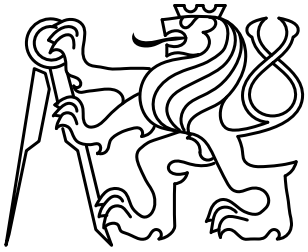
Czech Technical University in Prague
Faculty of Electrical Engineering

Doctoral Thesis

July 2014 Branislav Bošanský

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Cybernetics



Iterative Algorithms for Solving Finite Sequential Zero-Sum Games

Doctoral Thesis

Branislav Bošanský

Prague, July 2014

Ph.D. Programme: Electrical Engineering and Information Technology (P2612)
Branch of study: Artificial Intelligence and Biocybernetics (3902V035)

Supervisor: doc. Ing. Lenka Lhotská, CSc.
Supervisor-Specialist: prof. Dr. Ing. Michal Pěchouček, M.Sc.

Dedicated to my parents, my sister, and Pavlína for their endless support.

Acknowledgments

I am grateful for having an opportunity to work with great researches that make this thesis possible. First of all I would like to thank my supervisor, doc. Lenka Lhotská for giving me the opportunity as a PhD student and her support throughout the years. I would like to thank prof. Michal Pěchouček for introducing me to multi-agent research and game theory in Agent Technology Center (ATG). My thanks also goes to Viliam Lisý for great collaboration and for discovering the beauty of game theory throughout our PhD studies. I would like to thank prof. Christopher Kiekintveld for endless consultations and collaboration on the main algorithms in this thesis. I would also like to thank many of the current or former colleagues in ATG, namely Peter Novák for shaping my perspective on research, Ondřej Vaněk and Michal Jakob for reminding me that one should always think about real-world applications, Jiří Čermák for helping me with the implementation of the framework and the algorithms, and other ATG members for valuable discussions about our research topics. I am also grateful to Albert Xin Jiang and Milind Tambe for their time and collaboration on the newest algorithm in this thesis during my stay at University of Southern California, and to prof. Antonín Kučera, Vojtěch Řehák, Tomáš Brázdil, and Jan Krčál for their collaboration on the problem of patrolling and introducing me to more theoretic perspectives of algorithmic game theory and stochastic games.

The work in this thesis was supported by Czech Science Foundation grant P202/12/2054 (Security Games in Extensive Form), Czech Ministry of Education, Youth and Sports project LH11051 (Formal Models and Effective Algorithms for Intelligent Protection of Transportation Infrastructure), and the Grant Agency of the Czech Technical University in Prague (grant no. OHK3-060/12).

Abstract

Non-cooperative game theory belongs to theoretic foundations of multi-agent systems describing the optimal behavior of rational agents in competitive scenarios (termed games). Problem of finding optimal strategies and solving games is one of the classical problems of computer science and there is a continuous effort to push the scalability by designing new efficient algorithms able to solve large games.

We focus on finite, strictly competitive sequential scenarios where agents sequentially perform their actions and observe some changes in the environment. We introduce a collection of novel exact algorithms that improve scalability in three different classes of sequential games compared to existing state-of-the-art exact algorithms.

First, we study games where agents act simultaneously in each state of the game, while they are able to perfectly observe this state. Second, we move to a more general class of games where agents cannot perfectly observe the current state and study extensive-form games with imperfect information. In these games we assume that agents can perfectly remember the history of their own actions and all information gained during the course of the game. Third, we relax this restriction and allow an agent to forget certain information, while assuming limited observability of the actions of the opponent.

We experimentally evaluate the performance of each of the novel algorithms on a collection of games from the corresponding class and compare against the state-of-the-art algorithms. The results confirm significant improvement in memory and typically also computation time, often in orders of magnitude, demonstrating that our novel algorithms are able to solve much larger games.

Abstrakt

Nekooperativní teorie her popisuje optimální chování agentů v kompetitivních scénářích (v hrách) a patří k teoretickým základům multiagentních systémů. Problém nalezení optimální strategie a řešení her je součástí umělé inteligence a počítačových věd od jejich počátků. Nejen v poslední době je však kladen důraz na škálovatelnost algoritmů a možnost řešení velkých her.

V práci se věnujeme konečným sekvenčním hrám, ve kterých mají agenti navzájem protichůdné cíle. V těchto hrách agenti mění stav prostředí pomocí akcí, přičemž některé tyto změny mohou pozorovat. V práci představujeme několik nových exaktních algoritmů navržených pro tři podtřídy sekvenčních her, přičemž každý z těchto nových algoritmů posouvá hranice poznání a překonává doposud existující algoritmy.

V první řadě se zaměřujeme na hry se simultánními tahy, ve kterých hráči plně pozorují své prostředí, ale akce provádějí simultánně. Následně se zabýváme nejobecnější třídou her, hrami v extenzivní formě s neúplnou informací. Jediným předpokladem v těchto hrách je, že agenti nezapomínají historii vlastních akcí ani žádnou informaci získanou během hry. Jako poslední představujeme třídu her, ve kterých tento předpoklad odstraňujeme, přičemž předpokládáme omezené možnosti pozorování akcí oponenta.

Každý z nových algoritmů experimentálně porovnáváme s existujícími algoritmy na sadě her z příslušné třídy. Výsledky experimentů ukazují, že nově představené algoritmy umožňují řešení mnohem větších her a často dosahují zrychlení na úrovni několika řádů.

Contents

1 Introduction **1**
- 1.1 Topic and Research Goals of this Thesis . . . 2
  - 1.1.1 Aims and Contributions of this Thesis . . . 4
  - 1.1.2 Organization of the Thesis . . . 5

2 Introduction to Game Theory **7**
- 2.1 Introduction to Game Theory and Basic Definitions . . . 8
  - 2.1.1 Solution Concepts . . . 9
- 2.2 Extensive-Form Games . . . 10
  - 2.2.1 Definitions . . . 10
  - 2.2.2 Strategies in Extensive-form Games . . . 12
  - 2.2.3 Nash Equilibrium (NE) in Extensive-Form Games . . . 12

3 Computing Nash Equilibrium **15**
- 3.1 Solving Normal-Form Games . . . 15
- 3.2 Solving Extensive-Form Games . . . 16
  - 3.2.1 Sequence-Form Representations of Strategies . . . 17
  - 3.2.2 Sequence-Form Linear Program . . . 18
- 3.3 Approximating Nash Equilibrium . . . 19
  - 3.3.1 Counterfactual Regret Minimization . . . 20
  - 3.3.2 Excessive Gap Technique . . . 20
  - 3.3.3 Monte Carlo Tree Search . . . 21
- 3.4 Double-Oracle Algorithm for Normal-Form Games . . . 22

4 Double-Oracle and Alpha-Beta in Simultaneous-Move Games **25**
- 4.1 Definitions . . . 25
- 4.2 Related Work . . . 27
  - 4.2.1 Simultaneous-Move Alpha-Beta Search (SMABS) . . . 27
  - 4.2.2 Approximative Algorithms and Heuristic Approach . . . 28
- 4.3 Backward Induction with Double-Oracle and Serialized Bounds . . . 29
  - 4.3.1 Backward Induction with Serialized Alpha-Beta Bounds . . . 29
  - 4.3.2 Integrating Double-Oracle with Backward Induction . . . 32
- 4.4 Experimental Evaluation . . . 36

4.4.1 Experiment Domains . . . 36
4.4.2 Results . . . 39
4.4.3 Improved variants of DO$\alpha\beta$ . . . 43
4.5 Discussion of the Results and Summary . . . 44

5 Sequence-Form Double-Oracle Algorithm for Extensive-Form Games 47
5.1 Related Work . . . 47
5.2 Sequence-Form Double-Oracle Algorithm . . . 48
5.2.1 Restricted Game . . . 51
5.2.2 Best-Response Sequence Algorithms . . . 55
5.2.3 Main Loop Alternatives . . . 58
5.3 Theoretical Analysis . . . 59
5.4 Experimental Evaluation . . . 62
5.4.1 Experiment Domains . . . 63
5.4.2 Results . . . 65
5.5 Discussion of the Results and Summary . . . 71

6 Normal-form Games with Sequential Strategies 73
6.1 Definition and Background . . . 74
6.1.1 Solving Normal-Form Games with Sequential Strategies . . . 76
6.2 Compact-Strategy Double-Oracle Algorithm . . . 76
6.2.1 Restricted Game . . . 77
6.2.2 Best-Response Algorithm . . . 80
6.2.3 Theoretical Analysis . . . 80
6.3 Experimental Evaluation . . . 81
6.3.1 Experiment Domains . . . 81
6.3.2 Results . . . 82
6.3.3 Support Size Analysis . . . 84
6.4 Discussion of the Results and Summary . . . 85

7 Conclusions and Future Work 87
7.1 Thesis Achievements . . . 87
7.2 Future Work . . . 88
7.2.1 Beyond Current Domain-Independent Double-Oracle Algorithms . . . 89
7.2.2 Theoretical Analysis of Double-Oracle Algorithms . . . 90

Appendix A Game-Theoretic Library 97
A.1 Domains . . . 97
A.2 Algorithms . . . 100

Appendix B Publications 103

Chapter 1

Introduction

> "We must not forget that when radium was discovered no one knew that it would prove useful in hospitals. The work was one of pure science. And this is a proof that scientific work must not be considered from the point of view of the direct usefulness of it. It must be done for itself, for the beauty of science, and then there is always the chance that a scientific discovery may become like the radium a benefit for humanity."
>
> *– Marie Curie, Lecture at Vassar College, May 14, 1921*

Non-cooperative game theory provides mathematical models of behavior of rational agents (or players) in competitive scenarios. Designing algorithms and analyzing computational complexity of the problem of solving games has been an important part of research in computer science. Recently, the number of real-world applications of game-theoretic models has increased, supported by the focus of research on designing and implementing new algorithms that improve scalability and allow solving larger games. The most prevalent are the applications in the security domain (Tambe, 2011; Ordónez et al., 2013). Deploying checkpoints at LAX international airport (Pita et al., 2008), scheduling air marshals to flights over US (Tsai et al., 2009), securing the maritime transit (Vanek et al., 2011a; Vanek, 2013), or protecting the transit network (Luber et al., 2013) are among the best known applications. However, the application of game-theoretic results is not restricted to security and successful applications include scenarios from auctions and mechanism design for charging electric cars (Vytelingum et al., 2010; Robu et al., 2013), or problems of social choice applied in recommendation systems (Popescu, 2013).

The interaction between agents in real-world scenarios is often complex. Agents perform actions to change the state of the world, they are able to observe some effects of these changes, and react to them accordingly. Moreover, the agents might not have a perfect information about the state of the world, or about the actions performed by the other agents, the actions of agents may fail, or there can be other form of uncertainty in the environment. An example of such complex interactions can be an auction, where the goal of an agent is to acquire some item. An agent can increase her bid and can observe the bids of the other agents. On the other hand, the agent does not know the true evaluation of the other agents for this item and may have some uncertainty about her own evaluation. Another examples

include security scenarios, where the members of a security team observe the current situation (e.g., monitoring a store or an airport using cameras), and then react to information according to the security protocol (e.g., there is a suspicious person near the entrance to a restricted area). Such a sequential reasoning and a proper response to different situations must be incorporated directly into the game-theoretic model. Otherwise it can be exploited by a strategically reasoning adversary (e.g., by using decoys or deception).

Game theory provides a model capable of representing this type of scenarios termed *extensive-form games*. The whole course of the scenario is represented as a game tree, where each node of the game tree corresponds to a state in the scenario. In each non-terminal state, one of the agents can execute an action leading to a different state. Each terminal state of the scenario correspond to a leaf in the game tree and has assigned utility values that represent the benefit of reaching this state for each player. Classes of games analyzed in this thesis follow two key assumptions: (1) we study finite games that (2) model strictly competitive situations.

Extensive-form games are finite in the number of (possible) actions the agents perform before the game terminates. This model is sufficiently expressive to capture many real-world scenarios (e.g., executing a security protocol, or participating in an auction, etc.). Moreover, this model is capable of modeling various types of uncertainty. Players may not be able to exactly observe the current state, or there can be uncertainty due to stochastic environment. On the other hand, finite games are substantially easier to solve from the computational perspective compared to a more general class of infinite *stochastic games* and thus allow us to model larger scenarios.

We restrict our focus on strictly competitive situations, where the gain of one player means loss for the opponent. These scenarios can be modeled as *zero-sum games*. Even though finite zero-sum games are the easiest from the viewpoint of theoretical computer science (they can be solved in polynomial time in size of the game), they still present a great challenge from the computational perspective when modeling real-world scenarios. Many situations are essentially strictly competitive. Examples include scenarios corresponding to area protection, where one player wants to cross undetected (Vanek et al., 2010; Vanek et al., 2012), more general security domains focusing on patrolling and protection of targets (Agmon et al., 2009; Bosansky et al., 2011). Also, many classical board and card games like Chess, Backgammon, or Poker belong to this class.

1.1 Topic and Research Goals of this Thesis

Solving large sequential games presents a great challenge in the computational game theory. The computational challenge in these games stems from an extremely large state space on which the algorithms need to operate. For example, No-Limit Texas Hold'em Poker game has over 10^{70} different states (and there are 10^{18} states in a limited setting (Sandholm, 2010)). Therefore, new algorithms and methods are necessary for pushing the boundaries of the state of the art and allowing to solve larger games.

The *main goal* of this thesis is to *advance the state of the art in algorithms for solving sequential zero-sum games*. This goal is achieved with a set of novel algorithms designed for

different variants of sequential games. All these algorithms share several key characteristics: (1) all presented algorithms are exact (they compute an exact Nash equilibrium), (2) the algorithms are domain-independent and do not need to use any specific domain knowledge, and finally (3) they all exploit iterative double-oracle framework.

The focus on exact algorithms is in contrast with the current trends, where most of the algorithmic approaches are approximative, exploiting stochastic sampling of the large state space, and/or methods from machine learning (Kocsis and Szepesvári, 2006; Zinkevich et al., 2008; Sandholm, 2010; Lanctot, 2013). The approximative algorithms are extremely successful in a so called on-line setting, where there are strict time limitations. Sampling algorithms are able to quickly return reasonably good strategy (or action) that should be executed. On the other hand, these algorithms are less successful when the goal is to solve the game and find pair of equilibrium strategies (so called off-line setting) without strict time restrictions. This means to compute the exact expected outcome of the game, or approximate this value to a very small error (e.g., 10^{-5}). These scenarios often arise in the security domain, where the game models are solved and then the computed strategies are used in daily practice (Tambe, 2011). Another possible application includes using the algorithms in mathematical applications, or in generic game-theoretic software tools (e.g., Gambit (McKelvey et al., 2014)). Finally, improving exact algorithms can affect and speed-up also the performance of approximative algorithms in the future. Combining deterministic exact algorithms with a stochastic approximative algorithm can be a very promising direction of research in artificial intelligence (e.g., in (Alliot et al., 2012)). In sequential games, one possible connection is via solving endgames – smaller parts of the game close to the end can be solved exactly yielding a better result overall in comparison to just using the strategies from the approximative algorithms (as for example demonstrated in (Ganzfried and Sandholm, 2013)).

We focus on domain-independent algorithms for three classes of sequential games. Games in the following classes can model various scenarios, however, we seek for the most generic and domain-independent improvement to maximize the possibility of future application and implementation in practice. Moreover, all the algorithms can be easily strengthened with the domain-specific knowledge. We clearly identify which subroutines can be replaced with a faster domain-specific methods, offering further computation speed-up. First, we study the simplest type of imperfect-information games, termed *simultaneous-move games*, in which both players can perfectly observe the state of the game and the imperfect information is only local due to simultaneous choice of the players in each state of the game. Secondly, we study the most general class of imperfect-information extensive-form games, where the only restriction is the *perfect recall* – the assumption the players perfectly remember the history of their actions and all information gained throughout the game play. Finally, we study one example of games where the assumption of perfect recall does not hold. The games with imperfect recall are in general much harder to solve (Koller and Megiddo, 1992). We focus on a specific subclass of games where the players have limited observation capabilities and they are not able to directly observe the actions of the opponent.

Finally, all presented algorithms exploit the iterative double-oracle framework. Oracle algorithms were introduced in the game theoretic setting in (McMahan et al., 2003) and they

can be seen as an application of methods of column/constraint generation used for solving large-scale optimization problems (Dantzig and Wolfe, 1960; Barnhart et al., 1998). The oracle algorithm exploits two characteristics commonly found in games. First, in many cases finding a solution to a game only requires using a small fraction of possible strategies, so it is not necessary to enumerate all of the strategies to find a solution (e.g., as analyzed in (Koller and Megiddo, 1996; Schmid et al., 2014)). Second, finding a best response to a specific strategy of the opponent in a game is computationally much less expensive than solving for an equilibrium (Wilson, 1972). In addition, best response calculations can often make use of domain-specific knowledge or heuristics to speed up the calculations even further. In the worst case, the oracle algorithms need to enumerate all possible strategies; hence, they do not bring any advantage from the theoretical perspective. On the other hand, experimental results show dramatic speed-up in larger games, often in orders of magnitude. Several previous works demonstrated the success of this method in practice for specific domains (McMahan et al., 2003; Halvorson et al., 2009; Vanek et al., 2010; Jain et al., 2011; Vanek et al., 2011b; Vanek et al., 2012). In this work we focus on novel domain-independent variants of the double-oracle algorithms for sequential games. Although there were some initial works that applied double-oracle framework for such games (McMahan and Gordon, 2007a; Zinkevich et al., 2007), the performance of the proposed algorithms was limited compared to the existing algorithms.

1.1.1 Aims and Contributions of this Thesis

There are three research goals that provide contributions of this thesis:

Set of Novel Algorithms The main goal is an introduction of new exact algorithms for solving three different classes of games: (1) simultaneous-move games, (2) extensive-form games with imperfect information, and (3) normal-form games with sequential strategies. Newly designed algorithms should outperform existing state-of-the-art algorithms and allow to solve much larger games.

Generalization of Double-Oracle Methods Second goal is to generalize the iterative double-oracle methods. All the previous work used double-oracle method strictly as an algorithm that iteratively expands the game using new strategies. However, this concept can be generalized and more fine-grained approach can be used, where the restricted game can be expanded with the actions, or sequences of actions. To enable such a generalization, more compact representation of the strategies need to be defined that allow us to compactly reason about strategies only in specific parts of the game.

Implementation and Framework The third goal is to provide domain-independent implementation of introduced algorithms in order to maximize the possibility of their future applications and usage in practice. In order to accomplish this goal, a generic framework for modeling and solving extensive-form games must be implemented, together with a collection of games on which the algorithms can be compared. The importance of such a framework is due to lack of a benchmark collection of games

for the classes of games of our interest. The best-known collection of games for purposes of computational benchmarks, GAMUT (Nudelman et al., 2004), offers only single-step games.

1.1.2 Organization of the Thesis

The organization of the thesis is based on the class of the game, for which the novel algorithms are designed.

- Chapter 2 provides the necessary background for computational game theory and describes two best-known and used mathematical representation for games – normal-form (also known as matrix, or strategic-form) games, and extensive-form games. Only the key and necessary concepts are defined formally.
- Afterwards, Chapter 3 presents the description of the algorithms for solving zero-sum games. Keeping aligned with the focus of the thesis on exact algorithms, these are described in more details. However, the chapter also provides a quick description of the most used approximative and heuristic algorithms used for sequential games.
- Chapter 4 introduces novel double-oracle algorithm for simultaneous-move games that exploits alpha-beta pruning. This chapter is based on works (Bosansky et al., 2013b; Bosansky, 2013; Bosansky et al., 2014c).
- Chapter 5 introduces the most generic double-oracle algorithm for imperfect-information extensive-form games. This chapter is based on works (Bosansky et al., 2012; Bosansky et al., 2013a; Bosansky, 2013; Bosansky et al., 2014b).
- Chapter 6 introduces a further generalization of the double-oracle algorithm and apply this method to a limited class of sequential games where the assumption of perfect recall does not need to hold. This chapter is based on work (Bosansky et al., 2014a)
- Finally, we conclude the thesis and offer several possible directions for future research along both theoretical and more applied lines.
- Appendix A describes the framework used for experimental evaluation and provides a complete list of implemented games and algorithms.

Chapter 2

Introduction to Game Theory

Game theory is a mathematical framework for modeling behavior of rational agents interacting in an environment. Game theory analyzes and prescribes what strategy should agents take in order to achieve their goals, to maximize their expected payoff. Solution of the game is a tuple of strategies (one strategy for each player; this tuple is called *strategy profile*) that optimizes the expected payoff of agents, and they cannot improve their outcome given certain assumptions described by a *solution concept*. Many various solution concepts have been defined over the years. The best known is the *Nash equilibrium* (NE) solution concept (Nash, 1950) that prescribes an optimal behavior for agents assuming all agents know the rules of the game and the rationality of each of the agents is a common knowledge. Informally speaking, a strategy profile is in NE, if all agents cannot improve their outcome by unilaterally changing their strategy. However, the class of solution concepts is rather large and include many other solution concepts (e.g., Correlated equilibrium (Aumann, 1974), Stackelberg equilibrium (von Stackelberg, 1934), or refinements of Nash equilibrium) that pose further assumptions on rules of the game or knowledge of the agents.

Significant research effort has been devoted to studying algorithmic and computational aspects of the problem of solving games over the last years. The result of this effort is twofold: (1) several fundamental results in theoretical computer science and computational complexity were introduced (e.g., proving the computational complexity of computing a Nash equilibrium (Daskalakis et al., 2006; Chen and Deng, 2006), or Stackelberg equilibrium (Conitzer and Sandholm, 2006)); (2) new algorithms with improved scalability were designed and deployed in many real-world scenarios (e.g., apparent in the security domain (Tambe, 2011)).

This chapter formally introduces basic concepts and definitions used in computational game theory[1]. We introduce two basic formal representations of games (normal-form and extensive-form games), we formally define actions that agents can perform and strategies in the game, and finally solution concepts that define equilibrium.

[1]Definitions of concepts from game theory are inspired by books (Shoham and Leyton-Brown, 2009; Maschler et al., 2013).

	H_2	T_2
H_1	$(1, -1)$	$(-1, 1)$
T_1	$(-1, 1)$	$(1, -1)$

Figure 2.1: Normal-form representation of a well known game called Matching Pennies.

2.1 Introduction to Game Theory and Basic Definitions

Throughout this thesis, we focus on strictly competitive games with only two players. The set of all players is denoted as $\mathcal{N} = \{1, 2\}$. We use index i when referring to a single player. We follow a standard notation used in game theory and use $-i$ to the opponent of player i.

Players change the environment by executing actions. We introduce the basic concepts of game theory on the simplest type of games, where the game ends after all players simultaneously perform a single action (known as *single-step*, or *one-shot games*). In this case, the actions are termed *pure strategies*, denoted as $\Pi = \Pi_1 \times \Pi_2$. Executing pure strategies yield some outcome of the game. This outcome is specified with a utility function u that for each player i assigns a utility value to each possible outcome of the game:

$$u_i : \Pi \to \mathbb{R}$$

The utility value determines how good the outcome of the game is in favor for this player. The assumption of the rationality of players is thus captured as their goal to maximize their own utility outcome of the game. Finally, in strictly competitive (or zero-sum) games always holds that $u_i = -u_{-i}$. Simple normal-form games can be conveniently visualized as matrices (see Figure 2.1). Rows of the matrix represent pure strategies for one player, columns represent pure strategies of the opponent. Every possible outcome of the game is written in the cell corresponding to the pair of pure strategies. We use M to refer to the matrix, where M_{rc} corresponds to the outcome of the game (i.e., utility values) if the row player plays strategy $r \in \Pi_1$ and the column player plays strategy $c \in \Pi_2$. The outcome is depicted as a tuple of numbers that correspond to the utility values for each player; (u_1, u_2).

However, a stable equilibrium strategy profile is not always possible if we allow the players to only use pure strategies. Therefore we generalize the concept and define *mixed strategies*; a mixed strategy for a player is a probabilistic distribution over the set of all pure strategies of this player. The set of all mixed strategies is denoted as $\Delta = \Delta_1 \times \Delta_2$, and we can extend the utility function to calculate an expected utility outcome for playing a pair of mixed strategies:

$$u_i(\delta_i, \delta_{-i}) = \sum_{\pi_i \in \Pi_i} \sum_{\pi_{-i} \in \Pi_{-i}} \mathcal{P}(\pi_i|\delta_i)\mathcal{P}(\pi_{-i}|\delta_{-i})u_i(\pi_i, \pi_{-i})$$

where $\mathcal{P}(\pi_i|\delta_i)$ denotes the probability of playing pure strategy π_i in a mixed strategy δ_i.

2.1.1 Solution Concepts

Next we formally define main game-theoretic solution concepts. The goal of the agents is to maximize their expected utility. Therefore, we first need to formally define the concept of a *best response*. Best response is a strategy of a player such that there does not exist any other strategy that yields better expected utility for this player against some fixed strategy of the opponent. Formally, a best response of player i to the opponent's strategy δ_{-i} is (generally mixed) strategy $\delta_i^{BR} \in \Delta_i$, for which $u_i(\delta_i^{BR}, \delta_{-i}) \geq u_i(\delta_i', \delta_{-i})$ for all strategies $\delta_i' \in \Delta_i$.

Nash Equilibrium (NE) A strategy profile δ is in a NE if and only if for each player i it holds that δ_i is a best response to δ_{-i}. It is known that in general can a game have multiple NE that yield different expected values to the players. In the zero-sum setting, however, all these equilibria share the same value (i.e., the expected utility for a player is the same in all equilibria). The expected utility of equilibrium strategies is called the *value of the game* and denoted $\mathcal{V}^*$.

Often, the size of the game prohibits us from finding exact solution and the algorithms seek for approximative equilibrium. First, we again define approximative best response, but we allow some small error $\varepsilon \geq 0$. We say that strategy $\delta_i^{\varepsilon BR}$ is an ε-best response to the opponent's strategy δ_{-i}, if for all strategies $\delta_i' \in \Delta_i$ it holds $u_i(\delta_i^{\varepsilon BR}, \delta_{-i}) + \varepsilon \geq u_i(\delta_i', \delta_{-i})$ A strategy profile δ is an ε-NE if and only if for each player i it holds that δ_i is a ε-best response to δ_{-i}.

Stackelberg equilibrium (SE) Many recent works in computational game theory focus on Stackelberg equilibrium (von Stackelberg, 1934; Conitzer and Sandholm, 2006) the introduces additional assumptions. Players have specific roles in this case, one player (also called the *leader*) commits herself to a publicly known mixed strategy and then the second player (or the *follower*) plays the best response to this strategy. The strategy is in Stackelberg equilibrium, when the strategies are as described and the expected outcome is maximal for the leader. This assumption introduces a new dynamic into the game, since the leader announces which (mixed) strategy she is about to play and the follower can perfectly react to this strategy. Although counterintuitive, leader can not be worse off by following strategy from Stackelberg equilibrium, since she can always commit herself to a strategy prescribed by Nash equilibrium. However, these two solution concepts coincide in zero-sum games and the expected utility of the leader in Stackelberg equilibrium equals to the value of the game $\mathcal{V}^*$.

General-Sum Games By restricting our work to zero-sum games we avoid number of complications that arise in general-sum games. Many of the following issues make real-world applications of general-sum models more difficult. As we will describe in the next chapter, computing NE is more complex for general-sum games than for zero-sum games. Moreover, there may be (infinitely) many equilibria in a single game and each of these equilibria can have a different expected payoff for a player in general-sum games. Finding an equilibrium that maximizes a payoff for certain player is computationally even more difficult (NP-hard (Gilboa and Zemel, 1989; Conitzer and Sandholm, 2008)). Finally, there

is a well-known equilibrium selection problem apparent in general-sum games stating that if a player plays a strategy from one Nash equilibrium, and the opponent plays a strategy from a different Nash equilibrium, the result may be arbitrarily bad for both agents. Therefore, NE strategies do not give any guarantees for expected outcome in general-sum games.

Some of these complications can be addressed by using Stackelberg equilibria (SE). First of all, SE can be computed in polynomial time even for general-sum normal-form games (Conitzer and Sandholm, 2006). Secondly, since strategy of the leader is publicly known, the problem with equilibrium selection is reduced to deterministically decide which pure best response the follower plays in case there are multiple pure best-response strategies. This problem is typically addressed by using Strong Stackelberg equilibrium, where the follower favors the leader (e.g., see (Yin et al., 2010)). However, correctness of these assumptions in practice must be guaranteed and robustness against possibly imperfect utility of the follower is often analyzed (e.g., in (Pita et al., 2010)). Furthermore, we are interested in sequential games, where computing SE is harder than NE due to exponentially many pure strategies (it is NP-hard to compute SE in extensive-form games (Letchford and Conitzer, 2010)).

2.2 Extensive-Form Games

The normal-form representation described in the previous section is generic enough for capturing any game with finite number of strategies. The disadvantage is that the representation can be rather inefficient and the matrix representing the game rather large. This is particularly apparent in sequential games, where players are able to observe effects of the performed actions and react to these information.

Therefore, more compact representations were designed to model the games with sequential interactions. For sequential games with finite number of steps of the game, *the extensive form* is typically used. Games in the extensive form are often visualized as game trees (see Figure 2.2). Nodes in the game tree represent states of the game; each state of the game corresponds to a history of moves executed by all players in the game. Each node is assigned to a player that acts in the game state associated with this node. An edge in the game tree outgoing from a node corresponds to an action that can be performed by the player acting in this node. Extensive-form games model limited observations of the players by grouping the nodes into *information sets*, so that one player cannot distinguish between nodes that belong to the same information set. The model also represents uncertainty about the environment and stochastic events by using a special *Nature player* that acts according to the fixed probability distribution known to all players. This representation is exponentially smaller compared to the normal-form representation and thus allows us to analyze and solve larger instances.

2.2.1 Definitions

We extend the definition from the previous sections by defining states of the game. Formally, a two-player extensive-form game G is defined as a tuple $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, \rho, u, \mathcal{C}, \mathcal{I})$,

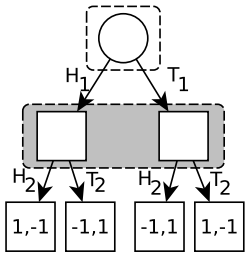


Figure 2.2: Matching Pennies game from Figure 2.1 depicted as an extensive-form game.

where $\mathcal{N} = \{1, 2\}$ is again the set of two players. $\mathcal{H}$ denotes a finite set of *nodes* in the game tree. Each node corresponds to a unique *history* of actions taken by all players and Nature from the root of the game; hence, we use the terms history and node interchangeably. We denote by $\mathcal{Z} \subseteq \mathcal{H}$ the set of all *terminal nodes* of the game. $\mathcal{A}$ denotes the set of all actions and we overload the notation and use $\mathcal{A}(h) \subseteq \mathcal{A}$ to represent the set of actions available to the player acting in node $h \in \mathcal{H}$. We specify $ha = h' \in \mathcal{H}$ to be node h' reached from node h by performing action $a \in \mathcal{A}(h)$. We say that h is a *prefix* of h' and denote it by $h \sqsubseteq h'$. Utility value is defined for every terminal node $z \in \mathcal{Z}$ and we again use $u_i : \mathcal{Z} \to \mathbb{R}$ to denote the utility value for player i

The function $\rho : \mathcal{H} \setminus \mathcal{Z} \to \mathcal{N} \cup \{c\}$ assigns each node to a player who takes an action in the node, where c means that the Nature player selects an action in the node based on a fixed probability distribution known to all players. We use function $\mathcal{C} : \mathcal{H} \to [0, 1]$ to denote the probability of reaching node h due to Nature (i.e., when both players play actions that lead to node h); the value of $\mathcal{C}(h)$ is equal to the product of the probabilities assigned to all actions of the Nature player in history h. Imperfect observation of player i is modeled via *information sets* $\mathcal{I}_i$ that form a partition over the nodes assigned to player i $\{h \in \mathcal{H} \mid \rho(h) = i\}$. Every information set contains at least one node and each node belongs to exactly one information set. Nodes in an information set of a player are not distinguishable to the player. All nodes h in a single information set $I_i \in \mathcal{I}_i$ have the same set of possible actions $\mathcal{A}(h)$; hence, an action a from $\mathcal{A}(h)$ uniquely identifies information set I_i and there cannot exist any other node $h' \in \mathcal{H}$ that does not belong to information set I_i and for which a is allowed to be played (i.e., $a \in \mathcal{A}(h')$). Therefore we overload notation and use $\mathcal{A}(I_i)$ to denote the set of actions defined for each node h in this information set. We assume *perfect recall*, which means that players perfectly remember their own actions and all information gained during the course of the game. As a consequence, all nodes in any information set I_i have the same history of actions for player i.

Finally, we define a *sub-game* of a game G induced by a node $h \in \mathcal{H}$. A sub-game is defined as a subset of nodes, actions, and information sets of the original game G such that node h is a (not necessarily direct) predecessor of each node in the sub-game (i.e., in the history of any node in the sub-game there must exist an action taken in node h). For every information set it must hold that it is either fully included in the sub-game (i.e., all nodes from this information set are in the sub-game), or fully disjoint (i.e., none of the nodes from this information set is in the sub-game). This definition is particularly useful

for perfect-information games, or simultaneous-move games, where sub-games correspond to sub-trees of the game tree. However, for generic imperfect-information extensive-form games, the concept of sub-game is not particularly useful since the assumption of the complete involvement (or complete exclusion) of information sets is too strong.

2.2.2 Strategies in Extensive-form Games

Any extensive-form game can also be represented and visualized as a normal-form game. An example of such a transformation is the game of Matching Pennies depicted in Figure 2.1 in the normal form and in Figure 2.2 in the extensive form. We can see that the circle player plays first and chooses either heads (H_1) or tails (T_1), following by two states that are not distinguishable to the box player and they belong to the same information set. After the box player chooses her action, the game terminates with the same outcome as defined before.

For extensive-form games we can define similar concepts of strategies that are used for normal-form games. A pure strategy π_i of player i in an extensive-form game is an assignment of which action to play in every possible situation that can occur during the course of the game. Formally, this corresponds to an assignment of an action from set $\mathcal{A}_i(I_i)$ for each information set $I_i \in \mathcal{I}_i$ of player i. Using the definition of pure strategies we can represent an extensive-form game as a matrix game, where rows correspond to pure strategies of one player, and columns to the strategies of the opponent. Since pure strategies prescribe actions for every information set, the outcome of the game corresponds to the utility value of the leaf reached by sequentially playing the actions in pure strategies, and it is well defined.

This definition of pure strategies in sequential games causes the combinatorial explosion and makes the final matrix exponentially large in the size of the game tree. Therefore, even though we can proceed by using the same definition of mixed strategies as in standard normal-form games (i.e., the probability distribution over pure strategies), this approach is not scalable and therefore used only for small games. Alternatively we can use a more natural representation of strategies in extensive-form games – in each information set assigned to player i there is a probability distribution over the actions applicable in this situation. This representation is called *behavioral strategies*. Its significance is particular in games with perfect recall, since the behavioral strategies can be transformed to mixed strategies and vice versa. On the other hand, behavioral strategies are also sought in games of imperfect recall since they better correspond to the decision process of the agent that can observe current information set and needs to make a decision, although they are less expressive than mixed strategies in this case. Finally, behavioral strategies can be compactly represented by using notion of sequences in games with perfect recall (see Section 3.2.1).

2.2.3 Nash Equilibrium (NE) in Extensive-Form Games

Using the mixed strategies in extensive-form games, the definition of the NE solution concept is the same as for normal-form games. However, strategies prescribed by NE in extensive-form games do not always correspond to behavior of players that can be considered rational. There are two main issues arising with NE strategies in extensive-form

games. For general-sum games, players may use so called *non-credible threats* to enforce some behavior to the opponent. A non-credible threat is considered a strategy that prescribes to play an action $a_i \in \mathcal{A}_i(I_i)$ in information set I_i even though there exists a different action $a'_i \in \mathcal{A}_i(I_i)$ applicable in this situation yielding a higher utility value player i. However, by choosing this "worse" action a_i, player i can also decrease the utility of the opponent, and thus force the opponent to avoid this information set by playing differently in the earlier stages of the games. This behavior is called non-credible, because once the game actually reaches information set I_i during the game-play, it is rational for player i to play action a'_i that yields a higher utility value instead of a_i.

Non-credible threats are not present in zero-sum games and it holds that NE strategies guarantee the value of the game (i.e., by following Nash strategies the player cannot achieve lower expected utility than $\mathcal{V}$). However, Nash strategies cannot exploit the mistakes that the opponent can make during the course of the game. Nash equilibrium assumes both players to act rationally; hence, strategies selected for the parts of the game tree that should not be reachable when both players follow NE strategies can be irrational. The primary reason is that NE solution concept does not expect this part of the game to be played and cannot restrict strategies in these information sets in any way. To overcome these drawbacks, a number of refinements of NE have been introduced imposing further restrictions with an intention to describe more sensible strategies. Examples include *subgame-perfect equilibrium* (Selten, 1965) used in perfect-information EFGs. The subgame-perfect equilibrium forces the strategy profile to be a Nash equilibrium in each sub-game of the original game. Since sub-games are not particularly useful in imperfect-information EFGs, there are other refinements including *strategic-from perfect equilibrium* (Selten, 1975), *sequential equilibrium* (Kreps and Wilson, 1982), or *quasi-perfect equilibrium* (van Damme, 1984; Miltersen and Sørensen, 2010). The first refinement avoids using weakly dominated strategies in equilibrium strategies for two-player games ((van Damme, 1991), p. 29) and it is also known as the *undominated equilibrium*. Sequential equilibrium tries to exploit the mistakes of the opponent by using the notion of beliefs consistent with the strategy of the opponent even in information sets off the equilibrium path. The main intuitions behind the first two refinements are then combined in quasi-perfect equilibrium.

In the remaining of this thesis we primarily focus on Nash equilibrium and algorithms for computing NE strategies. Even though the solution described by NE does not always prescribe rational strategies off the equilibrium path, it is still valuable to compute exact NE of large extensive-form games since they provide guarantees on the expected outcome. Moreover, a refined equilibrium is still a NE and calculating the value of the game is often a starting point for many of the algorithms that compute these refinements (e.g., it is used for computing undominated equilibrium (e.g. see (Ganzfried and Sandholm, 2013; Cermak et al., 2014), or normal-form proper equilibrium (Miltersen and Sørensen, 2008)).

Chapter 3

Computing Nash Equilibrium

This chapter describes the state of the art in algorithms for computing Nash equilibrium (NE) in zero-sum games. There is a large number of different algorithms, however, this thesis focuses on exact algorithms. Therefore, this chapter primarily describes the exact algorithms that are typically based on mathematical programming, where the problem of finding an equilibrium strategies is formulated as an optimization mathematical problem. First, we describe algorithms for solving normal-form games, following by the specific algorithms for extensive-form games. Afterwards we briefly highlight the best-known approximative algorithms for extensive-form games and finally we describe the iterative framework of double-oracle algorithms that would be further applied in sequential games.

3.1 Solving Normal-Form Games

Problem of finding NE strategies is known to be hard in general. It was shown (Daskalakis et al., 2006; Chen and Deng, 2006) that the computational complexity of computing equilibrium strategies is PPAD-complete for two-player general-sum finite games (PPAD stands for "polynomial parity argument, directed version"). Very informally speaking, this class of problems corresponds to searching for a specific node in a graph of exponential size that does not need to be explicitly represented in memory and functions that provide predecessors and successors are polynomial. In contrast to the well-known class of NP-complete problems, the solution is guaranteed to exist for problems in PPAD class.

Solving a zero-sum normal-form game is known to be a polynomial problem in the size of the game. This is mainly due to minimax theorem by von Neumann (von Neumann and Morgenstern, 1947). This theorem states that the expected values of strategies that maximize the worst-case payoff of player and strategies that minimize the best-case payoff are equal:

$$\max_{\delta_i \in \Delta_i} \min_{\delta_{-i} \in \Delta_{-i}} u_i(\delta_i, \delta_{-i}) = \min_{\delta_i \in \Delta_i} \max_{\delta_{-i} \in \Delta_{-i}} u_{-i}(\delta_i, \delta_{-i}) = \mathcal{V}^* \tag{3.1}$$

Exploiting this equation, the problem of solving a zero-sum game can be formulated as a linear program as follows:

$$\max_{x_{\pi_i} \in \mathbb{R}} \mathcal{V}^* \tag{3.2}$$

$$\sum_{\pi_i \in \Pi_i} x_{\pi_i} u_i(\pi_i, \pi_{-i}) \geq \mathcal{V}^* \qquad \forall \pi_{-i} \in \Pi_{-i} \tag{3.3}$$

$$1 \geq x_{\pi_i} \geq 0 \qquad \forall \pi_i \in \Pi_i \tag{3.4}$$

$$\sum_{\pi_i \in \Pi_i} x_{\pi_i} = 1 \tag{3.5}$$

where $x_{\pi_i} \in \mathbb{R}$ are variables of the linear program representing the probability of playing pure strategy π_i. The semantics of the constraints are following. While the first player is maximizing the expected value of the game in the objective (Equation 3.2), the opponent is minimizing this value by selecting such a pure strategy π_{-i} of all her pure strategies (Equation 3.3). Finally, we need to constrain the variables x to form a valid probability distribution (Equations 3.4-3.5).

Example Consider a game of Matching Pennies depicted in Figure 2.1. Linear program constructed for this game:

$$\begin{aligned}
\max_{x_{H_1}, x_{T_1}} \quad & \mathcal{V}^* \\
x_{H_1} \cdot 1 + x_{T_1} \cdot (-1) &\geq \mathcal{V}^* \\
x_{H_1} \cdot (-1) + x_{T_1} \cdot 1 &\geq \mathcal{V}^* \\
x_{H_1} + x_{T_1} &= 1 \\
1 \geq x_{H_1} &\geq 0 \\
1 \geq x_{T_1} &\geq 0
\end{aligned}$$

Solving this program yields value of the game $\mathcal{V}^* = 0$ and strategy $x_{H_1} = x_{T_1} = 0.5$.

3.2 Solving Extensive-Form Games

One approach to solving extensive-form games is by transforming them to normal-form games and solve in this representation. However, since the transformation is exponential (i.e., the created normal-form game is exponentially larger than the extensive-form game), the algorithms that specifically exploit the sequential structure of extensive-form games are more practical. The state-of-the-art exact algorithm is based on linear programming and exploits so-called *sequence-form representation* (Koller et al., 1996; von Stengel, 1996) of the strategies that allows to reason about behavioral strategies using linear constraints and polynomial number of variables.

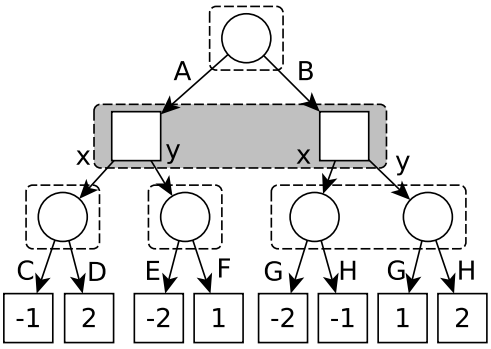


Figure 3.1: Example of an extensive-form game depicted as a game tree. Dashed boxes visualize the information sets. Utility values are for the circle player, the box player minimizes the value.

3.2.1 Sequence-Form Representations of Strategies

In an extensive-form game G, a *sequence* σ_i is an ordered list of actions of player i in a history $h \in \mathcal{H}$. Number of these actions (i.e., the length of sequence σ_i) is denoted as $|\sigma_i|$, empty sequence (i.e., a sequence with no actions) is denoted as $\emptyset$. The set of all possible sequences in a game for player i is denoted by Σ_i and the set of sequences for all players is $\Sigma = \Sigma_1 \times \Sigma_2$. A sequence $\sigma_i \in \Sigma_i$ can be extended by a single action a taken by player i, denoted as $\sigma_i a = \sigma_i'$ (we again use $\sigma_i \sqsubseteq \sigma_i'$ to denote that σ_i is a prefix of σ_i'). Due to perfect recall, all nodes in an information set I_i share the same sequence of actions of player i and we use $\mathsf{seq}_i(I_i)$ to denote this sequence. We overload the notation and also use $\mathsf{seq}_i(h)$ to denote the sequence of actions of player i leading to node h, and $\mathsf{seq}_i(\mathcal{H}') \subseteq \Sigma_i$, where $\mathsf{seq}_i(\mathcal{H}') = \bigcup_{h' \in \mathcal{H}'} \mathsf{seq}_i(h')$ for some $\mathcal{H}' \subseteq \mathcal{H}$. Since action a uniquely identifies information set I_i and all nodes in an information set share the same history, then each sequence uniquely identifies an information set – we use function $\mathsf{inf}_i(\sigma_i')$ to denote the information set in which the last action of sequence σ_i' (i.e., a in this case) is defined. For an empty sequence, function $\mathsf{inf}_i(\emptyset)$ denotes the information set of the root node.

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

$$g_i(\sigma_i, \sigma_{-i}) = \sum_{h \in \mathcal{Z} \,:\, \forall j \in \mathcal{N}\ \sigma_j = \mathsf{seq}_j(h)} u_i(h) \cdot \mathcal{C}(h) \tag{3.6}$$

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

Example Consider the example game depicted in Figure 3.1. The following table lists the sequences available in this game:

player i	set of sequences Σ_i
◯	$\emptyset, A, B, AC, AD, AE, AF, BG, BH$
□	$\emptyset, x, y$

Note that the sequences of not maximal length belong to set of sequences (e.g., sequence $A \in \Sigma_{\bigcirc}$) and that empty sequences also belong to the sets of all sequences.

3.2.2 Sequence-Form Linear Program

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

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

$$v_{\mathsf{inf}_{-i}(\sigma-i)} - \sum_{I'_{-i} \in \mathcal{I}_{-i} : \mathsf{seq}_{-i}(I'_{-i}) = \sigma_{-i}} v_{I'_{-i}} \leq \sum_{\sigma_i \in \Sigma_i} g_i(\sigma_{-i}, \sigma_i) \cdot r_i(\sigma_i) \qquad \forall \sigma_{-i} \in \Sigma_{-i} \tag{3.7}$$

$$r_i(\emptyset) = 1 \tag{3.8}$$

$$\sum_{\forall a \in A(I_i)} r_i(\sigma_i a) = r_i(\sigma_i) \qquad \forall I_i \in \mathcal{I}_i,\ \sigma_i = \mathsf{seq}_i(I_i) \tag{3.9}$$

$$1 \geq r_i(\sigma_i) \geq 0 \qquad \forall \sigma_i \in \Sigma_i \tag{3.10}$$

Solving this LP yields a realization-plan strategy for player i using variables r_i, and expected values for the information sets of player $-i$ (variables $v_{I_{-i}}$). The LP works as follows: player i maximizes the expected value in information sets by selecting the values for the variables of realization plan that is constrained by equations (3.8–3.10). The probability of playing the empty sequence is defined to be 1 (Equation (3.8)), and the probability of playing a sequence σ_i is equal to the sum of the probabilities of playing sequences extended by exactly one action (Equation (3.9)). Finding such a realization plan is also constrained by the best responding opponent, player $-i$. This is ensured by Equation (3.7), where player $-i$ selects in each information set I_{-i} such action that minimizes the expected utility value in this information set $v_{I_{-i}}$. There is one constraint defined for each sequence σ_{-i}, where the last action of this sequence determines the best action to be played in information

set $\mathsf{inf}_{-i}(\sigma_{-i})$. The expected utility is composed of the expected utilities of the information sets reachable after playing sequence σ_{-i} (sum of v variables on the left side) and of the expected utilities of leafs to which this sequence leads (sum of g values on the right side of the constraint). The value in the information set of the root $v_{\mathsf{inf}_{-i}(\emptyset)}$ is equal to the value of the game $\mathcal{V}^*$.

Example Consider again our extensive-form game from Figure 3.1 as an example. We construct a sequence-form linear program that computes Nash equilibrium for this game:

$$\begin{aligned}
&\max_{r,v} v_{\mathsf{inf}_\square(\emptyset)} \\
v_{\mathsf{inf}_\square(\emptyset)} - v_{\mathsf{inf}_\square(x)} &\leq 0 \\
v_{\mathsf{inf}_\square(x)} &\leq -r_\bigcirc(AC) + 2 \cdot r_\bigcirc(AD) - 2 \cdot r_\bigcirc(BG) - r_\bigcirc(BH) \\
v_{\mathsf{inf}_\square(y)} &\leq -2 \cdot r_\bigcirc(AE) + r_\bigcirc(AF) + r_\bigcirc(BG) + 2 \cdot r_\bigcirc(BH) \\
r_\bigcirc(\emptyset) &= 1 \\
r_\bigcirc(A) + r_\bigcirc(B) &= r_\bigcirc(\emptyset) \\
r_\bigcirc(AC) + r_\bigcirc(AD) &= r_\bigcirc(A) \\
r_\bigcirc(AE) + r_\bigcirc(AF) &= r_\bigcirc(A) \\
r_\bigcirc(BG) + r_\bigcirc(BH) &= r_\bigcirc(B)
\end{aligned}$$

The objective function is maximizing the expected utility in the root information set $v_{\mathsf{inf}_\square(\emptyset)}$. Optimal value in this set equals to the value of the game. Following, there are 3 constraints corresponding to the sequences of the box player. These 3 constraints are instantiation of constraint (3.7). First constraint is associated with the empty sequence, second one with sequence x, third one with sequence y. The left side of the constraints consider the value of the information set $\mathsf{inf}_\square(\sigma_\square)$ and all information set this sequence leads to (note the constraint for the empty sequence that leads to information set, where actions x and y are applicable). Note that $\mathsf{inf}_\square(x) = \mathsf{inf}_\square(y)$ since both actions are applicable in the same information set. The right side of the constraints consider the immediate leafs that can be reached after playing the sequence. This is apparent in the latter two constraints, where utility values of leaves reached either by playing x or y are listed. Finally, there are 5 equations constraining the realization plan of the circle player.

Solving this linear program we calculate value of the game ($\mathcal{V}^* = 1.25$) and equilibrium realization plan for the circle player. We can calculate strategy for the box player by constructing similar program, where variables r correspond to sequences x and y.

3.3 Approximating Nash Equilibrium

Solving the linear program for extensive-form games has a limited scalability since the size of the program is linear in the size of the game tree that grows exponentially with the number of sequential actions in the game. A common practice for solving very large games is thus to use an approximation method. The best known approximate algorithms include counterfactual regret minimization (CFR) (Zinkevich et al., 2008), improved versions of CFR with

sampling methods (Lanctot et al., 2009; Gibson et al., 2012); Nesterov's Excessive Gap Technique (EGT) (Hoda et al., 2010); and variants of Monte Carlo tree search algorithms applied to imperfect-information games (e.g., see (Ponsen et al., 2011; Lisy et al., 2012a; Lanctot et al., 2014; Lisy et al., 2014)).

Approximative algorithms typically use the concept of *regret* that expresses the expected loss of the algorithm for choosing some strategy instead of choosing the best possible strategy. This is an alternative reformulation of seeking the equilibrium strategies in any game representation: instead of seeking for a pair of equilibrium strategies, approximative algorithms seek for strategies that minimize the regret.

3.3.1 Counterfactual Regret Minimization

The family of Counterfactual Regret Minimization (CFR) algorithms (Zinkevich et al., 2008) is based on learning methods that can be informally described as follows. The algorithm repeatedly traverses the game tree and in each information set applies a no-regret learning rule that minimizes a specific local variant of regret (termed *counterfactual regret*) in each iteration. The authors prove that by minimizing local counterfactual regret in each information set, the regret of the strategy in the game is minimized as well.

CFR algorithms are well established in the community and many different variants of this algorithm have been introduced over the years. The improvements of CFR algorithm focus primarily on the methods for selecting information set to be updated using the learning rule. In standard CFR, all information sets are updated in each iteration. In the sampling versions, on the other hand, the algorithm selects only a subset of information set updated during a single iteration of the algorithm that correspond to a single branch of the tree (Lanctot et al., 2009; Gibson et al., 2012). Finally, the latest variants (termed Online Outcome Sampling) also iteratively construct the game tree similarly to the Monte Carlo tree search (see next section) (Lanctot et al., 2013a; Lanctot et al., 2014; Bosansky et al., 2014c). The flexibility of CFR algorithms is also apparent in algorithms like CFR-BR (Johanson et al., 2012), where the algorithm is learning the best strategy on an abstracted game, while the regret is calculated against a best-responding opponent that uses a non-abstracted game. Variant of a CFR is also one of the few algorithms used in practice to solve some of the EFGs with imperfect recall (Lanctot et al., 2012).

However, the algorithm operates on the whole game tree and requires convergence in all information sets. This results in large memory requirements (all information sets need to be stored in memory) and, as we will see in Chapter 5, slow convergence when seeking strategies with very small error.

3.3.2 Excessive Gap Technique

Another fast approximative method is Excessive Gap Technique (EGT). EGT exploits the convex properties of the sequence-form representation and uses recent mathematical results on finding extreme points of smooth functions (see (Hoda et al., 2010) for the details). The main idea is to approximate the problem of finding a pair of equilibrium strategies by two smoothed functions (i.e., Equation 3.1). Since these smoothed functions are convex

and differentiable, the algorithm uses gradients to guide these functions to approximate the solution. Although this approach achieves faster convergence in comparison with CFR, exploitation of the mathematical foundations make the algorithm less robust (it is not known whether a similar approach can be done for more generic classes of games) and it is less used in practice (e.g., in Poker competition).

3.3.3 Monte Carlo Tree Search

Monte Carlo tree search (MCTS) is a simulation-based sampling algorithm (Kocsis and Szepesvári, 2006) that incrementally constructs the game tree starting from the root node. In each iteration MCTS algorithm executes four methods in the following order: (1) *Selection* starts in the root note and descends down the already constructed part of the game tree based on the statistical information about the actions that estimate their quality; (2) *Expansion* is performed after reaching a leaf of the constructed part of the game tree and at least one succeeding node of the current node is added into the constructed part of the game tree; (3) The value of added node is estimated using a *simulation* of the play using random strategy until a terminal state is reached; (4) During the *backpropagation* phase the algorithm updates the statistics for actions selected in the previous steps leading to the node selected in step 3.

This generic template can be instantiated with different functions and algorithms for selecting actions during the first step. The first and best-known variant of MCTS (Kocsis and Szepesvári, 2006) uses Upper Confidence Bounds (UCB) selection function and the algorithm is called UCT (UCB applied on trees). UCB selection function optimizes the problem of exploration and exploitation in perfect information settings. However, the situation is more complex in generic imperfect information EFGs. As demonstrated in (Shafiei et al., 2009), UCT applied on simultaneous-move games can converge to an incorrect solution. Therefore, other variants of selection function in MCTS are considered for imperfect information games: Exp3 (Auer et al., 2003) optimizes exploration/exploitation problem in stochastic and adversarial setting, or simple regret matching (Hart and Mas-Colell, 2000) can be applied. However, even with different selection functions MCTS algorithms do not in general guarantee finding an (approximate) optimal solution in imperfect-information games. One exception is the recent proof of convergence of MCTS with certain selection methods for simultaneous-move games (Lisy et al., 2013).

In spite of the negative theoretic results, using MCTS (and UCT) is very common since MCTS in general does not require heuristic domain-specific knowledge and has much lower memory requirements as it does not build the complete game tree in memory. Therefore MCTS is also used for game playing of imperfect-information games where it can produce good strategies in practice (e.g., see (Ciancarini and Favini, 2010; Lisy et al., 2012a; Lisy et al., 2012b; Lisy et al., 2014)).

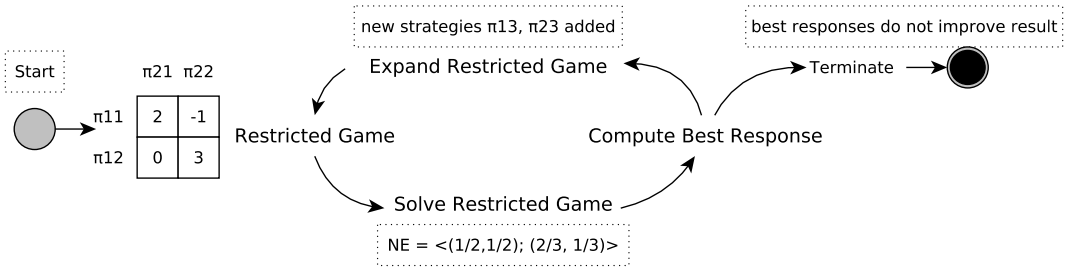


Figure 3.2: Schemata of the double-oracle method.

3.4 Double-Oracle Algorithm for Normal-Form Games

Now we turn to an alternative approach for addressing large linear programs. In decision theory, solving large-scale optimization problems can be improved by constraint/column generation techniques (Dantzig and Wolfe, 1960; Barnhart et al., 1998). In game theory, this principle has been adopted as *oracle algorithms* introduced in (McMahan et al., 2003). This work describes the double-oracle algorithm, proves convergence to a Nash equilibrium, and experimentally verifies that the algorithm achieves computation time improvements on a search game where an evader is trying to cross an environment without being detected by sensors placed by the opponent. The double-oracle algorithm reduced the computation time from several hours to tens of seconds and allowed for solving much larger instances of this game. Similar success with the double-oracle methods has been demonstrated on a variety of different domains inspired by pursuit-evasion games (Halvorson et al., 2009) and security games played on a graph (Jain et al., 2011; Letchford and Vorobeychik, 2013).

Figure 3.2 shows a visualization of the main structure of the algorithm. The algorithm repeats the following three steps until convergence:

1. create a restricted game by limiting the set of pure strategies that each player is allowed to play

2. compute a pair of Nash equilibrium strategies in this restricted game

3. for each player, compute a pure best response strategy against the equilibrium strategy of the opponent, which may be *any* pure strategy in the original unrestricted game

The best response strategies computed in step 3 are added to the restricted game, the game matrix is expanded by adding new rows and columns, and the algorithm 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, players play best response strategies to the strategy of the opponent. The algorithm maintains the values of expected utilities of the best-response strategies throughout the iterations of the algorithm. These values provide bounds on the value of the original unrestricted game $\mathcal{V}^*$ — from the perspective of a player i, the minimal value from all her past best-response calculations represents an upper bound of the value of the original game, $\mathcal{V}_i^{UB}$, and maximal value from

all past best-response calculations of the opponent represents the lower bound on the value of the original game, $\mathcal{V}_i^{LB}$. Note that for the bounds it holds that the lower bound for player i equals to the negative value of the upper bound for the opponent:

$$\mathcal{V}_i^{LB} = -\mathcal{V}_{-i}^{UB}$$

In general, computing best responses is typically computationally less demanding than solving the game, since the problem is reduced to a single-player optimization (Wilson, 1972). Since best-response algorithms can operate very quickly (e.g., also by exploiting additional domain-specific knowledge), they are called *oracles* in this context. If the algorithm incrementally adds strategies only for one player, the algorithm is called a *single-oracle algorithm*, if the algorithm incrementally adds the strategies for both players, the algorithm is called a *double-oracle algorithm*. Double-oracle algorithms are typically initialized with an arbitrary pair of strategies (one pure strategy for each player). However, a larger set of initial strategies selected with a domain-specific knowledge can also be used.

Double-oracle algorithm runs in polynomial time in the size of the game matrix since there is a linear number of iterations (at least one new pure strategy is added in each iteration), and each iteration takes polynomial time (solving the LP of the restricted game and calculating the best responses). In the worst case, the algorithm adds all pure strategies and solves the original game, however, this is rarely the case in practice. Unfortunately, there are no known estimations on the average number of iterations needed for the double-oracle algorithm to converge.

Towards Extensive-Form Games A straightforward method of using the double-oracle algorithm for any sequential game is to use pure strategies (i.e., assignments of action for every possible situation) and use the very same algorithm described in this section; hence, iteratively add strategies from the unrestricted extensive-form game into the restricted game matrix. However, this approach still suffers from the exponential transformation of sequential (and extensive-form) games into normal-form games; hence, the restricted game can be exponential in size of the extensive-form game and there can be exponentially many iterations of the double-oracle algorithm. In the remaining of this thesis we present algorithms that exploit the key ideas of the double-oracle framework, however, do not use this transformation. Each of the newly designed algorithm is designed specifically for a subclass of extensive-form games which allows them to achieve significant improvements in computation time.

Chapter 4

Double-Oracle and Alpha-Beta in Simultaneous-Move Games

This chapter focuses on a specific subclass of extensive-form games (EFGs) with imperfect information called *perfect-information simultaneous-move games*. Simultaneous-move EFGs is the simplest class of imperfect-information games, where both players can perfectly observe the current state of the game, however in this state of the game both players act simultaneously. Class of simultaneous-move games is very important and many real-world scenarios can be either directly modeled as simultaneous-move games, or approximated by this class of games. Examples include pursuit-evasion scenarios, where one player tries to capture the opponent (e.g., in (Vieira et al., 2009)), or close combat between multiple units (Kovarsky and Buro, 2005; Churchill et al., 2012), both applicable in security scenarios. Moreover, the class of simultaneous-move games became popular among game search community, including card games like Oshi-Zumo (Buro, 2003), Goofspiel (Ross, 1971; Saffidine et al., 2012; Rhoads and Bartholdi, 2012; Lanctot et al., 2013a), PacMan (Nguyen and Thawonmas, 2013; Pepels et al., 2014), or Tron (Samothrakis et al., 2010; Lanctot et al., 2013b). Finally, simultaneous-move games are the formal model for games in general-game playing community (Genesereth et al., 2005).

In this chapter we first give the formal definitions specific for simultaneous-move games following by more detailed description of the algorithms. Afterwards, we describe our novel algorithm DO$\alpha\beta$ that enhance the classical backward induction with double-oracle framework and fast calculation of bounds for values of the sub-games. Finally, we demonstrate the computation speed-up in practical experiments.

4.1 Definitions

For reasoning about simultaneous-move games we primarily use the formalism of extensive-form games; hence, we use the same symbols for denoting nodes $\mathcal{H}$, actions $\mathcal{A}$, players $\mathcal{N}$, and utility function u. We formalize the games always from perspective of one of the players (say i), assuming this player moves first (i.e., plays action $a \in \mathcal{A}_i(h)$) in the state of the

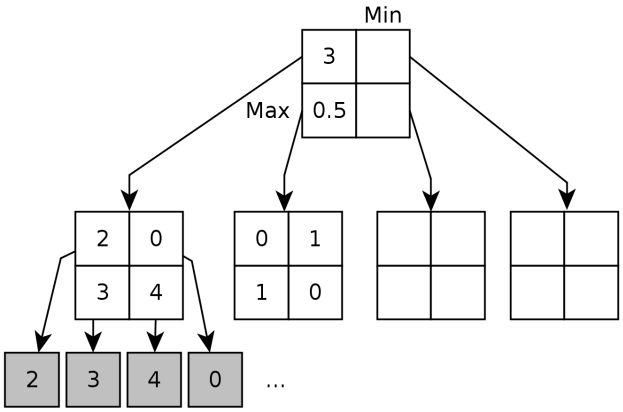


Figure 4.1: Example of a two-player zero-sum simultaneous-move game without chance nodes. In each stage of the game both players act simultaneously. Utility values of the row player are in the leafs, the values in the stage matrices correspond to the values of the sub-games as calculated by backward induction.

game $h \in \mathcal{H}$, then the opponent (player $-i$) moves. Note that player $-i$ cannot observe the first move of player i; hence, all nodes achievable by playing some action from $\mathcal{A}_i(h)$ belong to the same information set I_i:

$$\forall h' \in I_i \; \exists a \in \mathcal{A}_i(h) \; : h' = ha$$

and

$$\forall a \in \mathcal{A}_i(h) \; : ha = h' \in I_i$$

However, since both players are acting simultaneously, we can use an alternative representation where this simultaneous act is represented as a normal-form (i.e., a matrix) game. We term the situation where both players act simultaneously as *stage*, but still denoted as $\mathcal{H}$. Using stages simplifies the notation because we can assume that both players have defined actions in $h \in \mathcal{H}$ and playing a joint action (i.e., a pair of actions; one for each player) leads to another stage of the game, or a terminal state. This representation can be visualized as tree that has a matrix game in each node of the tree, and each edge corresponds to a joint action of both players (see Figure 4.1). We use M to refer to the stage matrix game, where M_{ab} refers to the value of sub-game reachable when first player chooses $a \in \mathcal{A}_1(h)$ and the second player chooses $b \in \mathcal{A}_2(h)$. We allow stochastic events to be part of our model; hence, an inner node can be a chance node. For brevity, we use function $\mathcal{T} : \mathcal{H} \times \mathcal{A} \to [0, 1]$ to refer to the transition probabilities due to Nature (i.e., what is the probability of an action a selected in chance node h):

$$\mathcal{T}(h, a) = \frac{\mathcal{C}(ha)}{\mathcal{C}(h)}$$

Algorithm 1 Backward Induction for simultaneous-move games.

Require: $h \in \mathcal{H}$ – current stage of the game; i – searching player

1: **if** $h \in \mathcal{Z}$ **then**
2: **return** $u_i(h)$
3: **end if**
4: **for** $a \in \mathcal{A}_i(h)$ **do**
5: **for** $b \in \mathcal{A}_{-i}(h)$ **do**
6: $h' \leftarrow hab$
7: **if** $\rho(h') = c$ **then**
8: $M_{ab} \leftarrow 0$
9: **for** $a_n \in \mathcal{A}_c(h')$ **do**
10: $M_{ab} \leftarrow M_{ab} + \mathcal{T}(h', a_n) \cdot \text{BI}(h'a_n, i)$
11: **end for**
12: **else**
13: $M_{ab} \leftarrow \text{BI}(h', i)$
14: **end if**
15: **end for**
16: **end for**
17: $v_h \leftarrow$ solve matrix game M
18: **return** v_h

4.2 Related Work

The baseline algorithm for solving zero-sum simultaneous-moves EFGs is the *backward-induction* algorithm (e.g., see (Ross, 1971; Buro, 2003)). The algorithm searches through the game tree in the depth-first manner, in each stage it computes the values of all the succeeding sub-games, and afterwards it solves the stage matrix game (i.e., computes a NE of the matrix game in the current stage of the game). Finally, the value is propagated to the predecessor. The result of the backward-induction algorithm is the value of the game and strategies in *subgame-perfect Nash equilibrium*. Since our novel algorithms further build on the backward induction, we describe backward induction using pseudocode in Algorithm 1. Note that in case the succeeding node ($h' = hab$) of the current node h is a chance node (line 7), all actions of Nature are evaluated and the expected utility value is calculated (line 10) by weighting the utility value with the probability of Nature player choosing the action a_n. Otherwise, the value of the sub-game rooted in the succeeding node h' is simply calculated using the recursive call (line 13). After evaluating all the actions, the matrix game composed of all values of the sub-games is solved using the linear program for normal-form games (see the linear program described in Section 3.1).

4.2.1 Simultaneous-Move Alpha-Beta Search (SMABS)

Existing state-of-the-art pruning method for the backward-induction algorithm for simultaneous-move games was proposed by (Saffidine et al., 2012). 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 stage game. The algorithm keeps bounds on the val-

ues of the sub-game for each successor $h' = ha_ib_j$ in stage h. The lower and upper bounds represent the threshold values, for which neither action a_i, nor action b_j, is dominated by any other action in the current matrix game M. These bounds are calculated by linear programs in the stage h given existing exact values (or appropriate bounds) of the values of all the sub-games succeeding the stage h. If they form an empty interval (the lower bound is higher than the upper bound), a pruning occurs and the dominated action is not considered in this stage any more. This way, the algorithm is able to save computation time by pruning and avoiding sub-games, to which the pruned actions lead.

SMABS outperforms classical backward induction, however, the computation speed-up is only marginal. The reason is that testing for dominance of actions in each stage is a costly operation that does not prune many actions; hence, a significant portion of the game tree is still fully evaluated. The results reported in (Saffidine et al., 2012) state that the algorithm evaluates at best only 29% of nodes. Since our algorithm achieves computation speed-up in several orders of magnitude compared to the backward induction and often evaluates less than 2% of nodes (as we will show in experimental section of this chapter), we do not explicitly use SMABS for comparison.

4.2.2 Approximative Algorithms and Heuristic Approach

Besides the exact methods based on the backward induction, heuristic and approximative algorithms are also used in practice. Even though we discussed in Section 3.3 that Shafiei et al. (2009) showed that straightforwardly using Monte Carlo UCT algorithm does not converge to NE in simultaneous move games, UCT is used in many actual game-playing algorithms. The success of UCT algorithm was demonstrated by success of Cadiaplayer (Finnsson, 2007; Finnsson, 2012) that was the top performing player of the GGP competition between 2007 and 2009. On the other hand, variants with Exp3 selection function, or regret matching selection function, are shown to converge to NE (Lisy et al., 2013).

Alternatively, sampling variants of CFR algorithm are also applicable in simultaneous-move games. Lanctot et al. (2013a) introduced a variant of Monte Carlo algorithm based on the counterfactual regret minimization called Online Outcome Sampling (OOS) for simultaneous-move games. When comparing OOS and MCTS algorithms, MCTS algorithm with regret matching selection function performs much better in online game playing (Lanctot et al., 2013a; Bosansky et al., 2014c).

Finally, several works used heuristic approach without any theoretical guarantees. In one of the first papers on combat scenarios in real-time strategy games, the authors used randomized serialization of the game (Kovarsky and Buro, 2005), or strategy simulation from scripts was used to build a single matrix of values from which an equilibrium strategy was computed using linear programming (Sailer et al., 2007). This search could be extended to multiple nodes where internal nodes would correspond to scripts being interrupted to replan, similarly to (Lisy et al., 2009). In later work, alpha-beta search was run on a serialized (sequential) version of the simultaneous move game for combat scenarios (Churchill et al., 2012). In the next section we also exploit serialization of the game and use classical alpha-beta search on this serialized games, however, we give theoretical guarantees of correctness of our approach.

4.3 Backward Induction with Double-Oracle and Serialized Bounds

This section describes the main algorithm that improves the standard backward-induction algorithm by employing the double-oracle normal-form games in stages of the game. However, to make the algorithm even more efficient, we first introduce concept of serialized bounds and their application when solving simultaneous-move games.

4.3.1 Backward Induction with Serialized Alpha-Beta Bounds

Backward induction algorithm can be easily enhanced by using bounds on the value of the sub-games succeeding the current stage. These bounds can be calculated very quickly by transforming the simultaneous-move game into a perfect information extensive-form game.

Consider a matrix game representing a simultaneous choice of both players. This matrix can be serialized by discarding the notion of information sets; hence, letting one player to play first following by the play of the second player. The crucial difference between a serialized and a simultaneous-move matrix game is that the second player to move has an advantage of knowing what action has been played in a serialized game. Therefore, the value of a serialized game where player i is second to move is an upper bound on the value

Lemma 4.3.1 *Let M be a matrix game with expected value v_h for player i. Let v_h^i be the value of the perfect information game created form the matrix game by letting player $-i$ move first and player i move second with the knowledge of the first player's action selection. Then*

$$v_h \leq v_h^i$$

Proof

$$\begin{aligned} v_h &= \min_{\delta_{-i}^h \in \Delta_{-i}} \max_{\delta_i^h \in \Delta_i} \sum_{a \in \mathcal{A}_i(h)} \sum_{b \in \mathcal{A}_{-i}(h)} \mathcal{P}(a|\delta_i^h)\mathcal{P}(b|\delta_{-i}^h) M_{ab} \\ &= \min_{b \in \mathcal{A}_{-i}(h)} \max_{\delta_i^h \in \Delta_i} \sum_{a \in \mathcal{A}_i(h)} \mathcal{P}(a|\delta_i^h) M_{ab} \\ &\leq \min_{b \in \mathcal{A}_{-i}(h)} \max_{a \in \mathcal{A}_i(h)} M_{ab} = v_h^i \end{aligned}$$

The first equality is the definition of the value of a zero-sum game, the second is 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_h and some of the actions form its support would have lower expected utility, removing the action 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 improve the value. □

Example As an example, consider a following matrix game with utility values for the row player (i.e., the column player is minimizing):

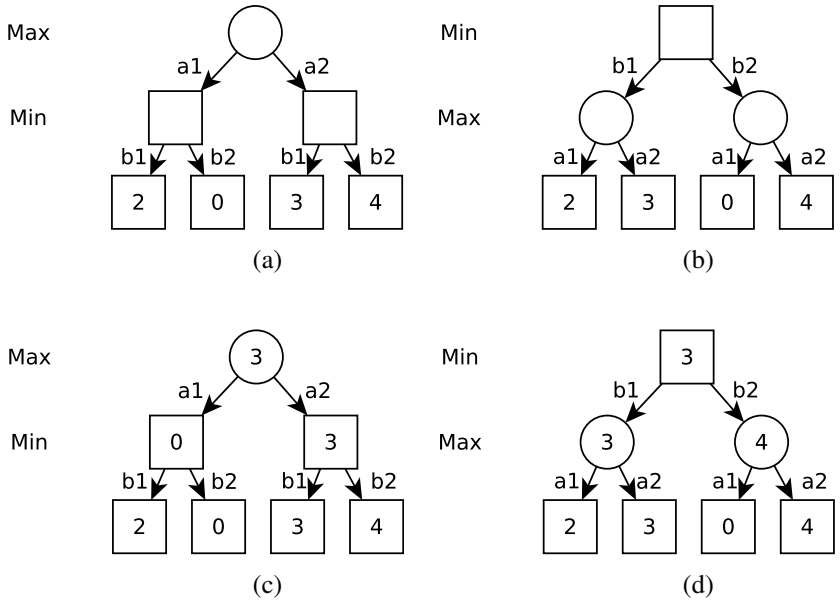


Figure 4.2: Different serialization of a simultaneous-move game. Utilities in the leafs are for the circle player.

	b_1	b_2
a_1	2	0
a_2	3	4

This game can be serialized in two ways that are depicted in Figure 4.2. If the row player i moves first (subfigure 4.2a), then the value of this serialized game is the lower bound of the value of the game (denoted v^{-i}); if the row player moves second (subfigure 4.2b), then the value of this serialized game is the upper bound of the value of the game (denoted v^{i}). Since the serialized games are zero-sum perfect-information games in the extensive form, they can be solved very quickly by using classical AI algorithms such as alpha-beta (Russell and Norvig, 2009) or negascout (Reinefeld, 1989). If the values of both serialized games are equal, then this is equal also to the value of the game. This situation occurs in our example in Figure 4.2, where both serialized games have value equal to 3 (see calculation in subfigures 4.2c and 4.2d). However, this is not the case when mixed strategies are required for NE strategies in the matrix game. Recall our example of Matching Pennies from Figure 2.1 – serializing this game yields values $v^{i} = 1$ and $v^{-i} = 0$.

Having the upper and lower bounds on values of sub-games can speed-up the backward induction algorithm. Algorithm 2 depicts the pseudocode. When the backward induction starts evaluating successors of the current state, the algorithm calculates upper and lower bounds using alpha-beta algorithm on serialized variants of the sub-game rooted in the successor h' (lines 10-11 and lines 19-20). The serialized game is solved using standard alpha-beta algorithm, where the opponent of the searching player i minimizes the utility

Algorithm 2 Backward Induction with Serialized Bounds (BI$\alpha\beta$)

Require: $h \in \mathcal{H}$ – current stage of the game; i – searching player

```
 1: if s ∈ Z then
 2:     return u_i(h)
 3: end if
 4: for a ∈ A_i(h) do
 5:     for b ∈ A_{-i}(h) do
 6:         h' ← hab
 7:         if ρ(h') = c then
 8:             M_ab ← 0
 9:             for a_n ∈ A_c(h') do
10:                 v^i_{h'a_n} ← alpha-beta(h'a_n, i)
11:                 v^{-i}_{h'a_n} ← alpha-beta(h'a_n, −i)
12:                 if v^i_{h'a_n} < v^{-i}_{h'a_n} then
13:                     M_ab ← M_ab + T(h', a_n) · BIαβ(h'a_n, i)
14:                 else
15:                     M_ab ← M_ab + T(h', a_n) · v^i_{h'a_n}
16:                 end if
17:             end for
18:         else
19:             v^i_{h'} ← alpha-beta(h', i)
20:             v^{-i}_{h'} ← alpha-beta(h', −i)
21:             if v^i_{h'a_n} < v^{-i}_{h'a_n} then
22:                 M_ab ← BIαβ(h', i)
23:             else
24:                 M_ab ← v^i_{h'}
25:             end if
26:         end if
27:     end for
28: end for
29: v_h ← solve matrix game M
30: return v_s
```

value. If the bounds are equal, the algorithm stores the value (line 24) instead of performing a recursive call (line 22). The algorithm has to again handle chance nodes by calculating the expected value and weighting the value of the succeeding sub-games (lines 7-15).

Corollary 4.3.2 *The serialized alpha-beta*(h', i) *algorithm called for player* i *computes the upper bound on the value of the simultaneous moves sub-game defined by stage* h'.

Proof Since it is known that the standard alpha-beta algorithm returns the same result as the standard minimax algorithm without pruning (i.e., backward induction on perfect information games), we disregard the pruning in this proof. We use Lemma 4.3.1 inductively. Let h be the current stage of the game and let M be the exact matrix game corresponding to h. By induction we assume the algorithm computes for stage h some $\overline{M}$ so that $\forall a, b\ \overline{M}_{ab} \geq M_{ab}$. It means that the worst case expected payoff of any fixed strategy in $\overline{M}$ is larger than in M;

hence, $v(\overline{M}) \geq v(M)$. The serialization used in alpha-beta(h', i) assumes that player $-i$ moves first; hence, by Lemma 4.3.1 the algorithm returns value $v^i(\overline{M}) \geq v(\overline{M}) \geq v(M)$. □

Backward induction algorithm with serialized alpha-beta algorithms is, to the best of our knowledge, not described in related work and as such can be considered as a contribution of this thesis. However, the following section describe a more advanced algorithm that, as we will show in the experiments, outperforms BI$\alpha\beta$ algorithm.

4.3.2 Integrating Double-Oracle with Backward Induction

Double-oracle algorithm for matrix games can be directly incorporated into the backward induction – 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 depicts this integration. In each stage of the game, the algorithm initializes the restricted game with an arbitrary action for each player (line 7). Afterwards, the algorithm needs to evaluate each of the successors of the restricted game, for which the current value M'_{ab} is not known (lines 9-25). This evaluation is the same as for the backward induction with alpha-beta algorithm (BI$\alpha\beta$); hence, the algorithm distinguishes possible succeeding chance node and calculate values of the sub-games either by serialized alpha-beta algorithms (lines 18 or 25), or by a recursive call (lines 16 or 23). Note that due to clarity we omit calls of alpha-beta algorithm on serialized variants of the game. These are called implicitly any time the algorithm requires to obtain a bound for a certain successor h'. Moreover, these values are cached for this stage of the game and removed from memory once the matrix game corresponding to stage h is solved and the value is propagated to its predecessor. Therefore, each value $v_h^{i/-i}$ is calculated at most once in stage h.

Once all values for the restricted game matrix M' are known, the algorithm solves the restricted game (line 30) and keeps the optimal strategies δ' of the restricted game. Next, the algorithm calculates best responses for each of the player (lines 31,32), and updates the lower and upper bounds (line 33). Finally, the algorithm expands the restricted game with best response actions (line 37) until the lower and upper bound are equal (line 38). In this case, 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 stage h.

Calculating Best Response

Next, we describe the algorithm for computing the best responses. The pseudocode of the algorithm is depicted in Algorithm 4. The goal of the algorithm is to find the best action for player i from the original unrestricted game against current strategy of the opponent δ'_{-i}. Throughout the algorithm we use, as before, $v^i_{h'}$ to denote the upper bound of the value of the sub-game rooted in state h' calculated using alpha-beta(h', i). Again, these values are calculated on demand, i.e., they are calculated once needed and cached until the game for

Algorithm 3 Double Oracle with Serialized Bounds (DO$\alpha\beta$)

Require: h – current state of the game; i – searching player; α_h, β_h – bounds for the game value rooted in state h

1: **if** $h \in \mathcal{Z}$ **then**
2: **return** $u_i(h)$
3: **end if**
4: **if** $v_h^{-i} = v_h^i$ **then**
5: **return** v_h^i
6: **end if**
7: initialize $\mathcal{A}'_i$, $\mathcal{A}'_{-i}$ with arbitrary actions from $\mathcal{A}_i(h), \mathcal{A}_{-i}(h)$
8: **repeat**
9: **for** $a \in \mathcal{A}'_i, b \in \mathcal{A}'_{-i}$ **do**
10: $h' \leftarrow hab$
11: **if** M'_{ab} is not initialized **then**
12: $M'_{ab} \leftarrow 0$
13: **if** $\rho(h') = c$ **then**
14: **for** $a_n \in \mathcal{A}_c(h')$ **do**
15: **if** $v^{-i}_{h'a_n} < v^i_{h'a_n}$ **then**
16: $M'_{ab} \leftarrow M'_{ab} + \mathcal{T}(h', a_n) \cdot \text{DO}\alpha\beta(h'a_n, i, v^{-i}_{h'a_n}, v^i_{h'a_n})$
17: **else**
18: $M'_{ab} \leftarrow M'_{ab} + \mathcal{T}(h', a_n) \cdot v^i_{h'a_n}$
19: **end if**
20: **end for**
21: **else**
22: **if** $v^{-i}_{h'} < v^i_{h'}$ **then**
23: $M'_{ab} \leftarrow \text{DO}\alpha\beta(h', i, v^{-i}_{h'}, v^i_{h'})$
24: **else**
25: $M'_{ab} \leftarrow v^i_{h'}$
26: **end if**
27: **end if**
28: **end if**
29: **end for**
30: $\langle v_h, \delta' \rangle \leftarrow$ solve matrix game M'
31: $\langle v_i^{BR}, a_i^{BR} \rangle \leftarrow$ best-response$(i, \delta'_{-i}, \beta_h)$
32: $\langle v_{-i}^{BR}, a_{-i}^{BR} \rangle \leftarrow$ best-response$(-i, \delta'_i, -\alpha_h)$
33: $\alpha_h \leftarrow \max(\alpha_h, -v_{-i}^{BR})$, $\beta_h \leftarrow \min(\beta_h, v_i^{BR})$
34: **if** $a_i^{BR} =$ **null** $\vee\ a_{-i}^{BR} =$ **null then**
35: **return** $-\infty$
36: **end if**
37: $\mathcal{A}'_i \leftarrow \mathcal{A}'_i \cup \{a_i^{BR}\}$, $\mathcal{A}'_{-i} \leftarrow \mathcal{A}'_{-i} \cup \{a_{-i}^{BR}\}$
38: **until** $\alpha_h = \beta_h$
39: **return** v_h

state h is not solved. Moreover, once the algorithm calculates the exact value of a particular sub-game, both upper and lower bounds are updated to be equal to actual value of the game.

The algorithm iteratively examines each action of player i from the unrestricted game (line 3). Every action a is evaluated against the actions of the opponent that are used in the

Algorithm 4 Best Response with Serialized Bounds

Require: h – current matrix game; i – best-response player; λ – bound for the best-response value; δ'_{-i} – strategy of the opponent

1: $v^{BR} \leftarrow \lambda$
2: $a^{BR} \leftarrow$ **null**
3: **for** $a \in \mathcal{A}_i(h)$ **do**
4: $\quad v_a \leftarrow 0$
5: $\quad$ **for** $b \in \mathcal{A}'_{-i}(h) \ : \ \delta'_{-i}(b) > 0$ **do**
6: $\quad\quad h' \leftarrow hab$
7: $\quad\quad \lambda_{ab} \leftarrow \frac{v^{BR} - \sum_{b' \in \mathcal{A}'_{-i} \setminus \{b\}} \delta'_{-i}(b') \cdot v^i_{hab'}}{\delta'_{-i}(b)}$
8: $\quad\quad$ **if** $\lambda_{ab} > v^i_{h'}$ **then**
9: $\quad\quad\quad$ continue from line 3 with next a_i
10: $\quad\quad$ **else**
11: $\quad\quad\quad$ **if** $\rho(h') = c$ **then**
12: $\quad\quad\quad\quad v_{ab} \leftarrow 0$
13: $\quad\quad\quad\quad$ **for** $a_n \in \mathcal{A}_c(h')$ **do**
14: $\quad\quad\quad\quad\quad$ **if** $v^{-i}_{h'a_n} < v^i_{h'a_n}$ **then**
15: $\quad\quad\quad\quad\quad\quad \lambda'_{ab} \leftarrow \frac{\lambda_{ab} - \sum_{a' \in \mathcal{A}_c(h') \setminus \{a_n\}} \mathcal{T}(h', a') \cdot v^i_{h'a'}}{\mathcal{T}(h', a_n)}$
16: $\quad\quad\quad\quad\quad\quad v_{ab} \leftarrow v_{ab} + \mathcal{T}(h', a_n) \cdot \text{DO}\alpha\beta(h'a_n, i, v^{-i}_{h'}, v^i_{h'a_n})$
17: $\quad\quad\quad\quad\quad$ **else**
18: $\quad\quad\quad\quad\quad\quad v_{ab} \leftarrow v_{ab} + \mathcal{T}(h', a_n) \cdot v^i_{h'a_n}$
19: $\quad\quad\quad\quad\quad$ **end if**
20: $\quad\quad\quad\quad$ **end for**
21: $\quad\quad\quad$ **else**
22: $\quad\quad\quad\quad$ **if** $v^{-i}_{h'} < v^i_{h'}$ **then**
23: $\quad\quad\quad\quad\quad v_{ab} \leftarrow \text{DO}\alpha\beta(h', i, v^{-i}_{h'}, v^i_{h'})$
24: $\quad\quad\quad\quad$ **else**
25: $\quad\quad\quad\quad\quad v_{ab} \leftarrow v^i_{h'}$
26: $\quad\quad\quad\quad$ **end if**
27: $\quad\quad\quad$ **end if**
28: $\quad\quad\quad v_a \leftarrow v_a + \delta'_{-i}(b) \cdot v_{ab}$
29: $\quad\quad$ **end if**
30: $\quad$ **end for**
31: $\quad$ **if** $v_a > v^{BR}$ **then**
32: $\quad\quad v^{BR} \leftarrow v_a$
33: $\quad\quad a^{BR} \leftarrow a$
34: $\quad$ **end if**
35: **end for**
36: **return** $\langle v^{BR}, a^{BR} \rangle$

optimal strategy from the restricted game (line 5). First, the algorithm determines whether the current action of the searching player, a, can still be the best response action (line 7). The value λ_{ab} represents the lowest possible expected utility this action must gain against the current action of the opponent b. If this value is strictly higher than the upper bound of the value of the successor h' (i.e., the value $v^i_{h'}$) then the algorithm knows that the action a can never be the best response action against δ'_{-i} since value λ_{ab} cannot be reached. In this

case, the algorithm proceeds with the next action (line 9). λ_{ab} is calculated by subtracting from the current best response value (v^{BR}) upper bound of the expected value against all other actions of the opponent ($\sum_{b'} v^i_{hab'}$). Recall, that these values are updated once the algorithm calculates exact values.

If the currently evaluated action a can still be the best response, the value of the successor is determined (first by comparing the bounds, and again we have to distinguish a case when the succeeding node is a chance node). If the upper and lower bounds are not equal, a recursive call of the double-oracle algorithm is executed (line 16 or 23). Once the expected outcome against all actions of the opponent is evaluated, the expected value of action a_i is compared against the current best-response value (line 31) and saved, if the expected utility is higher (line 33). If the succeeding node is a chance node, a new bound is calculated that takes into consideration all possible successors of the chance node and the probability of playing action a_n (line 15). Although the formula is much more complex, the main idea remains the same – λ'_{ab} represents the lower bound for utility value of the sub-game rooted in successor $h'a_n$ such that a can still be the best response. It is calculated by subtracting from the overall lower bound λ_{ab} the upper bounds $v^i_{h'a_n}$ and weighting accordingly.

We now discuss the correctness of the backward-induction algorithm with double-oracle and serialized bounds. Corollary 4.3.2 shows that the bounds $v^i_{h'}$ and $v^{-i}_{h'}$ in the algorithms are correct. Assuming inductively the values $v_{h'}$ from all successors of a stage h are correct, the best-response algorithms always return either a correct best response a^{BR} with maximal expected value given the strategy of the opponent, or **null** in case no action is better than the given bound. Now we can formally prove the correctness of the main algorithm.

Lemma 4.3.3 *Let i be the searching player, v_h be the value of the sub-game defined by stage h. Then double-oracle$(h, i, \alpha_h, \beta_h)$ returns*

$$\begin{cases} v_h & \text{if } \alpha_h \leq v_h \leq \beta_h \\ -\infty & \text{if } v_h < \alpha_h \vee v_h > \beta_h \end{cases}$$

Proof The basis for the correctness of this part of the algorithm comes from the standard double-oracle algorithm for zero-sum normal form games (McMahan et al., 2003).

Case 1: If the algorithm ends on line 39 and we assume that all the recursive calls of the algorithm are correct, the algorithm performs the standard double-oracle algorithm on exact values for sub-games. The only difference is that the best-response algorithms are bounded by α_h or β_h. However, if the algorithm did not end on line 35, these bounds were never restrictive and the best-response algorithm has returned strategies of the same value as it would without any bounds.

Case 2: If the algorithm returns on line 35, the situation is symmetric: WLOG we assume that $v_h < \alpha_h$. We have to consider two possibilities. If the bound α_h has not been updated during the iterations of the main loop (line 33) and the algorithm finds a strategy δ_i so that no best response of player $-i$ to this strategy has a higher expected value than α_h, the strategy δ_i proves that the value of the game M is lower than α_h and $-\infty$ is returned.

If the bound α_h has been updated on line 33, the algorithm can never stop at line 35. Modifying α_h means the algorithm has found a strategy δ_i for player i that guarantees the

reward of α_h against any strategy of the opponent. For any strategy δ_{-i} of the opponent that can occur in further iterations of the main cycle of the double-oracle algorithm, playing mixed strategy δ_i gains at least α_h for player i. It means that either all pure strategies from the support of δ_i gain payoff α_h, or there is at least one pure strategy that gains strictly more than α_h in order to achieve α_h in expectation. Either way, the best-response algorithm for player i does not return **null**. □

4.4 Experimental Evaluation

We experimentally evaluate our novel algorithm on three specific games and a set of randomly generated games. As a baseline we use algorithms based on full backward induction that solves the full linear program (LP) in each stage of the game (denoted BI). The same baseline algorithm and one of the games was also used in the most related previous work (Saffidine et al., 2012), which allows us to directly compare the results, since their pruning algorithm introduced only minor improvement. We use three variants of our novel algorithms: first of all we use backward-induction algorithm that uses alpha-beta on serialized games, denoted BI$\alpha\beta$. Secondly, we use double-oracle algorithm without using serialized alpha-beta search (calculating bounds $v_h^{i/-i}$ is disabled), denoted DO. Finally, we compare our most advanced algorithm combining both aspects, denoted DO$\alpha\beta$. This comparison allow us to evaluate separately the contribution to computation speed-up due to serialized alpha-beta search and due to normal-form double-oracle algorithm.

Our collection of games contain games used previously in benchmarks for simultaneous-move games, pursuit-evasion game that is an important example of simultaneous-move games, and randomly generated games. The games differ in possible utility values, size of the branching factor, and amount of required mixed strategies in equilibrium. The last factor affects how successful the serialized alpha-beta searches are, and how often the algorithms can avoid fully evaluating certain sub-games. None of the algorithms uses any domain-specific knowledge or heuristics and a different random move ordering is selected in each run of the experiments. Unless explicitly stated, none of the algorithms use any other form of cache other than remembering the results in the current stage of the game. All the compared algorithms were implemented in Java, using a single thread on a 2.8 GHz CPU and CPLEX v12.5 for solving LPs.

4.4.1 Experiment Domains

Goofspiel

Card game Goofspiel appear in many works focused on simultaneous-move games (e.g., (Saffidine et al., 2012; Rhoads and Bartholdi, 2012; Lanctot et al., 2013a)). There are 3 identical decks of cards with values $\{0, \ldots, (d-1)\}$ (one for nature and one for each player), where d is a parameter of the game. The game is played in rounds: at the beginning of each round, nature 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 nature's

card. In case both players select the card with the same value, the nature'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. We follow the assumption made in (Saffidine et al., 2012) that both players know the sequence of the nature's cards. The game is win-tie-loss; hence, the players receive utility from set $\{-1, 0, 1\}$. In the experiments we varied the size of the decks of cards d, and in each experimental run we used a randomly selected sequence of the nature's cards.

Instances of Goofspiel game correspond to game trees with interesting properties. First unique feature is that the branching factor 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:

Depth	0	1	2	3	4
Pure	0	17	334	3354	14400
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.

Pursuit-evasion Game

As a second game we use a pursuit-evasion game played on a graph for a pre-defined number of moves (d). Pursuit-evasion games are an important class of games with many robotic applications (see for example (Nguyen and Thawonmas, 2013; Lisy et al., 2014)). There is a single evader and a pursuer that controls 2 pursuing units. Since all units move simultaneously, the game has larger branching factor than in Goofspiel (there are up to 16 actions for pursuer). The evader wins, if she successfully avoids the units of the pursuer for the whole game; pursuer wins, if her units successfully capture the evader. The evader is captured at some turn of the game 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) in this turn. The game is win-loss and the players receive utility from set $\{-1, 1\}$. We used two grid-graphs for the experiments (4×4, and 5×5 nodes) without any obstacles or holes. In the experiments we varied the allowed number of moves d and we altered the starting positions of the players (the distance between the pursuers and the evader was always at most $\lfloor \frac{2}{3} d \rfloor$ moves, in order to provide a possibility for the pursuers to capture the evader).

Many instances of pursuit-evasion games have a pure Nash equilibrium. However, the mixing 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 (units of the pursuers are simply going towards the evader, or the position is clearly winning or loosing for some player).

Once the units are close to each other, the game may require mixed strategies for final coordination. This can be seen on our small example on a graph 4×4 nodes and depth 5:

Depth	0	1	2	3	4
Pure	1	12	261	7656	241986
Mixed	0	0	63	1008	6726

Oshi-Zumo

Oshi-Zumo is a board game analyzed from the perspective of computational game theory in (Buro, 2003). There are two players in the game, both starting with certain number of coins, and there is a playing board represented as a one-dimensional playing field. There are $2K + 1$ locations on the field (indexed $0, \ldots, 2K$), where K is a parameter of the game. At the beginning, there is a stone (or a wrestler) located in the center of the playing field (i.e., at position K). In each move, both players simultaneously place their bid from the amount of coins they have (but at least 1 if they still have some coins). Afterwards, the bids are revealed, both bids are subtracted from the number of coins of the players, and the highest bidder pushes 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 looses the game. If the final position of the wrestler at in the center, the game is a draw. Again, the utility values are restricted to $\{-1, 0, 1\}$. In the experiments we varied number of coins and parameter K. Although the size of the minimal bid is also a parameter, we fixed this parameter to 1.

Many instances of the Oshi-Zumo game have pure Nash equilibrium. With the increasing number of the coins the game necessarily 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 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	2012	14767	48538	79926	69938	33538	8351	861
Mixed	0	1	4	17	8	0	0	0	0	0

Random 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 players can play in each stage to 4

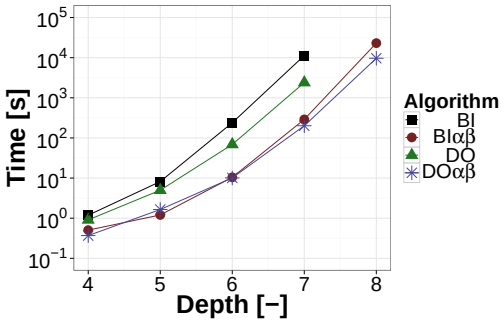


Figure 4.3: Comparison of running times on Goofspiel with increasing size of the deck.

and 5 (the results were similar for different branching factors) and we varied 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 a joint action in each state (i.e., to each combination of actions) and the utility value in a leaf is a sum of all values on 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 search, or double-oracle methods, since there is no correlation between actions and utility values in sibling leafs. The two latter methods are based on random *T-games* (Smith and Nau, 1995), that create more realistic games using the intuition of good and bad moves.

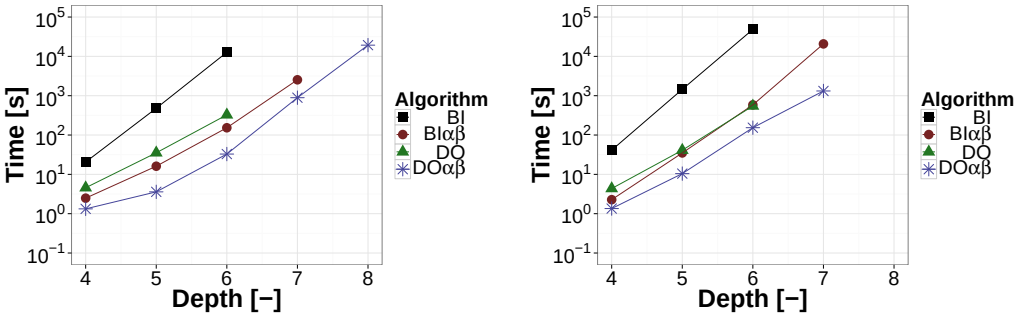
Randomly generated games represent games that require mixed strategies in most of the states. This holds even for games of the second type with correlated utility values in the leaves. The following table shows the number of states depending on the depth for randomly generated game of depth 5 with 4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

Depth	0	1	2	3	4
Pure	0	2	29	665	20093
Mixed	1	14	227	3431	45443

4.4.2 Results

We compare the overall running time for each of the algorithms. We measure the overall computation time for each of the algorithms and number of evaluated nodes – i.e., nodes for which the main method of the backward induction algorithm executed (i.e., 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 of at least 30 runs.

Goofspiel Figure 4.3 depicts the results for the card game Goofspiel (note the logarithmic y-scale). The results show that a significant number of sub-games has a pure sub-game



(a) (b)

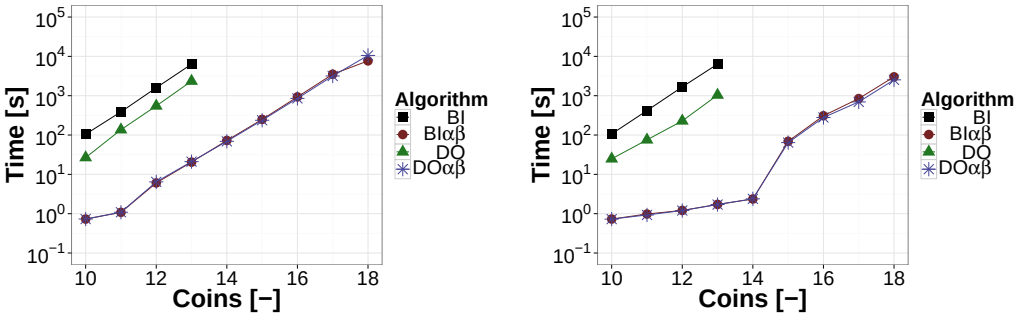
Figure 4.4: Comparison of running times on pursuit-evasion game with increasing number of moves: sub-figure (a) depicts the results on 4×4 grid graph, (b) depicts results for 5×5 grid.

Nash equilibrium that can be computed using 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 DO$\alpha\beta$ with the increasing size of the game. Both of these algorithms significantly reduce the number of states visited by the backward induction algorithm (i.e., excluding the number of states evaluated by serialized alpha-beta algorithm). While BI algorithm evaluates on average more than $32 \cdot 10^6$ nodes in the setting with 7 cards in more than 3 hours, BI$\alpha\beta$ evaluates only 198986 nodes in less than 5 minutes. The performance is further improved by DO$\alpha\beta$ that evaluates on average 51035 nodes in less than 4 minutes. However, the overhead of the algorithm is slightly higher in case of DO$\alpha\beta$; hence, the difference between DO$\alpha\beta$ and BI$\alpha\beta$ is relatively small in this case. Finally, the results show that even simple DO algorithm without serialized alpha-beta search can improves the performance of BI. In the setting with 7 cards, DO evaluates more than $6 \cdot 10^6$ nodes which takes on average almost 40 minutes.

The results on Goofspiel highly contrast with the Goofspiel results of the pruning algorithm SMABS presented in (Saffidine et al., 2012). In their work, the number of evaluated nodes was at best around 29%, and the running time improvement was only marginal.

Pursuit-Evasion Games The results on pursuit-evasion games show more significant improvement when comparing DO$\alpha\beta$ and BI$\alpha\beta$ (see Figure 4.4). For both graph the DO$\alpha\beta$ is significantly the fastest. When we compare the performance on graph 5×5 with depth set to 6, BI evaluates more than $49 \cdot 10^6$ nodes that takes more than 13 hours. On the other hand, BI$\alpha\beta$ evaluates on average 42001 nodes taking almost 10 minutes (584 seconds). Interestingly, the benefits of pure integration with alpha-beta search is not that helpful in this game. This is apparent from results of DO algorithm that evaluates less than $2 \cdot 10^6$ nodes but it takes slightly over 9 minutes on average (547 seconds). Finally, DO$\alpha\beta$ evaluates only 6692 nodes and it takes the algorithm less than 3 minutes.

Large parts of pursuit-evasion game can be solved by serialized alpha-beta algorithms. These parts typically corresponds to clearly winning, or clearly loosing positions for a



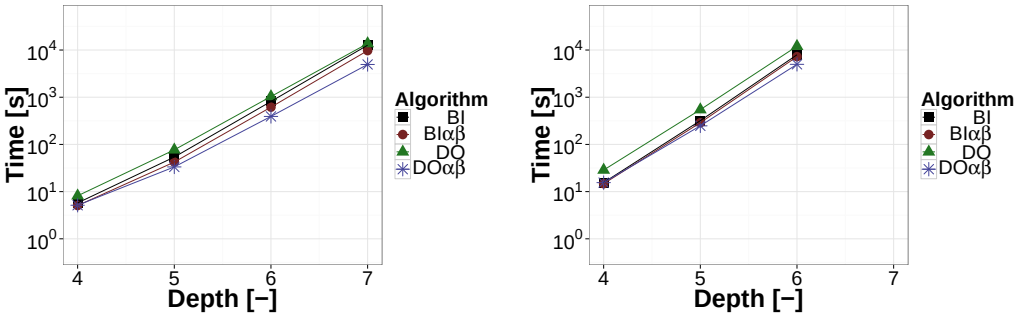
(a) (b)

Figure 4.5: Comparison of running times on the Oshi-Zumo game with increasing number of coins: sub-figure (a) depicts the results on with K set to 3, (b) depicts the results with $K = 4$.

player. However, since there are only two pursuit units, it is still necessary to use mixed strategies for 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 sub-games, while double-oracle provides additional pruning and cause the improvement in computation time for DO$\alpha\beta$ compared to BI$\alpha\beta$ and all the other algorithms.

Oshi-Zumo Many instances of the Oshi-Zumo game have Nash equilibria in pure strategies. Although this does not hold for all the instances, the size of the sub-games with pure NE are rather large and cause dramatic computation speed-up for both algorithms using 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 to pursuit-evasion games, where mixed strategies were necessary close to the end of the game tree. Figure 4.5 depicts the results for two different setting of the playing field K is either set to 3 or 4. In both cases, the graphs clearly highlights when the whole game does not have an equilibrium in pure strategies – for K equal to 3, the change occurs when the number of coins increase from 11 to 12, for $K = 4$ the first setting with non-pure equilibria is in case with 15 coins for each player.

The consequence of the advantage of BI$\alpha\beta$ and DO$\alpha\beta$ algorithms that exploit serialized variants of alpha-beta algorihtms is dramatic in Oshi-Zumo game. We can see that both BI and DO scale rather badly. The algorithms were able to scale up to 13 coins in reasonable time. For setting with $K = 4$ and 13 coins, it takes almost 2 hours for BI to solve the game (the algorithm evaluates $15 \cdot 10^6$ nodes). DO improves the performance slightly (the algorithm evaluates nearly $6 \cdot 10^6$ nodes in 40 minutes), however, the difference between alpha-beta algorithm is dramatic. Both BI$\alpha\beta$ and DO$\alpha\beta$ are in essence solved by executing a single alpha-beta search on each serialization. Therefore, their performance is identical and it takes 21 seconds to solve the game. With an increasing number of coins the algorithms need to find mixed Nash equilibria, however, their performance is in both cases very similar.



(a) (b)

Figure 4.6: Comparison of running times on randomly generated games with increasing depth: sub-figure (a) depicts the results with branching factor set to 4 actions for each player, (b) depicts the results with branching factor 5.

Random Games Finally we turn to randomly generated games. In the first variant of the randomly generated games we used games with utility values randomly drawn from a uniform distribution $[0, 1]$. Such games represent an extreme case, where neither alpha-beta search, nor double-oracle algorithm can save much computation time, since each action can lead to arbitrarily good or bad terminal state. In these games, BI algorithm is typically the fastest. Even though both BI$\alpha\beta$ and DO$\alpha\beta$ evaluate marginally less nodes ($\approx 90\%$), the overhead of the algorithms (repeated calculation of alpha-beta algorithm, repeatedly solving linear programs, etc.) causes slower runtime 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. Figure 4.6 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 remaining algorithms, although the difference is rather small (however still statistically significant, as we will show in the next section). On the other hand, DO without serialized alpha-beta is not able to outperform BI. This is caused by a larger support in mixed sub-game equilibria that cause enumerating most of the actions by the double-oracle algorithm. 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 case with branching factor set to 4 for both players, and depth 7, BI evaluates almost $18 \cdot 10^6$ nodes in almost 3.5 hours, while BI$\alpha\beta$ evaluates more than $\approx 10 \cdot 10^6$ nodes in almost 3 hours. DO evaluates even more nodes compared to BI$\alpha\beta$ ($\approx 12 \cdot 10^6$) and it is slower compared to both BI and BI$\alpha\beta$. Finally, DO$\alpha\beta$ evaluates $\approx 2 \cdot 10^6$ nodes on average and it takes the algorithm slightly over 80 minutes.

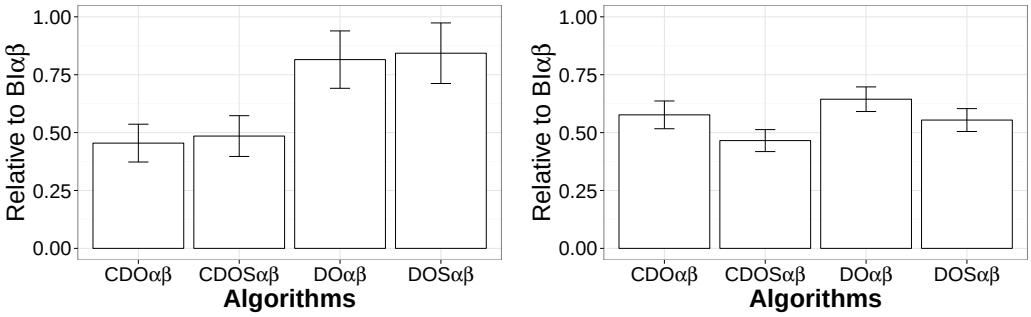
4.4.3 Improved variants of DO$\alpha\beta$

The results in the previous section show that DO$\alpha\beta$ outperforms other algorithms. The computation speed-up over standard backward induction is dramatic thanks to the integration of two components: double-oracle algorithm and alpha-beta search on serialized variants of the game. We also evalute separate contributions of each of the components. The results show that serialized alpha-beta offers dramatic speed-up if there exist large sub-games with pure Nash equilibria. On the other hand, using double-oracle algorithm in stage matrix games improve the performance in case mixed strategies are required for Nash equilibria.

However, described DO$\alpha\beta$ algorithm can be further improved by standard methods used before in search community when solving perfect-information games. First of all, DO$\alpha\beta$ can use information from serialized bounds also to determine move ordering. It is known from classical alpha-beta algorithm that move ordering can have a significant impact on the performance of the algorithm. In DO$\alpha\beta$ during the computation of the best-response for player i, the algorithm evaluates the actions of player i (see line 3 in Algorithm 4). These actions can be sorted according optimistic value v^i against the current strategy of the opponent and evaluate in descending order. This way, the algorithm should evaluate the actions that are more likely to be successful first. In further text we refer to the variant of double-oracle algorithm with DOS (i.e., DOS$\alpha\beta$ refer to the double-oracle with sorting the actions during the best response and using serialized alpha-beta algorithms).

Secondly, the DO$\alpha\beta$ algorithm does not use any global cache; hence, certain calculations can be done repeatedly. By using a cache (in the search literature known as transposition tables), we can re-use already calculated results and further speed-up the computation time. This is primarily concerned with repeated calculations of serialized alpha-beta search, however, we can also use double-oracle algorithm with tightening bounds. Recall the calculation of the best response in Algorithm 4. When the algorithm determines that certain action can still be a best response and needs to be fully evaluated against a particular action of the opponent, a call for DO$\alpha\beta$ is performed (lines 16 and 23). However, we can call the double-oracle algorithm with a tighter bound by selecting the lower bound as a $\max(\lambda'_{ab}, v^{-i}_{h'})$ – the maximum of the lower bound in order for action a to become the best response, and lower bound provided by the serialized alpha-beta search. Using this modification, the double-oracle algorithm can visit states repeatedly because selecting λ'_{ab} as the lower bound can cause a cut-off. However, this value depends on the current strategy of the opponent that can change in the next iteration and the same sub-game can be recalculated. Therefore, using a global cache is essential. In the following we use a simple unlimited global cache, where all calculated values of sub-games are kept for every state evaluated by the double-oracle algorithm (i.e., not those nodes visited only by serialized alpha-beta search). We denote using this cache with letter C at the beginning of the algorithm.

By combining these two improvements (i.e., sorting the own actions and using a cache with tightening bounds in best-response calculations), we introduce 3 new variants: DOS$\alpha\beta$, CDO$\alpha\beta$, and CDOS$\alpha\beta$ (sorting actions, cache, and both improvements). We compared the results against the DO$\alpha\beta$ algorithm as before and visualized relative performance on large scenarios of selected games the relative performance of these algorithms against the closest



(a) (b)

Figure 4.7: Relative performance of improved variants of the DO$\alpha\beta$ algorithm compared to BI$\alpha\beta$. sub-figure (a) depicts the results on goofspiel with depth 7, (b) depicts the results on a randomly generated game with branching factor 4 and depth 6.

competitor – algorithm BI$\alpha\beta$. Figure 4.7 depicts the results in with error bars representing 95% confidence interval based on 30 different instances of the games.

The results for Goofspiel (Figure 4.7a) show that all variants of double-oracle algorithm are statistically significant faster compared to BI$\alpha\beta$. The speed-up of the variants without cache is rather small, however, both variants that use cache (i.e., CDO$\alpha\beta$ and CDOS$\alpha\beta$) are much faster. Interestingly, sorting of actions (i.e., variants DOS$\alpha\beta$ and CDOS$\alpha\beta$) is marginally decreasing the mean performance and increases the variance. We conjecture that this result is most likely caused by not always accurate estimation given by serialized alpha-beta and domain-dependent move-ordering heuristics are more likely to succeed (e.g., see (Bosansky et al., 2013b) or experiments with Phantom Tic-Tac-Toe in the next chapter). The results for the random games (Figure 4.7b) again show that all variants of double-oracle algorithm are faster compared to BI$\alpha\beta$. However, the impact of the improvements on the performance of the algorithms is different in this case. Both DOS$\alpha\beta$ and CDOS$\alpha\beta$ outperform variants that do not sort the actions. On the other hand, using cache again improves the performance making CDOS$\alpha\beta$ algorithm the fastest.

4.5 Discussion of the Results and Summary

The results show that our novel algorithm DO$\alpha\beta$ outperforms existing state-of-the-art algorithms on all games. Comparing to the existing algorithm described in the previous work, the computation speed-up is dramatic and our algorithm is able to solve much larger games by pruning a substantial number of all nodes. Second in the comparison is our algorithm BI$\alpha\beta$ that is typically slightly slower than DO$\alpha\beta$ in all scenarios. The difference is more apparent in games that require mixed strategies, such as random games or pursuit-evasion games, where the performance of double-oracle is significantly better. However, BI$\alpha\beta$ still significantly outperforms backward-induction algorithm due to the serialized alpha-beta search.

We have identified two immediate improvements for our algorithm – sorting the examined actions during the best response phase, and using a global cache to keep intermediate results. These improvements show further computation speed-up, although less significant. However, they indicate that using domain-specific knowledge and further improvements applying from years of research on perfect-information games can bring additional advantage to the DO$\alpha\beta$ algorithms and further improve the performance in practice.

Chapter 5

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

In this chapter we move to the most generic class of games in this thesis – imperfect information extensive-form games (EFGs) with perfect recall – and we present our novel algorithm *sequence-form double-oracle algorithm*. As we have outlined in Section 3.4, double-oracle algorithm can be applied straightforwardly for solving extensive-form games (EFGs) by transforming an EFG into a normal-form game and apply double-oracle algorithm for normal-form games. However, this approach suffers from an exponential transformation (there is an exponential number of pure strategies in an extensive-form game). Our algorithm overcome this issue by working directly with the compact sequence-form representation.

We begin by describing relevant related work focusing on previous applications of the double-oracle algorithm in EFGs. Then we formally describe the algorithm, all the main components, and we provide a step-by-step example to illustrate the dynamics of the algorithm. Afterwards, we give a formal proof that our algorithm converges to a Nash equilibrium. Finally, we again demonstrate the performance on a set of experiments. Besides comparing our algorithm to state-of-the-art exact algorithm based on solving complete sequence-form linear program (see Section 3.2.2), we also provide a comparison with a state-of-the-art approximative algorithm Counterfactual Regret Minimization (CFR).

5.1 Related Work

The main algorithms solving EFGs are described in Section 3.2. Here we focus on previous work that applied the iterative framework of oracle algorithms to EFGs. The first work that exploited the iterative principle is described in (Koller and Megiddo, 1992), the predecessor of the sequence-form linear-program formulation. In this algorithm, the authors use a representation similar to the sequence form for only one player, while the strategies for the opponent are iteratively added as constraints into the linear program (there is an exponential

number of constraints in their formulation). This approach can be seen as the single-oracle method, where the strategy space is expanded gradually for a single player.

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

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

In contrast, our algorithm directly uses the compact sequence-form representation of EFGs and uses the sequences as the building blocks (i.e., the restricted game is expanded by allowing new sequences to be played in the next iteration). While this is a strong advantage, working directly with sequences also introduces new challenges when constructing and maintaining a valid restricted game and ensuring convergence to a Nash equilibrium that we must solve for our algorithm to converge to a correct solution.

5.2 Sequence-Form Double-Oracle Algorithm

We now describe sequence-form double-oracle algorithm for solving extensive-form games with imperfect information. First, we give an informal overview of our algorithm following by the detailed description of the key components.

The overall scheme of our algorithm is based on the double-oracle framework described in Section 3.4. The main difference is that our algorithm uses the sequences to define the

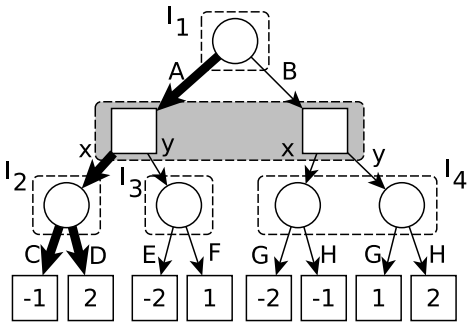


Figure 5.1: Example of a game tree, the bold edges represent sequences of actions added to the restricted game.

restrictions in the game tree. The restricted game in our model is defined by allowing players to use (i.e., to play with a non-zero probability) only a subset of the sequences from the original unrestricted game. This restricted subset of sequences then defines the subsets of reachable actions, nodes, and information sets from the original game tree. Consider our example in Figure 5.1. A restricted game can be defined by sequences $\emptyset, A, AC, AD$ for the circle player, and $\emptyset, x$ for the box player. These sequences represent allowed actions in the game, they define reachable nodes (using history we can reference them as $\emptyset, A, Ax, AxC, AxD$), and reachable information sets (I_1, I_2 for the circle player and the only information set $I_\square$ for the box player).

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

There are two primary complications that arise when we use sequences instead of full strategies in the double-oracle algorithm, both due to the fact that sequences do not necessarily define actions in all information sets: (1) a strategy computed in the restricted game may not be a complete strategy in the original game because it does not define behavior for information sets that are not in the restricted game, and (2) it may not be possible to play every action from a sequence that is allowed in the restricted game because playing a sequence can depend on having a compatible sequence of actions for the opponent. In our example game tree in Figure 5.1, no strategy of the circle player in the restricted game specifies what to play in information sets I_3 and I_4. The consequence of the second issue is that some inner nodes of the original unrestricted game can (temporarily) become leafs in the restricted game. For example, the box player can add sequence y into the restricted

game making node Ay a leaf in the restricted game, since there are no other actions of the circle player in the restricted game applicable in this node.

Our algorithm solves these complications using two novel ideas. The first idea is the concept of a *default pure strategy* (denoted $\pi_i^{\text{DEF}} \in \Pi_i$). Informally speaking, the algorithm assumes that each player has a fixed implicit behavior that defines what the player does by default in any information set that is not part of the restricted game. This is described by the default strategy π_i^{DEF}, which specifies an action for every information set. Note that this default strategy does not need to be represented explicitly (which could use a large amount of memory), but instead can be defined implicitly using rules, such as selecting the first action from a deterministic method for generating the ordered set of actions $A(h)$ in node h. We use the concept of default pure strategies to map every strategy from the restricted game into a valid strategy in the full game. To do this, the strategy in the original unrestricted game selects actions according to the probabilities specified by a strategy for the restricted game in every information set that is part of the restricted game, and for all other information sets it plays according to the default pure strategy. Recall our example in Figure 5.1, where pure default strategy for the circle player can be $\langle A, C, E, G \rangle$ (i.e., selecting the leftmost action in each information set); hence, a strategy in the original unrestricted game can use strategy from the restricted game in information sets I_1 and I_2, and select pure actions in E, G in information sets I_3 and I_4 respectively.

The second key idea is to use *temporary utility values* for cases where there are no allowed actions that can be played in a some node in the restricted game that is an inner node in the original game (so called *temporary leaf*). To ensure correct convergence of the algorithm these temporary utilities must be assigned so that they provide a bound on the value of continuing to play from the given node for the player making a choice in the node. Our algorithm uses a value that corresponds to an expected outcome of continuing the game play, assuming the player making choice in the temporary leaf uses the default strategy, while the opponent plays the best response. Assume we add sequence y for the box player into the restricted game in our example tree in Figure 5.1. The temporary utility value for node Ay would correspond to value -2, since the default strategy in information set I_3 is to play E for the circle player. Below we formally describe this method and prove the correctness of the algorithm given these temporary values.

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

5.2.1 Restricted Game

This section formally defines the restricted game that is essentially a subset of the original unrestricted game. The restricted game is derived from the set of allowed sequences; hence, we define all other sets of nodes, actions, and information sets as subsets of the original unrestricted sets based on the allowed sequences. We denote the original unrestricted game as a tuple $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, \rho, u, \mathcal{C}, \mathcal{I})$ and the restricted game as $G' = (\mathcal{N}, \mathcal{H}', \mathcal{Z}', \mathcal{A}', \rho, u', \mathcal{C}, \mathcal{I}')$ (all sets and functions associated with the restricted game use prime in the notation; the set of players, and functions ρ and $\mathcal{C}$ remain the same).

The restricted game is outlined with a set of allowed sequences that are returned by the best response algorithms. We term these *allowed sequences*, denoted $\Phi' \subseteq \Sigma$. As we have indicated above, however, even an allowed sequence $\sigma_i \in \Phi'$ might not be playable in the full length due to missing compatible sequences of the opponent. Therefore, the restricted game is formally defined as the maximal compatible set of sequences $\Sigma' \subseteq \Phi'$ defined by the set of allowed sequences Φ'. Therefore, Σ' is *the maximal subset* of the sequences from Φ' such that:

$$\Sigma'_i \leftarrow \{\sigma_i \in \Phi'_i \ : \ \exists \sigma_{-i} \in \Phi'_{-i} \exists h \in \mathcal{H} \ \forall j \in \mathcal{N} \ \mathsf{seq}_j(h) = \sigma_j\} \quad \forall i \in \mathcal{N} \tag{5.1}$$

Equation (5.1) means that for each player i and each sequence σ_i in Σ'_i, there exists a compatible sequence of the opponent σ_{-i} that allows the sequence σ_i to be executed in full.

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

$$\mathcal{H}' \leftarrow \{h \in \mathcal{H} \ : \ \forall i \in \mathcal{N} \ \mathsf{seq}_i(h) \in \Sigma'\} \tag{5.2}$$

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

$$\mathcal{A}'(h) \leftarrow \{a \in \mathcal{A}(h) \ : \ ha \in \mathcal{H}'\} \quad \forall h \in \mathcal{H}' \tag{5.3}$$

Note that nodes from the restricted game corresponding to inner nodes in the original unrestricted game are not necessarily also inner nodes in the restricted game. Therefore, the set of leaves in the restricted game is defined as a union of leaves of the original game included in the restricted game and those originally inner nodes that currently do not have a continuation in the restricted game:

$$\mathcal{Z}' \leftarrow \left(\mathcal{Z} \cap \mathcal{H}'\right) \cup \{h \in \mathcal{H}' \setminus \mathcal{Z} \ : \ \mathcal{A}'(h) = \emptyset\} \tag{5.4}$$

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

The information sets in the restricted game correspond to the information sets in the original unrestricted game – i.e., if some node h belongs to an information set $I_{\rho(h)}$ in the

original game, then the same holds in the restricted game. We define an information set to be a part of the restricted game if and only if at least one inner node that belongs to this information set is included in the restricted game:

$$\mathcal{I}'_i \leftarrow \{I_i \in \mathcal{I}_i \ : \ \exists h \in I_i \ h \in \mathcal{H}' \setminus \mathcal{Z}'\} \tag{5.5}$$

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

Finally, we need to define a modification of the utility function u' for the restricted game. The primary reason for the modified utility function is the definition of the temporary utility values for leafs in the set $\mathcal{Z}' \setminus \mathcal{Z}$. For some $h \in \mathcal{Z}' \setminus \mathcal{Z}$ it must hold that the assigned utility value $u'_i(h)$ is a lower bound of the expected utility value in this node if both players continue to play Nash strategies. By setting the temporary utility this way the algorithm ensures that the player acting in this node adds sequences continuing from this node as a part of the best response if there is some other continuation that improves the pessimistic temporary value for the node. To calculate a lower bound estimation, we set the utility value such that it corresponds to the outcome in the original unrestricted game if player acting in node h continues by playing the default strategy $\pi^{\text{DEF}}_{\rho(h)}$ and the opponent plays a best response $\delta^{BR}_{-\rho(h)}$ to this default strategy. Finally, for all other leafs $h \in \mathcal{Z}' \cap \mathcal{Z}$ we set $u'_i(h) \leftarrow u_i(h)$.

Solving the Restricted Game The restricted game defined in this section is a valid zero-sum extensive-form game, and as such can be solved using the sequence-form linear programming as described in Section 3.2.1. The algorithm computes NE of the restricted game by solving pair of linear programs and using the restricted sets Σ', $\mathcal{H}'$, $\mathcal{Z}'$, $\mathcal{I}'$, and the modified utility function u'.

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

$$\overline{r}'_i(\sigma_i) \leftarrow \begin{cases} r'_i(\sigma_i) & \sigma_i \in \Sigma'_i \\ r'_i(\sigma'_i) \cdot \pi^{\text{DEF}}_i(\sigma_i \setminus \sigma'_i) & \sigma_i \notin \Sigma'_i;\ \sigma'_i = \arg\max_{\sigma''_i \in \Sigma'_i;\ \sigma''_i \sqsubseteq \sigma_i} |\sigma''_i| \end{cases} \tag{5.6}$$

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

In each iteration of the double-oracle algorithm one sequence-form LP is solved for each player; this way the algorithm computes a pair of NE strategies in the restricted game.

We denote these strategies as (r_i^*, r_{-i}^*) and $(\overline{r}_i^*, \overline{r}_{-i}^*)$ when they are extended to the original unrestricted game by the default strategies.

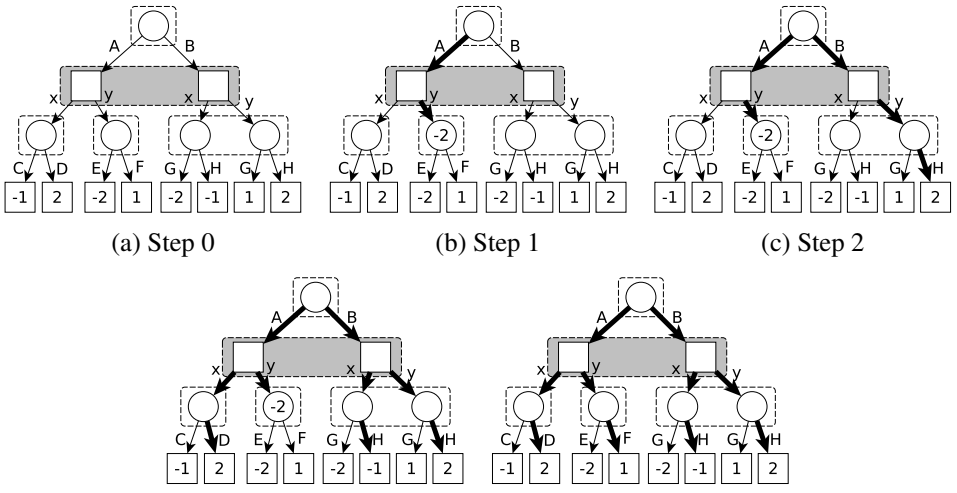
Expanding the Restricted Game The restricted game is expanded by adding new sequences to the set Φ' and updating the remaining sets according to their definition. Moreover, after adding new sequences, the algorithm calculates and keeps the temporary utility values for leafs in $\mathcal{Z}' \setminus \mathcal{Z}$, so they can be used in the sequence-form LP.

After updating the restricted game, the linear programs are modified to correspond to the new restricted game: for each newly added information sets and sequences, new variables are created in the linear programs, and constraints (3.7),(3.9) corresponding to these information sets/sequences are created. Moreover, some of the constraints already existing in the linear program need to be updated. If a sequence σ_i is newly added into the set Σ_i' and the immediate prefix sequence (i.e., sequence $\sigma_i' \sqsubseteq \sigma_i$ such that $|\sigma_i'| + 1 = |\sigma_i|$) has already been a part of the restricted game, then we need to update constraints (3.9) for information set I_i for which $\sigma_i' = \mathsf{seq}_i(I_i)$ to ensure the consistency of the strategies, and constraints (3.7) corresponding to sequence σ_i'. Moreover, the algorithm needs to update constraints (3.7) assigned to the sequences of the opponent σ_{-i}, for which $g(\sigma_i, \sigma_{-i}) \neq 0$. Finally, the algorithm needs to update all constraints that used temporary utility values of leafs that become internal nodes in this iteration so updated values of the utility function u' are reflected in the LP.

New sequences for each player are calculated by best response sequence algorithms (BRS). The details of the calculation are described in Section 5.2.2. From the perspective of the sequence-form double-oracle algorithm, the BRS algorithm calculates a pure best response for player i against a fixed strategy of the opponent in the unrestricted game. This pure best response specifies an action to play in each information set that is currently reachable, given the opponent's extended strategy $\overline{r}_{-i}^*$, in the form of pure realization plan r_i^{BR} (i.e., a realization plan that assigns only integer values 0 or 1 to the sequences). Note that this pure best response realization plan is not necessarily a pure strategy in the original unrestricted game, because there does not have to be an action specified for every information set. Specifically, there is no action specified for information sets that are not reachable (1) due to choices of player i, and (2) due to zero probability in the realization plan of the opponent $\overline{r}_{-i}^*$. Omitting these actions does not affect value of the best response because these information sets are never reached; hence, for r_i^{BR} it holds $\forall \overline{r}_i' \in \Pi_i \; u_i(r_i^{\text{BR}}, \overline{r}_{-i}^*) \geq u_i(\overline{r}_i', \overline{r}_{-i}^*)$ and there exists a pure best response strategy $\pi_i^{\text{BR}} \in \Pi_i$ such that $u_i(r_i^{\text{BR}}, \overline{r}_{-i}^*) = u_i(\pi_i^{\text{BR}}, \overline{r}_{-i}^*)$. The sequences that are used in the best-response pure realization plan with probability 1 are returned by BRS algorithm and we call these sequences to be the *best-response sequences*:

$$\{\sigma_i \in \Sigma_i \; : \; r_i^{\text{BR}}(\sigma_i) = 1\} \tag{5.7}$$

Example We now demonstrate the sequence-form double-oracle algorithm on an example game depicted in Figure 5.2a. In our example, there are two players, circle and box; circle aims to maximize the utility value in the leafs, box aims to minimize the utility value.



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

(d) Step 3 (e) Step 4

Figure 5.2: Example of the steps of the sequence-form double-oracle algorithm in a two-player zero-sum game, where circle player aims to maximize the utility value, box aims to minimize the utility value. Bold edges correspond to the sequences of actions added into the restricted game. The dashed boxes indicate the information sets.

Finally we assume that choosing the leftmost action in each information set is the default strategy for both players in this game.

The algorithm starts with an empty set $\Phi' \leftarrow \emptyset$; hence, the algorithm sets the current pair of $(\overline{r}_i^*, \overline{r}_{-i}^*)$ strategies to be equivalent to $(\pi_i^{\text{DEF}}, \pi_{-i}^{\text{DEF}})$. Next, the algorithm adds new sequences that correspond to pure realization plan best response to the default strategy of the opponent; in our example the best response sequences for the circle player are $\{\emptyset, \mathcal{A}, AD\}$, and $\{\emptyset, y\}$ for the box player. These sequences are added to the set of allowed sequences Φ'. Next, the set of sequences of the restricted game Σ' is updated. The maximal compatible set of sequences from set Φ' cannot contain sequence AD because the compatible sequence of the box player (i.e., x in this case) is not allowed in the restricted game yet ($x \notin \Phi'$) and sequence AD cannot be executed in the full length. Moreover, by adding sequences $\mathcal{A}$ and y, the restricted game will contain node Ay for which actions E and F are defined in the original unrestricted game. However, there is no continuation in the current restricted game yet; hence, this node is a leaf, belongs to $\mathcal{Z}' \setminus \mathcal{Z}$, and the algorithm needs to define a new value for a modified utility function u' for this node. The value $u'(Ay)$ is equal to -2 and corresponds to the outcome of the game if the circle player continues by playing the default strategy and the box player plays the best response. To complete the first step of the algorithm we summarize the nodes and information sets included in the restricted game; $\mathcal{H}'$ contains 3 nodes (the root, the node after playing an action $\mathcal{A}$ and the node Ay), and two information sets (the information set for node Ay is not added into the restricted game, because this node is now a leaf in the restricted game). The situation is depicted in Figure 5.2b, the sequences in Σ' are shown as bold edges.

The algorithm proceeds further and the complete list of steps of the algorithm is summarized in Table 5.1. In the second iteration, new sequences B and BH are added into the restricted game (box player does not add new sequences in this iteration). In the third iteration the situation changes and box player adds new sequence x, while there are no new allowed sequences for the circle player. Note that after adding sequence x, sequence AD also becomes a part of the set $\Sigma'_{\bigcirc}$ as it can now be executed in the full length thanks to sequence x. In the fourth iteration the algorithm adds sequence AF to the restricted game, which removes the assigned value $u'(Ay)$ since the node no longer belongs to set $\mathcal{Z}'$. The algorithm stops after four iterations, no other sequences are added into the restricted game, and the solution of the restricted game can be translated to the solution in the original unrestricted game – i.e., $(\overline{r}^*_i, \overline{r}^*_{-i})$ is Nash equilibrium of the original game.

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

Table 5.1: Steps of the sequence-form double-oracle algorithm in our example game.

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

5.2.2 Best-Response Sequence Algorithms

As indicated above, the purpose of the best-response sequence (BRS) algorithm is to generate new sequences that will be added to the restricted game in the next iteration, or to prove that no more sequences need to be added to the restricted game. Throughout this section we use the term *searching player* to represent the player for whom the algorithm computes the best response sequences. We denote this player as i. The BRS algorithm calculates the expected value of a pure best response to the opponent's strategy $\overline{r}^*_{-i}$. The algorithm returns both the set of best-response sequences as well as the expected value of the strategy against the extended strategy of the opponent.

The algorithm is based on a depth-first search that traverses the original unrestricted game tree. The behavior of the opponent $-i$ is fixed to the strategy given by the extended realization plan $\overline{r}^*_{-i}$. To save computational time, best-response algorithms use methods

Algorithm 5 BRS_i in the nodes of other players.

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

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

of branch and bound during the search for best-response sequences. The algorithm uses a bound on the expected value for each inner node, denoted as λ. This bound represents the minimal utility value that the node currently being evaluated needs to gain in order to be a part of a best-response sequence. Using this bound during the search, the algorithm is able to prune branches that will certainly not be a part of any best-response sequences. For the root node the bound is set to *MinUtility*.

We distinguish 2 cases in the search algorithm: either the algorithm is evaluating an information set (or more specifically a node h) assigned to the searching player i, or the node is assigned to one of the other players (either to the opponent (player $-i$), or it is a chance node). The pseudocode for these two cases is depicted in Algorithms 5 and 6.

Nodes of the Other Players We first describe the case when the algorithm evaluates a node h assigned either to the opponent of the searching player, or to Nature player (see Algorithm 5). The main idea is to calculate an expected utility value for this node according to the (fixed) strategy of the player. Note that the strategy is known, because it is either given by the extended realization plan $\overline{r}'_{-i}$ or by stochastic environment ($\mathcal{C}$). Throughout the algorithm, variable w represents the probability of this node based on realization probability of the opponent and stochastic environment (line 1). This value is iteratively decreased by

values w_a that represent realization probabilities of currently evaluated action $a \in \mathcal{A}(h)$. Finally, v_h is the expected utility value for this node.

The algorithm evaluates the actions in the descending order according to the probability of being played (based on $\overline{r}'_{-i}$ and $\mathcal{C}$; lines 9–21). First, we calculate a new lower bound λ' for the successor ha (line 10). The new lower bound is the minimal value that must be returned from the recursive call $\text{BRS}_i(ha)$ under the optimistic assumption that all the remaining actions will yield the maximum possible utility. If the lower bound does not exceed the maximum possible utility in the game, the algorithm recursively the successor (line 12). Note that the algorithm does not evaluate the branches with zero realization probability (line 9).

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

Nodes of the Searching Player In nodes assigned to the searching player, the algorithm (depicted in Algorithm 6) evaluates every action in each state that belongs to the current information set. The algorithm traverses the states in descending order according to the probability of occurrence given the strategies of the opponent and Nature (line 8). Similar to the previous case, in each iteration the algorithm calculates a new lower bound for successor nodes (line 17). The new lower bound is the minimal value that must be returned from the recursive call $\text{BRS}_i(h'a)$ in order for the action a to be selected as the best action for this information set, under the optimistic assumption that this action yields the maximum possible utility value after applying it in each of the remaining states in this information set. The algorithm performs a recursive call (line 20) only for an action that still could be the best in this information set (i.e., the lower bound does not exceed the maximal possible utility in the game). Note that if a cut-off occurs in one of the successors, this action can no longer be the best action, v_a is set to $-\infty$, and will not be evaluated for any of the remaining nodes. Overall, when the algorithm determines which action will be selected as the best one in this information set, it evaluates only this action for all remaining nodes in the information set. Finally, the algorithm stores the values for the best action for all nodes in this information set (line 30). These are reused if the same information set is visited again (i.e., the algorithm reaches a different node h' from the same information set I_i; line 5).

A cut-off occurs, when the maximal possible value v_a^h is smaller than the lower bound λ after evaluating node h. Therefore, regardless which action will be selected as the best action in this information set, the lower bound for node h will not be reached; hence, the cut-off occurs (line 27). Note that if a cut-off occurs in an information set, this information set cannot be reached again and the sequences of the searching player leading to this infor-

Algorithm 6 BRS$_i$ in the nodes of the searching player.

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

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

mation set cannot be a part of the best response. This is due to propagating the cut-off to at least one previous information set of the searching player, otherwise there will be no tight lower bound set (the bound is first set only in the information sets of the searching player). Therefore, there exists at least one action of the searching player that will never be evaluated again (after a cut-off, the value v_a for this action is set to $-\infty$) and cannot be selected as the best action in the information set. Due to perfect recall, all nodes in information set I_i share the same sequence of actions $\mathsf{seq}_i(I_i)$; hence, no node $h' \in I_i$ can be reached again.

5.2.3 Main Loop Alternatives

We now introduce several alternative formulations for the main loop of the sequence-form double-oracle algorithm. The general approach in double oracle is to solve the restricted

game, get the optimal strategy for each of the players, and then compute the best responses for both of the players before resolving the restricted game. However, the extensive-form LP is formulated in our double-oracle scheme in such that we can solve the restricted game from the perspective of a single player i. In other words, we formulate a single LP as described in Section 3.2.2 that computes the optimal strategy of the opponent in the restricted game (player $-i$), and then computes the best response of player i to this strategy. This means that on each iteration we can select player i, for which we calculate the best response in this iteration. We call this selection process the *player-selection policy*.

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

$$\arg\max_{i \in N} \left| \mathcal{V}_i^{UB} - \mathcal{V}_i^{LP} \right| \tag{5.8}$$

where $\mathcal{V}_i^{LP}$ is the last calculated value of the restricted game for player i. The intuition behind this choice is that either this bound is precise and there are some missing sequences of this player in the restricted game that need to be added, or the upper bound is overestimated. In either case, the best-response sequence algorithm should be run for this player in the next iteration, either to add new sequences or to tighten bound. In case of a tie, the alternating policy is applied in order to guarantee regular switching of the players. We experimentally compare these policies to show their impact on the overall performance of the sequence-form double-oracle algorithm (see Section 5.4).

5.3 Theoretical Analysis

In this section we prove that our sequence-form double-oracle algorithm will always converge to a Nash equilibrium of the complete game. First, we formally define the strategy computed by the best-response sequence (BRS) algorithm, then we prove lemmas about the characteristics of the BRS strategies, and finally we prove the main convergence result. Note that modifications of the main loop described in Section 5.2.3 do not affect the correctness of the algorithm as long as the player-selection policy ensures that if no improvement is made by the BRS algorithm for one player that the BRS algorithm is run for the opponent on the next iteration.

Lemma 5.3.1 *Let r'_{-i} be a realization plan of player $-i$ on some restricted game G'. $BRS(r'_{-i})$ returns sequences corresponding to realization plan r_i^{BR} in the unrestricted game, such that r_i^{BR} is a part of a pure best response strategy to $\overline{r}'_{-i}$. The value returned by the algorithm is the value of executing the pair of strategies $u_i(\overline{r}'_{-i}, r_i^{BR})$.*

Proof $BRS(r'_{-i})$ searches the game tree and takes the action that maximizes the value of the game for player i in all information sets I_i assigned to player i reachable given the strategy of the opponent $\overline{r}'_{-i}$. In the opponent's nodes, it takes the expected value of playing r'_{-i} where it is defined and the value of playing the pure action of the default strategy π^{DEF}_{-i} where r'_{-i} is not defined. In chance nodes, it returns the expected value of the node as the sum of successors weighted by their probabilities. In each node h, if the successors have the maximal possible value for i then also node h has the maximal possible value for i (when playing against $\overline{r}'_{-i}$). The selections in the nodes that belong to i achieves this maximal value; hence, they form a best response to strategy $\overline{r}'_{-i}$. □

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

Lemma 5.3.2 *Let r'_{-i} be a realization plan of player $-i$ on some restricted game G' and let $\mathcal{V}_i^*$ be the value of the original unrestricted game G for player i, then*

$$v(BRS(r'_{-i})) \geq \mathcal{V}_i^*. \tag{5.9}$$

Proof Lemma 5.3.1 showed that $v(BRS(\overline{r}'_{-i}))$ is a value of the best response against $\overline{r}'_{-i}$ which is a valid strategy in the original unrestricted game G. If $v(BRS(r'_{-i})) < \mathcal{V}_i^*$ then $\mathcal{V}_i^*$ cannot be the value of the game since player $-i$ has a strategy $\overline{r}'_{-i}$ that achieves better utility, which is a contradiction. □

Lemma 5.3.3 *Let r'_{-i} be a realization plan of player $-i$ on returned by the LP for some restricted game G' and let $\mathcal{V}_i^{LP}$ be the value of the restricted game returned by the LP, then*

$$v(BRS(r'_{-i})) \geq \mathcal{V}_i^{LP}. \tag{5.10}$$

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

Case 1: The best response strategy uses an action in a temporary leaf of G'. Player i makes the decision in the leaf, because otherwise the value of the temporary leaf would be directly returned by BRS. The value of the temporary leaf has been under-estimated for i in the LP computation and it is over-estimated in the BRS computation as the best response to the default strategy π^{DEF}_{-i}. The value of the best response can only increase by including this action.

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

Lemma 5.3.4 *Under the assumptions of the previous lemma, if $v(BRS(r'_{-i})) > \mathcal{V}_i^{LP}$ then it returns sequences that extend game G' in the next iteration.*

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

Note, that Lemmas 5.3.2 and 5.3.4 would not hold if the utility values u' for temporary leafs ($h \in \mathcal{Z}' \setminus \mathcal{Z}$) are set arbitrarily. Currently, the algorithm sets the values in temporary leaf h as if the player $\rho(h)$ continues by playing the default strategy and the opponent ($-\rho(h)$) is playing the best response. If the utility values for the temporary leafs are set arbitrarily and used in BRS algorithms to speed-up the calculation as proposed (see the algorithm in Figure 5, line 5), then the Lemma 5.3.2 does not need to hold in cases the value in node h strictly overestimates the optimal expected value for player $\rho(h)$. In this case, the best-response value of the opponent may be lower than the optimal outcome, $v\left(BRS(r_{\rho(h)})\right) < \mathcal{V}^*_{-\rho(h)}$. On the other hand, if the BRS algorithm does not use the cached values u' for such a node, then Lemma 5.3.4 is violated because the best-response value will be strictly higher for player $-\rho(h)$ even though no new sequences are to be added into the restricted game.

Theorem 5.3.5 *The double-oracle algorithm for extensive form games described in the previous section stops if and only if*

$$v(BRS(r'_{-i})) = -v(BRS(r'_i)) = \mathcal{V}_i^{LP} = \mathcal{V}_i^*, \tag{5.11}$$

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

Proof First we show that the algorithm continues until all equalities (5.11) hold. If $v(BRS(r'_{-i})) \neq -v(BRS(r'_i))$ then from Lemma 5.3.2 and Lemma 5.3.4 we know that for some player i it holds that $BRS(r'_{-i}) > \mathcal{V}_i^{LP}$, so the restricted game in the following iteration is larger by at least one action and the algorithm continues. In the worst case, the restricted game equals the complete game $G' = G$, and it cannot be extended any further. In this case the BRS cannot find a better response then v_i^* and the algorithm stops due to Lemma 5.3.4.

If the condition in the theorem holds the algorithm has found a NE in the original unrestricted game, because from Lemma 5.3.1 we know that $r_{-i}^{BR} = BRS(r'_i)$ is the best response to $\overline{r}'_i$ in the original unrestricted game. However, if the value of the best response to a strategy in a zero-sum game is the value of the game, then the strategy is optimal and it is part of a Nash equilibrium of the game. □

5.4 Experimental Evaluation

We now turn to the experimental evaluation of the performance of the double oracle algorithm for EFG. The full sequence-form LP is used as a baseline (referred to as FULLLP from now on). The purpose of the experiments is twofold. Primarily, the experimental results demonstrate the improvement of the double-oracle algorithm in practice on several games compared to the baseline FULLLP. The experiments also test the impact of the different variants of the main loop of the algorithm described in Section 5.2.3.

We compare three variants of the sequence-form double-oracle algorithm: (1) DOB is a variant in which the best-responses are calculated for both players in each iteration; (2) DOSA calculates the best-response for a single player on each iteration according to the simple alternating policy; and (3) DOSWP is a variant in which the best-response is calculated for a single player according to the worse-player selection policy. For all of the variants of the double-oracle algorithm we used the same default strategy where the first action applicable in a state is played by default. The ordering of the actions is randomized and does not use any domain-specific knowledge unless stated otherwise.

Since there is no standardized collection of zero-sum extensive-form games for benchmark purposes, we use several specific games to evaluate the double-oracle algorithm and identify the strengths and weaknesses of the algorithm. The games were selected to evaluate the performance of the double-oracle algorithm under different conditions; the games differ in maximal utility the players can gain, in cause of the imperfect information, and structure of the information sets. One of the key characteristics that clearly affects the performance of the algorithm is the relative size of the support of Nash equilibria (i.e., the number of sequences used in NE with non-zero probability). If there is no NE with small support, the algorithm must necessarily add a large fraction of the sequences into the restricted game in order to find the solution, mitigating the advantages of the iterative solution approach. Generally, the relative performance of the double-oracle algorithm compared to the FULLLP depends on other characteristics of the games, however, yet unknown. The experimental evaluation should help to identify some of the limitations of the current version of the algorithm and suggest directions for future work. We present the results on two specific games where the double-oracle significantly outperforms the FULLLP on all instances: a search game motivated by border patrol and phantom tic-tac-toe. We also present results on a simplified versions of Poker, in which double-oracle algorithm does not always improve the computation time. However, the FULLLP has a limited scalability due to larger memory requirements and cannot find solutions for larger variants of Poker, while the double-oracle algorithm is able to solve these instances.

Our principal interest is in developing new generic methods for solving extensive-form games. Therefore, we implemented the algorithm in a generic framework for modeling arbitrary extensive-form games. The algorithms do not use any domain-specific knowledge in the implementation, nor any specific ordering of the actions. The drawback of this generic approach are higher memory requirements and additional overhead for the algorithms. A domain-specific implementation could improve the performance by eliminating some of the auxiliary data structures. We run all of the experiments using a single thread on an Intel i7 CPU running at 2.8 GHz. Each of the algorithms was given at maximum of 10 GB of

memory for Java heap space. We used IBM CPLEX 12.5 for solving the linear programs, with parameter settings to use a single thread and the barrier solution algorithm.

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

5.4.1 Experiment Domains

Search Games

Our first test belong to class of search (or pursuit-evasion) games, often used in experimental evaluation of double-oracle algorithms (McMahan et al., 2003; Halvorson et al., 2009) The search game has two players: the *patroller* (or the defender) and the *evader* (or the attacker). The game is played on a directed graph (see Figure 5.3), where the evader aims to cross safely from a starting node (E) to a destination node (D). The defender controls two units that move in the intermediate nodes (the shaded areas) trying to capture the evader by occupying the same node as the evader. During each turn both players move their units simultaneously from the current node to an adjacent node or stay in the same location. The only exception is that the evader cannot stay in the two leftmost nodes. If a pre-determined number of turns is made without either player winning, the game is a draw. This is an example of a win-tie-loss game and the utility values are from a set $\{-1, 0, 1\}$.

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

Figure 5.3 shows examples of the graphs used in the experiments. The patrolling units can move only in the shaded areas (P1,P2), and they start at any node in the shaded areas. Even though the graph is small, the concurrent movement of all units implies a large branching factor (up to ≈ 50 for one turn) and thus large game trees (up to $\approx 10^{11}$ nodes). In the experiments we used three different graphs, varied the maximum number of turns of the game (from 3 to 7), and whether the attacker has the option to perform slow moves (labeled *SA* if the slow moves are allowed, and *SD* otherwise).

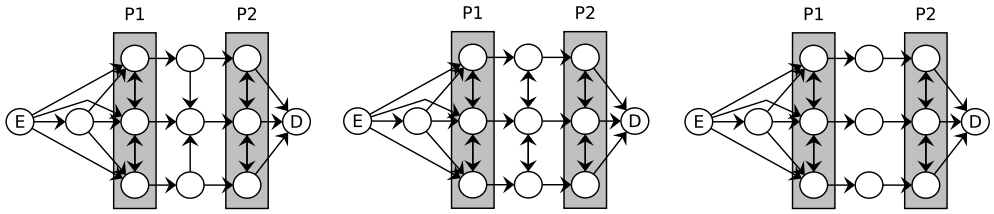


Figure 5.3: Three variants of the graph used in the experiments; we refer to them as G1 (left), G2 (middle), and G3 (right).

Phantom Tic-Tac-Toe

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

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

Poker Games

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

In our version of poker each player starts with the same amount of chips, and both players are required to put some number of chips in the pot (called the *ante*). In the next step the Nature player deals a single card to each player (the opponent is unaware of the card), and the betting round begins. A player can either *fold* (the opponent wins the pot), *check* (let the opponent make the next move), *bet* (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, the Nature player deals one 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.

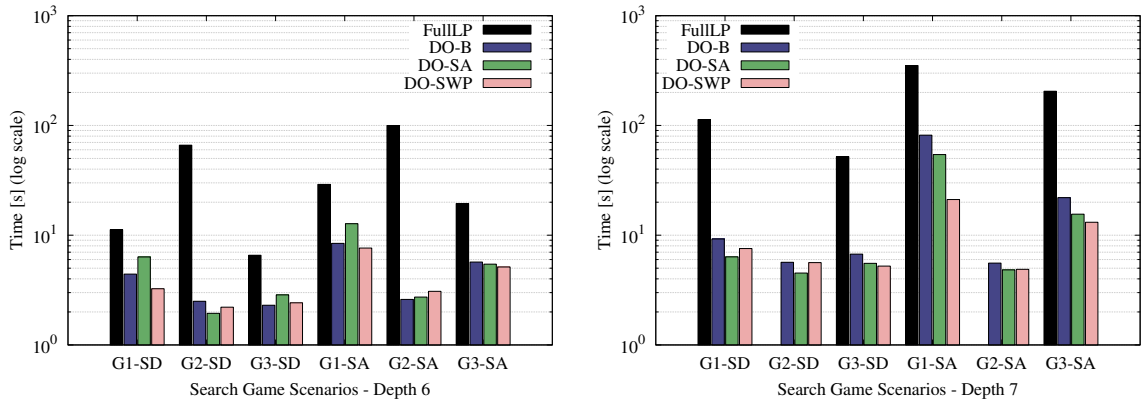


Figure 5.4: Comparing the running time on 3 different graphs with either slow moves allowed (SA) or disallowed (SD), with the depth set to 6 (left subfigure) or 7 (right subfigure). Missing values for FULLLP algorithm indicates that the algorithm run out of memory.

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

5.4.2 Results

Search Games

The results for the search game scenarios show that the sequence-form double-oracle algorithm is particularly successful when applied on games where NE with extremely small support exist. Figure 5.4 shows the comparison of the running times for FULLLP and variants of the double-oracle algorithm (note the logarithmic y-scale). All variants of the double-oracle algorithm are several magnitudes faster. This is most apparent on the fully-connected graph (G2) that generates the largest game tree. When the slow moves are allowed and the depth is set to 6, it takes almost 100 seconds for FULLLP to solve the instance of the game but all variants of the double-oracle algorithms solve the game in less than 3 seconds. Moreover, when the depth is increased to 7, FULLLP was unable to solve the game due to the memory constraints, while the fastest variant DOSWP solved the game in less than 5 seconds. Similar results were obtained on other graphs as well. The graph G1 resulted in the game-tree that was the most difficult for the double-oracle algorithm: when the depth is set to 7, it takes almost 6 minutes for FULLLP to solve the instance, while the fastest variant DOSWP solved the game in 21 seconds. The reason is that even though the game tree is not the largest, there is a more complex structure of the information sets. This is due to limited compatibility among sequences of the players (when the patroller P1 observes the

tracks in the top-row node, the second patroller P2 can capture the evader in the top, or in the middle-row node).

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

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

Table 5.2: Table of the running times for different components of the double-oracle algorithm, iterations, and size of the restricted game in terms of the number of sequences compared to the size of the complete game. The results are shown for scenario G1, depth 7, and allowed slow moves.

Table 5.2 shows a breakdown of the cumulative computational time spent in the different components of the double-oracle algorithm. The results show that due to the size of the game, the computation of the best-response sequences takes the majority of the time (typically around 75% on larger instances), while creating the restricted game and solving it takes only a small fraction of the total time. It is also noticeable that the size of the final restricted game is very small compared to the original game, since the number of sequences for the second player (the defender) is lower than 1% (there is 273099 sequences of the defender).

Finally, we analyze the convergence rate of the variants of the double-oracle algorithm. The results are depicted in Figure 5.5, where the size of the interval given by bounds $\mathcal{V}_i^{UB}$ and $\mathcal{V}_i^{LB}$ defines the current error of the double-oracle algorithm as $|\mathcal{V}_i^{UB} - \mathcal{V}_i^{LB}|$. Moreover, the convergence rate of the CFR algorithm is also depicted. The error of CFR is calculated similarly – i.e., as a sum of the best-response values to the strategies from CFR algorithm. We can see that all variants of the double-oracle algorithm perform very similar – the error very quickly drops to 1 and few iterations later each version of the algorithm quickly converges to an exact solution. This results show that in this game the double-oracle algorithm can very quickly find the correct sequences of actions and compute an exact solution, in spite of the size of the game. In comparison, CFR algorithm can also quickly learn the correct strategies in most of the information sets, however, the convergence has a very long tail. After 200 seconds, the error of the CFR was equal to 0.0657 and it was dropping very slowly (0.0158 after 1 hour). The error of CFR is quite significant considering the value of the game in this case ($-0.\overline{3333}$).

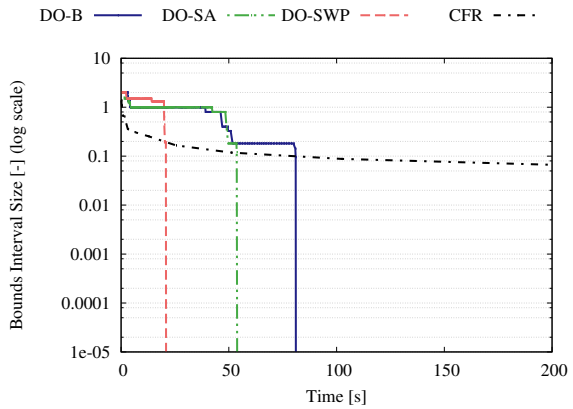


Figure 5.5: Convergence of variants of double-oracle algorithm and CFR algorithm on the search game domain: y-axis displays the current approximation error.

Phantom Tic-Tac-Toe

The results on Phantom Tic-Tac-Toe confirm that this game is also suitable for the sequence-form double-oracle algorithm. Due to the size of the game, both baseline algorithms (the FULLLP and CFR) ran out of memory and were not able to solve the game[1]. Therefore, we only compare the times for different variants of the double-oracle algorithm. Figure 5.6 (left subfigure) shows the overall performance of all three variants of the double-oracle algorithm as a convergence graph. We see that the performance of two of the variants is similar, with the performance of DOSA and DOSWP almost identical. On the other hand, the results show that DOB converges significantly slower.

Algorithm	Overall [s]	Core LP [s]	BR [s]	Validity [s]	Iterations	$\lvert\Sigma'_1\rvert(\frac{\lvert\Sigma'_1\rvert}{\lvert\Sigma_1\rvert})$	$\lvert\Sigma'_2\rvert(\frac{\lvert\Sigma'_2\rvert}{\lvert\Sigma_2\rvert})$
FULLLP	N/A	–	–	–	–	–	–
DOB	21197	2635	17562	999	335	7676 (0.60%)	10095 (0.23%)
DOSA	17667	2206	14560	900	671	7518 (0.59%)	9648 (0.22%)
DOSWP	17589	2143	14582	864	591	8242 (0.65%)	8832 (0.20%)

Table 5.3: Comparing the running time for different components of the double-oracle algorithm for the game of phantom tic-tac-toe.

The time breakdown of the variants of double-oracle algorithm is shown in Table 5.3. Similarly to the previous case, the majority of the time ($\approx 83\%$) is spent in calculating the best responses. Out of all variants of the double-oracle algorithm, the DOSWP variant is the fastest one. It converged in significantly fewer iterations compared to the DOSA variant (iterations are twice as expensive in the DOB variant). Similarly to the search game, the

[1]Based on the personal communication with author of (Lanctot et al., 2012), a hand-tuned version of CFR for solving Phantom Tic-Tac-Toe took more than 100 hours to reach error 0.01.

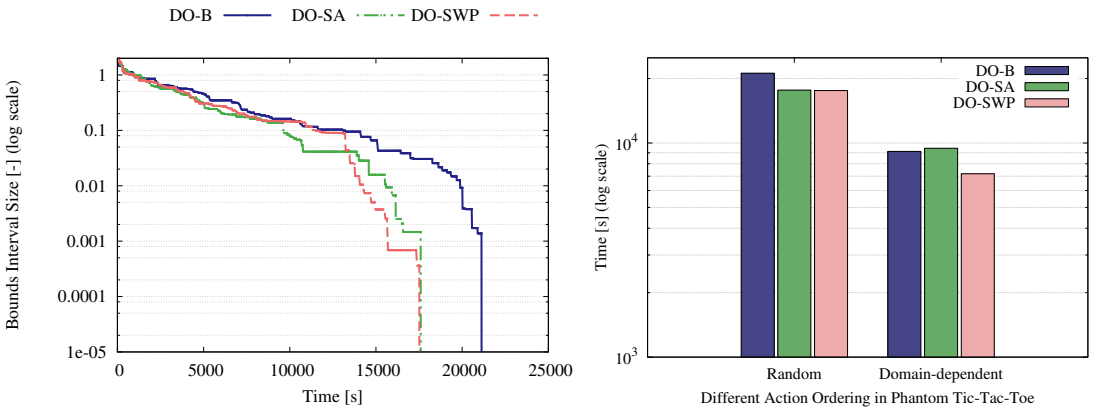


Figure 5.6: (left) Comparison of the convergence rate of the double-oracle variants for Phantom Tic-Tac-Toe (the interval size correspond to value $|\mathcal{V}_i^{UB} - \mathcal{V}_i^{LB}|$);
(right) Comparison of the performance of the double-oracle variants for Phantom Tic-Tac-Toe when domain-specific move ordering and default strategy is used.

final restricted game is extremely small and less than 1% of all sequences is added into the restricted game.

Finally, we demonstrate the potential of combining the sequence-form double oracle algorithm with a domain-specific knowledge. Every variant of the double-oracle algorithm can use a move ordering based on a domain-specific heuristics. Move ordering determines the default strategy (recall our algorithm uses the first action as the default strategy for each player), and the direction of the search in the best response algorithms. By replacing the randomly generated move ordering with a heuristic one the results show a significant improvement in performance of all variants (see Figure 5.6, right subfigure) even though the rest of the algorithms is not changed. Each of the variant was able to solve the game in less than 3 hours and it took 2 hours for the fastest DOSWP variant.

Poker

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

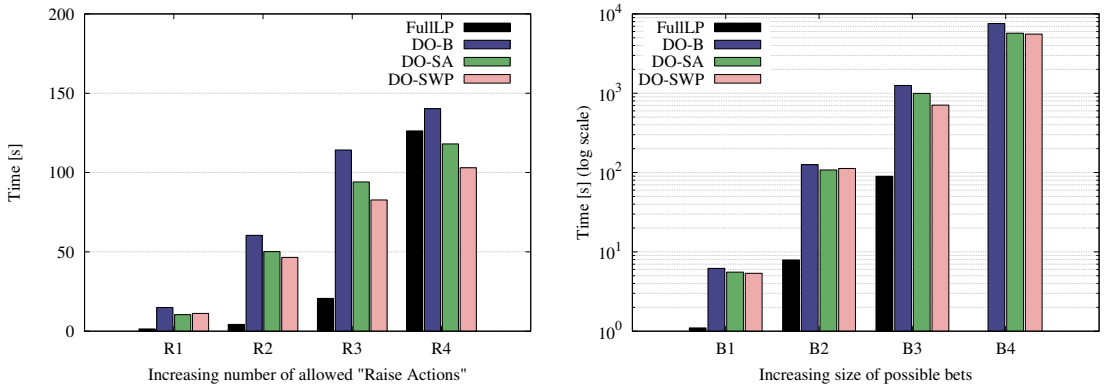


Figure 5.7: Comparison of the algorithm on different versions of simplified Poker game. The left sub-figure shows the computation times with an increasing number of raise actions allowed, and the right sub-figure shows the computation times with an increasing number of different bet sizes for raise/bet actions.

Figure 5.7 shows the selected results for simplified Poker variants with increasing size of the game. The results in the left subfigure show the computation time with increasing depth of the game by allowing the players to re-raise (i.e., the players are allowed to re-raise their opponent for a certain number of times). The remaining parameters were fixed to 3 types of cards, 2 cards of each type, and 2 different betting sizes. The size of the game grows exponentially, with the number of all sequences increasing to 210937 for each player for the $R4$ scenario. The computation time of the FULLLP copies the exponential grow and increases exponentially with the increasing depth (note that there is a standard y scale). On the other hand, the increase is less dramatic for all of the variants of the double-oracle algorithm. Moreover, the DOSWP variant double-oracle algorithm is the fastest in the largest scenario – while FULLLP solved the instance in 126 seconds, it took only 103 seconds for the DOSWP. Finally, FULLLP is not able to solve the games if we increase the length to $R5$ due to memory constraints, while the computation time of all double-oracle algorithms increase only marginally.

The right sub-figure of Figure 5.7 shows the increase in computation time with an increasing number of different bet sizes for raise/bet actions. The remaining parameters were fixed to 4 types of cards, 3 cards of each type, and 2 raise actions were allowed. Again, the game grows exponentially with the increasing branching factor. The number of all sequences increases exponentially up to 685125 for each player for the $B4$ scenario, and the computation time of all algorithms increases exponentially (note the logarithmic y-scale) as well. The results show that even with the increasing branching factor, the double-oracle variants tend to be slower than solving the FULLLP. However, while the FULLLP ran out of memory for the largest $B4$ setting, all of the double-oracle variants were able to find the exact solution using less memory.

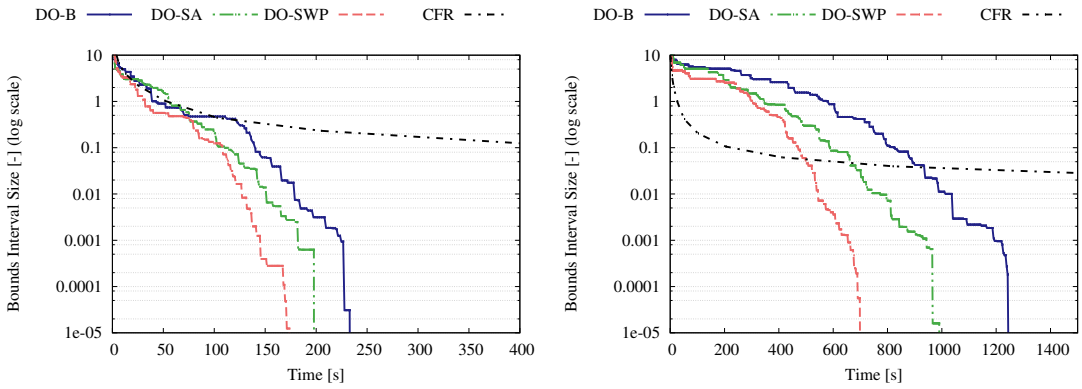


Figure 5.8: Comparison of the convergence of the variants of the double-oracle algorithm and CFR algorithm for a two variants of Poker game with 4 types of cards, and 3 cards of each type. There are 4 raise actions allowed, 2 different bet sizes in the left subfigure; there are 2 raise actions allowed, 3 different bet sizes in the right subfigure.

Algorithm	Overall [s]	Core LP [s]	BR [s]	Validity [s]	Iterations	$\lvert\Sigma'_1\rvert(\frac{\lvert\Sigma'_1\rvert}{\lvert\Sigma_1\rvert})$	$\lvert\Sigma'_2\rvert(\frac{\lvert\Sigma'_2\rvert}{\lvert\Sigma_2\rvert})$
FULLLP	278.18	–	–	–	–	–	–
DOB	234.60	149.32	56.04	28.61	152	6799 (1.81%)	6854 (1.83%)
DOSA	199.24	117.71	51.25	29.59	289	6762 (1.80%)	6673 (1.78%)
DOSWP	182.68	108.95	48.25	24.8	267	6572 (1.75%)	6599 (1.76%)

Table 5.4: Comparing the running time for different components of the double-oracle algorithm, iterations, and size of the restricted game in number of sequences compared to the size of the complete game. The results are shown for scenario with 4 raise actions allowed, 2 different betting values, 4 types of cards, and 3 cards of each type.

Comparing the different variants of the double oracle algorithm using a convergence graph (see Figure 5.8) and a decomposition of the computation times (see Table 5.4) shows that DOSWP is the fastest variant in the selected scenario (and almost in all of Poker scenarios). Decomposition of the overall time shows that the majority of the computational time is spent in solving the restricted game LP (up to 65%). The decomposition also shows that DOSWP is typically faster due to lower number of iterations. Also, the final size of the restricted game is typically the smallest. On average through all instances of the Poker games, DOSA spends 86.57% of the computation time of DOB, DOSWP spends 82.3% of the computation time.

Convergence in Poker games is slower compared to search games of similar size (note the logarithmic scale in Figure 5.8). Comparing the double-oracle algorithm variants with CFR yields an interesting result in the left subfigure. Due to the size of the game, the speed of the CFR convergence is nearly the same as for the double-oracle algorithms during the first iterations. However, while the double-oracle algorithms continue in the convergence at the same rate and are able to find an exact solution, the error of the CFR algorithm is decreasing only slowly. In the scenario depicted in the left subfigure, CFR algorithm converged to an error of 0.1212 (the value of the game in this case is ≈ -0.09963) after

400 seconds. After 1 hour, the error dropped to 0.0268. For scenarios with more shallow game trees and larger branching factor, the convergence of CFR is faster at the beginning compared to the double-oracle algorithms (right subfigure of Figure 5.8). However, the main disadvantage of CFR having the long tail remains and the error after 1600 seconds is still over 0.0266 (the value of this game is ≈ -0.09828).

5.5 Discussion of the Results and Summary

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

We also compare the speed of convergence of the double-oracle algorithms with the state-of-the-art approximative algorithm CFR. The results show that CFR quickly approximates the solution during the first iterations, however, the convergence of CFR has a very long tail and CFR is not able to find an exact solution for larger games in a reasonable time. Moreover, for some games the convergence rate of both, double-oracle algorithms and CFR, is similar in during the first iterations. Afterwards, the double-oracle algorithms are able to find an exact solution, while CFR algorithm again suffers from a long-tail convergence and still have a significant error. Recall, that the implementation of CFR has an advantage of having the complete game tree in memory.

Unfortunately, it is difficult to characterize the exact properties of the games for which the double-oracle algorithms perform better than exact or approximative state-of-the-art algorithms. Certainly, the double-oracle algorithm is not suitable for games with large support due to the necessity of large number of iterations. However, having a small support is not a sufficient condition. This is apparent due to two graphs shown in the Poker experiments, where either depth of the game tree or the branching factor was increased. Even though the game grows exponentially and the size of the support decreases to $\approx 2.5\%$ in both cases, the behavior of the double-oracle algorithms is completely different. Our conjecture is that the games with longer sequences suit the double-oracle algorithms better, since several actions that form the best-response sequences can be added during a single iteration. This contrasts with shallow game trees with large branching factors, where more iterations are necessary to add multiple actions. However, a deeper analysis to identify the exact properties of the games that are suitable is a topic for future work.

The advantage over FULLLP is not surprising since the game tree (and thus the size of the game tree) increases exponentially with depth and branching factor. In the next chapter we introduce a double-oracle algorithm that operates on a different class of games, size of which does not increase exponentially. However, also in this setting our algorithm will

prove to be faster and more scalable. Moreover, smaller sizes of games allow us to compare our variants of double-oracle algorithm with the previous variants that iteratively add pure strategies into the restricted game.

Chapter 6

Normal-form Games with Sequential Strategies

Our sequence-form double-oracle algorithm described in the previous chapter significantly builds on the assumption of perfect recall that allows the strategies to be compactly represented using the sequence-form. However, relaxing this assumption has immediate effects on the computational complexity of the problem of finding a Nash equilibrium. First of all, the mixed strategies are more expressive and a player can express more complex behavior using probability distribution over pure strategies than when relying only on behavior strategies (see e.g. (Wichardt, 2008; Waugh et al., 2009; Marple and Shoham, 2012)). Therefore, the algorithms for solving extensive-form games with imperfect recall seek NE in behavioral strategies. However, as shown for example in (Wichardt, 2008), NE might not always exist in these games in behavioral strategies. This issue can either be solved by searching for optimal strategies that guarantee certain payoff against the opponent (e.g. analyzed in (Koller and Megiddo, 1992)), or by defining new modified solution concepts that are guaranteed to exist (e.g. in (Marple and Shoham, 2012)). Computing the best guaranteed strategy is in general a computationally hard task. It has been proven that solving this problem restricting only to pure strategies is Σ_2^P-complete (Koller and Megiddo, 1992)[1]. The computational complexity of the problem in mixed strategies is, to the best of our knowledge, not determined yet. Related computational complexity results in extensive-form games regarding different solution concepts suggest that the problem is at least NP-hard (Letchford and Conitzer, 2010).

In this chapter we study a simplified class of games with imperfect recall, where each player performs a finite sequential plan, but they have only limited information about the actions of the opponent – the players are unable to observe the actions of the opponent until the game terminates. Typical example is a two-player search game, where one player (*evader*) aims to cross an area unobserved, while the second player (*pursuer*) aims to encounter the evader. We term this class as *normal-form games with sequential strategies* (NFGSS). The instances of NFGSS appear in many existing works (sometimes restricted to a single mo-

[1] Σ_2^P-complete class of problems are known to be still in NP even under assumption they have an oracle that can solve coNP problems.

bile agent) including the maritime security (Vanek et al., 2012; Vanek et al., 2010; Vanek, 2013), fare evasion (Yin et al., 2012; Jiang et al., 2013), network interdiction (Washburn and Wood, 1995; Letchford and Vorobeychik, 2013), and security and search games (McMahan et al., 2003; Jain et al., 2011). All these examples share a large underlying graph (e.g., public transport network in large cities) and a time component (e.g., schedules of police units). These factors cause very large strategy space that prohibits us from using standard sequential model of the extensive form. For even the smallest example in the experiments in this chapter, using perfect recall and EFGs leads to over $2 \cdot 10^{10}$ sequences for just a single player. Therefore, disregarding certain information simplifies the strategy space, allows us to scale up, and to solve more realistic scenarios.

The strategy space of each player consists of local states and actions to be performed in each state (these actions are often termed *marginal actions* in related work). Transitions between the states are stochastic; hence, an action leads to one of possible succeeding states. The graph structure of the strategy space induces an exponential number of pure strategies in the game. There are two main approaches to tackle this computational complexity. One is to use compact *network-flow* representation of the strategy space and solve the game using linear programming (e.g., in (McMahan, 2006; Jiang et al., 2013)). Alternative approach is to use the iterative double-oracle method, but with pure strategies (e.g., in (McMahan et al., 2003; Vanek et al., 2012; Vanek et al., 2010; Vanek, 2013)).

Our algorithm combines these two methods and it is to some extent similar to the sequence-form double-oracle algorithm described in the previous chapter. This algorithm, however, further fine-grades the method of expanding the restricted game by working with marginal actions. Moreover, due to relaxed assumption of perfect recall the double-oracle algorithm needs to be modified and having a pure default strategy is no longer sufficient to ensure the correctness of the algorithm. After providing formal definitions specific for NFGSS, we present the double-oracle algorithm for NFGSS termed *compact-strategy double-oracle* algorithm (CS-DO). The experimental section shows the computation speed-up of our novel algorithm. Moreover, relatively small size of the games allows us to compare our CS-DO algorithm to the standard normal-form double-oracle algorithm for this class of games.

6.1 Definition and Background

In NFGSS, the strategy space of a player formally corresponds to a finite-horizon, acyclic Markov decision process (MDP). States of this MDP correspond to the states observable by a player; hence, each player follows a different MDP and we use lower index to indicate the player. We again use $\mathcal{H}_i$ to denote the set of all states in MDP, $\mathcal{A}_i$ to be the set of actions in the MDP applicable in states of player i (we again use $\mathcal{A}_i(h) \subseteq \mathcal{A}_i$ to specify the actions applicable in state h, aligned with the related work we term these actions *marginal actions*). Similarly as for simultaneous-move games we specifically denote stochastic transitions between states and use $\mathcal{T} : \mathcal{H} \times \mathcal{A} \times \mathcal{H} \to \mathbb{R}$ as a function that defines the transition probabilities. Similarly to EFGs, a pure strategy is a selection of an action to play in each state of the MDP, and a mixed strategy is a probability distribution over pure strategies.

Finally, we need to specify utility function for NFGSS. Similarly to EFG, we can define the utility values only for combination of terminal states of players. However a more general approach can be used and we assume that each combination of actions (a_i, a_{-i}) (i.e., one action for each player) applicable in some states h_i and h_{-i} can have assigned a utility value (further termed as *marginal utilities*). The overall expected outcome of the game can be then linearly composed from the utility values assigned to combinations of marginal actions. We denote this utility value as $u\left((h_i, a_i), (h_{-i}, a_{-i})\right)$ and for an expected utility of a mixed strategy profile $\delta = (\delta_i, \delta_{-i})$ we assume:

$$u(\delta) = \sum_{\mathcal{H}_i \times \mathcal{A}_i} \sum_{\mathcal{H}_{-i} \times \mathcal{A}_{-i}} \delta_i(h_i, a_i)\delta_{-i}(h_{-i}, a_{-i}) \cdot \; u\left((h_i, a_i), (h_{-i}, a_{-i})\right) \tag{6.1}$$

where $\delta_i(h_i, a_i)$ represents the probability that state h_i is reached and then action a_i is played in this state when player i follows mixed strategy δ_i.

This assumption follows the from previous works. It has been used already in works by McMahan et al. (McMahan, 2006; McMahan and Gordon, 2007b), and also in the work (Jiang et al., 2013), where a term *separability condition* was used. Note that this assumption does not weaken the theoretic properties of NFGSS model. In case the overall utility is not a linear combination of marginal utilities, the model needs to distinguish and remember histories in states. In this situation, we can still model the game as a NFGSS, and the utility values are only assigned to terminal states/actions at the cost of exponential size of MDPs.

Compact Representation of Strategies

A separable utility function allows the mixed strategies to be compactly represented using a network flow representation, similar to realization plans used in sequence-form for EFGs. We again use $r_i : \mathcal{H}_i \times \mathcal{A}_i \to \mathbb{R}$ to represent the marginal probability of an action being played in a mixed strategy of player i. We also use $r_i(h_i)$ to refer to the probability that the state would be reached following this strategy. Formally, this probability is calculated as the sum of marginal probabilities incoming to state h_i and it must be equal to the sum of the marginal probabilities assigned to the actions played in this state:

$$r_i(h_i) = \sum_{h'_i \in \mathcal{H}_i} \sum_{a'_i \in \mathcal{A}(h'_i)} r_i(h'_i, a'_i) \cdot \mathcal{T}(h'_i, a'_i, h_i) = \sum_{a_i \in \mathcal{A}(h_i)} r_i(h_i, a_i) \qquad \forall h_i \in \mathcal{H}_i \tag{6.2}$$

where $\mathcal{T}(h'_i, a'_i, h_i)$ represents the nature probability that action a'_i played in a state h'_i leads to state h_i, and:

$$0 \leq r_i(h_i, a_i) \leq 1 \qquad \forall (h_i, a_i) \in \mathcal{H}_i \times \mathcal{A}(h_i) \tag{6.3}$$

We assume that each MDP has a starting root state, denoted h_i^r for player i, with probability $r(h_i^r) = 1$. Note that we can have an artificial root state with a single action for domains that require some initial probability distribution over the states.

6.1.1 Solving Normal-Form Games with Sequential Strategies

We solve for a pair of equilibrium strategies r^* using the following optimization problem:

$$\max_{r_i} \min_{r_{-i}} \sum_{h_i,a_i} \sum_{h_{-i},a_{-i}} r_i(h_i,a_i) r_{-i}(h_{-i},a_{-i}) \cdot u\left((h_i,a_i),(h_{-i},a_{-i})\right) \tag{6.4}$$

We can reformulate this objective as a linear program (LP) similar to sequence-form LP presented in Section 3.2.2. Let $v_{h_{-i}}$ be the expected utility value that can be achieved from state h_{-i} for player $-i$. Note that in equilibrium the strategies of both players have the same expected value in their root states (i.e., in $v_{h_i^r}$ and $v_{h_{-i}^r}$), equal to the value of the game $\mathcal{V}^*$. We give the formulation of the LP from the perspective of player i assuming that player $-i$ is playing a best response. This means that player $-i$ selects an action a_{-i} in each state h_{-i} such that the expected utility value of the state $v_{h_{-i}}$ is minimal. Therefore, the expected utility of a state $v_{h_{-i}}$ is smaller than the expected value of each state-action (h_{-i},a_{-i}) combination that consists of immediate expected utility of this action and the expected utility of the succeeding states h'_{-i}, weighted by the probability that this state would be reached $\mathcal{T}(h_{-i},a_{-i},h'_{-i})$. Formally:

$$\max_{r_i} \; v_{h^r_{-i}} \tag{6.5}$$

$$v_{h_{-i}} \leq \sum_{h_i,a_i \in \mathcal{H}_i \times \mathcal{A}_i} r_i(h_i,a_i) \cdot u((h_i,a_i),(h_{-i},a_{-i})) + \sum_{h'_{-i} \in \mathcal{H}_{-i}} v_{h'_{-i}} \mathcal{T}(h_{-i},a_{-i},h'_{-i}) \quad \forall (h_{-i},a_{-i}) \in \mathcal{H}_{-i} \times \mathcal{A}(h_{-i}) \tag{6.6}$$

where r_i satisfies network-flow constraints (6.2)-(6.3). A similar program can be constructed for player $-i$ with a minimizing objective and reversing the inequality in constraint (6.6).

6.2 Compact-Strategy Double-Oracle Algorithm

This section describes the new compact-strategy double-oracle (CS-DO) algorithm. The main idea of the algorithm is to iteratively expand MDP for each player by allowing them to select new actions to play. This differs from the double-oracle algorithms described in the previous chapters that operated on a single game tree, while CS-DO iteratively expands two MDPs separately. The algorithm again initially restricts the players to play a specific *default strategy* in each node of their respective MDPs. The algorithm then gradually relaxes this restriction for specific states based on best-response calculations in the unrestricted game. The main idea of this algorithm is similar to one for EFGs presented in Chapter 5, therefore we omit the full details of the algorithm and focus on the key differences in three components: (1) creating the restricted game and methods for its expansion, (2) methods that keep the restricted game valid, and (3) best-response algorithms. There are two main complications that CS-DO algorithm needs to address. First, the utility values can be defined for any pair of state-action combination; hence, defining a modified utility function to use in the

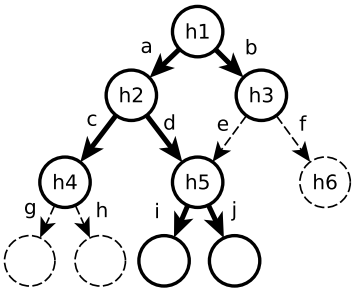


Figure 6.1: An example of a partially expanded deterministic MDP representing a strategy of one player. Solid nodes and edges represent states and actions included in the restricted game, the dashed ones are not included in the restricted game.

restricted game is more complex. Second, due to imperfect recall, the concept of the default strategy is no longer sufficient for ensuring the validity of the restricted game and therefore the correctness of the algorithm.

6.2.1 Restricted Game

The restricted game in CS-DO is defined by the sets of states $\mathcal{H}' \subseteq \mathcal{H}$ and actions $\mathcal{A}' \subseteq \mathcal{A}$ for each player (an example is depicted in Figure 6.1). We say that state h is *included* in the restricted game if $h \in \mathcal{H}'$ (states h_1–h_5 in the example); we say that it is *expanded* if there is some action playable in this state in the restricted game (i.e., $\mathcal{A}'(h) \neq \emptyset$; states h_1,h_2, and h_5 in the example). Finally, we say that action $a \in \mathcal{A}'$ is *fully expanded* if and only if all the succeeding states that can be reached by playing this action (i.e., $h' \in \mathcal{H}$ s.t. $\mathcal{T}(h, a, h') > 0$) are expanded in the restricted game (actions a, d, i, and j in the example). The remaining actions in the restricted game are not fully expanded (actions b and c).

The CS-DO algorithm again assumes a default strategy (e.g, choosing the leftmost action in Figure 6.1) will be played in nodes that are not included, or not expanded in the restricted game. Formally, the default strategy is a strategy[2] $\pi_i^{\text{DEF}} \in \Pi_i$. The restricted game is expanded by adding new states and actions into the sets $\mathcal{H}'$ and $\mathcal{A}'$. In a state added to the restricted game, a player can choose a strategy that differs from the default strategy and that consists of actions added to the restricted game. If there is no action added for some state $h \in \mathcal{H}'$ (i.e., $\mathcal{A}'(h) = \emptyset$), then the default strategy is played in h. The algorithm assumes that whenever action a is added into the restricted game, all immediate successor states are also added into the restricted game (i.e., all states $h' \in \mathcal{H}$ for which $\mathcal{T}(h, a, h') > 0$).

Finally, we again use $\overline{r}$ to represent the *extended strategy* in the complete game, which is created by extending the strategy from the restricted game r with the default strategy.

[2]This strategy can be any arbitrary but fixed, generally mixed strategy. In our approach we use a pure default strategy that selects the first action according to an arbitrary but fixed ordering of $\mathcal{A}(h)$.

Depending on which states and actions are included in the restricted game, we put $\overline{r}(h, a)$ equal to:

- 0, if state h is expanded, but action a is not in $\mathcal{A}'(h)$
- $r(h, a)$, if state h is expanded and action a is in $\mathcal{A}'(h)$
- $\overline{r}(h) \cdot \pi^{\text{DEF}}((h, a)|h)$ otherwise, where the marginal probability is calculated as a product of probability that this state would be reached from its predecessors $\overline{r}(h)$, and probability distribution over actions in this state according to the default strategy calculated as conditional probability $\pi^{\text{DEF}}((h, a)|h)$.

Maintaining a Valid Restricted Game

Naïvely adding new actions and states into the restricted game can make the extended strategies malformed, causing the double-oracle algorithm to converge to an incorrect result. The primary cause of this situation is imperfect recall, since one state can be reached using different combinations of actions. The algorithm needs to ensure that the strategy in the restricted game can always be extended into a well-formed strategy in the complete game as described in the previous section – i.e., to keep the restricted game valid. Recall that the concept of default strategies was sufficient for the double-oracle algorithm extensive-form games described in the previous chapter to keep the restricted game valid precisely due to the assumption of perfect recall.

An example of such situation is depicted in Figure 6.1. Recall that choosing the leftmost action is the default strategy in our example and the strategy in the restricted game is a network flow on the bold actions. Expanding the strategy from the restricted game into the complete game violates the network-flow constraints in state h_5 in this example. According to the LP of the restricted game, the sum of marginal probabilities of actions i and j equals the marginal probability of action d. However, the overall incoming probabilities into the state h_5 is equal to $d + b$ in the extended strategy due to playing the default strategy in state h_3.

We now describe the modifications of the algorithm to detect and resolve these situations. Whenever a new state h is added into the restricted game, the algorithm checks: (1) whether this state can be reached by following the default strategy from a different state h' that is already included, but not yet expanded in the restricted game, and (2) if this state h is not expanded in the restricted game yet, whether it is possible to reach some different state h' that is already included in the restricted game by following the default strategy from h. If such a sequence of actions and state h' is detected, all states and actions corresponding to this sequence are also added into the restricted game.

Utility Values in Restricted Game

The solution of a valid restricted game is found by solving the LP constructed from the restricted sets $\mathcal{H}'$ and $\mathcal{A}'$ for each player. Similarly to the extensive-form case, the algorithm needs to take the concept of default strategy into account and to use a modified utility

function u'. Consider again the MDP in Figure 6.1. State h_4 is included in the restricted game, but it is not expanded yet and the default strategy is used in this state (i.e., action g is played). Since action g does not appear in the restricted game LP ($g \notin \mathcal{A}'$), the solution of the restricted game might not be optimal because it may overestimate the benefits of action c in case there is a large negative utility assigned to actions g and h.

To overcome this issue, the algorithm must use a modified utility function u' for each action in the restricted game that is not fully expanded. This modified utility function takes the default strategy into account and propagates the utility values of all actions that will be played using the default strategy in not expanded states from now on. Formally we define the new utility function u' to be equal to u for each pair of actions (a_i, a_{-i}) in states (h_i, h_{-i}), if either:

1. *both* of these actions *are fully expanded* in the restricted game, or
2. *neither* of these actions *is included* in the restricted game.

Otherwise, we add to utility of this action also the utility value corresponding to the continuation of the strategy according to the default strategy. From the perspective of player i, for each not fully expanded action a_i played in state h_i, we use:

$$u'_i((h_i, a_i), (h_{-i}, a_{-i})) = u_i((h_i, a_i), (h_{-i}, a_{-i})) + \sum_{h'_i \in \mathcal{H}_i \,:\, \mathcal{A}'(h'_i) = \emptyset} \; \sum_{a'_i \in \mathcal{A}(h'_i)} u'_i((h'_i, a'_i), (h_{-i}, a_{-i})) \cdot \pi_i^{\text{DEF}}((h'_i, a'_i) | (h_i, a_i)) \tag{6.7}$$

where $\pi_i^{\text{DEF}}((h'_i, a'_i)|(h_i, a_i))$ denotes the conditional probability of playing an action a'_i in state h'_i after playing action a_i in h_i (this probability is zero if state h'_i is not reachable after playing a_i). The algorithm calculates utility values for each state and action of the opponent h_{-i}, a_{-i} that can have a non-zero extended strategy $\overline{r}_{-i}$.

Note that we use u'_i for any subsequent nodes h'_i and actions a'_i in the definition. This is due to possibility of action of the opponent a_{-i} also being not fully expanded. Since we assume that both MDPs forming strategy spaces for the players are finite and acyclic, this modified utility function is well-defined. Our CS-DO algorithm uses dynamic programing to calculate these values in order not to repeatedly calculate the same values. The algorithm do this calculation in two steps, one from perspective of each player:

Step 1: In the first step, the algorithm calculates utility values u'_i for each action of player i that is included but not fully expanded in the restricted game, against all actions of the opponent (h_{-i}, a_{-i}) that are (1) either included in the restricted game (i.e., $h_{-i} \in \mathcal{H}'_{-i} \wedge a_{-i} \in \mathcal{A}'_{-i}$), or (2) they can have a non-zero probability in extended strategy from the restricted game (i.e., $\exists h' \in \mathcal{H}'_{-i} \;:\; \pi_{-i}^{\text{DEF}}\left((h_{-i}, a_{-i})|h'_{-i}\right) > 0$). Algorithm can now calculate the values according Formula 6.7, and using original utility function u_i on the right side of the equation.

Step 2: In the second step, the situation changes and the algorithm calculates utility values u'_i for each action of player $-i$ that is included but not fully expanded in the restricted game according Formula 6.7, and using utility values u'_i calculated during the first step when necessary on the right side of the equation.

Algorithm 7 The best-response algorithm for the player i (BR_i).

Require: h - current state,$\overline{r}_{-i}$ - extended strategy of the opponent
1: **if** h is terminal **then**
2: **return** 0
3: **end if**
4: **for** $a \in \mathcal{A}_i(h)$ **do**
5: $v_a \leftarrow \sum_{h_{-i} \in \mathcal{H}_{-i}, a_{-i} \in \mathcal{A}_{-i}} \overline{r}_{-i}(h_{-i}, a_{-i}) \cdot u_i((h, a), (h_{-i}, a_{-i}))$
6: **for** $h' \in \mathcal{H}_i$ s.t. $\mathcal{T}(h, a, h') > 0$ **do**
7: $v_a \leftarrow v_a + BR_i(h', \overline{r}_{-i}) \cdot \mathcal{T}(h, a, h')$
8: **end for**
9: **end for**
10: *maxAction* $\leftarrow \arg\max_{a \in \mathcal{A}_i(h)} v_a$
11: **if** *maxAction* is not in the default strategy **or**
there is a change in some successor of h **then**
12: **backup** h,*maxAction*
13: **end if**
14: **return** $v_{maxAction}$

6.2.2 Best-Response Algorithm

Similarly to the best response algorithms for the extensive-form games, the best-response algorithm (BR) determines which states and actions are to be added into the restricted game. For each player, BR returns the set of best-response actions as well as the expected utility value of the best response. Best response is calculated against the current optimal *extended strategy* of the opponent, so the default strategy of the opponent is taken into account. The BR algorithm traverses the MDP in a depth-first search seeking for the best actions to play in each state. Figure 7 depicts the algorithm for player i. For each state the algorithm calculates the expected utility value for each action applicable in this state (line 5) and then recursively calculates the expected utility for each of the successors (line 7). Note that the expected utility is calculated considering all states and actions from the game ($\mathcal{H}_{-i}$ and $\mathcal{A}_{-i}$) and unmodified utility function u_i. Finally, the algorithm selects the best action for this node (line 10) and keeps it as a best-response candidate (line 12). Best-response actions are then found by following the best-response candidates from the root state.

There is one notable exception to the described behavior. If the selected $maxAction$ for some state s corresponds to an action prescribed by the pure default strategy, this action is stored as a best-response candidate only if some action a' was selected in one of the successors of s and this action a' is not prescribed by the pure default strategy (line 11). This condition ensures that the algorithm adds the default-strategy actions into the restricted game only if necessary.

6.2.3 Theoretical Analysis

The CS-DO algorithm converges to a Nash equilibrium of the original unrestricted game. The proof follows from three steps that are similar to the proof described in Section 5.3:

(1) note that a best-response algorithm returns a value that is strictly better for the player if and only if there exists an action not included in the restricted game that is a part of the best response; (2) if such an action does not exist for either of the players, the value of the best responses must be equal to the current value of the restricted game, as well as the value of the restricted game; (3) the algorithm is finite because in each iteration either at least one action is added to the restricted game, or the algorithm terminates.

6.3 Experimental Evaluation

We experimentally compare the performance of the CS-DO algorithm with the standard normal-form double-oracle algorithm that uses pure strategies (denoted PS-DO), as well as a baseline that solves the full linear program using compact strategies (FULLLP). We use variants of search games, where an evader tries to cross unobserved from one side to the other of an area represented as a graph, while the defender tries to discover the evader. The evader receives a reward of 1 for reaching the goal undetected, while the defender receives a reward of 1 for encountering the evader. The games are played for a fixed number of time steps. They differ in the capabilities of the players, the number of units the defender controls, and uncertainty in transitions. The first game is inspired by the maritime security problem (*Transit Game* (Vanek et al., 2010; Vanek et al., 2012)), and the second game is inspired by the border security (*Border Protection Game*) we already used in experiments for extensive-form games in Section 5.4. The games differ in the proportional size of the strategy space for the players as well as the size of the support for typical solutions. They are both parameterizable, allowing us to evaluate performance across different conditions.

Again, neither of the algorithms use any domain-specific knowledge. Experiments were run using a single thread on a standard PC. Each of the algorithms was given a maximum of 7 GB of memory for a process, and we used IBM CPLEX 12.4 to solve the linear programs.

6.3.1 Experiment Domains

Transit Game

The game is played on a grid, undirected graph (see Figure 6.2, left subfigure). The evader crosses the graph from left to right and receives a small penalty (0.01) for each step. The defender controls a single patrolling unit that starts in it's base (the black node). The patrolling unit has limited resources and receives a large penalty if it does not return to the base by the end of the game. The movement of the units may fail with probability 0.1 (the unit stays in the same node) causing the uncertainty in the game. We vary the size of the graph by increasing the number of the rows in the graph (*width*), number of the columns in the graph (*length*), and number of steps for both players.

The state of an MDP of a player consists of the node in the graph and the current time step. The graph is undirected; hence, there is a large number of possible paths that lead to certain state in MDPs, thus making them more connected than in the second game. The size of the strategy space is very similar for both players.

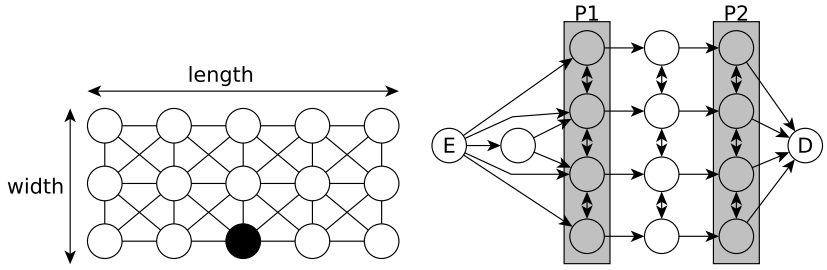


Figure 6.2: Examples of the graphs for the search games. The evader aims to cross from left to right; (left subfigure) transit game graph, the black node refers to the base of the defender; (right subfigure) border protection game graph, patrolling units of the defender operate in the shaded areas P1 and P2.

Border Protection Game

The game is similar to the game described in Section 5.4.1, the evader is again aiming for crossing safely from a starting node (E) to a destination node (D), and the defender controls two patrolling units operating in the selected nodes of the graph (the shaded areas P1 and P2). As before, all units move simultaneously. We use a larger graph for NFGSS depicted in Figure 6.2 (right subfigure).

The state of an MDP of a player consists of the position of the units in the graph and the current time step. The graph is directed and the movement of the units is limited; hence, there is the MDPs are less dense (primarily for the evader). The size of the strategy space is fairly small for the evader compared to the strategy space of the defender that controls two units.

6.3.2 Results

The results are means from several runs (at least 10 for smaller instances). In each run we used different random ordering of actions in states of MDPs to eliminate effect of some particular ordering, which also determines the default strategy in CS-DO.

Transit Game

Results in the Transit Game demonstrate the benefits of our algorithm. Figure 6.3 shows the dependence of the computation time on the number of steps in the game for selected graph sizes (note the logarithmic y-scale). CS-DO algorithm outperforms the other two algorithms by orders of magnitude. It creates the restricted game by adding only $\approx 20\%$ of all marginal actions for each player. On the other hand, PS-DO has the worst performance. This is caused by the exponential number of possible pure strategies and relatively large support in the complete game (the size of the support is typically $10 - 25\%$ of all marginal actions for each player). These factors contribute to the large number of iterations and slow convergence of PS-DO. FULLLP was the second fastest due to the compact strategy

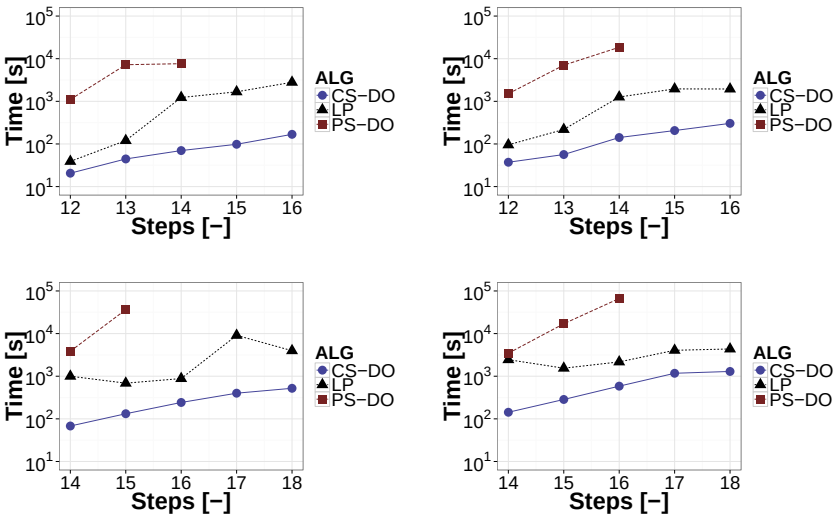


Figure 6.3: Comparison of computation times for Transit Games without uncertainty (left column) and with uncertainty (right column). The graphs are sized: width 5 and length 10 (top row), and with width 6 and length 12 (bottom row).

representation and the fact that the depicted graph sizes fit into memory (the largest game in this comparison has 8423 marginal actions for the evader and 10237 for the defender).

The following table shows the cumulative times the algorithms spent while solving LP, calculating best responses (BR), and constructing the restricted game (RG), in a scenario with width 6, length 14, and 17 steps:

Algorithm	Iterations	Total [s]	LP [s]	BR [s]	RG [s]
FULLLP	-	1,424	1,376	-	-
PS-DO	2,167	123,340	114,264	2,774	3,547
CS-DO	276	243	50	187	2

We can see the large number of iterations for PS-DO that causes the algorithm to be the slowest. The overall time in this setting is more than 36 hours for PS-DO, while CS-DO solves the game in 4 minutes. The majority of time is used to solve the restricted LP and to construct the restricted game, since calculating the utility value for a pair of pure strategies is much more expensive than calculating utility for pair of marginal actions. CS-DO, on the other hand, converges rather quickly and the majority of time is used to calculate best-responses. This is because the size of LP for the restricted game is rather small (on average it adds 984 evader actions (15% of all marginal actions) and 1365 defender actions (16%)).

The results show that the CS-DO scales much better compared to the other algorithms with increasing graph size. We compared the scalability in an experiment where we increased the size of the graph for a fixed ratio of the properties (width, length, and number of steps). CS-DO successfully solves even very large graphs (width 11) in an hour, where

FULLLP fails to construct this game due to the memory constraints (the game has 26610 marginal actions of the evader and 33850 actions of the defender). Moreover, FULLLP is often unable to solve the largest instances that fit into memory (width 10) in 60 hours, while CS-DO solves such instances in 32 minutes.

Border Protection Game

This game is better suited to the standard double-oracle algorithm, since it has unequal sizes of the strategy space, and extremely small support solutions ($\approx 3\%$ of the marginal actions for each player). The left subfigure in Figure 6.4 shows the computation times. The results confirm that the performance of PS-DO dramatically improves. Both double-oracle algorithms perform comparably and they both significantly outperform FULLLP. The time breakdown for the game with 5 steps is shown in the following table:

Algorithm	Iterations	Total [s]	LP [s]	BR [s]	RG [s]
FULLLP	-	12,177	12,174	-	-
PS-DO	9	9.5	0.05	8.13	0.52
CS-DO	11	13	0.2	8.8	1.92

Both double-oracle algorithms are extremely efficient at finding all strategies needed for solving the game. CS-DO adds less than 7% of all marginal actions for the evader and less than 3% for the defender. The slightly worse performance of CS-DO is a result of additional overhead when expanding the restricted game, and the slightly larger number of iterations due to more conservative expansion of the restricted game.

6.3.3 Support Size Analysis

The previous results, and also results in the previous chapter suggest that there is a correlation between the size of the support in the complete game and the performance of the DO algorithms. We analyze this factor in more detail by using the Transit Game to produce games with differing sizes of support. We alter the utility values for the evader's reward from 1 to 20 and her penalty for a time step from 0.01 to 2. We ran the experiments on small graphs of width 4 and 5, where FULLLP performs reasonably well.

The right subfigure in Figure 6.4 shows the relative performance of the double-oracle algorithms in comparison with FULLLP. The x-axis shows the size of the support for the solution relative to the complete game, calculated by multiplying the proportional support sizes for both players. The results show that there is a strong correlation between these two factors (correlation coefficients are 0.715 for CS-DO and 0.729 for PS-DO); the double-oracle algorithms typically perform worse with larger support. Moreover, the comparison shows that CS-DO is less sensitive to the changes in the size of the support (see the linear fit of data, note the logarithmic y-scale). These experimental results demonstrate that the size of the support can be a good indicator of the expected performance of the iterative approach in practice. However, this correlation is not absolute and there are situations in which the

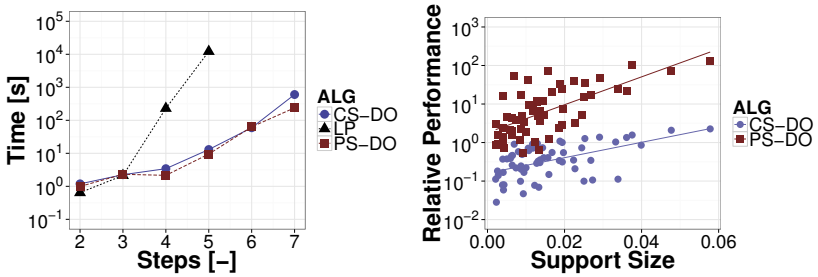


Figure 6.4: (Left) Comparison of computation times for Border Protection Game with graph from Figure 6.2 and increasing number of steps. (Right) Comparison of the relative performance of the DO algorithms depending on the size of the support.

double-oracle enumerates all the actions (and thus has weak performance) even though the original game admits small equilibrium refinement.

6.4 Discussion of the Results and Summary

The results in this section demonstrated the computation speed-up of our compact-strategy double-oracle algorithm (CS-DO) in normal-form games with sequential strategies. The results show that while both variants of double-oracle algorithms perform comparably good in a game with extremely small support, our CS-DO algorithm significantly outperforms existing PS-DO algorithm in games with large support. Moreover, in all setting our algorithm outperforms solving the full linear program FULLLP and it is able to scale to much larger instances. Finally, the experimental analysis suggests that there is a strong correlation between the relative performance of CS-DO and the size of the support in the original game; hence, size of the support is a reasonable indicator for the relative performance. However, a further analysis discovering other characteristics is necessary.

Chapter 7

Conclusions and Future Work

Solving large sequential games is a great challenge for computational game theory. Increasing deployment of game-theoretic models in practice causes further need for scaling-up and being able to solve much larger and more complex models than before. This thesis presents a set of domain-independent algorithms for finding exact solutions for three different classes of finite sequential games with strictly competitive players. First, we have analyzed simultaneous-move games, where both players act simultaneously, but they are able to perfectly observe the state of the game. Secondly, we analyzed extensive-form games with imperfect information, where the only assumption is that the players are able to perfectly remember all their moves and information gained during the course of the game (they have perfect recall). Finally, we analyzed one of the classes of sequential games, where the assumption of perfect recall does not hold. We term this class normal-form games with sequential strategies, since the necessary simplifying assumption is that the players are not able to immediately observe the actions of the opponent.

7.1 Thesis Achievements

This thesis achieved all set goals and offers a collection of contributions:

Set of Novel Algorithms The main contribution of this thesis is the set of three exact algorithms for solving different classes of games: (1) double-oracle algorithm with serialized alpha-beta search (DO$\alpha\beta$) for solving simultaneous-move games, (2) sequence-form double-oracle (DOSWP) algorithm for solving extensive-form games, and (3) compact-strategy double-oracle algorithm (CS-DO) for solving normal-form games with sequential strategies. The immediate advantage of our novel algorithms is the computation speed-up, often in several orders of magnitude compared to the existing state-of-the-art algorithms, and allowing to solve much larger instances of games. We evaluated the performance of our algorithms on a collection of games and identified some of the characteristics of the games suitable for our algorithms. Besides the immediate improvement in scaling-up, these algorithms also offer a different perspective on the problem of computing Nash equilibria. Mainly, double-oracle algorithms decompose the problem of computing an exact Nash equilibrium into several

subproblems, where each of these subproblems can be further enhanced and/or replaced with a domain-specific method while applying the algorithm. Our practical experiments show that even a simple domain-dependent move-ordering can have a substantial impact on the performance and offer further speed-up in practice.

Generalization of Double-Oracle Methods Secondly, this thesis offers a generalization of the iterative double-oracle methods. All the previous work used double-oracle method strictly as an algorithm that iteratively expands the game using new strategies. Our algorithms generalize this concept and use more fine-grained approach, where the restricted game can be expanded with the actions, or sequences of actions. This generalization is one of the components of the success of our double-oracle algorithms and allows the algorithms to construct more efficient restricted games, since only actions and sequences that are part of the best responses are added. Moreover, this generalization also allows us to reduce the exponential number of iterations in comparison to the previous double-oracle algorithms. In order to make such a generalization possible, our algorithms use a novel concept of a compact representation termed *the default strategy* that defines behavior of players if it is not specified in the restricted game. This representation can be compact in practice, because by using the default strategies we can represent only the deviations from sufficiently good strategy (that can be either guessed, or obtained using domain-specific knowledge). Alternatively, the default strategy can be useful in parts of the game that are not relevant for the final expected outcome, and where most of the strategies can be used.

Implementation and Framework Finally, all described algorithms were implemented in a unified framework for solving extensive-form games (see Appendix A). The framework, jointly implemented with Jiří Čermák, Viliam Lisý, and Ondřej Vaněk, offers a unique set of domain-independent algorithms including also several variants of approximative algorithms, and a collection of sequential games, on which the algorithms can be easily compared.

7.2 Future Work

The algorithms presented in this thesis open up a set of directions for future work. Primarily, our novel algorithms are still in the early stage of the development (compared to well established algorithms like Counterfactual Regret Minimization) and there are several opportunities for their further improvement in a domain-independent way. Secondly, each of the presented algorithms has a significant potential of being applied for solving games modeling various real-world scenarios. Finally, the thesis opened several theoretical questions that deserves to be answered in the future work.

7.2.1 Beyond Current Domain-Independent Double-Oracle Algorithms

Simultaneous-Move Games

Each of the presented algorithms has its own specific features that can be further enhanced. DO$\alpha\beta$, introduced for simultaneous-move games, currently exploits only the standard alpha-beta algorithm for solving the serialized variants of simultaneous-move games. However, the research in zero-sum perfect information extensive-form games has a long history and techniques that employ null-window search (searching with more restricted bounds), transposition tables (more complex variant of caching the calculated results), or move-ordering heuristics can further significantly improve the algorithm.

Extensive-Form Games

There are several direct ways for improving further the double-oracle algorithm for generic extensive-form games DOSWP. First step is to restrict ourselves to games with publicly observable actions but imperfect information caused by the Nature player. Instances of games in this class include games of Poker that have been the center of interest of many researchers in computational game theory. By exploiting recent results in fast calculation of best responses for this class of games (Johanson et al., 2011), the double-oracle algorithm might be much more competitive against currently used CFR algorithm. Secondly, recent theoretical results in extensive-form games provide new bounds on the size of the support of equilibria in each information set (Schmid et al., 2014). This theoretical result can be directly used in the double-oracle algorithm by removing sequences that are provably no longer necessary. Contrary to the existing double-oracle approaches that remove the strategies from the restricted game (McMahan and Gordon, 2007a), this approach would not increase the number of iterations. Finally, DOSWP algorithm (as well as CS-DO for normal-form games with sequential strategies) uses the concept of default strategies. Currently, these strategies are fixed and implicitly encoded as a parameter of the algorithm. However, this restriction can be relaxed and the default strategy can be changed during iterations of the algorithm adapting to the current situation and the strategy of the opponent.

Computing Refinements of Nash Equilibria

As we have mentioned at the beginning of the thesis, the Nash solution concept has certain limitations when used in extensive-form games. However, modifying our double-oracle algorithm to calculate any of the refined solution concept is non trivial. Moreover, many of the existing algorithms for computing refined Nash strategies suffer from numerical-precision errors (Miltersen and Sørensen, 2008; Miltersen and Sørensen, 2010) and they are unable to scale for large games. Therefore, a completely different algorithm must be designed in order to improve the scalability. Moreover, we have investigated the practical performance of refined strategies in larger extensive-form games (Cermak et al., 2014). The results show that the simplest undominated equilibrium (or strategic-form perfect equilibrium (Selten, 1975)) often produces strategies similar to more advanced concepts and does not suffer from the numerical precision errors. However, a simple integration of the sequence-form

double-oracle algorithm with the algorithm solving undominated Nash equilibrium does not bring significant reduction in iterations and final size of the restricted game (Cermak, 2014); hence, further analysis is necessary.

Combining Exact and Approximative Algorithms

Comparison of DOSWP with counterfactual regret minimization algorithm (CFR) in Chapter 5 shows that the existing approximative algorithms have typically very quick convergence rate during the first iterations. Therefore, they are often used in game search on-line setting, where the algorithms need to return strategies to play in a given time-limit (Lisy et al., 2012a; Lisy et al., 2014; Bosansky et al., 2014c). Afterwards, the game advances to the next state, and the algorithm is given a new position, for which new strategies need to be found. Double-oracle algorithm is not very competitive in this setting since the quality of the strategies increases more slowly. However, the approximative algorithms typically suffer from a very long convergence and they are often unable to find exact equilibrium (or to approximate it to a very small error in a reasonable time). One of the main cause of the long convergence is the necessity for the algorithm to ensure that all branches of the game tree are visited infinitely often in order to guarantee the theoretical results about convergence to a Nash equilibrium. Combining approximative and exact algorithms is a challenging opportunity, where either the restricted game of a double-oracle algorithm can initialized with a set of strategies from approximative algorithms, or the approximative algorithms could benefit from executing the exact algorithm on specific parts of the game tree and proving that certain actions are no longer needed and will not be evaluated any more.

7.2.2 Theoretical Analysis of Double-Oracle Algorithms

The algorithms in this thesis are built on top of the double-oracle framework. In the worst-case, the performance of double-oracle algorithms is worse than using standard linear programming since all pure strategies or sequences need to be enumerated and added into the restricted game, while the restricted game is solved repeatedly. However, the experimental results show that this is rarely the case in practice. Unfortunately, there are no known results about the expected number of iterations, nor about the way the exact characteristics of games affect the relative performance of the double-oracle algorithms. The experimental evaluation on different models provided some intuition behind certain characteristics and we showed that there is a strong correlation between the relative performance and the size of the support of the original game. However, exact and formal estimation of the expected number of iterations would help to better identify the scenarios where double-oracle algorithms are guaranteed to be better than solving full linear programs. Since these properties are not known even for simple normal-form games, this class of games should be investigated first and such an investigation is beyond the scope of this thesis.

Bibliography

Agmon, N., Kraus, S., Kaminka, G. A., and Sadov, V. (2009). Adversarial uncertainty in multi-robot patrol. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCAI)*, pages 1811–1817. Morgan Kaufmann Publishers Inc.

Alliot, J.-M., Durand, N., Gianazza, D., and Gotteland, J.-B. (2012). Finding and Proving the Optimum: Cooperative Stochastic and Deterministic Search. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*.

Auer, P., Cesa-Bianchi, N., and Freund, Y. (2003). The non-stochastic multiarmed bandit problem. *SIAM Journal on Computing*.

Aumann, R. (1974). Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, 1:67–96.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329.

Bosansky, B. (2013). Solving Extensive-form Games with Double-oracle Methods (Extended Abstract). In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Bosansky, B., Jiang, A. X., Tambe, M., and Kiekintveld, C. (2014a). Combining Compact Representation and Incremental Generation in Large Games with Sequential Strategies. Unpublished.

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

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

Bosansky, B., Kiekintveld, C., Lisy, V., and Pechoucek, M. (2014b). An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information. *Journal of Artificial Intelligence Research (under review)*.

Bosansky, B., Lisy, V., Cermak, J., Vitek, R., and Pechoucek, M. (2013b). Using Double-oracle Method and Serialized Alpha-Beta Search for Pruning in Simultaneous Move Games. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 48–54.

Bosansky, B., Lisy, V., Jakob, M., and Pechoucek, M. (2011). Computing Time-Dependent Policies for Patrolling Games with Mobile Targets. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 989–996.

Bosansky, B., Lisy, V., Lanctot, M., Cermak, J., and Winands, M. H. (2014c). Algorithms for Computing Strategies in Two-Player Simultaneous Move Games. *Artificial Intelligence (submitted)*.

Buro, M. (2003). Solving the Oshi-Zumo Game. *Advances in Computer Games*, 10:361–366.

Cermak, J. (2014). Using Refinements of Nash Equilibria for Solving Extensive-Form Games. Master's thesis, Czech Technical University in Prague.

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

Chen, X. and Deng, X. (2006). Settling the complexity of two-player nash equilibrium. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 261–272.

Churchill, D., Saffidine, A., and Buro, M. (2012). Fast heuristic search for RTS game combat scenarios. In *8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 112–117.

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

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

Conitzer, V. and Sandholm, T. (2008). New complexity results about Nash equilibria. *Games and Economic Behavior*, 63(2):621 – 641.

Dantzig, G. and Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, 8:101–111.

Daskalakis, C., Goldberg, P. W., and Papadimitriou, C. H. (2006). The Complexity of Computing a Nash Equilibrium. In *Proceedings of the 38th annual ACM symposium on Theory of computing*.

Finnsson, H. (2007). Cadia-player: A general game playing agent. Master's thesis, Reykjavík University.

Finnsson, H. (2012). *Simulation-Based General Game Playing*. PhD thesis, Reykjavík University.

Ganzfried, S. and Sandholm, T. (2013). Improving performance in imperfect-information games with large state and action spaces by solving endgames. In *Computer Poker and Imperfect Information Workshop at the National Conference on Artificial Intelligence (AAAI)*.

Genesereth, M., Love, N., and Pell, B. (2005). General game playing: Overview of the AAAI competition. *AI magazine*, 26(2):73–84.

Gibson, R., Lanctot, M., Burch, N., Szafron, D., and Bowling, M. (2012). Generalized sampling and variance in counterfactual regret minimization. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, pages 1355–1361.

Gilboa, I. and Zemel, E. (1989). Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93.

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

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

Hoda, S., Gilpin, A., Peña, J., and Sandholm, T. (2010). Smoothing techniques for computing nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2):494–512.

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

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 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 207–214.

Johanson, M., Bard, N., Burch, N., and Bowling, M. (2012). Finding Optimal Abstract Strategies in Extensive-Form Games. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1371–1379.

Johanson, M., Bowling, M., Waugh, K., and Zinkevich, M. (2011). Accelerating best response calculation in large extensive games. In *In Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 258–265.

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo Planning. In *ECML-06*.

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

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

Koller, D., Megiddo, N., and von Stengel, B. (1996). Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2):247–259.

Kovarsky, A. and Buro, M. (2005). Heuristic search applied to abstract combat games. *Advances in Artificial Intelligence*, pages 55–77.

Kreps, D. M. and Wilson, R. (1982). Sequential equilibria. *Econometrica*, 50(4):863–94.

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

Lanctot, M., Gibson, R., Burch, N., Zinkevich, M., and Bowling, M. (2012). No-Regret Learning in Extensive-Form Games with Imperfect Recall. In *ICML*, pages 1–21.

Lanctot, M., Lisy, V., and Bowling, M. (2014). Search in Imperfect Information Games using Online Monte Carlo Counterfactual Regret Minimization. In *AAAI Workshop on Computer Poker and Imperfect Information*.

Lanctot, M., Lisy, V., and Winands, M. H. (2013a). Monte Carlo Tree Search in Simultaneous Move Games with Applications to Goofspiel. In *Proceedings of IJCAI 2013 Workshop on Computer Games*.

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 (NIPS)*, pages 1078–1086.

Lanctot, M., Wittlinger, C., Winands, M. H. M., and Teuling, N. G. P. D. (2013b). Monte Carlo tree search for simultaneous move games: A case study in the game of Tron. In *Proceedings of the Twenty-Fifth Benelux Conference on Artificial Intelligence (BNAIC)*, pages 104–111.

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

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

Lisy, V., Bosansky, B., Jakob, M., and Pechoucek, M. (2009). Adversarial Search with Procedural Knowledge Heuristic. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 899–906, Budapest, Hungary.

Lisy, V., Bosansky, B., and Pechoucek, M. (2012a). Anytime Algorithms for Multi-agent Visibility-based Pursuit-evasion Games (Extended Abstract). In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1301–1302.

Lisy, V., Bosansky, B., and Pechoucek, M. (2014). Monte-Carlo Tree Search for Imperfect-Information Pursuit-Evasion Games. In *Transactions on Computational Intelligence and Artificial Intelligence in Games (submitted)*.

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

Lisy, V., Pibil, R., Stiborek, J., Bosansky, B., and Pechoucek, M. (2012b). Game-theoretic Approach to Adversarial Plan Recognition. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*, pages 546–551.

Luber, S., Yin, Z., Delle Fave, F., Xin Jiang, A., Tambe, M., and Sullivan, J. P. (2013). Game-theoretic patrol strategies for transit systems: the TRUSTS system and its mobile app. In *Proceedings of the 12th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*, pages 1377–1378.

Marple, A. and Shoham, Y. (2012). Equilibria in Finite Games with Imperfect Recall. pages 1–30.

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

McKelvey, R. D., McLennan, A. M., and Turocy, T. L. (2014). *Gambit: Software Tools for Game Theory, Version 13.1.2*.

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

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

McMahan, H. B. and Gordon, G. J. (2007b). A unification of extensive-form games and Markov decision processes. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 86. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

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

Miltersen, P. B. and Sørensen, T. B. (2008). Fast algorithms for finding proper strategies in game trees. In *Proceedings of Symposium on Discrete Algorithms (SODA)*.

Miltersen, P. B. and Sørensen, T. B. (2010). Computing a quasi-perfect equilibrium of a two-player game. *Economic Theory*, 42(1):175–192.

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

Nguyen, K. Q. and Thawonmas, R. (2013). Monte carlo tree search for collaboration control of ghosts in ms. pac-man. *Computational Intelligence and AI in Games, IEEE Transactions on*, 5(1):57–68.

Nudelman, E., Wortman, J., Shoham, Y., and Leyton-Brown, K. (2004). Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems*.

Ordónez, F., Tambe, M., Jara, J. F., Jain, M., Kiekintveld, C., and Tsai, J. (2013). Deployed security games for patrol planning. In *Handbook of Operations Research for Homeland Security*, pages 45–72. Springer.

Pepels, T., Winands, M. H., and Lanctot, M. (2014). Real-Time Monte-Carlo Tree Search in Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*.

Pita, J., Jain, M., Tambe, M., Ordóñez, F., and Kraus, S. (2010). Robust solutions to Stackelberg games: Addressing bounded rationality and limited observations in human cognition. *Artificial Intelligence*, 174(15):1142 – 1171.

Pita, J., Jain, M., Western, C., Portway, C., Tambe, M., Ordonez, F., Kraus, S., and Parachuri, P. (2008). Deployed ARMOR protection: The application of a game-theoretic model for security at the Los Angeles International Airport. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*.

Ponsen, M. J. V., de Jong, S., and Lanctot, M. (2011). Computing approximate nash equilibria and robust best-responses using sampling. *J. Artificial Intell. Res. (JAIR)*, 42:575–605.

Popescu, G. (2013). Group recommender systems as a voting problem. In *Online Communities and Social Computing*, pages 412–421. Springer.

Reinefeld, A. (1989). *Spielbaum-Suchverfahren*, volume 200. Springer.

Rhoads, G. C. and Bartholdi, L. (2012). Computer solution to the game of pure strategy. *Games*, 3(4):150–156.

Robu, V., Gerding, E. H., Stein, S., Parkes, D. C., Rogers, A., and Jennings, N. R. (2013). An online mechanism for multi-unit demand and its application to plug-in hybrid electric vehicle charging. *Journal of Artificial Intelligence Research*.

Ross, S. M. (1971). Goofspiel — the game of pure strategy. *Journal of Applied Probability*, 8(3):621–625.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition.

Saffidine, A., Finnsson, H., and Buro, M. (2012). Alpha-beta pruning for games with simultaneous moves. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*, pages 22–26.

Sailer, F., Buro, M., and Lanctot, M. (2007). Adversarial planning through strategy simulation. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 37–45.

Samothrakis, S., Robles, D., and Lucas, S. M. (2010). A UCT agent for Tron: Initial investigations. In *Proceedings of the 2010 IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 365–371.

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

Schmid, M., Moravcik, M., and Hladik, M. (2014). Bounding the Support Size in Extensive Form Games with Imperfect Information. In *Proceedings of the National Conference of Artificial Intelligence (AAAI)*.

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

Selten, R. (1975). Reexamination of the perfectness concept for equilibrium points in extensive games. *International Journal of Game Theory*, 4:25–55.

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

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

Smith, S. and Nau, D. (1995). An analysis of forward pruning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1386–1386.

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

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

van Damme, E. (1984). A relation between perfect equilibria in extensive form games and proper equilibria in normal form games. *Game Theory*, 13:1–13.

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

Vanek, O. (2013). *Computational Methods for Transportation Security*. PhD thesis, Czech Technical University in Prague.

Vanek, O., Bosansky, B., Jakob, M., Lisy, V., and Pechoucek, M. (2012). Extending Security Games to Defenders with Constrained Mobility. In *Proceedings of AAAI Spring Symposium GTSSH*.

Vanek, O., Bosansky, B., Jakob, M., and Pechoucek, M. (2010). Transiting Areas Patrolled by a Mobile Adversary. In *Proceedings of IEEE Conference on Compuational Intelligence and Games*.

Vanek, O., Jakob, M., Bosansky, B., Hrstka, O., and Pechoucek, M. (2011a). AgentC: Agent-based System for Securing Maritime Transit (Demonstration). In *Proceedings of The 10th International*

Conference on Autonomous Agents and Multiagent Systems (AAMAS).

Vanek, O., Jakob, M., Lisy, V., Bosansky, B., and Pechoucek, M. (2011b). Iterative Game-theoretic Route Selection for Hostile Area Transit and Patrolling. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Vieira, M., Sukhatme, G., and Govindan, R. (2009). Scalable and Practical Pursuit-Evasion. *Proceedings of the 2nd International Conference on Robotic Communication and Coordination*.

von Neumann, J. and Morgenstern, O. (1947). *The Theory of Games and Economic Behavior*. Princeton University Press, 2nd edition.

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

von Stengel, B. (1996). Efficient computation of behavior strategies. *Games and Economic Behavior*, 14:220–246.

Vytelingum, P., Ramchurn, S. D., Voice, T. D., Rogers, A., and Jennings, N. R. (2010). Trading agents for the smart electricity grid. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 897–904.

Washburn, A. and Wood, K. (1995). Two-person Zero-sum Games for Network Interdiction. *Operations Research*, 43(2):243–251.

Waugh, K., Zinkevich, M., Johanson, M., Kan, M., Schnizlein, D., and Bowling, M. H. (2009). A Practical Use of Imperfect Recall. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*.

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.

Wilson, R. (1972). Computing equilibria of two-person games from the extensive form. *Management Science*, 18(7):448–460.

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)*.

Yin, Z., Korzhyk, D., Kiekintveld, C., Conitzer, V., and Tambe, M. (2010). Stackelberg vs. Nash in security games: Interchangeability, equivalence, and uniqueness. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pages 1139–1146.

Zinkevich, M., Bowling, M., and Burch, N. (2007). A new algorithm for generating equilibria in massive zero-sum games. In *Proceedings of National Conference on Artificial Intelligence*, pages 788–793.

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

Appendix A

Game-Theoretic Library

All algorithms described in this thesis were implemented in Java framework jointly designed and implemented with Jiří Čermák, Viliam Lisý, and Ondřej Vaněk[1]. The source codes of this framework are released under LGPL license and available at

`http://agents.felk.cvut.cz/topics/Computational_game_theory`.

A.1 Domains

The framework provides a unified environment for implementing algorithms for solving of various classes of extensive-form games. We designed and implemented generic structure of an extensive form game consisting of four main classes:

GameInfo encapsulates information about the game, such as list of the players, or parameters of the game

GameState represents the state of the game, keeps the history of the actions performed in the game so far, and based on the domain-dependent information provides identification of the information set, to which this state belongs

Action represent and encapsulates information necessary for each action in the game

Expander generates new actions based on a game state

A new domain (i.e., a new extensive-form game) can be implemented and integrated with the framework by extending these four classes. Currently, there is a number of parameterizable games that can be scaled in order to evaluate performance of algorithms for solving extensive-form games, and a collection of other smaller games used for experimental purposes.

Goofspiel

A card game with 3 identical decks of cards with values $\{0, \ldots, (d-1)\}$ (one for nature and one for each player), where d is a parameter of the game. The game is played in rounds: at the beginning of each round, nature 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 nature's card. In case both players select

[1] Games Oshi-Zumo and Tron were implemented by Marc Lanctot.

the card with the same value, the nature'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 4 implemented variants of this game – either the moves of Nature can be fixed and known to both players, or not; there is standard simultaneous-move variant, or imperfect-information variant (the players do not learn what card the opponent had used, only whether they have won or lost the last bid).

Poker

Poker in the framework is based on a simplified variant known as Leduc Hold'em Poker. In our version of poker each player starts with the same amount of chips, and both players are required to put some number of chips in the pot (called the *ante*). In the next step the Nature player deals a single card to each player (the opponent is unaware of the card), and the betting round begins. A player can either *fold* (the opponent wins the pot), *check* (let the opponent make the next move), *bet* (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, the Nature player deals one 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.

There are several parameters for this game: we can set the number of types of the cards, the number of cards of each type, the maximum length of sequence of raises in a betting round, and the number of different sizes of bets (i.e., amount of chips added to the pot) for *bet*/*raise* actions.

Phantom Tic-Tac-Toe

Phantom Tic-Tac-Toe is a blind variant of the well-known game of Tic-Tac-Toe. The game is played on a 3×3 board, where two players (cross and circle) attempt to place 3 identical marks in a horizontal, vertical, or diagonal row to win the game. In the blind variant, the players are unable to observe opponent's moves and each player only knows that the opponent made a move and it is her turn. Moreover, if a player tries to place her mark on a square that is already occupied by an opponent's mark, the player learns this information and can place the mark in some other square.

Oshi-Zumo

Oshi-Zumo is a board game with two players in the game, both starting with certain number of coins, and there is a playing board represented as a one-dimensional playing field. There are $2K + 1$ locations on the field (indexed $0, \ldots, 2K$), where K is another parameter of the game. 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). Afterwards, 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 looses the game. If the final position of the wrestler at in the center, the game is a draw. Number of coins, size of the board, and minimal bid are parameters of this game.

Tron

Tron is a two-player simultaneous-move game played on a discrete grid, possibly obstructed by walls. At each step in Tron both players move to adjacent cells, and a wall is placed in the cells the players started on that turn. A player looses if he hits the wall. The goal of both players is to survive as long as possible. If both players can only move into a wall, can only move off the board or move into each other at the same turn, the game ends in a draw. The game can be parametrized with different graphs, possibly containing obstacles.

Border Protection Game

Example of a search game with two players: the *patroller* (or the defender) and the *evader* (or the attacker). The game is played on a directed graph, where the evader aims to cross safely from a starting node to a destination node. The defender controls two units that move in the intermediate nodes trying to capture the evader by occupying the same node as the evader. During each turn both players move their units simultaneously from the current node to an adjacent node or stay in the same location. If a pre-determined number of turns is made without either player winning, the game is a draw. Players are unaware of the location and the actions of the other player with one exception – the evader leaves tracks in the visited nodes that can be discovered if the defender visits the nodes later. The game also includes an option for the evader to avoid leaving tracks using a special move (a *slow move*) that requires two turns to simulate an evader covering the tracks. The game can be parametrized with an arbitrary graph, number of moves, and initial positions of the units. Allowing the evader to perform slow moves is also a parameter.

Pursuit-Evasion Game

As a second example of a search game we use a pursuit-evasion game played on a graph for a pre-defined number of moves (d). There is a single evader and a pursuer that controls 2 pursuing units. Both players have perfect information about the position of the units of the opponent. The evader wins, if she successfully avoids the units of the pursuer for the whole game; 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 can be parametrized with an arbitrary graph, number of moves, and initial position of the units. By default, the units are placed randomly, but the distance between the pursuers and the evader is 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.

Transit Game

Third example of a search game. The game is played on a grid, undirected graph. The evader crosses the graph from left to right and receives a small penalty for each step. The defender controls a single patrolling unit that starts in it's base. The patrolling unit has limited resources and receives a large penalty if it does not return to the base by the end of the game. The movement of the units may fail with some probability (the unit stays in the same node) causing the uncertainty in the game. We vary the size of the graph by increasing the number of the rows in the graph (*width*), number of the columns in the graph (*length*), and number of steps for both players. The dimensions of the graph, position of the base, probabilities of failing actions, the penalty for each move of the evader, and the overall horizon are the parameters of this game.

Randomly Generated Games

We also use randomly generated extensive-form games, where we can alter number of actions for each player (branching factor), depth of the game tree. Moreover, each action generates an observation for the opponent from a restricted set determining the number of information sets in the game. Finally, we use 4 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. the utility values are binary, uniformly selected from the set $\{0, 1\}$;
3. we randomly assign either -1, 0, or $+1$ value to actions and the utility value in a leaf is a sum of all values on edges on the path from the root of the game tree to the leaf;
4. as in the previous case, however, the utility is equal to the signum of the sum of all values on the edges, which corresponds to win-tie-loose

A.2 Algorithms

The framework contains a collection of algorithms for solving different classes of games. We list the algorithms based on the primary class, for which they are designed. However, this is not a strict restriction and algorithms for solving imperfect-information extensive-form games with perfect recall can also be used for solving simultaneous-move games (although they are not that efficient).

Simultaneous-Move Games

- backward induction (as described in Section 4.2)
- backward induction with alpha-beta search (as described in Section 4.3.1)
- backward induction with double-oracle algorithm (as described in Section 4.3.2)
- backward induction with double-oracle algorithm and serialized alpha-beta (as described in Section 4.3.2)
- simultaneous-move variants of Monte Carlo Tree Search with several selection functions (see (Lanctot et al., 2013a; Lisy et al., 2013; Bosansky et al., 2014c))
- simultaneous-move variant of (online) outcome sampling (see (Lanctot et al., 2013a; Bosansky et al., 2014c))

Extensive-Form Games

- sequence-form linear programming (as described in Section 3.2.2)
- sequence-form double-oracle algorithm (as described in Section 5.2)
- sequence-form linear programming for computing undominated Nash equilibrium (see for example (Cermak et al., 2014))
- algorithm for computing quasi-perfect equilibrium (integrated with source code of GTF framework by T. B. Sørensen[2])

[2]available at `http://www.itu.dk/people/trbj/gtf.html`

- algorithm for computing normal-form proper equilibrium (see (Miltersen and Sørensen, 2008))
- counterfactual regret minimization (as defined in (Zinkevich et al., 2008), implementation based on (Lanctot, 2013))
- sampling variant of counterfactual regret minimization (outcome sampling) (see (Lanctot et al., 2014))
- imperfect-information variant of Monte Carlo tree search (see (Lanctot et al., 2014))

Normal-Form Games

- linear program (as described in Section 3.1)
- pure-strategy double-oracle for normal-form games (as defined in (McMahan et al., 2003; McMahan, 2006))

Normal-Form Games with Sequential Strategies

- network-flow based linear programming (as described in Section 6.1.1)
- pure-strategy double-oracle (as defined in (McMahan et al., 2003; McMahan, 2006))
- pure-strategy double-oracle algorithm with limited bundle size (as defined in (McMahan, 2006; McMahan and Gordon, 2007a))
- compact-strategy double-oracle algorithm (as described in Section 6.2)

Appendix B

Publications

Publications Related to the Topic of the Thesis

Journal Publications (with IF) (3):

- **B. Bošanský**, C. Kiekintveld, V. Lisý, and M. Pěchouček. An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information. *Journal of Artificial Intelligence Research (under review)* **(65%)**
- **B. Bošanský**, V. Lisý, M. Lanctot, J. Čermák, and M. M. H. Winands. Algorithms for computing strategies in two-player simultaneous move games. *Artificial Intelligence (submitted)* **(30%)**
- V. Lisý, **B. Bošanský**, and M. Pěchouček. Monte-Carlo Tree Search for Imperfect-Information Pursuit-Evasion Games. In *Transactions on Computational Intelligence and Artificial Intelligence in Games (submitted)* **(15%)**

In Proceedings (16):

- J. Čermák, **B. Bošanský**, and V. Lisý. (2014). Practical Performance of Refinements of Nash Equilibria in Extensive-Form Zero-Sum Games. In *Proceedings of the 21th European Conference on Artificial Intelligence (ECAI)* **(40%)**
- **B. Bošanský**, V. Lisý, J. Čermák, R. Vítek and 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 **(35%)**
- **B. Bošanský**, C. Kiekintveld, V. Lisý, J. Čermák and M. Pěchouček: Double-oracle Algorithm for Computing an Exact Nash Equilibrium in Zero-sum Extensive-form Games. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2013. **(50%)**
- **B. Bošanský**: Solving Extensive-form Games with Double-oracle Methods (Extended Abstract). In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2013. **(100%)**
- V. Lisý, V. Kovařík, M. Lanctot, **B. Bošanský**: Convergence of Monte Carlo Tree Search in Simultaneous Move Games. In *Advances in Neural Information Processing Systems (NIPS)*. 2013 **(10%)**

- **B. Bošanský**, C. Kiekintveld, V. Lisý and M. Pěchouček: Iterative Algorithm for Solving Two-player Zero-sum Extensive-form Games with Imperfect Information. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*. 2012 **(65%)**
- V. Lisý, R. Píbil, J. Stiborek, **B. Bošanský** and Michal Pěchouček: Game-theoretic Approach to Adversarial Plan Recognition. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*. 2012 **(15%)**
- O. Vaněk, **B. Bošanský**, M. Jakob, V. Lisý and M. Pěchouček: Extending Security Games to Defenders with Constrained Mobility. In *Proceedings of AAAI Spring Symposium GTSSH*. 2012. **(25%)**
- V. Lisý, **B.Bošanský**, M.Pěchouček: Anytime Algorithms for Multi-agent Visibility-based Pursuit-evasion Games (Extended Abstract). In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2012. **(20%)**
- **B. Bošanský**, V. Lisý, M. Jakob and M. Pěchouček: Computing time-dependent policies for patrolling games with mobile targets. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2011. **(40%)**
- O. Vaněk, M. Jakob, V. Lisý, **B. Bošanský**, and M. Pěchouček: Iterative game-theoretic route selection for hostile area transit and patrolling. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2011. **(10%)**
- O. Vaněk, **B. Bošanský**, M. Jakob and M. Pěchouček: Transiting Areas Patrolled by a Mobile Adversary. In *IEEE Conference on Computational Intelligence and Games (CIG)*. 2010 **(35%)**
- V. Lisý, **B. Bošanský**, R. Vaculín and M. Pěchouček: Agent Subset Adversarial Search for Complex Non-cooperative Domains. In *IEEE Conference on Computational Intelligence and Games (CIG)*. 2010 **(40%)**
- V. Lisý, **B. Bošanský**, M. Jakob and M. Pěchouček: Goal-Based Game Tree Search for Complex Domains. Chapter in *Agents and Artificial Intelligence. Communications in Computer and Information Science*. Volume 67, Part 2, 97-109, Springer Berlin / Heidelberg, 2010. **(35%)**
- V. Lisý, **B. Bošanský**, M. Jakob and M. Pěchouček: Adversarial Search with Procedural Knowledge Heuristic. In *The 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2009 **(35%)**
- V. Lisý, **B. Bošanský**, M. Jakob and M. Pěchouček: Goal-based Adversarial Search - Searching Game Trees in Complex Domains using Goal-based Heuristic. In *Proceedings of ICAART - First International Conference on Agents and Artificial Intelligenc*e. 2009 **(35%)**

Unpublished (1):

- **B. Bošanský**, A. X. Jiang, M. Tambe and C. Kiekintveld (2014). Combining Compact Representation and Incremental Generation in Large Games with Sequential Strategies

Other Publications

Journal Publications (without IF) (1):

- **B. Bošanský**: Process-based Multi-agent Architecture in Home Care. In *European Journal for Biomedical Informatics*, 2010.

In Proceedings (12):

- M. Abaffy, T. Brázdil, V. Řehák, **B. Bošanský**, A. Kučera, J. Krčál. Solving adversarial patrolling games with bounded error (Extended Abstract). In *Proceedings of the 13th international conference on Autonomous agents and multi-agent systems (AAMAS)*. 2014

- O. Vaněk, Z. Yin, M. Jain, **B. Bošanský**, M. Tambe and M. Pěchouček: Game-theoretic Resource Allocation for Malicious Packet Detection in Computer Networks. In *Proceedings of The 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2012.

- P. Novák, A. Komenda, V. Lisý, **B. Bošanský**, M. Čáp and M. Pěchouček: Tactical Operations of Multi-Robot Teams in Urban Warfare (Demonstration). In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2012.

- **B. Bošanský**, O. Vaněk and M. Pechoucek: Strategy Representation Analysis for Patrolling Games. In *Proceedings of AAAI Spring Symposium GTSSH*. 2012

- R. Píbil, V. Lisý, C. Kiekintveld, **B. Bošanský** and M. Pěchouček: Game Theoretic Model of Strategic Honeypot Selection in Computer Networks. In *Decision and Game Theory for Security, LNCS*, 2012.

- L. Lhotská, **B.Bošanský** and J. Doležal: Integration of Procedural Knowledge in Multi-Agent Systems in Medicine. In *Information Technology in Bio- and Medical Informatics, LNCS*, 2011

- **B. Bošanský**, V. Lisý and M. Pěchouček: Towards Cooperation in Adversarial Search with Confidentiality. In *Holonic and Multi-Agent Systems for Manufacturing, LNCS*, 2011

- O. Vaněk, M. Jakob, **B. Bošanský**, O. Hrstka, and M. Pěchouček: AgentC: Agent-based System for Securing Maritime Transit (Demonstration). In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2011.

- **B. Bošanský** and L. Lhotská: Agent-based process-critiquing decision support system, In *2nd International Symposium on Applied Sciences in Biomedical and Communication Technologies (ISABEL)*. 2009

- **B. Bošanský** and L. Lhotská: Medical Processes Agent-Based Critiquing System. In: *Hakl, F. (ed) Doktorandský den 2009., Ústav informatiky,AV ČR*, MATFYZPRESS, Praha

- **B. Bošanský** and L. Lhotská: Agent-based Simulation of Processes in Medicine. In: *Hakl, F. (ed) Doktorandský den 2008., Ústav informatiky, AV ČR*, MATFYZPRESS, Praha

- **B. Bošanský** and C. Brom: Agent-Based Simulation of Business Processes in a Virtual World. In *Hybrid Artificial Intelligence Systems.* 2008

Responses

Follows a list of publications with at least three citations based on Google Scholar as of July 2014 and *excluding self-citations*:

- B. Bošanský, V. Lisý, M. Jakob and M. Pěchouček: Computing time-dependent policies for patrolling games with mobile targets. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2011. **22 citations**
- O. Vaněk, B. Bošanský, M. Jakob and M. Pěchouček: Transiting Areas Patrolled by a Mobile Adversary. In *IEEE Conference on Computational Intelligence and Games (CIG)*. 2010 **8 citations**
- O. Vaněk, M. Jakob, V. Lisý, B. Bošanský, and M. Pěchouček: Iterative game-theoretic route selection for hostile area transit and patrolling. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2011. **5 citations**
- B. Bošanský, C. Kiekintveld, V. Lisý, J. Čermák and M. Pěchouček: Double-oracle Algorithm for Computing an Exact Nash Equilibrium in Zero-sum Extensive-form Games. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2013. **4 citations**
- V. Lisý, B.Bošanský, M.Pěchouček: Anytime Algorithms for Multi-agent Visibility-based Pursuit-evasion Games (Extended Abstract). In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2012. **3 citations**