peak time close to the end of hour two of the shift; (2) the passenger trips defined by the number of passengers that take each train between each pair of stations generated pseudo-randomly based on non-uniform station popularity values; and (3) the time intervals in which a train reaches a following station. The patrol starts in the middle of the line. In a station, the patrol can take the next train in either direction or check the tickets of the passengers in the station for 15 minutes. On a train, she can either check the tickets of the passengers on the train until the next stop, or exit the train. The patrol checks 5 passengers per minute and the passengers

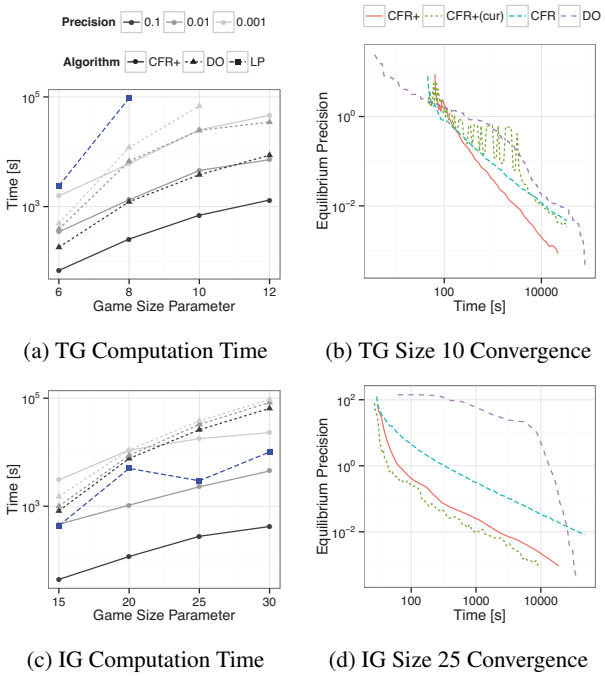
are assumed to be present in the stations 3 minutes before their train departs and 3 minutes after it arrives. Because of unexpected delays, the patrol can with probability 0.1 miss the train it intended to take and stay at the station, or fail to exit the train at the intended station and stay in the train until the next one. The game has originally been modelled as a Bayesian game with passengers taking the same trip as types, but it can be also seen as NFGSS. Instead of types, we model the passengers' strategy by the MDP in Figure 3b. The MDP starts with a dummy action with one outcome for each trip, occurring with the probability that a random passenger takes the trip. In each of these outcomes, the passenger chooses an action representing either buying a ticket or not buying a ticket. The MDP of the patroller starts with a dummy action of collecting the fare money, which gives him a reward of \$1.50 for each passenger that buys a ticket. Afterwards, the MDP describes her movement on the line and the patrol gets a reward of \$100 for each checked passenger which did not buy a ticket. The game is zero-sum, with the patrol maximizing its revenue per passenger and each passenger type minimizing its cost. The ticket inspection game of size s has s stations, s trains in each direction and approximately s thousand passengers.

Results

We compare the run time of the proposed NFGSS-CFR$^+$ algorithm to the full compact strategy LP formulation and the double oracle (DO) algorithms proposed in (Bosansky et al. 2015). Since CFR$^+$ is generally used as an approximative algorithm, we compare the run time of the algorithms with various target precision: 0.1, 0.01 and 0.001 in absolute utility value. For solving LPs, we used IBM CPLEX 12.51. The simplex algorithm is substantially faster than the barrier method on large instances of these problems, so we use simplex. The precision of CPLEX is by default set to 10^{-6}. Setting it to a higher value increased the run time, most likely because of "Harris' method"-like feasibility bounds shifts. The limit on these shifts is also controlled by the precision parameter. When these shifts are eventually removed, the solution is in a substantially more infeasible state, which is harder to correct and the correction has a larger negative effect on optimality (Klotz and Newman 2013). Therefore, we evaluate only the default setting. The results presented in Figures 4(a,c) are means of 10 runs with small a variance and Figures 4(b,d) are each a single representative run.

The precision of 0.01 in the transit game is half the penalty the evader pays for each move. We compute the utility in IG in cents; hence, the error of 0.01 per passenger means a loss of \$3 of overall revenue in the largest game with 30 thousand passengers. Since the models are always an approximation of the real world problems, we consider error of 0.1 to be perfectly acceptable and the error smaller than 0.01 irrelevant from the domain perspective.

Figure 4a presents the computation times required by the algorithms to reach the given precision in TG. This domain is suitable for the double oracle algorithm; hence, in games of all evaluated sizes, DO outperforms LP for any precision. LP was not able to solve the larger games within 30 hours. For precision of 0.1 and 0.01, CFR$^+$ clearly outperforms DO



(a) TG Computation Time (b) TG Size 10 Convergence

(c) IG Computation Time (d) IG Size 25 Convergence

Figure 4: Computation times and convergence curves for Transit (a,b) and Ticket inspection (c,d) games. CFR+(cur) is the current strategy of CFR$^+$.

for all game sizes by up to factor of 5 (note the log scale). For the largest game with width 12 (5041 and 6118 states in the MDPs), CFR$^+$ finds the solution with precision 0.01 in 119 minutes, while for DO, it takes almost 10 hours. Two hours are enough for DO to reach precision of 0.01 in game of size 8 with 1729 and 2036 MDP states. For very high precision, DO already outperforms CFR$^+$ on the smallest game instance. Figure 4b presents the progress of the convergence in time on a log-log plot. Besides CFR$^+$ and DO, we present also the adaptation of standard CFR to NFGSS analogous to the presented adaptation of CFR$^+$, and the performance of the current strategy of CFR$^+$. The current strategy has no formal guarantees, but has been observed to converge to an equilibrium in some poker games. We can see that the mean strategy of CFR$^+$ converges the fastest, but the current strategy also converges, which allows reducing the memory requirements of the algorithm by not storing of the average strategy. DO initially converges slowly, but eventually starts converging very quickly, which allows it to reach higher precisions before CFR$^+$.

We present the exact same experiments for the ticket inspection game in Figures 4c and 4d. The LP formulation of the game always outperforms DO. With the exception of the smallest game, CFR$^+$ reaches precision of 0.1 and 0.01 before the LP finishes. In the largest game (4699 and 15362 MDP states), CFR$^+$ reaches the precision of 0.01 after 74 minutes, while the LP requires almost 226 minutes. Even in this domain, CFR$^+$ converges faster than CFR. The current

strategy converges even faster than the average strategy.

Conclusions

We study an abstract class of games called NFGSS, suitable for modelling a variety of real world security problems. We transfer several key poker research results to this class. We extend the previously studied notion of well-formed imperfect-recall games and show that after this extension, it can model any game from NFGSS. We propose an adaptation of CFR$^+$ for this class and provide a formal guarantee that it converges to a Nash equilibrium. We empirically show that with a small loss of precision, it allows solving larger problem instances with up to five times less computation time than the currently used approaches. With CFR$^+$, we can solve the game models more frequently (e.g., if they are not entirely static) or solve substantially larger game instances in the same time as with standard methods.

This paper opens two natural directions of future research. First, since incremental game generation in double oracle algorithms is, to a large extent, orthogonal to the actual equilibrium solving algorithm, combining it with CFR$^+$ might lead to an additional speedup. Second, the link we established between NFGSS and CRSWF games can help to further extend these classes of games to allow modelling more complex interactions between players' strategies.

Acknowledgments

We thank Branislav Bosansky for providing the source codes and support for the LP based methods. This research was supported by Alberta Innovates Technology Futures through the Alberta Innovates Centre for Machine Learning and Reinforcement Learning and AI Lab, and used the computing resources of Compute Canada and Calcul Quebec.

References

Bosansky, B.; Jiang, A. X.; Tambe, M.; and Kiekintveld, C. 2015. Combining compact representation and incremental generation in large games with sequential strategies. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Bowling, M.; Burch, N.; Johanson, M.; and Tammelin, O. 2015. Heads-up limit holdem poker is solved. *Science* 347(6218):145–149.

Durkota, K.; Lisy, V.; Bosansky, B.; and Kiekintveld, C. 2015. Optimal network security hardening using attack graph games. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 526–532.

Fang, F.; Stone, P.; and Tambe, M. 2015. Defender strategies in domains involving frequent adversary interaction. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, 1663–1664. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Jain, M.; Korzhyk, D.; Vaněk, O.; Conitzer, V.; Pěchouček, M.; and Tambe, M. 2011. A double oracle algorithm for zero-sum security games on graphs. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 327–334. International Foundation for Autonomous Agents and Multiagent Systems.

Jiang, A. X.; Yin, Z.; Zhang, C.; Tambe, M.; and Kraus, S. 2013. Game-theoretic randomization for security patrolling with dynamic execution uncertainty. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, 207–214. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Klotz, E., and Newman, A. M. 2013. Practical guidelines for solving difficult linear programs. *Surveys in Operations Research and Management Science* 18(12):1 – 17.

Koller, D., and Megiddo, N. 1992. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior* 4(4):528 – 552.

Kroer, C., and Sandholm, T. 2014. Extensive-form game imperfect-recall abstractions with bounds. *arXiv preprint arXiv:1409.3302*.

Lanctot, M.; Gibson, R.; Burch, N.; Zinkevich, M.; and Bowling, M. 2012. No-regret learning in extensive-form games with imperfect recall. In Langford, J., and Pineau, J., eds., *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, ICML '12, 65–72. New York, NY, USA: Omnipress.

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 joint conference on Autonomous agents and multiagent systems: industrial track*, 125–132. International Foundation for Autonomous Agents and Multiagent Systems.

Tammelin, O.; Burch, N.; Johanson, M.; and Bowling, M. 2015. Solving heads-up limit texas hold'em. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*.

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 *Proc. of the 8th Int. Conf. on Autonomous Agents and Multiagent Systems*, 37–44.

Waugh, K.; Zinkevich, M.; Johanson, M.; Kan, M.; Schnizlein, D.; and Bowling, M. 2009. A practical use of imperfect recall. In *Proceedings of the 8th Symposium on Abstraction, Reformulation and Approximation (SARA)*, 175–182.

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.

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, M.; Johanson, M.; Bowling, M.; and Piccione, C. 2008. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, 1729–1736.

The AAAI-17 Workshop on
Computer Poker and Imperfect Information Games
WS-17-06

Eqilibrium Approximation Quality of Current No-Limit Poker Bots

Viliam Lisý[a,b]
[a]Artificial intelligence Center
Department of Computer Science, FEL
Czech Technical University in Prague
viliam.lisy@agents.fel.cvut.cz

Michael Bowling[b]
[b]Computer Poker Research Group
Alberta Machine Intelligence Institute
Dept. of Computing Science, University of Alberta
mbowling@ualberta.ca

Abstract

Approximating a Nash equilibrium is currently the best performing approach for creating poker-playing programs. While for the simplest variants of the game, it is possible to evaluate the quality of the approximation by computing the value of the best response strategy, this is currently not computationally feasible for larger variants of the game, such as heads-up no-limit Texas hold'em. In this paper, we present a simple and computationally inexpensive Local Best Response method for computing an approximate lower bound on the value of the best response strategy. Using this method, we show that existing poker-playing programs, based on solving abstract games, are remarkably poor Nash equilibrium approximations.

One very popular measure for progress in artificial intelligence is computers' performance in recreational games commonly played by humans. There has been a dramatic sequence of successes in the past two decades starting with Chinook with checkers (Schaeffer et al. 1996), Deep Blue with chess (Campbell, Hoane, and Hsu 2002), Watson with Jeopardy! (Ferrucci et al. 2013), and AlphaGo with go (Silver et al. 2016). Despite these successes, poker has proven to be a harder challenge for AI. Similar to checkers, chess, and go, poker can be easily and completely described by a simple set of rules. The size of the game ranges from being smaller than checkers (heads-up limit Texas hold'em; HULH) to as large as go (heads-up no-limit Texas hold'em; HUNL). However, it is a substantially more complex game due to the element of imperfect information. In poker, some information is private to specific players. Players need to infer the private information of others based on their actions in the game, while also seeking to avoid losing their own strategic advantage by revealing their private information via their actions. This is complicated by the fact that the information revealed by an opponents' play depends on their behaviour, and that behaviour naturally depends on the player's own private information.

Like other popular games, poker has been a challenge problem in artificial intelligence since the inception of the field (Kuhn 1950; Koller and Pfeffer 1997; Billings et al. 2002). Recent progress in approximating Nash equilibria in massive extensive-form games (Gilpin et al. 2007; Zinkevich et al. 2008; Lanctot et al. 2009; Tammelin et al. 2015) has allowed for some initial successes in the smallest variant of poker played by humans, HULH. Polaris, a program built around the CFR-family of methods, defeated professional poker players for the first time in a meaningful match of HULH in 2008 (Rehmeyer, Fox, and Rico 2008). In 2015, this was taken a step further by the program Cepheus, which essentially solved the game the HULH, with the resulting strategy requiring 11TB of storage and using over 900 CPU years of computation (Bowling et al. 2015).

The size of the game of HULH, though, is trivial compared to the more popular no-limit variants of the game. In limit poker variants, there are at most 3 different actions available to a player at any situation (fold, check/call, bet/raise) as all bet amounts are fixed in advance. In no-limit games, players can bet any number of remaining chips, leading to thousands of possible actions from a single situation. As a result, HUNL can have over 10^{160} decision points in the game, as in the case of the variant played in the Annual Computer Poker Competition (ACPC) [1]. This makes HULH's size of 10^{14} decision points seem trivial.

Much of the progress in HULH was enabled by the ability to measure the approximation quality of a strategy in HULH (Johanson et al. 2011). This has been a challenge for research in HUNL where to date evaluation has been limited to tournament evaluation,where strategies are evaluated by having them play against each other. The results of such a tournament are necessarily relative and cannot give an absolute strength of any particular program. Furthermore, it can substantially depend on small details in the design of the tournament, such as the winnings cap used in the ACPC (Bard 2016). The absolute measure of performance through computing a strategy's approximation quality made possible a number of important strides in HULH research, for example, investigating the effect of restricted opponent modelling, asymmetric abstractions (Bard, Johanson, and Bowling 2014), translation, and payoff tilts (Johanson et al. 2011).

This paper presents a simple method to quickly approximate a lower-bound to a strategy's exploitability in the HUNL game. It is trivial to parallelize, makes no card abstraction commitments, can be applied even to strategies

Copyright © 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1]www.computerpokercompetition.org

that use dynamic (and expensive) endgame solving techniques, and can probe a far larger portion of the total betting space than any current techniques use in their solving approach. Using this technique, we compare a number of the top HUNL programs from the ACPC. We show that even though the differences among the top performing agents in the ACPC are tiny fractions of a blind per hand, the exploitability of the players is several whole blinds per hand. For every program tested, it would be far less exploitable to immediately fold every hand than to use even such a state-of-the-art startegy. Furthermore, we can tease apart the source of the approximation error, observing that considerably more exploitability can be attributed to card abstraction as opposed to betting abstraction.

Current No-Limit Poker Bots

While we are aware there are poker programs (or bots) playing online in real money games, this paper focuses on bots submitted to the ACPC. These bots are developed by top research teams, use principled AI approaches, and the techniques they use are to large extent well documented.

Heads-up No-Limit Texas Hold'em

Heads-up no-limit Texas hold'em is a variant of poker played with a standard deck of 52 cards. At the beginning of each hand or *game*, the first player enters a *big blind* into the *pot*; the second player enters half of that size, or the *small blind*; and both players are then dealt two *private cards*. The second player then starts the first round of betting. The players alternate in choosing to *fold* – ending the game and letting the opponent take the pot; *call* – matching the amount of chips entered by the opponent and ending the betting round; or *raise* by x – adding x more chips than the opponent to the pot. A raise of all remaining chips is called an *all in* bet. After the first round, three board or *public cards* are dealt face up, and the first player now starts an identical round of betting to the first round. In the third and fourth rounds, one additional public card is dealt and betting starts again with the first player. If none of the players folds before the end of round 4, the game enters *show-down*: The private cards are revealed and the pot is won by the player that can compose the strongest hand of 5 cards using his 2 private and the 5 public cards. A *match* consist of large number of games, in which the players alternate their positions as the first and the second player.

Best ACPC Players

The bots annually submitted to the ACPC include programs based on hand-crafted rules, learning systems trained on logs of past games, or advanced linear programming methods. The bots which have seen the most success in HUNL all have the same basic structure.

The bots are based on creating a smaller abstract version of the game, approximating the equilibrium strategy in the abstract game, and executing this strategy in the original game using a translation method to map real game situations into the abstraction. The abstract games abstract card information (called information abstraction) based on clustering hands with similar strength and potential to improve after additional cards are dealt. The abstract games abstract betting information by restricting the available bets to a small handful, usually expressed as fractions of the current size of the pot. The most successful bots approximate the Nash equilibrium in the abstract games using some variant of the Counterfactual Regret Minimization algorithm (Zinkevich et al. 2008).

Bot Name	Authors	Winnings (mBB/h)
Baby Tartanian8	Carnegie Mellon Uni.	0.0 ± 0.0
Slumbot	Eric Jackson	-11.88 ± 10.33
Act1	Unfold Poker	-16.82 ± 14.35

Table 1: ACPC 2016 results.

While playing a hand, the bots find the abstract state that corresponds to (cluster that includes) the current state of the game represented by the exact private and public cards in the game. Furthermore, they have to map the real betting sequence in the game that can use any size bets to the "most similar" betting sequence represented in the abstraction. The abstract strategy is queried for the probability distribution over actions (pot fractions) included in the abstract game and these are then post-processed and played in the actual game, if they are applicable. There are many publications related to each step of this process in the AI literature.

All three top performing players in ACPC 2016 match the high level description above. Their Instant Runoff Competition Results are summarized in Table 1. Baby Tartanian8 won the competition, Slumbot lost on average 12 mBB/h in its matches with the winner and Act1 lost 17 mBB/h on average against the other two agents.

Local Best Response

This section presents the local best response algorithm for fast approximation of a lower bound on the exploitability of no-limit poker strategies. We call the player that computes the best response "LBR", and its opponent "the opponent". The key concept in this algorithm is the probability that the opponent holds each of the possible private hands, which we call the opponent's *range*. At the very beginning of the game, it is equally likely that the opponent holds any pair of private cards, which is not in conflict with the cards held by the player. The probabilities of actions performed by the opponent depend on the private hand she holds. Therefore, with access to the strategy of the opponent, we can use Bayes' rule to infer the exact probabilities that the opponent holds each of the private hands. It is important that there is no abstraction or other approximation needed to exactly represent these probabilities. Based on the range, local best response greedily approximates best response actions, assuming a simple heuristic for behavior in the future.

Let $\mathcal{H}$ be the set of all possible *private hands*. In HUNL, each private hand consists of two cards. Ignoring their ordering, $|\mathcal{H}| = 1326$. A player's range is a probability distribution over hands and we denote it $\pi : \mathcal{H} \to [0, 1]$. We denote $\mathcal{S}$ the set of *public states* in the game. Each public state consists of the board cards, the order in which they came, and the complete sequence of bets by both players up to some

LocalBR(π - range, $s \in \mathcal{S}, h_i \in \mathcal{H}$)

1: $wp = WpRollout(h_i, \pi, s)$
2: $asked = pot_{-i}(s) - pot_i(s)$
3: $U(call) = wp \cdot pot(s) - (1 - wp) \cdot asked$
4: **for** action a in considered bets / raises **do**
5: $fp = 0$
6: **for** opponent's hands $h_{-i} \in \mathcal{H}$ **do**
7: $fp = fp + \pi(h_{-i}) \cdot \sigma(s, h_{-i}, \text{fold})$
8: $\pi'(h) = \pi(h) \cdot (1 - \sigma(s, h_{-i}, \text{fold}))$
9: normalize π'
10: $wp = WpRollout(h_i, \pi', s)$
11: $U(a) = fp \cdot pot(s) +$
$+(1 - fp) \cdot (wp \cdot (pot(s) + a) - (1 - wp) \cdot (asked + a))$
12: **if** $\max_a U(a) > 0$ **then**
13: **return** $\arg\max_a U(a)$
14: **else**
15: **return** fold

Figure 1: The algorithm for approximating lower bound on strategy exploitability.

point in the game. The *strategy* of a player is a probability distribution on actions (fold, call, all different bet sizes) from $\mathcal{A}$, available to a player: $\sigma : \mathcal{S} \times \mathcal{H} \times \mathcal{A} \to [0, 1]$.

The algorithm does not compute a best response strategy explicitly, but uses its local approximation to directly play against the evaluated strategy. At the beginning of each hand, LBR initializes the opponent's range uniformly for all private hands that do not include the cards dealt to LBR. After each action a of the opponent performed at a public state s, LBR updates the opponent's range using her strategy σ:

$$\pi(h) = \pi(h) \cdot \sigma(s, h, a)$$

and normalizes the distribution to sum to one.

LBR chooses its action to maximize its expected utility, under the assumption that the game will be checked/called until the end, unless the opponent folds right after LBR's action. The pseudocode is given in Figure 1. First, it computes the probability of winning the current hand if the game continues until the show-down in function *WpRollout*. This function exhaustively deals all possible remaining board cards and computes the mean probability of winning with hand h_i against the opponent's range π. The expected utility of actions is computed with respect to utility 0 for fold. On line 3, the utility of action *call* for LBR is computed as the chips currently in the pot in case LBR wins, and the negative of the money LBR has to add to the pot in order to continue playing if it loses. Afterward, the algorithm computes the expected utility for all other considered actions. These are typically defined as a fixed set of pot fractions, but they can be arbitrary. For each action, lines 6-8 compute the probability that the opponent will fold after LBR performs the action given the current range (fp) and the new range that would hold for the opponent in case she does not fold (π'). The expected utility of the action for LBR (line 11) is computed as getting the whole pot if the opponent folds, getting the pot and the size of the bet (a) if the opponent does not fold and LBR wins, and losing the chips asked for and added otherwise.

This algorithm computes an approximation of the best response, looking only one action ahead and assuming that the players will check until the end of the game after performing the action (and not folding). The main advantage it exploits is that it perfectly understands the cards it holds and the state of the game without any abstraction. It may be easily extended to longer look-ahead or more complicated heuristics for estimating the value of the remainder of the game, however, it would substantially increase its computational requirements and even this simple version is very effective, as we show in the following section.

Computing Lower Bound on Exploitability

Recall that LBR does not pre-compute the best response approximation, but rather directly uses it to evaluate an input strategy. The evaluation consists of playing a large number of regular poker hands. The cards are dealt randomly as in a regular game. Every time it is the opponent's turn to play an action, her strategy is queried for the right probability distribution and an action is sampled based on the actual private hand held by the opponent. Every time it is LBR's turn to play, it updates the opponent's range and selects the best action based on the LocalBR algorithm in Figure 1 and its private hand. The estimate of the exploitability is the average number of chips LBR wins in these games.

LBR queries the opponent's strategy for each hand after each action it considers. Therefore, if we want to run LBR with n different pot fractions, completing 1 hand with LBR generally requires at most $(n|\mathcal{H}| + 1)$ times more computation time than playing a regular match with the strategy. Furthermore, the most expensive computation currently used in advanced poker bots is endgame solving (Ganzfried 2016), which solves for all possible hands in one computation. If this is the dominant part of the computation for a strategy, LBR evaluation requires approximately $(n + 1)$ times the computation required for playing a hand. If it is possible to play a game within few minutes on a single computational node, it is most likely also feasible to get reliable LBR estimates on a cluster of these nodes.

An important property of this algorithm is that it computes a lower bound on the exploitability of strategies. Since LBR actually plays a legal poker strategy, it can never win more in expectation than the worst case opponent of a strategy. Similarly, using longer look-ahead or different heuristic evaluation, as suggested above, would still have this property. Therefore, a strategy with substantially better performance against LBR is likely to be closer to an equilibrium.

Sampled soft translation

When mapping the solution of the abstract game to the actual game played, it may be convenient to use sampling (Schnizlein, Bowling, and Szafron 2009; Ganzfried and Sandholm 2013). As a result, it may be difficult to obtain the exact probability distribution over actions in a particular public state of the real game. The proposed local best response method can still work in this situation. We can approximate the actual distribution by averaging a larger number of samples of the strategy at the same state. If we use a

Betting	Rounds	Call	$\frac{1}{2}$Call $\frac{1}{2}$Raise	Random
fc	1-4	0	7.1 ± 0.5	15.7 ± 0.4
fc	3-4	0	16.2 ± 0.3	2.2 ± 0.7
fcpa	1-4	34.0 ± 0.5	23.1 ± 0.5	39.1 ± 0.6
fcpa	3-4	49.0 ± 0.4	24.4 ± 0.6	80.7 ± 0.7

Table 2: Results of LBR with trivial strategies in BB/h.

new independent sample from the strategy to pick the actual action played by the opponent, LBR does not learn any extra information about the action played and therefore computes a lower bound on the exploitability of the strategy.

Variance reduction

Since the evaluation plays-out standard poker hands, we can use any of the previously developed variance reduction techniques (White and Bowling 2009; Davidson, Archibald, and Bowling 2013; Burch et al. 2017) to reduce the number of hands required to produce statistically significant results. For the experiments presented in this paper, we used duplicate matches and imaginary observations of expected outcomes of all hands the opponent could hold for a given line of play, instead of just the actual hand she holds. These two techniques combine to reduce the size of the confidence intervals by roughly 20% with the same number of matches.

Experimental evaluation

In this section, we show that the proposed LBR computation is a fast and effective method to compute a lower bound approximation on exploitability of no-limit poker strategies. We present results from running LBR on simple chump strategies; bots created at University of Alberta for past ACPCs; the two of the top three bots from the ACPC in 2016; and a huge strategy with a very sparse betting abstraction, but no card abstraction. Most results in this section will be presented in milli-big-blinds per hand (mBB/h) or whole big blinds per hand (BB/h). The evaluation is stochastic; hence, we also present 95% confidence intervals. In order to understand the common magnitude of these values, a bot that always folds as the first action would loose 750 mBB/h. The results of the one-on-one matches of the best three players in ACPC 2016 were all decided by less than 24 mBB/h.

In addition to showing the efficiency of the LBR computation, our experimental results also show that a large portion of the exploitability uncovered by the tool is caused by card abstraction. We show that using bets outside of the opponent's betting abstraction does add to the bot's exploitability, but at a significantly less magnitude.

Chumps

In order to better understand the strengths and limitations of LBR, we first use it to evaluate simple rule-based strategies that ignore the cards completely.

First, we consider always calling, regardless of the cards. A best response (optimal counter-strategy) against this strategy is to wait until all cards are dealt and go all-in if the probability of winning is higher than 0.5. This strategy gains on average $\frac{1}{4}$ of all players chips per hand. The best response would go all-in on half of the hands and it would win $\frac{3}{4}$ and lose $\frac{1}{4}$ of these bets. With the stack of 200 big blinds used in the ACPC, the exploitability of the always call strategy is approximately 50 BB/h.

The results of LBR are presented in Table 2. If we limit LBR to choose only actions fold or call (note, it has no reason to fold), both players play always call and the expected value of LBR is 0. If we include any bets, LBR always uses only the largest one against an opponent that never folds. If we use LBR in all rounds of the game, LBR gains 33 BB/h. The reason it does not achieve the best-response value is that LBR greedily bets all-in as soon as the probability of winning is higher than 0.5. In the situations in which the probability drops below the threshold until the end of the game, it loses utility compared to the actual best response. If we force LBR to call in the first two rounds and allow other bets only in rounds 3-4, this effect is minimized and LBR gains almost the whole 50 BB/h. If we use LBR only in the last round and call in the remaining 3, LBR actually gets the full 50 BB/h in expectation.

Second chump strategy we evaluate is calling with 50% of actions and playing a random raise otherwise. Since this strategy raises often, LBR can already gain several blinds per hand even using only fold and call. Even with this strategy, checking until more public cards are dealt increases the performance of LBR. Overall, this strategy seems to be less exploitable than always calling, since it forces LBR to fold and to commit more chips with less information about the cards.The results in Table 2 show that LBR can already gain several blinds per hand.

The last chump strategy we evaluate is a random legal action. This strategy is most exploitable. The reason is that after the all in bet, only fold and call are legal actions. Hence, the bot will fold half of his hands in this situation, even the hands it would otherwise clearly win.

ACPC agents

We continue with evaluating the agents from past ACPC competitions. All these agents are based on solving a smaller abstract game by CFR and using a translation mechanism to use the strategy in the full game. The agents differ mainly in construction of the abstract game and modifications of the CFR algorithm to speed-up convergence at relevant parts of the game. The specific bots are Hyperborean 2013 and 2014 created by the University of Alberta, and the second and third best performing bots from ACPC2016. We approached the authors of all three placing submissions from the competition, but only the two were able to support this evaluation before the paper deadline. Furthermore, the winning submission used purification to meet the competition disk limit (Brown and Sandholm 2016) and therefore we expect it to be highly exploitable.

Table 3 summarizes the results. We evaluate the players with LBR's betting restricted to just fold and call (fc); fold, call, pot, and all in (fcpa); the bets that are used by the agent in its abstract game solution (on-tree); and fold, call, all in, and 55 pot fractions computed as $0.05 \cdot (1.15)^k$ for $k = 0 \dots 54$ (56bets). If a pot fraction is not applicable, min-bet or all in is evaluated instead. The column Rounds in

Betting	Rounds	Hyp. 2013	Hyp. 2014	Slumbot 2016	Act1 2016	Full Cards
fc	1-4	1048 ± 68	721 ± 56	522 ± 50	407 ± 47	-424 ± 37
fc	3-4	1006 ± 76	608 ± 61	496 ± 55	390 ± 55	-819 ± 52
fcpa	3-4	4040 ± 147	3852 ± 141	4020 ± 115	2597 ± 140	-536 ± 87
on-tree	3-4	4743 ± 163	4789 ± 156			-536 ± 87
56 bets	1-4	619 ± 117	574 ± 125	763 ± 84	2429 ± 134	1607 ± 76
56 bets	3-4	5062 ± 152	4675 ± 152	3763 ± 104	3302 ± 122	2403 ± 87

Table 3: Lower bound on exploitability (in mBB/h) of ACPC bots and a bot with no card abstraction and fold-call-pot-all_in betting computed using Local Best Respons restricted to the given betting options and to check/call out of the denoted rounds.

the table defines the rounds (or streets) in which LBR actually computed the local best response to choose actions. The remaining rounds were always check/call. All the results are averaged over $2\times50{,}000$ duplicate hands. It means that each hand was played by each player on the first as well as the second position to reduce the effect of luck.

The main result is that with 97.5% confidence, exploitability of each evaluated bot is over 3180 mBB/h. It means that folding every hand would cost the bots at least 4 times less money than playing their strategy against their worst case opponent. Second, it is important to wait when using LBR. If we use all 56 bets from the very beginning, LBR often exploits the opponent almost a full order of magnitude less than if we force it to check in the first two rounds and use the 56 bets only afterwards. The reason, as in the case of always call, is the greedy nature of LBR. It places large bets too early, without sufficiently exploiting the information it might learn later. Alternatively, it pushes the opponent to folds before she places more money in to the pot to make a larger gain. Third, all bots lose substantially even if LBR is allowed to play only fold and call. Allowing fold only later in the game does not seem to be beneficial, since the ability to exploit the information LBR learns during the hand is very limited. Most of the exploitation of LBR against all bots is realized with only using a single pot-bet option ("fcpa" row in Table 3). For Hyperboreans, exploitability is further increased by adding more actions, but adding actions beyond the actions used in the abstract game (on-tree) helps only with the 2013 version and does not help at all for the 2014 version. For Slumbot and Act1, we do not know the exact betting abstraction used in their abstract game, but the betting options in "fcpa" are almost always included. If these bets are included, as in case of Hyperboreans, there is no need for any translation, the play will never leave the precomputed abstract tree and all the exploitability is caused only by the errors in the card abstraction. Recent APCP bots are generally well converged within their abstract games.

The least exploitable bot in these experiments is Act1, even though it was beaten in one-on-one play by Slumbot in the competition. It confirms that as with full-game best response, even LBR may not be indicative of actual one on one performance (and vice versa).

The experiments with ACPC bots were performed on a cluster of AMD Opteron 6172 nodes with 24 cores, 32 GB of RAM and the strategies on a shared network drive. Typically, we were running 10-20 instances of LBR on each node in batches of 1000 hands. One batch for fold-call betting completed within half an hour, one batch of 56 bets experiments generally took up to 8 hours. Since an important bottle neck is the disk/network bandwidth which is further improved by caching, the variance on required resources is rather large. Still, computing good LBR values even for complicated strategies with endgame solving is perfectly feasible with the presented method.

Full cards

The last bot we evaluate is a large bot that uses complete non-abstracted information about the cards and the sparse "fcpa" betting abstraction. It plays a slightly smaller game with a 100 BB stack. The results on this bot (Table 3) show that LBR does not realize the actual best response and can even be substantially beaten (i.e., an uninformative lower bound approximation). When restricted to the same "fcpa" abstraction used in the opponent's abstraction, a full (non-local) best response shows the opponent is exploitable for 90 mBB/h, while LBR loses 536 mBB/h. However, the solution with "fcpa" abstraction is not sufficient to ensure low exploitability in the whole game with existing translation techniques. With hard translation used in bets off-tree, LBR wins 2403 mBB/h against this bot using 56 bets only in the last two rounds of the game. Soft translation seems to mitigate this problem a little, but definitely does not solve it. Using sampled soft translation with 10 samples for estimating the strategy, LBR on the last two rounds is winning 1981 ± 224. The larger confidence interval is caused by playing fewer hands, since even the look-up in the huge compressed strategy without card abstraction is expensive. Note that not all 56 actions are necessary to see the high exploitability using LBR. It is sufficient to use several bets out of the original betting abstraction. For example, the bot with hard translation looses 1849 mBB/h with only fold, call, min-bet, 2, 4, 8 times pot bet, and the all in bet.

Conclusions

This paper presents the Local Best Response method for fast approximation of exploitability of large poker strategies. If a bot is able to provide a strategy for all hands it could hold in a specific public state of the game within a reasonable time (i.e., minutes), this method can generally be used to approximate its exploitability. This is also the case for all published endgame solving techniques, which resolve a subgame for all possible private hands at once.

Using this method we show that the existing poker bots, including the second and the third best performing bots in

the ACPC in 2016, all have exploitability substantially larger than folding all hands. The bots that use card abstraction are losing over 3 big blinds per hand on average against their worst case opponent. Exploitability can be reduced by not using card abstraction, but that necessarily leads to using a very sparse betting abstraction, which can be heavily exploited as well. Therefore, we assume that a substantial paradigm shift is necessary to create bots that would closely approximate equilibrium in full no-limit Texas hold'em.

Acknowledgements

We would like to thank ACPC 2016 poker bot authors Eric Jackson and Tim Reiff for providing their bots and implementing the interface that allowed us to use LBR to evaluate them. Our tool is based on the University of Alberta Computer Poker Research Group's code base and we are grateful to all current and previous members that contributed to its development. Computing resources were provided by Calcul Quebec, Westgrid, and Compute Canada. This work was partially supported by Czech Science Foundation (15-23235S).

References

Bard, N.; Johanson, M.; and Bowling, M. 2014. Asymmetric abstractions for adversarial settings. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, 501–508. International Foundation for Autonomous Agents and Multiagent Systems.

Bard, N. D. C. 2016. *Online Agent Modelling in Human-Scale Problems*. Ph.D. Dissertation, University of Alberta.

Billings, D.; Davidson, A.; Schaeffer, J.; and Szafron, D. 2002. The challenge of poker. *Artificial Intelligence* 134(1):201–240.

Bowling, M.; Burch, N.; Johanson, M.; and Tammelin, O. 2015. Heads-up limit holdem poker is solved. *Science* 347(6218):145–149.

Brown, N., and Sandholm, T. 2016. Baby tartanian8: Winning agent from the 2016 annual computer poker competition. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, 4238–4239.

Burch, N.; Schmid, M.; Moravcik, M. M.; and Bowling, M. 2017. Aivat: A new variance reduction technique for agent evaluation in imperfect information games. In *AAAI-17 Workshop on Computer Poker and Imperfect Information Games*.

Campbell, M.; Hoane, A. J.; and Hsu, F.-h. 2002. Deep blue. *Artificial intelligence* 134(1):57–83.

Davidson, J.; Archibald, C.; and Bowling, M. 2013. Baseline: practical control variates for agent evaluation in zero-sum domains. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 1005–1012. International Foundation for Autonomous Agents and Multiagent Systems.

Ferrucci, D.; Levas, A.; Bagchi, S.; Gondek, D.; and Mueller, E. T. 2013. Watson: beyond jeopardy! *Artificial Intelligence* 199:93–105.

Ganzfried, S., and Sandholm, T. 2013. Action translation in extensive-form games with large action spaces: Axioms, paradoxes, and the pseudo-harmonic mapping. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, 120–128. AAAI Press.

Ganzfried, S. 2016. Reflections on the first man vs. machine no-limit texas hold'em competition. *ACM SIGecom Exchanges* 14(2):2–15.

Gilpin, A.; Hoda, S.; Pena, J.; and Sandholm, T. 2007. Gradient-based algorithms for finding nash equilibria in extensive form games. In *International Workshop on Web and Internet Economics*, 57–69. Springer.

Johanson, M.; Waugh, K.; Bowling, M.; and Zinkevich, M. 2011. Accelerating best response calculation in large extensive games. In *IJCAI*, volume 11, 258–265.

Koller, D., and Pfeffer, A. 1997. Representations and solutions for game-theoretic problems. *Artificial intelligence* 94(1):167–215.

Kuhn, H. W. 1950. A simplified two-person poker. *Contributions to the Theory of Games* 1:97–103.

Lanctot, M.; Waugh, K.; Zinkevich, M.; and Bowling, M. 2009. Monte carlo sampling for regret minimization in extensive games. In *Advances in Neural Information Processing Systems*, 1078–1086.

Rehmeyer, J.; Fox, N.; and Rico, R. 2008. Ante up, human: The adventures of polaris the poker-playing robot. *Wired* 16:186–191.

Schaeffer, J.; Lake, R.; Lu, P.; and Bryant, M. 1996. Chinook the world man-machine checkers champion. *AI Magazine* 17(1):21.

Schnizlein, D.; Bowling, M. H.; and Szafron, D. 2009. Probabilistic state translation in extensive games with large action sets. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 278–284.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Tammelin, O.; Burch, N.; Johanson, M.; and Bowling, M. 2015. Solving heads-up limit texas holdem. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 645–652.

White, M., and Bowling, M. H. 2009. Learning a value analysis tool for agent evaluation. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 1976–1981. Citeseer.

Zinkevich, M.; Johanson, M.; Bowling, M.; and Piccione, C. 2008. Regret minimization in games with incomplete information. *Advances in Neural Information Processing Systems* 20:1729–1736.

Convergence of Monte Carlo Tree Search in Simultaneous Move Games

Viliam Lisý[1] **Vojtěch Kovařík**[1] **Marc Lanctot**[2] **Branislav Bošanský**[1]

[1]Agent Technology Center
Dept. of Computer Science and Engineering
FEE, Czech Technical University in Prague
<name>.<surname>
@agents.fel.cvut.cz

[2]Department of Knowledge Engineering
Maastricht University, The Netherlands
marc.lanctot
@maastrichtuniversity.nl

Abstract

We study Monte Carlo tree search (MCTS) in zero-sum extensive-form games with perfect information and simultaneous moves. We present a general template of MCTS algorithms for these games, which can be instantiated by various selection methods. We formally prove that if a selection method is ϵ-Hannan consistent in a matrix game and satisfies additional requirements on exploration, then the MCTS algorithm eventually converges to an approximate Nash equilibrium (NE) of the extensive-form game. We empirically evaluate this claim using regret matching and Exp3 as the selection methods on randomly generated games and empirically selected worst case games. We confirm the formal result and show that additional MCTS variants also converge to approximate NE on the evaluated games.

1 Introduction

Non-cooperative game theory is a formal mathematical framework for describing behavior of interacting self-interested agents. Recent interest has brought significant advancements from the algorithmic perspective and new algorithms have led to many successful applications of game-theoretic models in security domains [1] and to near-optimal play of very large games [2]. We focus on an important class of two-player, zero-sum extensive-form games (EFGs) with perfect information and simultaneous moves. Games in this class capture sequential interactions that can be visualized as a game tree. The nodes correspond to the states of the game, in which both players act simultaneously. We can represent these situations using the normal form (*i.e.,* as matrix games), where the values are computed from the successor sub-games. Many well-known games are instances of this class, including card games such as Goofspiel [3, 4], variants of pursuit-evasion games [5], and several games from general game-playing competition [6].

Simultaneous-move games can be solved exactly in polynomial time using the backward induction algorithm [7, 4], recently improved with alpha-beta pruning [8, 9]. However, the depth-limited search algorithms based on the backward induction require domain knowledge (an evaluation function) and computing the cutoff conditions requires linear programming [8] or using a double-oracle method [9], both of which are computationally expensive. For practical applications and in situations with limited domain knowledge, variants of simulation-based algorithms such as Monte Carlo Tree Search (MCTS) are typically used in practice [10, 11, 12, 13]. In spite of the success of MCTS and namely its variant UCT [14] in practice, there is a lack of theory analyzing MCTS outside two-player perfect-information sequential games. To the best of our knowledge, no convergence guarantees are known for MCTS in games with simultaneous moves or general EFGs.

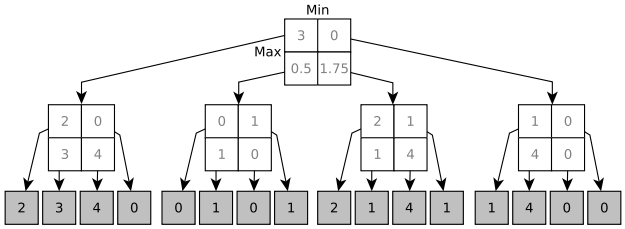


Figure 1: A game tree of a game with perfect information and simultaneous moves. Only the leaves contain the actual rewards; the remaining numbers are the expected reward for the optimal strategy.

In this paper, we present a general template of MCTS algorithms for zero-sum perfect-information simultaneous move games. It can be instantiated using any regret minimizing procedure for matrix games as a function for selecting the next actions to be sampled. We formally prove that if the algorithm uses an ϵ-Hannan consistent selection function, which assures attempting each action infinitely many times, the MCTS algorithm eventually converges to a subgame perfect ϵ-Nash equilibrium of the extensive form game. We empirically evaluate this claim using two different ϵ-Hannan consistent procedures: regret matching [15] and Exp3 [16]. In the experiments on randomly generated and worst case games, we show that the empirical speed of convergence of the algorithms based on our template is comparable to recently proposed MCTS algorithms for these games. We conjecture that many of these algorithms also converge to ϵ-Nash equilibrium and that our formal analysis could be extended to include them.

2 Definitions and background

A finite zero-sum game with perfect information and simultaneous moves can be described by a tuple $(\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, \mathcal{T}, u_1, h_0)$, where $\mathcal{N} = \{1, 2\}$ contains player labels, $\mathcal{H}$ is a set of inner states and $\mathcal{Z}$ denotes the terminal states. $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ is the set of joint actions of individual players and we denote $\mathcal{A}_1(h) = \{1 \dots m^h\}$ and $\mathcal{A}_2(h) = \{1 \dots n^h\}$ the actions available to individual players in state $h \in \mathcal{H}$. The transition function $\mathcal{T} : \mathcal{H} \times \mathcal{A}_1 \times \mathcal{A}_2 \mapsto \mathcal{H} \cup \mathcal{Z}$ defines the successor state given a current state and actions for both players. For brevity, we sometimes denote $\mathcal{T}(h, i, j) \equiv h_{ij}$. The utility function $u_1 : \mathcal{Z} \mapsto [v_{\min}, v_{\max}] \subseteq \mathbb{R}$ gives the utility of player 1, with v_{min} and $v_{\max}$ denoting the minimum and maximum possible utility respectively. Without loss of generality we assume $v_{\min} = 0$, $v_{\max} = 1$, and $\forall z \in \mathcal{Z}, u_2(z) = 1 - u_1(z)$. The game starts in an initial state h_0.

A *matrix game* is a single-stage simultaneous move game with action sets $\mathcal{A}_1$ and $\mathcal{A}_2$. Each entry in the matrix $M = (a_{ij})$ where $(i, j) \in \mathcal{A}_1 \times \mathcal{A}_2$ and $a_{ij} \in [0, 1]$ corresponds to a payoff (to player 1) if row i is chosen by player 1 and column j by player 2. A strategy $\sigma_q \in \Delta(\mathcal{A}_q)$ is a distribution over the actions in $\mathcal{A}_q$. If σ_1 is represented as a row vector and σ_2 as a column vector, then the expected value to player 1 when both players play with these strategies is $u_1(\sigma_1, \sigma_2) = \sigma_1 M \sigma_2$. Given a profile $\sigma = (\sigma_1, \sigma_2)$, define the utilities against best response strategies to be $u_1(br, \sigma_2) = \max_{\sigma_1' \in \Delta(\mathcal{A}_1)} \sigma_1' M \sigma_2$ and $u_1(\sigma_1, br) = \min_{\sigma_2' \in \Delta(\mathcal{A}_2)} \sigma_1 M \sigma_2'$. A strategy profile (σ_1, σ_2) is an ϵ-Nash equilibrium of the matrix game M if and only if

$$u_1(br, \sigma_2) - u_1(\sigma_1, \sigma_2) \leq \epsilon \qquad \text{and} \qquad u_1(\sigma_1, \sigma_2) - u_1(\sigma_1, br) \leq \epsilon \tag{1}$$

Two-player perfect information games with simultaneous moves are sometimes appropriately called *stacked matrix games* because at every state h each joint action from set $\mathcal{A}_1(h) \times \mathcal{A}_2(h)$ either leads to a terminal state or to a subgame which is itself another stacked matrix game (see Figure 1).

A *behavioral strategy* for player q is a mapping from states $h \in \mathcal{H}$ to a probability distribution over the actions $\mathcal{A}_q(h)$, denoted $\sigma_q(h)$. Given a profile $\sigma = (\sigma_1, \sigma_2)$, define the probability of reaching a terminal state z under σ as $\pi^\sigma(z) = \pi_1(z)\pi_2(z)$, where each $\pi_q(z)$ is a product of probabilities of the actions taken by player q along the path to z. Define Σ_q to be the set of behavioral strategies for player q. Then for any strategy profile $\sigma = (\sigma_1, \sigma_2) \in \Sigma_1 \times \Sigma_2$ we define the expected utility of the strategy profile (for player 1) as

$$u(\sigma) = u(\sigma_1, \sigma_2) = \sum_{z \in Z} \pi^\sigma(z) u_1(z) \tag{2}$$

An ϵ-Nash equilibrium profile (σ_1, σ_2) in this case is defined analogously to (1). In other words, none of the players can improve their utility by more than ϵ by deviating unilaterally. If the strategies are an ϵ-NE in each subgame starting in an arbitrary game state, the equilibrium strategy is termed subgame perfect. If $\sigma = (\sigma_1, \sigma_2)$ is an exact Nash equilibrium (*i.e.,* ϵ-NE with $\epsilon = 0$), then we denote the unique value of the game $v^{h_0} = u(\sigma_1, \sigma_2)$. For any $h \in \mathcal{H}$, we denote v^h the value of the subgame rooted in state h.

3 Simultaneous move Monte-Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a simulation-based state space search algorithm often used in game trees. The nodes in the tree represent game states. The main idea is to iteratively run simulations to a terminal state, incrementally growing a tree rooted at the initial state of the game. In its simplest form, the tree is initially empty and a single leaf is added each iteration. Each simulation starts by visiting nodes in the tree, selecting which actions to take based on a selection function and information maintained in the node. Consequently, it transitions to the successor states. 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.

In Simultaneous Move MCTS (SM-MCTS), the main difference is that a joint action of both players is selected. The algorithm has been previously applied, for example in the game of Tron [12], Urban Rivals [11], and in general game-playing [10]. However, guarantees of convergence to NE remain unknown. The convergence to a NE depends critically on the selection and update policies applied, which are even more non-trivial than in purely sequential games. The most popular selection policy in this context (UCB) performs very well in some games [12], but Shafiei et al. [17] show that it does not converge to Nash equilibrium, even in a simple one-stage simultaneous move game. In this paper, we focus on variants of MCTS, which provably converge to (approximate) NE; hence we do not discuss UCB any further. Instead, we describe variants of two other selection algorithms after explaining the abstract SM-MCTS algorithm.

Algorithm 1 describes a single simulation of SM-MCTS. T represents the MCTS tree in which each state is represented by one node. Every node h maintains a cumulative reward sum over all simulations through it, X_h, and a visit count n_h, both initially set to 0. As depicted in Figure 1, a matrix of references to the children is maintained at each inner node. The critical parts of the algorithm are the updates on lines 8 and 14 and the selection on line 10. Each variant below will describe a different way to select an action and update a node. The standard way of defining the value to send back is RetVal$(u_1, X_h, n_h) = u_1$, but we discuss also RetVal$(u_1, X_h, n_h) = X_h/n_h$, which is required for the formal analysis in Section 4. We denote this variant of the algorithms

```
1  SM-MCTS(node h)
2    if h ∈ Z then return u_1(h)
3    else if h ∈ T and ∃(i,j) ∈ A_1(h) × A_2(h) not previously selected then
4        Choose one of the previously unselected (i,j) and h' ← T(h,i,j)
5        Add h' to T
6        u_1 ← Rollout(h')
7        X_h' ← X_h' + u_1;  n_h' ← n_h' + 1
8        Update(h,i,j,u_1)
9        return RetVal(u_1, X_h', n_h')
10   (i,j) ← Select(h)
11   h' ← T(h,i,j)
12   u_1 ← SM-MCTS(h')
13   X_h ← X_h + u_1;  n_h ← n_h + 1
14   Update(h,i,j,u_1)
15   return RetVal(u_1, X_h, n_h)
```

Algorithm 1: Simultaneous Move Monte Carlo Tree Search

with additional "M" for mean. Algorithm 1 and the variants below are expressed from player 1's perspective. Player 2 does the same except using negated utilities.

3.1 Regret matching

This variant applies regret-matching [15] to the current estimated matrix game at each stage. Suppose iterations are numbered from $s \in \{1, 2, 3, \cdots\}$ and at each iteration and each inner node h there is a mixed strategy $\sigma^s(h)$ used by each player, initially set to uniform random: $\sigma^0(h, i) = 1/|\mathcal{A}(h)|$. Each player maintains a cumulative regret $r_h[i]$ for having played $\sigma^s(h)$ instead of $i \in \mathcal{A}_1(h)$. The values are initially set to 0.

On iteration s, the Select function (line 10 in Algorithm 1) first builds the player's current strategies from the cumulative regret. Define $x^+ = \max(x, 0)$,

$$\sigma^s(h, a) = \frac{r_h^+[a]}{R_{sum}^+} \text{ if } R_{sum}^+ > 0 \text{ oth. } \frac{1}{|\mathcal{A}_1(h)|}, \text{ where } R_{sum}^+ = \sum_{i \in \mathcal{A}_1(h)} r_h^+[i]. \tag{3}$$

The strategy is computed by assigning higher weight proportionally to actions based on the regret of having not taken them over the long-term. To ensure exploration, an γ-on-policy sampling procedure is used choosing action i with probability $\gamma/|\mathcal{A}(h)| + (1 - \gamma)\sigma^s(h, i)$, for some $\gamma > 0$.

The Updates on lines 8 and 14 add regret accumulated at the iteration to the regret tables r_h. Suppose joint action (i_1, j_2) is sampled from the selection policy and utility u_1 is returned from the recursive call on line 12. Define $x(h, i, j) = X_{h_{ij}}$ if $(i, j) \neq (i_1, j_2)$, or u_1 otherwise. The updates to the regret are:

$$\forall i' \in \mathcal{A}_1(h), r_h[i'] \leftarrow r_h[i'] + (x(h, i', j) - u_1).$$

3.2 Exp3

In Exp3 [16], a player maintains an estimate of the sum of rewards, denoted $x_{h,i}$, and visit counts $n_{h,i}$ for each of their actions $i \in \mathcal{A}_1$. The joint action selected on line 10 is composed of an action independently selected for each player. The probability of sampling action a in Select is

$$\sigma^s(h, a) = \frac{(1 - \gamma)\exp(\eta w_{h,a})}{\sum_{i \in \mathcal{A}_1(h)} \exp(\eta w_{h,i})} + \frac{\gamma}{|\mathcal{A}_1(h)|}, \text{ where } \eta = \frac{\gamma}{|\mathcal{A}_1(h)|} \text{ and } w_{h,i} = x_{h,i}{}^{1}. \tag{4}$$

The Update after selecting actions (i, j) and obtaining a result (u_1, u_2) updates the visits count $(n_{h,i} \leftarrow n_{h,i} + 1)$ and adds to the corresponding reward sum estimates the reward divided by the probability that the action was played by the player ($x_{h,i} \leftarrow x_{h,i} + u_1/\sigma^s(h, i)$). Dividing the value by the probability of selecting the corresponding action makes $x_{h,i}$ estimate the sum of rewards over all iterations, not only the once where action i was selected.

4 Formal analysis

We focus on the eventual convergence to approximate NE, which allows us to make an important simplification: We disregard the incremental building of the tree and assume we have built the complete tree. We show that this will eventually happen with probability 1 and that the statistics collected during the tree building phase cannot prevent the eventual convergence.

The main idea of the proof is to show that the algorithm will eventually converge close to the optimal strategy in the leaf nodes and inductively prove that it will converge also in higher levels of the tree. In order to do that, after introducing the necessary notation, we start by analyzing the situation in simple matrix games, which corresponds mainly to the leaf nodes of the tree. In the inner nodes of the tree, the observed payoffs are imprecise because of the stochastic nature of the selection functions and bias caused by exploration, but the error can be bounded. Hence, we continue with analysis of repeated matrix games with bounded error. Finally, we compose the matrices with bounded errors in

[1] In practice, we set $w_{h,i} = x_{h,i} - \max_{i' \in \mathcal{A}_1(h)} x_{h,i'}$ since $\exp(x_{h,i})$ can easily cause numerical overflows. This reformulation computes the same values as the original algorithm but is more numerically stable.

a multi-stage setting to prove convergence guarantees of SM-MCTS. Any proofs that are omitted in the paper are included in the appendix available in the supplementary material and on http://arxiv.org (arXiv:1310.8613).

4.1 Notation and definitions

Consider a repeatedly played matrix game where at time s players 1 and 2 choose actions i_s and j_s respectively. We will use the convention $(|\mathcal{A}_1|, |\mathcal{A}_2|) = (m, n)$. Define

$$G(t) = \sum_{s=1}^{t} a_{i_s j_s}, \quad g(t) = \frac{1}{t} G(t), \quad \text{and} \quad G_{max}(t) = \max_{i \in \mathcal{A}_1} \sum_{s=1}^{t} a_{i j_s},$$

where $G(t)$ is the *cumulative payoff*, $g(t)$ is the *average payoff*, and G_{max} is the *maximum cumulative payoff* over all actions, each to player 1 and at time t. We also denote $g_{max}(t) = G_{max}(t)/t$ and by $R(t) = G_{max}(t) - G(t)$ and $r(t) = g_{max}(t) - g(t)$ the *cumulative and average regrets*. For actions i of player 1 and j of player 2, we denote t_i, t_j the number of times these actions were chosen up to the time t and t_{ij} the number of times both of these actions has been chosen at once. By *empirical frequencies* we mean the strategy profile $(\hat{\sigma}_1(t), \hat{\sigma}_2(t)) \in \langle 0,1\rangle^m \times \langle 0,1\rangle^n$ given by the formulas $\hat{\sigma}_1(t,i) = t_i/t$, $\hat{\sigma}_2(t,j) = t_j/t$. By *average strategies*, we mean the strategy profile $(\bar{\sigma}_1(t), \bar{\sigma}_2(t))$ given by the formulas $\bar{\sigma}_1(t,i) = \sum_{s=1}^{t} \sigma_1^s(i)/t$, $\bar{\sigma}_2(t,j) = \sum_{s=1}^{t} \sigma_2^s(j)/t$, where σ_1^s, σ_2^s are the strategies used at time s.

Definition 4.1. We say that a player is *ϵ-Hannan-consistent* if, for any payoff sequences (e.g., against any opponent strategy), $\limsup_{t\to\infty}, r(t) \leq \epsilon$ holds almost surely. An algorithm A is ϵ-Hannan consistent, if a player who chooses his actions based on A is ϵ-Hannan consistent.

Hannan consistency (HC) is a commonly studied property in the context of online learning in repeated (single stage) decisions. In particular, RM and variants of Exp3 has been shown to be Hannan consistent in matrix games [15, 16]. In order to ensure that the MCTS algorithm will eventually visit each node infinitely many times, we need the selection function to satisfy the following property.

Definition 4.2. We say that A is an *algorithm with guaranteed exploration*, if for players 1 and 2 both using A for action selection $\lim_{t\to\infty} t_{ij} = \infty$ holds almost surely $\forall (i,j) \in \mathcal{A}_1 \times \mathcal{A}_2$.

Note that most of the HC algorithms, namely RM and Exp3, guarantee exploration without any modification. If there is an algorithm without this property, it can be adjusted the following way.

Definition 4.3. Let A be an algorithm used for choosing action in a matrix game M. For fixed exploration parameter $\gamma \in (0,1)$ we define a modified algorithm A^* as follows: In each time, with probability $(1-\gamma)$ run one iteration of A and with probability γ choose the action randomly uniformly over available actions, without updating any of the variables belonging to A.

4.2 Repeated matrix games

First we show that the ϵ-Hannan consistency is not lost due to the additional exploration.

Lemma 4.4. *Let A be an ϵ-Hannan consistent algorithm. Then A^* is an $(\epsilon+\gamma)$-Hannan consistent algorithm with guaranteed exploration.*

In previous works on MCTS in our class of games, RM variants generally suggested using the average strategy and Exp3 variants the empirical frequencies to obtain the strategy to be played. The following lemma says there eventually is no difference between the two.

Lemma 4.5. *As t approaches infinity, the empirical frequencies and average strategies will almost surely be equal. That is,* $\limsup_{t\to\infty} \max_{i\in\mathcal{A}_1} |\hat{\sigma}_1(t,i) - \bar{\sigma}_1(t,i)| = 0$ *holds with probability* 1.

The proof is a consequence of the martingale version of Strong Law of Large Numbers.

It is well known that two Hannan consistent players will eventually converge to NE (see [18, p. 11] and [19]). We prove a similar result for the approximate versions of the notions.

Lemma 4.6. *Let $\epsilon > 0$ be a real number. If both players in a matrix game with value v are ϵ-Hannan consistent, then the following inequalities hold for the empirical frequencies almost surely:*

$$\limsup_{t\to\infty} u\left(br, \hat{\sigma}_2(t)\right) \leq v + 2\epsilon \quad \textit{and} \quad \liminf_{t\to\infty} u\left(\hat{\sigma}_1(t), br\right) \geq v - 2\epsilon. \tag{5}$$

The proof shows that if the value caused by the empirical frequencies was outside of the interval infinitely many times with positive probability, it would be in contradiction with definition of ϵ-HC. The following corollary is than a direct consequence of this lemma.

Corollary 4.7. *If both players in a matrix game are ϵ-Hannan consistent, then there almost surely exists $t_0 \in \mathbb{N}$, such that for every $t \geq t_0$ the empirical frequencies and average strategies form $(4\epsilon + \delta)$-equilibrium for arbitrarly small $\delta > 0$.*

The constant 4 is caused by going from a pair of strategies with best responses within 2ϵ of the game value guaranteed by Lemma 4.6 to the approximate NE, which multiplies the distance by two.

4.3 Repeated matrix games with bounded error

After defining the repeated games with error, we present a variant of Lemma 4.6 for these games.

Definition 4.8. We define $M(t) = (a_{ij}(t))$ to be a game, in which if players chose actions i and j, they receive randomized payoffs $a_{ij}\left(t, (i_1, ...i_{t-1}), (j_1, ...j_{t-1})\right)$. We will denote these simply as $a_{ij}(t)$, but in fact they are random variables with values in $[0, 1]$ and their distribution in time t depends on the previous choices of actions. We say that $M(t) = (a_{ij}(t))$ is a *repeated game with error η*, if there is a matrix game $M = (a_{ij})$ and almost surely exists $t_0 \in \mathbb{N}$, such that $|a_{ij}(t) - a_{ij}| < \eta$ holds for all $t \geq t_0$.

In this context, we will denote $G(t) = \sum_{s \in \{1...t\}} a_{i_s j_s}(s)$ etc. and use tilde for the corresponding variables without errors ($\tilde{G}(t) = \sum a_{i_s j_s}$ etc.). Symbols v and $u\left(\cdot, \cdot\right)$ will still be used with respect to M without errors. The following lemma states that even with the errors, ϵ-HC algorithms still converge to an approximate NE of the game.

Lemma 4.9. *Let $\epsilon > 0$ and $c \geq 0$. If $M(t)$ is a repeated game with error $c\epsilon$ and both players are ϵ-Hannan consistent then the following inequalities hold almost surely:*

$$\limsup_{t \to \infty} u\left(br, \hat{\sigma}_2\right) \leq v + 2(c+1)\epsilon, \quad \liminf_{t \to \infty} u\left(\hat{\sigma}_1, br\right) \geq v - 2(c+1)\epsilon \tag{6}$$

$$\text{and} \quad v - (c+1)\epsilon \leq \liminf_{t \to \infty} g(t) \leq \limsup_{t \to \infty} g(t) \leq v + (c+1)\epsilon. \tag{7}$$

The proof is similar to the proof of Lemma 4.6. It needs an additional claim that if the algorithm is ϵ-HC with respect to the observed values with errors, it still has a bounded regret with respect to the exact values. In the same way as in the previous subsection, a direct consequence of the lemma is the convergence to an approximate Nash equilibrium.

Theorem 4.10. *Let $\epsilon, c > 0$ be real numbers. If $M(t)$ is a repeated game with error $c\epsilon$ and both players are ϵ-Hannan consistent, then for any $\delta > 0$ there almost surely exists $t_0 \in \mathbb{N}$, such that for all $t \geq t_0$ the empirical frequencies form $(4(c+1)\epsilon + \delta)$-equilibrium of the game M.*

4.4 Perfect-information extensive-form games with simultaneous moves

Now we have all the necessary components to prove the main theorem.

Theorem 4.11. *Let $\left(M^h\right)_{h \in H}$ be a game with perfect information and simultaneous moves with maximal depth D. Then for every ϵ-Hannan consistent algorithm A with guaranteed exploration and arbitrary small $\delta > 0$, there almost surely exists t_0, so that the average strategies $(\hat{\sigma}_1(t), \hat{\sigma}_2(t))$ form a subgame perfect*

$$\left(2D^2 + \delta\right) \epsilon\text{-Nash equilibrium for all } t \geq t_0.$$

Once we have established the convergence of the ϵ-HC algorithms in games with errors, we can proceed by induction. The games in the leaf nodes are simple matrix game so they will eventually converge and they will return the mean reward values in a bounded distance from the actual value of the game (Lemma 4.9 with $c = 0$). As a result, in the level just above the leaf nodes, the ϵ-HC algorithms are playing a matrix game with a bounded error and by Lemma 4.9, they will also eventually return the mean values within a bounded interval. On level d from the leaf nodes, the errors of returned values will be in the order of $d\epsilon$ and players can gain $2d\epsilon$ by deviating. Summing the possible gain of deviations on each level leads to the bound in the theorem. The subgame perfection of the equilibrium results from the fact that for proving the bound on approximation in the whole game (i.e., in the root of the game tree), a smaller bound on approximation of the equilibrium is proven for all subgames in the induction. The formal proof is presented in the appendix.

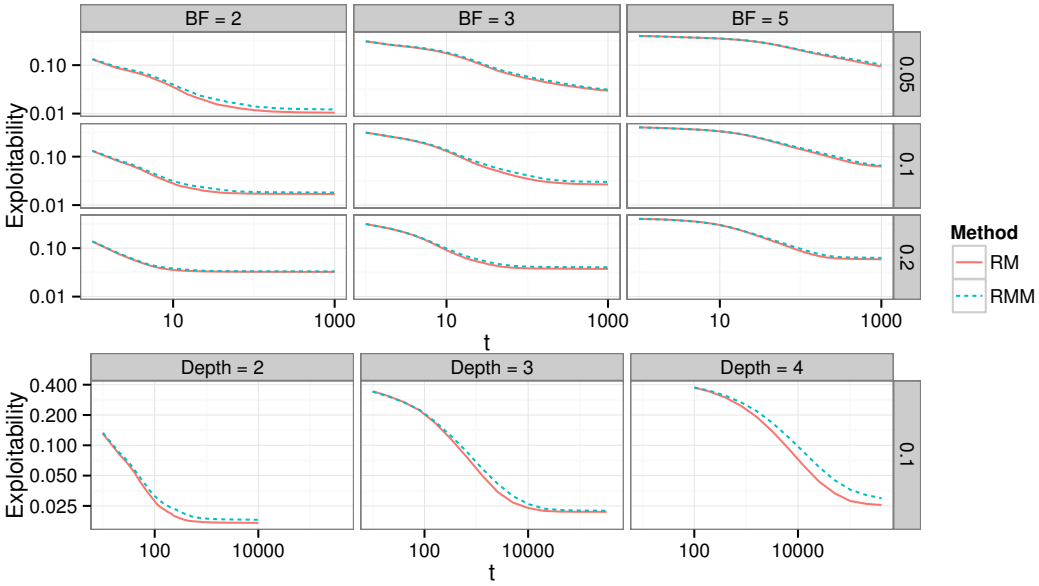


Figure 2: Exploitability of strategies given by the empirical frequencies of Regret matching with propagating values (RM) and means (RMM) for various depths and branching factors.

5 Empirical analysis

In this section, we first evaluate the influence of propagating the mean values instead of the current sample value in MCTS to the speed of convergence to Nash equilibrium. Afterwards, we try to assess the convergence rate of the algorithms in the worst case. In most of the experiments, we use as the bases of the SM-MCTS algorithm Regret matching as the selection strategy, because a superior convergence rate bound is known for this algorithm and it has been reported to be very successful also empirically in [20]. We always use the empirical frequencies to create the evaluated strategy and measure the exploitability of the first player's strategy (i.e., $v^{h_0} - u(\hat{\sigma}_1, br)$).

5.1 Influence of propagation of the mean

The formal analysis presented in the previous section requires the algorithms to return the mean of all the previous samples instead of the value of the current sample. The latter is generally the case in previous works on SM-MCTS [20, 11]. We run both variants with the Regret matching algorithm on a set of randomly generated games parameterized by depth and branching factor. Branching factor was always the same for both players. For the following experiments, the utility values are randomly selected uniformly from interval $\langle 0, 1 \rangle$. Each experiment uses 100 random games and 100 runs of the algorithm.

Figure 2 presents how the exploitability of the strategies produced by Regret matching with propagation of the mean (RMM) and current sample value (RM) develops with increasing number of iterations. Note that both axes are in logarithmic scale. The top graph is for depth of 2, different branching factors (BF) and $\gamma \in \{0.05, 0.1, 0.2\}$. The bottom one presents different depths for $BF = 2$. The results show that both methods converge to the approximate Nash equilibrium of the game. RMM converges slightly slower in all cases. The difference is very small in small games, but becomes more apparent in games with larger depth.

5.2 Empirical convergence rate

Although the formal analysis guarantees the convergence to an ϵ-NE of the game, the rate of the convergence is not given. Therefore, we give an empirical analysis of the convergence and specifically focus on the cases that reached the slowest convergence from a set of evaluated games.

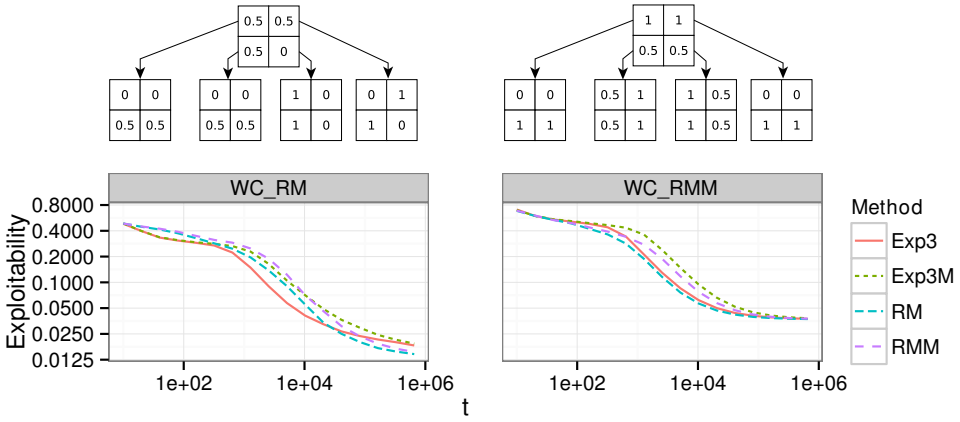


Figure 3: The games with maximal exploitability after 1000 iterations with RM (left) and RMM (right) and the corresponding exploitabililty for all evaluated methods.

We have performed a brute force search through all games of depth 2 with branching factor 2 and utilities form the set $\{0, 0.5, 1\}$. We made 100 runs of RM and RMM with exploration set to $\gamma = 0.05$ for 1000 iterations and computed the mean exploitability of the strategy. The games with the highest exploitability for each method are presented in Figure 3. These games are not guaranteed to be the exact worst case, because of possible error caused by only 100 runs of the algorithm, but they are representatives of particularly difficult cases for the algorithms. In general, the games that are most difficult for one method are difficult also for the other. Note that we systematically searched also for games in which RMM performs better than RM, but this was never the case with sufficient number of runs of the algorithms in the selected games.

Figure 3 shows the convergence of RM and Exp3 with propagating the current sample values and the mean values (RMM and Exp3M) on the empirically worst games for the RM variants. The RM variants converge to the minimal achievable values (0.0119 and 0.0367) after a million iterations. This values corresponds exactly to the exploitability of the optimal strategy combined with the uniform exploration with probability 0.05. The Exp3 variants most likely converge to the same values, however, they did not fully make it in the first million iterations in WC_RM. The convergence rate of all the variants is similar and the variants with propagating means always converge a little slower.

6 Conclusion

We present the first formal analysis of convergence of MCTS algorithms in zero-sum extensive-form games with perfect information and simultaneous moves. We show that any ϵ-Hannan consistent algorithm can be used to create a MCTS algorithm that provably converges to an approximate Nash equilibrium of the game. This justifies the usage of the MCTS as an approximation algorithm for this class of games from the perspective of algorithmic game theory. We complement the formal analysis with experimental evaluation that shows that other MCTS variants for this class of games, which are not covered by the proof, also converge to the approximate NE of the game. Hence, we believe that the presented proofs can be generalized to include these cases as well. Besides this, we will focus our future research on providing finite time convergence bounds for these algorithms and generalizing the results to more general classes of extensive-form games with imperfect information.

Acknowledgments

This work is partially funded by the Czech Science Foundation (grant no. P202/12/2054), the Grant Agency of the Czech Technical University in Prague (grant no. OHK3-060/12), and the Netherlands Organisation for Scientific Research (NWO) in the framework of the project Go4Nature, grant number 612.000.938. 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 appreciated.

References

[1] Manish Jain, Dmytro Korzhyk, Ondrej Vanek, Vincent Conitzer, Michal Pechoucek, and Milind Tambe. A double oracle algorithm for zero-sum security games. In *Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pages 327–334, 2011.

[2] Michael Johanson, Nolan Bard, Neil Burch, and Michael Bowling. Finding optimal abstract strategies in extensive-form games. In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, pages 1371–1379, 2012.

[3] S. M. Ross. Goofspiel — the game of pure strategy. *Journal of Applied Probability*, 8(3):621–625, 1971.

[4] Glenn C. Rhoads and Laurent Bartholdi. Computer solution to the game of pure strategy. *Games*, 3(4):150–156, 2012.

[5] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *In Proceedings of the Eleventh International Conference on Machine Learning (ICML-1994)*, pages 157–163. Morgan Kaufmann, 1994.

[6] M. Genesereth and N. Love. General game-playing: Overview of the AAAI competition. *AI Magazine*, 26:62–72, 2005.

[7] Michael Buro. Solving the Oshi-Zumo game. In *Proceedings of Advances in Computer Games 10*, pages 361–366, 2003.

[8] Abdallah Saffidine, Hilmar Finnsson, and Michael Buro. Alpha-beta pruning for games with simultaneous moves. In *Proceedings of the Thirty-Second Conference on Artificial Intelligence (AAAI-12)*, pages 556–562, 2012.

[9] Branislav Bosansky, Viliam Lisy, Jiri Cermak, Roman Vitek, and Michal Pechoucek. Using double-oracle method and serialized alpha-beta search for pruning in simultaneous moves games. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, pages 48–54, 2013.

[10] H. Finnsson and Y. Björnsson. Simulation-based approach to general game-playing. In *The Twenty-Third AAAI Conference on Artificial Intelligence*, pages 259–264. AAAI Press, 2008.

[11] Olivier Teytaud and Sébastien Flory. Upper confidence trees with short term partial information. In *Applications of Eolutionary Computation (EvoApplications 2011), Part I*, volume 6624 of *LNCS*, pages 153–162, Berlin, Heidelberg, 2011. Springer-Verlag.

[12] Pierre Perick, David L. St-Pierre, Francis Maes, and Damien 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)*, pages 242–249, 2012.

[13] Hilmar Finnsson. *Simulation-Based General Game Playing*. PhD thesis, Reykjavik University, 2012.

[14] L. Kocsis and C. Szepesvári. Bandit-based Monte Carlo planning. In *15th European Conference on Machine Learning*, volume 4212 of *LNCS*, pages 282–293, 2006.

[15] S. Hart and A. Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

[16] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

[17] M. Shafiei, N. R. Sturtevant, and J. Schaeffer. Comparing UCT versus CFR in simultaneous games. In *Proceeding of the IJCAI Workshop on General Game-Playing (GIGA)*, pages 75–82, 2009.

[18] Kevin Waugh. Abstraction in large extensive games. Master's thesis, University of Alberta, 2009.

[19] A. Blum and Y. Mansour. Learning, regret minimization, and equilibria. In Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, chapter 4. Cambridge University Press, 2007.

[20] Marc Lanctot, Viliam Lisý, and Mark H.M. Winands. Monte Carlo tree search in simultaneous move games with applications to Goofspiel. In *Workshop on Computer Games at IJCAI*, 2013.

Game-theoretic Approach to Adversarial Plan Recognition

Viliam Lisý and **Radek Píbil** and **Jan Stiborek** and **Branislav Bošanský** and **Michal Pěchouček** [1]

Abstract. We argue that the problem of adversarial plan recognition, where the observed agent actively tries to avoid detection, should be modeled in the game theoretic framework. We define the problem as an imperfect-information extensive-form game between the observer and the observed agent. We propose a novel algorithm that approximates the optimal solution in the game using Monte-Carlo sampling. The experimental evaluation is performed on a synthetic domain inspired by a network security problem. The proposed method produces significantly better results than several simple baselines on a practically large domain.

1 Introduction

Intention and plan recognition is an important capability of an intelligent agent. In cooperative settings, it enables coordination with other agents in situations where explicit communication is not possible or desirable. The examples of such situations are unobtrusive intelligent assistance to people or coordination of robotic systems. In competitive or adversarial settings, plan recognition allows, e.g., detecting malicious behavior, or predicting the opponent's future actions. Most of the research on plan recognition assumes that the observed agent is either indifferent about the recognition process, or actively cooperates and provides hints that make the recognition process easier. In many (security) applications the situation is exactly opposite. An agent is required to recognize plans or intentions of its adversaries that are aware of the presence of the recognition process and actively try to avoid it while pursuing their own goals. The adversary may choose actions that are hard to detect, or use deception to mislead the observer. The problem of recognizing plans in such settings is termed *adversarial plan recognition*. It is present, for example, in warfare scenarios, different security domains, such as airport security, or computer networks. The latter is the main scenario we use for experimental validation of the proposed techniques.

The previous studies on adversarial plan recognition identified the problem to be different from other forms of plan recognition (i.e., keyhole and intended plan recognition). They list the differences from the domain perspective and identify the possibility of missing observations to be the most crucial one [7, 10]. However, they perform the recognition as if the observations were missing by accident. They do not model that the observed agent intentionally selects actions to avoid detection and how it influences the recognition process. In this paper, we focus on exactly this aspect of adversarial plan recognition under the assumption of rationality of both agents. Further, we complete the model by allowing the observer to perform actions that influence its chances to observe specific actions of the observed.

We define the problem of the adversarial plan recognition as a two-player zero-sum extensive-form game (EFG) between the observer and the observed agent (further called the *actor* for clarity). None of the players can directly observe the actions taken by the other player; however, we assume that the observer receives noisy probabilistic observations of the actions of the actor from the environment. The solution of this game (in the form of Nash equilibrium) provides the optimal strategy for both players. It allows the actor to find the best tradeoff between its plan cost and observability and it prescribes the observer how to choose its actions to maximize the chance of observing and identifying the plan.

In order to achieve better scalability of the approach, we show that thanks to the special structure of the game, the size of the game representation can be substantially reduced. Using the reduced representation, we introduce an algorithm for solving the adversarial plan recognition game. It approximates the optimal solution under real-world time constraints, based on the Monte-Carlo Tree Search (MCTS). Finally, we evaluate the approach experimentally, showing it performs significantly better than a set of baseline algorithms.

2 Related Work

The most crucial difference between the adversarial plan recognition and the other kinds of plan recognition is that the actor actively tries to prevent the recognition process. The previous work on adversarial plan recognition [7, 10] focused only on the effect of this intention – the missing observations, but did not model the intention itself and its effect to the adversary's plan selection. Reasoning of the actor was not part of their models. In this paper, we model the problem more completely, including the intention of both players under the assumption of their rationality.

The concept of rationality is not new in the field of plan recognition. It was first introduced by Mao and Gratch [9] in the problem of utility-based plan recognition. They assume that the actor is rational and tries to maximize its utility. If more plan hypotheses are consistent with the current observations, they propose to use the one with highest expected utility for the actor as the most likely one. This paper, however, uses the assumption of rationality only partially. They assume all plans are equally probable a priory, which is inconsistent with the assumption of rationality. In fact, only the plans that lead to the maximal expected utility can be selected by a truly rational agent.

The problem of utility-based plan recognition was studied also in [3]. In this paper, the authors assume rationality of the observer. It prefers to detect the less likely, but more dangerous behaviors. After computing the probabilities of the hypothesis, they are multiplied

[1] Agent Technology Center, Dept. of Computer Science and Engeneering, FEE, Czech Technical University, Czech Republic, email: {lisy, pibil, stiborek, bosansky, pechoucek}@agents.felk.cvut.cz

by their utility (harm to the observer) and the one with the highest expected utility is the result of the recognition process.

In case of adversarial plan recognition between rational agents, there is often only a small difference between considering the utility of the observer and the utility of the actor. The plans that are more important to be detected for the observer are those that the actor wants to keep hidden. Hence, we model the problem as a zero-sum game. A similar approach was taken by Braynov in [4]. He presents a conceptual framework of planning and plan recognition as a deterministic full-information simultaneous-moves game and argues that rational players would play the Nash equilibrium in the game. The goal of the actor is to traverse an attack graph from a source to one of few targets and the observer can remove one of the edges in the attack graph per move. The approach relays on a plan recognition algorithm as a plugin and the paper does not provide any experimental validation.

Example The following example demonstrates the main difference between the existing and the proposed plan recognition approaches. Assume the actor can execute plans A, B, C with expected utilities $9, 9, 7$ for the observed agent. The classical (even adversarial) plan recognition does not take the utilities into account and assumes uniform a priori probability of all plans. If the observations indicate the probabilities of the plans are $0.2, 0.3, 0.5$, the classical plan recognition [7] answers plan C was most likely executed. The utility-based plan recognition [9] multiplies the expected utilities and the posterior probabilities of the plans to get $1.8, 2.7, 3.5$. It answers plan C was most likely executed by a "rational" agent, because it gives the highest product. In this paper, we use the assumption of rationality strictly. We identify that the rational player can never play C, as this choice is suboptimal. The remaining two plans have the same prior probability. Hence we choose B due to observations.

Furthermore, our model defines different utility values for executing the plan based on its successful observation. It allows defining actions of the observer influencing chances of detecting individual actions in the actor's plan and automatic reasoning about the optimal plan and its recognition.

3 Adversarial Plan Recognition Game

The *adversarial plan recognition game* (APRG) is an imperfect-information zero-sum EFG between the actor and the observer.

3.1 Game Definition

APRG is defined as a tuple (A, P, g, D, m), where

- A is a set of actions available to the actor. It can contain the action None, corresponding to actor not performing any action.
- P is the set plans (sequences of actions) of the actor. We assume the actions have preconditions and effects; therefore, not all sequences are legal plans. We often assume the plans are organized as a tree by identical prefixes.
- $g : P \to \mathbb{R}$ is the actor's payoff for the actor in case the plan is performed unobserved.
- D is a set of actions available to the observer. Generally, it corresponds to using a specific sensor / behavior classifier, each with different chances to detect the actor's actions correctly.
- $m : D \to \mathbb{R}^{|A|\times|A|}$ is a collection of one confusion matrix for each observer's action $d \in D$. Such matrix expresses the probability that the observer observes action $o \in A$ in case the actor executes $a \in A$ (i.e., for each $d \in D$ we have $p(o|a, d)$).[2]

[2] Note that the set of observations can be different form the set of actions. However, it would require a more complex definition of the utility function.

The game is played in discrete time steps. In each time step, the players choose actions simultaneously. After the actor has performed $(a_1 \dots a_n) \in A^n$, it chooses an action $a \in A$, such that $(a_1 \dots a_n, a)$ is a prefix of some sequence in P. The observer simultaneously selects an action $d \in D$. Based on these two actions, the nature selects stochastically the observed action according to $m(d)$.

Utility The game ends when the actor completes any sequence from P. The utility function of the game captures the aims of the players. The utility value for the actor depends on the payoff g for the executed plan and it is discounted by the number of times an action from the executed plan has been correctly observed:

$$u(a_1 \dots a_h, o_1 \dots o_h) = \frac{g(a_1 \dots a_h)}{1 + \sum_{i \in \{1 \dots h\}; o_i = a_i} 1} \qquad (1)$$

We assume the game to be zero-sum, which is a natural assumption in the adversarial plan recognition. The utility value is maximized by the actor and minimized by the observer. The method we propose for solving this game in Section 4 does not depend on the definition of the utility function so it can easily be adjusted for needs of a specific domain. Even if the zero-sum assumption does not hold, the model can still be used to compute solutions with a guaranteed (possibly sub-optimal) quality against any adversary.

Information We assume that both players know all parts of the definition of the game. The information about the progress of the game is much more limited. Neither of the players can directly observe the actions of the other player. The observer knows its own actions (selection of the classifiers) and the observations generated by nature based on the actions. The actor can observe only its own actions.

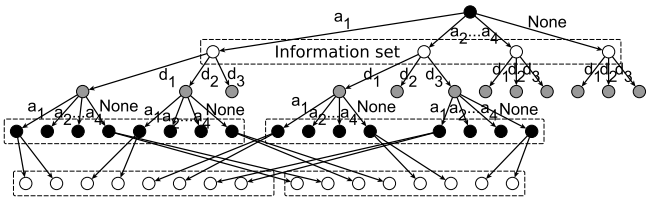


Figure 1: Portion of the extensive-form representation of the game. The leafs in the picture are not actual leafs in the game, the game continues for large (possibly infinite) number of steps.

APRG as an Extensive Form Game The restrictions on the available information create a specific structure of information sets[3]. Part of the complete extensive-form tree of the game is presented in Figure 1. The tree of the game can be arbitrarily deep. The longest branch is three times longer than the length of the longest plan in P. Hence, the "leaves" in the figure are just root nodes of subtrees omitted for clarity. Three plies of nodes are periodically repeating in the tree. The actor's decision nodes (black), the observer's decision nodes (white) and the observation nodes (grey). The actor's decision nodes must respect the plan tree P; hence, the available actions in one stage of the game are based on the actions performed by the actor in the previous stages. The observer's decision nodes contain always the same set of actions corresponding to all available classifiers (D). The observation nodes are chance nodes with probability distribution based on the classifier quality matrices (m) and the preceding actions of the actor and the observer. The structure of the information

[3] Information set is a set of game states that cannot be distinguished from each other by a player.

sets is consistent with the assumptions stated above as well as with the intuition of simultaneous moves of the players. We refer to the three plies of different players starting with the actor's decision as a (simultaneous) *move*.

3.2 Two Solutions of the Game

The proposed APRG is both a game and a plan recognition problem. As a result, it has two solutions. From the game perspective, the solutions are the action selection strategies of the players that optimize the tradeoffs of plan value and detectability. From the plan recognition perspective, it is the plan of the actor that is most probable given the observations. Both solutions are based on the assumption of rationality of the players.

Game-theory perspective For the game-theoretic solution, we use the well-known concept of Nash equilibrium (NE). It is a pair of strategies, such that none of the players has an incentive to deviate from its strategy if the other player's strategy stays unchanged, i.e., the observer cannot improve the chance to detect the plans the actor is likely to play and the actor cannot pick a plan that would have a better utility after the discount for detection. Further, we allow the agents to use *mixed strategies*, i.e., they can choose their actions based on fixed probability distributions in each information set of the game. This is necessary to achieve the optimal behavior in this game. For example, if the actor has two similarly good actions that can be detected by different actions of the observer, it should randomize among these options to make detection more difficult.

The most fundamental refinement of Nash equilibrium for imperfect-information extensive-form games is the *Sequential Equilibrium*, an imperfect-information-equivalent of the better known subgame-perfect Nash equilibrium used in perfect-information extensive-form games [11]. For this paper, it is sufficient to know that the strategies in this equilibrium can be represented as *behavioral strategies*, i.e., the probabilities of selecting each action in each information set.

Plan recognition perspective Even before the game starts, many of the plans in set P are clearly not going to be selected by a rational actor. Only the plans that have non-zero probability in some Nash equilibrium of the game can be selected. If the game has a single Nash equilibrium that does not require randomization, it can be recognized as the plan of the actor without any observations. On the other hand, more complex games require randomization. In that case, the task of the observer is to identify what are the random choices of the actor in the equilibrium strategy it currently plays. The result of the plan recognition process is the posterior probability distribution on the set P. This can be used to identify the plan that has been most probably executed by the actor or determine the probability of a specific plan chosen in advance.

4 Solution method

The game defined above is a standard extensive-form game with imperfect information and chance nodes and, in principle, it can be optimally solved by a standard algorithm for solving the games, such as, the linear program for sequence-form representation of the game[11]. The size of the linear program is polynomial in the size of the game tree. However, the size of the game tree is exponential in the number of actions of the players and the length of the game. For example, if each player has 5 applicable choices in each decision node and the game is played for 10 moves (corresponds to 30 plies) the size of the full game tree is more than 10^{20}. As a result, even traversing the whole game tree is practically impossible with current hardware. Therefore, we propose a method that approximates the optimal solution of the game.

4.1 Reduced Game Representation

The full tree of APRG has a very specific structure caused by the limited amount of information available to the players. The actor never learns any information about the observer's actions and the observer learns only the outcomes of the chance nodes. As a result, the information sets of the players include a large number of nodes in the tree. This fact allows us to represent the game in a more compact way without any loss of information or the expressiveness of the computed strategies. The basis of the representation is the concept of a signal tree [5]. A signal tree represents the space of all possible developments of a game from a single player's perspective. In general, plies of player's actions and observations are alternating in the tree. After any observation node, a child corresponding to each of the possible observations that can follow the player's decision is added.

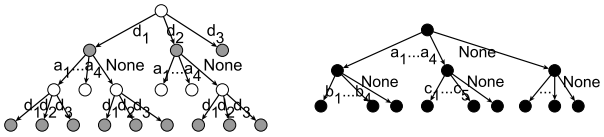


Figure 2: Signal trees of the observer (left) and the actor (right).

The actor does not obtain any observation in the game. Hence, its signal tree is simply the tree of its possible plans (see Figure 2). The observer obtains an observation after each decision. Hence, his signal tree contains both the observation and the decision plies. In general, each decision node has exactly $|D|$ children and each observation node has exactly $|A|$ children. However, forward pruning based on the domain-specific knowledge can be applied. The depth of the defender's signal tree, in the worst case, is uniform and it is two times deeper than the length of the longest attack plan. If the confusion matrices m contain sharp ones or zeros, the size of the tree can be reduced.

Any branch from one tree in combination with a branch of the matching length from the other tree defines a unique history h in the extensive-form game – the full game tree. All histories that correspond to the same decision node in the signal tree of a player belong to the same information set of the player. Also, all information sets that exist in the full EFG are represented by a node in one of the signal trees. As a result, any behavioral strategy can be represented as a probability distribution in each decision node of the signal tree.

The size of the two signal trees is much smaller than the size of the full game tree. With the 5 choices and 10 moves, the size of the attacker's signal tree is less than 10^7 and the size of the defender's tree is less than 10^{14}. It is still not possible to traverse the observer's signal tree in a reasonable time, but the reduction is substantial. In order to make this representation of the game equivalent to the full game tree, we have to solve two issues. First, the chance nodes in the observer's tree must become more complex. One chance node in the observer's signal tree corresponds to many chance nodes in the full game tree. However, the probabilities of the observations in the observer's signal tree depend only on the current (last) action of the actor. Hence, if the chance node stores a conditional probability $p(o|a)$, we do not lose any information about the distributions. The second issue are the utilities. In general, any leaf of the full game

tree can have a different utility. All this information cannot fit into the signal trees; it has to be stored/computed separately. For any pair of branches from the signal trees, the corresponding history in the full game can be easily computed and we assume there is an algorithm that computes the utility based on this information.

4.2 Concurrent Monte-Carlo Tree Search

We propose a method, for computing the strategy in the game, based on concurrent construction of both the actor's and the observer's signal tree by Monte-Carlo Tree Search (MCTS). Our method is a generalization of MCTS developed for perfect information games [8]. A similar approach has been proposed by [2] and applied to the game of phantom tic-tac-toe. However, we extend the algorithm in two directions to make it applicable to our problem. The first is handling of the conditional chance nodes that appear in the defender's signal tree. The second extension is enabling the algorithm to further improve the players' strategies during the game play.

The algorithm maintains two MCTS-like trees corresponding to the two signal trees defined in Section 4.1. We present the pseudocode of the algorithm in Figure 3. At the beginning of the algorithm, both trees contain only one node – the root. The trees are then gradually expanded. We denote the root nodes $aRoot$ and $dRoot$ for the actor and the observer respectively. The main loop of the algorithm performs identical iterations until it is out of time. It is an anytime algorithm; the more computation time it has allocated the better is the result it produces.

In each iteration, first the plan for the actor is selected (procedure $selectActorPlan$). Starting from the root node of the actor's tree, the actions of the actor's plan are selected based on a suitable MCTS selection function, which balances the exploration and exploitation (lines 1-5). When the selection reaches a leaf of the currently constructed part of the tree and it is not the end of some actor's plan, all possible direct successors of the node are added to the tree (line 7). Then, the plan is competed randomly to a full plan of the actor (lines 8-12). At the end, the actor's plan is returned $(a_1 \dots a_h) \in P$.

Next, a plan for the observer is selected in a similar way from his signal tree (procedure $selectObserverPlan$). In the observer's decision nodes, a suitable selection function is applied (line 4). In the observation nodes, conditional probability distributions $p(o|a,d)$ from m are used to select a child. The algorithm first selects the plan for the actor; hence, we can identify the right distribution in the matrix at this point (line 6). If the observation node is a leaf of the constructed part of the tree, the descent in the tree stops (line 8). Otherwise, the child corresponding to the generated observation is selected (line 9). The selection of the observer's plan continues until either depth h (i.e., the length of the actor's plan) is reached or a leaf in the observer's tree is selected. In the earlier case, the procedure ends (line 11). In the latter case, the children of the selected leaf are added to the tree (lines 13, line 19) and the variables are modified so that the simulation can start in an observer's decision node. The rest of the observer's actions are selected randomly and the resulting observations are computed (lines 22-27). At the end of this procedure, we have the full observer's plan and the observations it produced $(d_1 \dots d_h, o_1 \dots o_h)$.

With the plan of the actor and the observer's observations, we can compute the utility function of the players in the $Main$ procedure (line 4). The utility is then back-propagated in both trees as in MCTS (procedure $backPropagate$). Starting in the selected leafs of the trees, the statistics in the nodes are updated. Afterwards, next iteration of the $Main$ procedure starts.

Procedure: Main

```
 1: loop
 2:    (aLeaf, aPlan) = selectActorPlan(aRoot)
 3:    (dLeaf, dPlan, Obs) = selectObserverPlan(dRoot, aPlan)
 4:    u = utility(aPlan, Obs)
 5:    backPropagate(aLeaf, u); backPropagate(dLeaf, u)
 6: end loop
```

Procedure: selectActorPlan(aRoot)

```
 1: curNode = aRoot
 2: while curNode is not leaf do
 3:    action = select(curNode); aPlan = aPlan + action
 4:    curNode = child(curNode, action)
 5: end while
 6: if curNode is plan end then return (curNode, aPlan)
 7: addNewChildren(curNode)
 8: aLeaf = select(curNode); curNode = aLeaf
 9: while curNode is not plan end do
10:    action = random(curNode); aPlan = aPlan + action
11:    curNode = createNodeAfter(curNode, action)
12: end while
13: return (aLeaf, aPlan)
```

Procedure: selectObserverPlan(dRoot, aPlan)

```
 1: curNode = dRoot
 2: while curNode not leaf & length(aPlan) > 0 do
 3:    aAction = popFirst(aPlan)
 4:    dAction = select(curNode); dPlan = dPlan + dAction
 5:    obsNode = child(curNode, dAction)
 6:    obs = confusionMatrixSample(dAction, aAction)
 7:    Observations = Observations + obs
 8:    if obsNode is leaf then break
 9:    else curNode = child(obsNode, obs)
10: end while
11: if length(aPlan) = 0 then return (curNode, dPlan, Observations)
12: if curNode is leaf & length(aPlan) > 0 then
13:    addNewChildren(curNode)
14:    dAction = select(curNode); dPlan = dPlan + dAction
15:    obsNode = child(curNode, dAction)
16:    obs = confMatSample(dAction, aAction)
17:    Observations = Observations + obs
18: else
19:    addNewChildren(obsNode)
20:    curNode = child(obsNode, obs)
21: end if
22: while length(aPlan) > 0 do
23:    aAction = popFirst(aPlan)
24:    dAction = random(curNode); dPlan = dPlan + dAction
25:    obs = confMatSample(dAction, aAction)
26:    Observations = Observations + obs
27: end while
28: if curNode is leaf then return (curNode, dPlan, Observations)
29: else return (obsNode, dPlan, Observations)
```

Procedure: backPropagate(node, u)

```
1: updateStatistics(node,u)
2: if node is not root then backPropagate(parent(node),u)
```

Figure 3: The concurent monte carlo tree search algorithm for the first stage of the game. Procedures $select$ and $updateStatistics$ implement some MCTS selection strategy, such as UCT of EXP3.

4.3 Convergence of the Algorithm

We have implemented this algorithm under the hypothesis that with a suitable selection function, it converges to the Nash equilibrium of the game. We do not have a full formal proof supporting it yet.

It has been proven that in a one stage normal form game, the overall frequencies of using individual actions with EXP3.1[1] as the selection function converges to the Nash equilibrium, even if the utilities in the game and the number of actions of the opponent are unknown to the players [6]. This fact has been used to create an algorithm for playing an imperfect information version of tic-tac-toe [2], which is in spirit very similar to our algorithm. It uses separate trees for the players, but does not deal with chance nodes and continuous reasoning. The author claims that even in this setting, his algorithm converges to a NE of the extensive form game. However, he does not provide a formal proof or extensive experimental evidence to support this claim. On the other hand, his as well as our experimental results indicate that this method can be used to create successful players for imperfect information extensive form games and we believe that it is possible to proof the convergence for a large class of games.

If an agent uses EXP3.1 as the selection function in a repeated problem, its loss for not playing the optimal action all the time (i.e., regret) can be bounded by $O(1/\sqrt{(T)})$ [1], where T is the number of trials. It means the quality of the produced solution gradually improves. Even before full convergence of the algorithm, the computed strategy guarantees a bound on its sub-optimality.

4.4 Continuous Reasoning

The described algorithm finds a suitable course of action for both players at the beginning of the game. In theory, it can be run for a long time before the game starts and the computed trees can be used for playing the game without further computation. However, the size of the trees that would need to be stored and the time required to compute good strategies in their lower levels would be huge. For practical applications, we suggest computing an initial strategy and adding more iterations after each step of the game.

The continuous reasoning is slightly different for each player. We further describe the algorithm for the observer, which is the main focus of this paper. After each move, the root of the observer's tree is substituted by its grandchild corresponding to the selected observer's action and the actual observation in the game. The situation is more complex in the actor's tree. The observer generally cannot be sure about the current actor's tree node in the later stages of the game. Instead, the defender maintains a probability distribution among k most probable nodes where the attacker can be. We call the nodes the *root set*. The probability of the nodes in the set can be computed by Bayesian inference form the probabilities computed by the algorithm in the previous stage and the confusion matrices. For each node in the root set of the current stage, the concurrent MCTS converges to the probability of executing each of the actions by rational actor in the current situation ($p(a)$). The probability that the actor is in the child under the action a given the observation o can be computed as:

$$p(a|o) = \frac{p(o|a)p(a)}{p(o)} = \frac{p(o|a)p(a)}{\sum_{a_i \in A} p(o|a_i)p(a_i)} \qquad (2)$$

The probabilities of different observations given an attacker's action ($p(o|a_i)$) can be extracted from the confusion matrix that corresponds to the classifier selected by the observer in the game.

After the probability of each child of the nodes in the current root set is computed, the k most probable children are selected to the root set in the following stage. Further, for each actor's action, we can compute the probability that the action was the last action executed by the actor. It is the sum of probabilities of executing the action in each node of the root set. The action with the highest execution probability is the observation generated by the algorithm and used in evaluation as the input to the utility function (1).

When concurrent MCTS runs with a root set containing multiple nodes, one of them is randomly selected as the root of the actor's tree in each iteration proportionally to their probabilities.

5 Experimental Evaluation

We evaluate the presented approach on a simplified network security domain. The actor is the attacker that tries to attack a computer network. The attacker can perform one of 12 actions (set A), such as vertical ping scan, or brute-force attack to a specific service. These actions have preconditions and postconditions and their legal sequences generate over 150 thousand different attack plans of different lengths (set P). Each action provides the attacker with some knowledge about the network. The values g of the attack plans are defined in terms of this knowledge and vary between 0 and 120. The observer is the intrusion detection system operating on the network. It has three different settings; each of them can detect some of the attacker's actions better and other worse. The actions of the observer (set D) are selecting one of these settings. The three classes of actions that are detected better in different settings are producing high, medium, and small volume of network traffic. The classes include two, four, and six actions respectively. Each setting detect the actions from the class correctly with 90%, 60%, and 30% precision, misclassifying in remaining cases evenly as any other action. This defines the confusion matrices m.

We performed the experiments with a single-thread implementation of the algorithm presented in Section 4 on Intel(R) Core(TM) i7 CPU @ 3.20GHz. In order to select (close to) rational attack plans, we run the proposed algorithm for 10 minutes in the initial stage of the game. Then, we took five different high probability plans, that are presented in Figure 4. They are ordered by number of samples that used the plans in the attacker's tree, with the first plan having 30% of all samples and the fifth a little more than 1%.

The observations generated for the observer are stochastic; hence we run the recognition process 200 times for each plan. The algorithm was set to use 2 minutes of computational time per game stage. We compare the performance of the proposed algorithm in terms of the utility to several baseline values. The first three baselines always use only one of the actions available to the defender. We denote them by the first character of the volume class (H,M,L). The other two baselines are the ex post best response (BR) and worst response

first: *dns_requests, Horiz_scan_for_spec_service, web_attacks, connect_to_host, fingerprinting* ($g = 86$)
second: *dns_requests, SEND_SPAM, connect_to_host, Horiz_scan_for_spec_service, fingerprinting* ($g = 86$)
third: *SEND_SPAM, connect_to_host, Horiz_scan_for_spec_service, fingerprinting* ($g = 81$)
fourth: *dns_requests, Horiz_scan_for_spec_service, web_attacks, connect_to_host* ($g = 78$)
fifth: *Horiz_scan_for_spec_service, web_attacks, DDOS_TO_SPECIFIC_SERVICE, fingerprinting, connect_to_host* ($g = 93$)

Figure 4: Attack plans used in evaluation. The upper cased actions are in the high traffic category, medium traffic actions have capital letter at the beginning and the low traffic actions are lowercased.

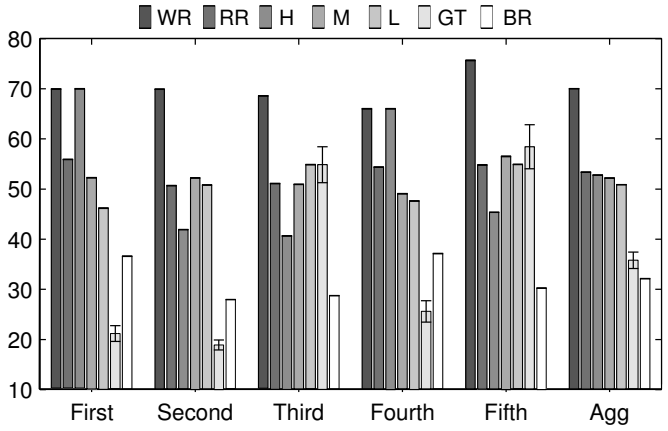


Figure 5: The mean attacker's utility with the proposed approach (GT) and the base lines; RR - random classifier selection; BR - the result of the ex post optimal classifier; and other baselines defined in the text. Lower values are better for the defender.

(WR). With the prior knowledge of the actual attacker's plan, we compute BR (WR) by selecting the defender's action with the highest (lowest) probability of observing the actual attacker action. The last baseline is the random selection of defender's action (RR). As all these baselines are independent of defender's observations, we can quickly compute a good approximation of their quality by averaging over 10^6 trials for each plan.

Figure 5 presents the utility of the proposed method (GT) and the baselines computed by formula (1). The values are presented for each attacker's plan separately and aggregated over all five plans (Agg). Overall, the GT approach is significantly better then all applicable baselines and it is only slightly worse than the BR. The error bars indicates the 95% confidence interval of the mean. For all baseline values, the confidence intervals are hardly noticeable in the graphs, because of their small width. Looking at the individual plans, the GT approach performs very well on the first two plans and the results are worst for the fifth plan. This can be expected as the first plans better approximate the rational behavior of the attacker and the later plans have higher chance to be irrational. Note that the result of GT can be better than BR, because it combines the information from the classifiers with the game theoretic reasoning and does not directly outputs the observation of the classifiers as BR.

In the plan recognition task, the plan actually performed by the attacker was ranked as the most likely by the GT approach in 38.6% of runs and its median position in the final list of the most probable plans was 5. This is a promising result considering there are over 150 thousand plans to choose from.

The number of iterations (samples) of the concurrent MCTS algorithm performed in a single stage varied with the stage of the game. The histogram of the number of instances in which certain number of iterations was performed follows roughly binomial distribution. In most instances, more than 10^7 iterations were performed, however, in some cases even 10^8 can be made in time. One instance of the algorithm never crossed 4.5GB memory limit during the experiment. The memory was used for storing the MCTS trees, size of which can be substantially limited by a more conservative expansion strategy.

6 Conclusion

In this paper, we argue that the problem of adversarial plan recognition, where the observed agent actively tries to avoid observation should be modeled as a game. The observed agent has to model reasoning of the observer and vice versa in order to derive the optimal strategies. We model the problem as a specific subclass of imperfect-information zero-sum extensive form games, which we term Adversarial Plan Recognition Games. We show how problems from this class can be compactly represented by a pair of signal trees of the players. Based on this representation, we propose an algorithm that approximates the optimal solution of the game. The algorithm is a novel generalization of Monte-Carlo tree search. In order to verify applicability of APRG, we use it to model a simplified game between the attacker and the intrusion detection system in the network security domain. The presented experimental evaluation shows that the proposed algorithm can be used to produce good behaviors for the agents in reasonably large domains. The observer using the proposed algorithm performed significantly better than any of the naive baseline approaches we tried.

The proposed model can be further extended in several interesting directions. In real world applications of the model, it is not likely that the agents involved will be perfectly rational. It would be interesting to relax the rationality assumption and use a suitable bounded rationality model. Furthermore, it would be interesting to extend the model to allow temporally extended actions. If various actions of the observed agent take different time, the model cannot be directly applied because of more complex synchronization between the observer and the observed agent. Solving this issue is also an important line of future research.

ACKNOWLEDGEMENTS

The authors thank Martin Rehák, Martin Grill, and Tomáš Pevný for helpful discussions forming this paper. This research was supported by the Air Force Office of Scientic Research (grant no. FA8655-10-1-3016), the Office of Naval Research Global (grant no. N62909-12-1-7019), and the Czech Science Foundation (grant no. P202/12/2054).

REFERENCES

[1] P. Auer, N Cesa-Bianchi, Y Freund, and R.E. Schapire, 'The non-stochastic multiarmed bandit problem', *SIAM Journal on Computing*, **32**(1), 48–77, (2003).

[2] D. Auger, 'Multiple Tree for Partially Observable Monte-Carlo Tree Search', in *Applications of Evolutionary Computation*, pp. 53–62. Springer, (2011).

[3] D. Avrahami-Zilberbrand and G.A. Kaminka, 'Incorporating observer biases in keyhole plan recognition (efficiently!)', in *Proc. of the National Conference on Artificial Intelligence*, volume 22, p. 944, (2007).

[4] S. Braynov, 'Adversarial planning and plan recognition: Two sides of the same coin', in *Secure Knowledge Management Workshop*, (2006).

[5] P. Ciancarini and G.P. Favini, 'Monte Carlo tree search in Kriegspiel', *Artificial Intelligence*, **174**(11), 670–684, (jul 2010).

[6] C. Daskalakis, A. Deckelbaum, and A. Kim, 'Near-optimal no-regret algorithms for zero-sum games', in *Proc. of the Twenty-Second Annual ACMSIAM Symposium on Discrete Algorithms*, pp. 235–254, (2011).

[7] C.W. Geib and R.P. Goldman, 'Plan recognition in intrusion detection systems', in *DARPA Information Survivability Conference & Exposition II*, volume 1, pp. 46–55. IEEE, (2001).

[8] L. Kocsis and C. Szepesvári, 'Bandit based Monte-Carlo Planning', in *ECML-06*, (2006).

[9] W. Mao and J. Gratch, 'A utility-based approach to intention recognition', in *AAMAS 2004 Workshop on Agent Tracking: Modeling Other Agents from Observations*, (2004).

[10] X. Qin and W. Lee, 'Attack Plan Recognition and Prediction Using Causal Networks', in *20th Annual Computer Security Applications Conference*, pp. 370–379. Ieee, (2004).

[11] Y. Shoham and K. Leyton-Brown, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*, Cambridge University Press, 2009.