Czech Technical University in Prague

Faculty of Electrical Engineering

Doctoral Thesis

August 2014

Viliam Lisý

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Cybernetics

Monte Carlo Tree Search in Imperfect-Information Games

Doctoral Thesis

Viliam Lisý

Prague, August 2014

Ph.D. Programme: Electrical Engineering and Information Technology
Branch of study: Artificial Intelligence and Biocybernetics

Advisor: prof. Dr. Ing. Michal Pěchouček, M.Sc.
Advisor-Specialist: Ing. Michal Jakob, Ph.D.

Acknowledgments

This work would not be possible without my colleagues from the Agent Technology Center and other collaborating institutions. Specifically, I would like to thank my advisor, Michal Pěchouček, for creating a secure and inspiring work environment. Thanks to close collaboration with Branislav Bošanský, I always had someone to discuss new ideas and new exciting results in computational game theory. I would like to thank Michal Jakob, Ondřej Vaněk, and Peter Novák, mainly for discussions shaping my view on what are the main principles of good and useful research. I am thankful to Katia Sycara and Roie Zivan for giving me the opportunity to do research at excellent foreign universities. My thanks also go to my direct collaborators that taught me a lot in our regular interaction, especially to Marc Lanctot, Christopher Kiekintveld, and Vojtěch Kovařík. Last, but not least, I would like to thank my parents and my brother for their support during my studies.

The work on this thesis was partially supported by the Czech Science Foundation (grant no. P202/12/2054), by the Air Force Office of Scientific Research, USAF, under grant number FA8655-07-1-3083, by the U.S. Army grant no. W15P7T-05-R-P209 and by the Grant Agency of the Czech Technical University in Prague, grant No. SGS10/188/OHK3/2T/13. Access to the computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the "Projects of Large Infrastructure for Research, Development, and Innovations" programme (LM2010005) is highly appreciated.

Abstract

Monte Carlo Tree Search (MCTS) is currently the most popular game playing algorithm for perfect-information extensive-form games. Its adaptation led, for example, to human expert level Go playing programs or substantial improvement of solvers for domain-independent automated planning. Inspired by this success, researchers started to adapt this technique also for imperfect-information games. Imperfect-information games provide several difficulties not present in perfect-information games, such as the necessity to use randomized strategies to ensure an optimal play. Even though the properties of the optimal strategies in these games are well-studied, MCTS literature on imperfect-information games does not build on this knowledge sufficiently. None of the pre-existing MCTS algorithms is known to converge to the optimal strategy in the game, even if they were given an infinite time for computation.

In this thesis, we study MCTS in two-player zero-sum extensive-form games with imperfect information and focus on game-theoretic properties of the produced strategies. We proceed in two steps. We first analyze in detail one of the simplest classes of games with imperfect information: games with simultaneous moves. Afterwards, we proceed to fully generic imperfect-information games. We survey the existing MCTS algorithms for these classes of games and classify them to few fundamentally distinct classes. Furthermore, we provide the following contributions.

First, we propose new MCTS algorithms that provably converge to Nash equilibrium of the game with increasing computation time. We introduce three such algorithms. One based on a minor modification of the standard MCTS template for simultaneous-move games and other two as an adaptation of the successful Monte Carlo Counterfactual Regret Minimization (MCCRF) to online search in both simultaneous-move and imperfect-information games.

Second, we focus on improving the performance of MCTS algorithms, mainly by proposing and evaluating novel selection functions for choosing the actions to sample in the later iterations based on the statistics collected from the earlier iterations. In generic imperfect-information games, we propose explicit modelling of player's beliefs about the probability of being in a specific game state during a match.

Third, we perform an extensive evaluation of the proposed and existing MCTS methods on five simultaneous-move games and four fully imperfect-information games with variable size and fundamentally different properties. We evaluate both the ability of the algorithms to quickly approximate Nash equilibrium strategy and their performance in head-to-head tournaments. We show that the algorithms based on MCCFR have very a fast convergence to an equilibrium, but classical MCTS with the novel selection functions has superior performance in tournaments in large games.

Finally, we present a case study of using MCTS for creating intelligent agents for a robotic visibility-based pursuit-evasion game. We design domain-specific variants of the previously introduced algorithms and evaluate their performance in a complex simulated environment. We show that the algorithms based on MCTS outperform the best previously known algorithm for this problem.

Abstrakt

Monte Carlo Tree Search (MCTS) je v poslední době velmi populární algoritmus pro hraní her v extenzivní formě s úplnou informací. Jeho adaptace vedla například k programům hrajícím hru Go na úrovni lidských expertů nebo podstatnému zlepšení algoritmů pro doménově nezávislé automatické plánování. Inspirováni těmito úspěchy začali výzkumníci přizpůsobovat tuto techniku i pro hry s neúplnou informací. Neúplná informace způsobuje v těchto hrách několik komplikací, které se u her s úplnou informací neobjevují. Příkladem je třeba nutnost používání randomizovaných strategií pro zajištění optimálního hraní. I když vlastnosti optimálních strategií v těchto hrách jsou dobře prozkoumány, předchozí výzkum v oblasti MCTS v hrách s neúplnou informací na těchto znalostech nestaví dostatečně. Žádný z existujících MCTS algoritmů u těchto her nekonverguje k optimální strategii, i kdyby měl neomezený čas na výpočet.

V této práci studujeme MCTS v hrách dvou hráčů s nulovým součtem v extenzivní formě s neúplnou informací a zaměřujeme se na herně teoretické vlastnosti produkovaných strategií. Postupujeme ve dvou krocích. Nejprve detailně analyzujeme jednu z nejjednodušších tříd her s neúplnou informací: hry se současnými tahy. Poté přistoupíme k plně generickým hrám s neúplnou informací. Zkoumáme stávající MCTS algoritmy pro tyto druhy her a zařazujeme je do několika principiálně odlišných tříd. Dále prezentujeme následující přínosy:

Za prvé se zaměřujeme na návrh nových MCTS algoritmů, které prokazatelně konvergují k Nashově rovnováze hry s rostoucím výpočetním časem. Navrhujeme tři takové algoritmy. Jeden je založený na menší modifikaci standardní MCTS šablony pro hry se současnými tahy a další dva na adaptaci úspěšného algoritmu Monte Carlo Counterfactual Regret Minimization (MCCRF) pro online hraní her jak v hrách se současnými tahy tak v obecných hrách s neúplnou informací.

Za druhé jsme se zaměřili na zlepšení výkonu MCTS algoritmů a to především tím, že navrhujeme a vyhodnocujeme nové funkce pro výběr akcí, které určují takové akce, které mají být vzorkovány v pozdějších iteracích na základě statistických údajů získaných z předchozích iterací. V obecných hrách s neúplnou informací navrhujeme explicitní modelování hráčových odhadů pravděpodobnosti, že se nachází v konkrétním herním stavu během hry.

Za třetí provádíme rozsáhlé vyhodnocení navržených a existujících MCTS metod na pěti hrách se současnými tahy a čtyřech hrách s neúplnou informací, které mají proměnnou velikost a zásadně odlišné vlastnosti. Hodnotíme schopnost algoritmů rychle konvergovat k Nashově rovnováze a jejich výkony srovnáváme ve vzájemných turnajích. Ukazujeme, že algoritmy založené na MCCFR mají velmi rychlou konvergenci k rovnováze, ale klasické MCTS s novými funkcemi pro výběr akcí vynikají při turnajích ve velkých hrách.

Závěrem prezentujeme případovou studii využití MCTS pro vytváření inteligentních agentů pro robotické hry pronásledování cílů. Navrhujeme doménově specifické varianty navržených algoritmů a vyhodnocujeme jejich úspěšnost ve složitém simulovaném prostředí. Ukazujeme, že algoritmy založené na MCTS překonávají nejlepší známý algoritmus pro tento problém.

Contents

Abbreviations

CFR	Counterfactual Regret Minimization
DRM	Decoupled Regret Matching
DUCT	Decoupled Upper Confidence Bounds applied to Trees
EFG	Two-player zero-sum imperfect-information extensive-form game
Exp3	Exponential-weight algorithm for Exploration and Exploitation
HC	Hannan consistency
IS	Information set
IS-MCTS	Information Set Monte Carlo Tree Search
IST	Information set tree
MAB	Multi-Armed Bandit
MCCFR	Monte Carlo Counterfactual Regret Minimization
NE	Nash equilibrium
NFG	Normal-form game
PEG	Pursuit-evasion game
PIMC	Perfect Information Monte Carlo sampling
RM	(Coupled) Regret Matching
RUCT	Randomized Upper Confidence Bounds applied to Trees
(SM-)MCTS	(Simultaneous-Move) Monte Carlo Tree Search
(SM-)OOS	(Simultaneous-Move) Online Outcome Sampling
SM-MCTS-A	Simultaneous-Move Monte Carlo Tree Search with Averaging
SMG	Simultaneous-move extensive-form game with perfect information
UCB	Upper Confidence Bounds
UCT	Upper Confidence Bounds applied to Trees

Chapter 1

Introduction

Game theory is a formal framework for analyzing the optimal behavior of self-interested agents interacting in a shared environment. It describes the possible developments of these interactions and the computational game theory has recently provided algorithms for computing the optimal strategies for increasingly complex games. These advancements led to expert level game playing algorithms for a range of artificial games (such as Chess [Hsu06], Go[GS11], or Poker[Joh07]), but also to creating strategies for real world problems, such as auctions [CSS06] and various physical or network security, and law enforcement scenarios [Tam11]. Complex game-theoretic strategies computed by non-trivial algorithms are in daily use for example by airport security patrols [JTP+10] and coast guards [SAY+12].

Many of these application domains require the agents to perform sequences of decisions in time. Agents participating in auctions may decide about their next actions based on sequences of the bids made by their opponents, security patrols select their actions based on alerts and observations caused by their earlier actions. Furthermore, many of these domains include hidden information about the actions performed by the opponents. One of the generic game-theoretic models of games with imperfect information that evolve in time are the two-player zero-sum extensive-form games with imperfect information. Computing near-optimal strategies for these games is the main subject of this thesis. One of the main challenges in creating strategies for these games is often their size, even though the problem of finding the optimal strategy for these games is polynomially solvable [KMvS96]. The games are often defined by compact game rules, but the actual explicit representation of the game is usually exponential in several parameters of the game. For example, the No-Limit Texas Hold'em game played at the computer Poker competition has more than 10^{165} terminal states [Joh13]. A robotic visibility-based pursuit-evasion game with two pursuers and one evader on a four-connected graph and 100 moves time limit has more than 10^{180} terminal states.

There are two main approaches to creating strategies in such large games. If the exact definition of the game is known sufficiently long in advance, a smaller abstract version of the game that captures the most important features of the game can be created. The optimal strategy for each situation in the abstract game can be computed or approximated using weeks of parallel computation. The resulting strategy is then used in the original game, based on some mapping from the actions in the abstract game to the actions in the original game. This approach has been successful for example in Poker [San10, Joh07], where the game allows natural abstractions, such as card bucketing. In general, making good abstractions by hand can be difficult and time-consuming and an automatic creation of abstractions may not be computationally feasible for large games [KS14]. Therefore, abstractions can be complemented, or even substituted, by online game playing also often referred to as online search in games. In online game playing, the complete strategy for the game is not computed in advance, but the relevant parts of the strategy are computed during the match, based on the current state of the game. This reduces the need to store the strategy for a large number of game states in advance and allows the programs to work without any, or with a much finer, abstraction of the game. A classic example of using this approach in perfect-information games is the minimax search with evaluation function, which is typically used in chess programs [RN09]. However, a significant jump in the performance of the state-of-the-art solvers for many perfect-information problems, such as the game of Go [GS11], or domain-independent planning under uncertainty [KE12] was recently caused by Monte Carlo Tree Search (MCTS) algorithm. The main idea of Monte Carlo Tree Search is running a large number of randomized simulations of the problem and learning the best actions to choose from the experience. The earlier simulations are generally used to create statistics that help to guide the latter simulations to more important parts of the search space and eventually decide on the best action to play in the current state of the game. In this thesis, we study how the success of MCTS can be translated to games with imperfect information.

Games with imperfect information are fundamentally more complicated than perfect-information games for several reasons. The most important complication is that the optimal strategies in imperfect-information games may require the players to make randomized decisions. For example, in the well-known game of Rock-Paper-Scissors, none of the available actions can be considered optimal. Playing any of the actions all the time can always be exploited by the opponent and the optimal strategy against a rational opponent is to play each action with the same probability. Another important complication is the strong inter-dependency between the strategies in different parts of the game. A player often does not know the exact state of the game during the match and the probability of the game being in the individual possible state can depend on the opponent's strategy in previous decisions in the game, which may generally depend on the optimal strategy in any other decision point in the game.

Game theory provides means to deal with all these complications, but previous attempts to adapt MCTS to imperfect-information games generally did not try to ensure optimality of the strategies from the perspective of game theory. The algorithms were developed mainly based on heuristics and analogies with the perfect-information case. We show that the lack of theoretical foundations of the algorithms causes pathologies, such as more computation time per move leading to worse, more exploitable, strategies. There are several fundamental concepts in game theory, which can help in designing algorithms for creating strategies in complex games. The notion of Nash equilibrium defines the optimal strategy against the strongest possible opponent with well-understood quality guarantees against suboptimal opponents. Moreover, refinements of this concept define what action to play if the opponent performs clearly irrational moves.

In this thesis, we review the existing MCTS algorithms for imperfect-information games and we propose several novel variants of these algorithms. We build on the foundations of game theory and propose algorithms that are either guaranteed to converge to a Nash equilibrium, or at least use the insights from game theory to allow reasonable approximations of a Nash equilibrium strategies. Our main aim is not creating the strongest computer player for a specific game, but rather providing general algorithms with reasonably good performance over a wide range of different games.

1.1 Research Questions

Based on the motivation above, this thesis tries to answer the following research questions:

What are the fundamental principles of the existing MCTS algorithms for imperfect-information games? We review the existing research on the use of Monte Carlo Tree Search algorithms in two-player zero-sum extensive-form games with imperfect information (EFGs). We focus both on the existing algorithms and data structures they use, as well as any possible theory describing their performance guarantees. We also briefly review the most important research in MCTS for perfect-information games and the basic results about algorithms for multi-armed bandit problem, which often form the basis of MCTS algorithms.

What theoretical guarantees are provided by the existing algorithms? The most desirable property of online game playing algorithms is a rapid and continual increase of the quality of the computed solution even with short computation times. In this thesis, we conclude that formal theoretical guarantees are known only for MCTS in perfect-information setting. We formally analyze the convergence of our novel MCTS algorithms to Nash equilibria and at least empirically evaluate this property for algorithms that do not have formal guarantees.

Can we design better MCTS algorithms for these games using insights from game theory? Eventual convergence to a Nash equilibrium or a similar well-understood game-theoretic solution concept is important for online game playing algorithms. It can guide their design and provide formal guarantees on the quality of the produced solutions. We design our novel algorithms so that they are guaranteed to converge to an approximate Nash equilibrium and use the game-theoretic concept of consistent beliefs over information sets to substantially improve the existing algorithms.

How are the theoretical properties connected to the actual performance in various domains? Formal properties of algorithms often do not correlate with their actual performance on specific problems. We have implemented most of the discussed algorithms in a unified domain-independent framework and compare the actual performance of the algorithms in mutual matches with their more abstract convergence properties on a wide range of specific domains.

Is it possible to use MCTS for imperfect-information games to solve real-world problems? The model of extensive-form games could possibly be too restrictive to model real-world problems of useful size or the computational requirements could be too high to compute good strategies in a reasonably short time. In the last chapter of this thesis, we focus on an existing model of a realistic problem of visibility-based pursuit and evasion. We show that selected MCTS algorithms adapted to this problem outperform the pre-existing state-of-the-art algorithm for this problem.

1.2 Organization of the Thesis

Chapter 2 summarizes the basic game-theoretic definitions related to the game models and solution concepts used throughout the thesis. Afterwards, it presents the most important results and algorithms related to the multi-armed bandit problem, which is a fundamental model of exploration-exploitation problems and an important component in many existing MCTS algorithms. Next, it briefly summarizes the most important results about MCTS in perfect-information problems. Finally, Chapter 2 presents the Counterfactual Regret Minimization algorithm, which is the basis for two of our novel MCTS algorithms.

Chapter 3 presents all the existing and novel MCTS algorithms for imperfect-information games. We first focus on the simplest case of imperfect information, which are game with simultaneous moves, but otherwise perfect information. We first present the algorithms and then discuss their convergence guarantees. After the simultaneous-move games, Chapter 3 continues with the description of the algorithms for generic two-player zero-sum imperfect-information extensive-form games without any further restrictions.

Chapter 4 presents an evaluation of the algorithms described in Chapter 3 on a range of artificial and randomly generated games based on either card and board games played by people or abstracted versions of real-world problems. This chapter first describes the rules of the games and then presents the results in one game after another, first for all simultaneous-move games and then for all generic imperfect-information games. The presentation of each of these sections is concluded by a summarizing discussion.

Chapter 5 presents a case study of application of MCTS algorithms in a realistic problem of a robotic visibility-based pursuit-evasion game. It first presents the previous research on the problem and then formally defines the problem as an extensive-form game. The chapter then presents the domain-specific modifications of the generic algorithms proposed in Chapter 3. The algorithms are then evaluated in a complex simulation platform developed in Agent Technology Center over a series of projects in the past years.

Chapter 6 concludes the thesis, highlights the most important contributions of this thesis, and presents suggestions for future work.

Chapter 2

Background

This chapter summarizes the basic game-theoretic definitions related to the game models and solution concepts used throughout the thesis. Afterwards, it presents the most important results and algorithms related to the multi-armed bandit problem, which is a fundamental model of exploration-exploitation problems and an important component in many existing MCTS algorithms. Next, it briefly summarizes the most important results about MCTS in perfect-information problems. Finally, it presents the Counterfactual Regret Minimization algorithm, which is the basis for two of our novel MCTS algorithms.

2.1 Basic Game-Theoretic Definitions

This section introduces the most fundamental concepts of game theory that will be used in this thesis. We base most of the definitions on [OR94] and [SLB08]. Game theory formally studies interactions between agents who are selecting actions to maximize their own reward. Such an agent is further termed a *rational* agent.

2.1.1 Normal-Form Games

Normal-form games are the most basic game model used in game theory. The basic components of this model must be present in any game.

Definition 2.1.1 (Normal-form game). *A* normal-form game *consists of*

- $\mathcal{N}$ *- a set of players;*
- $\mathcal{A}_i$ $i \in \mathcal{N}$ *- a set of actions also called pure strategies for each player;*
- $u_i : \Pi_{i\in\mathcal{N}}\mathcal{A}_i \to \mathbb{R}$ *- a real-valued utility function each of the players maximizes.*

In normal-form games, each player $i \in \mathcal{N}$ selects an action a_i from his action set $\mathcal{A}_i$ and then plays it simultaneously with all the other players in the game. Afterwards, it receives a pay-off $u_i(a)$ depending on the actions selected by other players. Besides choosing a specific action, the players can chose to play a *mixed strategy* $\sigma_i \in \Delta(\mathcal{A}_i)$, which is a probability distribution over their possible actions. In that case, the reward received by the players is the expected value with respect to the distributions of all players. As a result, we extend the utility function u_i to be applicable also to mixed strategies with the meaning of this expected value.

For each *action profile* $a \in \Pi_{i\in\mathcal{N}}\mathcal{A}_i$ containing one action for each player and player $i \in \mathcal{N}$, we denote the action of player i in a by a_i and the actions of all the other players besides i by a_{-i}. We use the same convention for *strategy profiles* $\sigma \in \Pi_{i\in\mathcal{N}}\Delta(\mathcal{A}_i)$. A *zero-sum* game is a two player game ($|\mathcal{N}| = 2$), such that for each action profile $a \in \Pi_{i\in\mathcal{N}} cA_i$, the reward for one player is the loss of the other player ($u_i(a) = -u_{-i}(a)$). A game is not substantially modified if the utility functions are shifted, hence zero-sum games are equivalent to *constant-sum* games, in which for all action profiles a holds $u_i(a) + u_{-i}(a) = c \in \mathbb{R}$. If a game is not constant sum, it is termed a *general-sum* game.

The utility function for zero-sum games can be represented as a matrix with rows indexed by the actions of one player and columns indexed by the actions of the other player. Hence, the zero-sum normal-form games are also called *matrix games.* If $\sigma_1 \in \Delta(\mathcal{A}_1)$ is first player's strategy represented as a row vector and $\sigma_2 \in \Delta(\mathcal{A}_2)$ if second player's strategy represented as a column vector, then the expected value of player 1 when both players play with these strategies is $u_1(\sigma_1, \sigma_2) = \sigma_1 M \sigma_2$.

2.1.2 Extensive-Form Games

We adapt the definition of the Extensive-form game (EFG) from [OR94].

Definition 2.1.2 (Extensive-form game). *An extensive-form game with imperfect information consists of:*

- *$\mathcal{N}$ is the set of players including the* nature *player (c) that represents the dynamics of the game;*
- *for each $i \in \mathcal{N}$, $\mathcal{A}_i$ is the set of actions available to player i;*
- *$\mathcal{H}$ is the set of possible states of the game, each corresponding to a* history *of actions performed by all players;*
- *$\mathcal{Z} \subseteq \mathcal{H}$ is the set of terminal game states or histories;*
- *$\mathcal{P} : \mathcal{H} \setminus \mathcal{Z} \to \mathcal{N}$ is a function assigning each non-terminal state a player that selects an action in the state;*
- *$\mathcal{A} : \mathcal{H} \setminus \mathcal{Z} \to 2^{\mathcal{A}_i}$ is a function assigning each non-terminal game state the actions applicable by the acting player;*
- *$\mathcal{T} : \mathcal{H} \times A_i \to \mathcal{H} \cup \mathcal{Z}$ is the transition function realizing one move of the game;*
- *$u_i : \mathcal{Z} \mapsto \langle v_{\min}, v_{\max}\rangle \subseteq \mathbb{R}$ gives the utility of player 1, with v_{min} and $v_{\max}$ denoting the minimum and maximum possible utility respectively. Without loss of generality we sometimes assume $v_{\min} = 0$, $v_{\max} = 1$, and $\forall z \in \mathcal{Z}, u_2(z) = 1 - u_1(z)$.*

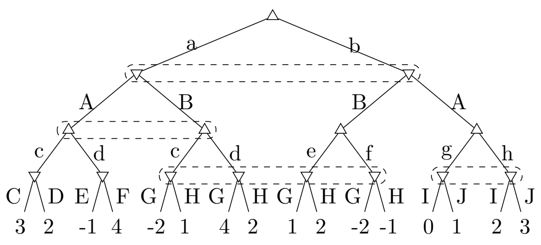


Figure 2.1: Example of a zero-sum imperfect-information extensive-form game.

- *$\mathcal{I}_i$ for player i represent player's imperfect information about the game. It is a partition of $\mathcal{H}_i = \{h \in \mathcal{H} : \mathcal{P}(h) = i\}$ termed* information sets. *Each information set $I \in \mathcal{I}_i$ represents the set of histories that are indistinguishable for player i.*

- $\Delta_c(h) \in \Delta(\mathcal{A}(h))$ *is the commonly known probability distribution of nature player's actions.*

The game starts with the empty history $\emptyset$. In each history h, player $P(h)$ selects an action $a \in \mathcal{A}(h)$. After the action is performed, the game proceeds to history $h' = \mathcal{T}(h, a)$, often also denoted ha. The game ends when it reaches a terminal history $h \in \mathcal{Z}$. At that point, each player $i \in \mathcal{N}$ receives the pay-off $u_i(h)$. We denote by $h \sqsubseteq h'$ the fact that history h is a *preffix* of history h' and by $h \sqsubset h'$ that $h \sqsubseteq h'$ and $h \neq h'$.

Extensive-form games can be represented by a game tree, such as the one in Figure 2.1. The set of histories $\mathcal{H}$ are the nodes in the tree and $\mathcal{Z}$ are the leaves. In zero-sum games, we denote the nodes where first player selects an action ($\mathcal{H}_1$) by $\triangle$ and the nodes of the second player (minimizing the utility of the first player) by $\bigtriangledown$. The nodes of the chance player are denoted by $\bigcirc$. In this example, the action sets of the players are $\mathcal{A}_1 = \{a, b, \ldots, h\}$, $\mathcal{A}_2 = \{A, B, \ldots, J\}$. We denote the information sets as the ellipses around the tree nodes. After the history $h = aA$ player $\triangle$ decides about the next move from $\mathcal{A}(h) = \{c, d\}$, but in the game, he has exactly the same information as if $h' = aB$ was the current state of the game. The numbers in the leaves denote the utility of the first player.

In order for the have the information sets well-defined, we require that $\forall h, h' \in I \in \mathcal{I}_i \; \mathcal{A}(h) = \mathcal{A}(h') \;\&\; \mathcal{P}(h) = \mathcal{P}(h')$; hence, we extend the notation $\mathcal{P}(I)$ and $\mathcal{A}(I)$ to be applicable also to information sets. We denote $I(h)$ the information set containing h. Another natural requirement to information sets is that the player does not forget his own actions during the game. If we denote for history $h \in \mathcal{H}$ and player $i \in \mathcal{N}$ by h_i the sequence of only the actions of player i in history h, then we say that a game has *perfect recall* if $\forall h, h' \in I \in \mathcal{I}_i \; h_i = h'_i$. A game in which each information set contains only one history is called a *perfect-information* game. If at least one information set contains more than one history, it is a game with *imperfect-information.*

A *pure strategy* in an extensive-form game is a function, which assigns each information set $I \in \mathcal{I}_i$ of player i an action from $\mathcal{A}(I)$. Example pure strategies in Figure 2.1 is $s_\triangle = \{a, d, f, g\}$ for the maximizer and $s_\bigtriangledown = \{B, D, G, J\}$ for the minimizer. Any extensive-form game can be seen as a normal-form game, in which the set of actions is the set of pure strategies of the extensive-form game. Each pair of pure strategies naturally defines the utility function as the expected value of playing this pair of strategies. The expectation is taken over the actions in the chance nodes, in which the action is selected based on a commonly known probability distribution. The pair of strategies $s_\triangle, s_\triangledown$ in the game without chance nodes uniquely defines utility of 4 for player $\triangle$ and -4 for player $\triangledown$. The set of *mixed strategies* of an extensive-form game is defined as the set of all probability distributions over the pure strategies. In games with perfect recall, a theorem proposed by Khun [Kuh53] allows us to use the following equivalent, but much more compact, representation of the mixed strategies. A *behavioral strategy* $\sigma_i : \mathcal{I}_i \to \Delta(\mathcal{A}_i)$ assigns each information set a probability distribution over the available actions. If it is not specified otherwise, we mean by a strategy in an extensive-form game the behavioral strategy and we denote the set of all strategies of player i by Σ_i.

For any fixed strategy profile $\sigma \in \Sigma = \Pi_{i\in\mathcal{N}}\mathcal{A}_i$, we define the *belief* B_I^σ over the information set I given σ to be a probability distribution over the histories in I consistent with σ:

$$B_I^\sigma(h) = \Pi_{i\in\mathcal{N}}\Pi_{a_i\in h}\sigma_i(I(h), a_i)/\pi^\sigma(I) \tag{2.1}$$

Where $\pi^\sigma(I)$ is the normalization factor corresponding to the probability of reaching the information I when players play strategy profile σ. In two player games of perfect recall, the belief depends only on the strategy of the opponent of the player deciding in I and the probabilities of nature moves.

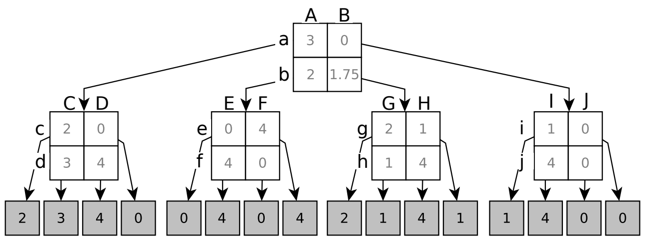
We call a *match* of an extensive-form game the process in which a the game environment starts the game in its initial state, informs the players about the current information set it it is their turn, collects the actions selected by the players and select the actions for the nature based on distribution δ_c. Each complete match corresponds to a terminal history in the extensive from game and its outcome is the utility of the players in this terminal history.

2.1.3 Perfect-Information Simultaneous-Move game

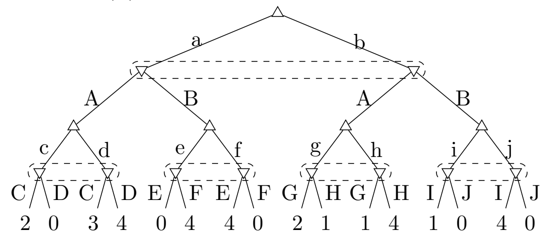
One of the simplest forms of imperfect information in an extensive-form game are simultaneous moves of the players. Simultaneous-move extensive-form games progresses in steps. In each step, both players perfectly know the current state of the game, but decide on the action to play simultaneously. After the actions are played, the game transitions to the following state, which is disclosed to both players. Simultaneous-move games can be viewed as a tree of normal-form games (see Figure 2.2a), in which each action profile leads to another subtree or a terminal node containing the rewards. The numbers in the non-terminal nodes of the game are not a part of the game and correspond to the optimal solution in the subtree.

Formally, a finite zero-sum game with perfect information and simultaneous moves (SMG) can be described by a tuple $(\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, \mathcal{T}, u_1, h_0)$, where

- $\mathcal{N} = \{1, 2\}$ contains player labels;
- $\mathcal{H}$ is a set of histories (states) in the game;



(a) Simultenoeus move game



(b) Equivalent extensive-form game

Figure 2.2: A game tree of a game with perfect information and simultaneous moves. Only the leaves contain the actual rewards; the remaining numbers are the expected reward for the optimal strategy.

- $\mathcal{Z} \subseteq \mathcal{H}$ denotes the terminal histories;
- $\mathcal{C} \subseteq \mathcal{H}$ denotes the histories in which chance select the next state;
- $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ is the set of joint actions of individual players and we denote $\mathcal{A}_1(h)$ and $\mathcal{A}_2(h)$ the actions available to individual players in state $h \in \mathcal{H}$;
- $\mathcal{T} : \mathcal{H} \times \mathcal{A}_1 \times \mathcal{A}_2 \mapsto \mathcal{H} \cup \mathcal{Z}$ defines the successor state given a current state and actions for both players. For brevity, we sometimes denote $\mathcal{T}(h, i, j) \equiv h_{ij}$.
- $\Delta_c : \mathcal{H} \mapsto \Delta(\mathcal{H})$ is the chance transition function;
- $u_i : \mathcal{Z} \to \mathbb{R}$ for each $i \in \mathcal{N} \setminus \{c\}$ are the utility functions of the players.

The simultaneous-move games can be represented as extensive-form games with imperfect information. Figure 2.2b presents the equivalent representation of the SMG as EFG. In EFG, the players choose their moves sequentially, but information sets are used to create information structure required by the simultaneous-move games. The decision about the move of the player that decides first is hidden form the second player until he selects his own move. The chance decisions not included in this example map to chance nodes. Regardless of the equivalence to EFGs, we use the definition of SMG, because it simplifies definition of algorithms and their formal analysis. The main advantage is the ability to easily refer to the joint actions of the players and properties of the "matrix games" in the inner nodes.

2.1.4 Basic Solution Concepts

The following solution concepts are applicable to all game definitions we introduce above. The only difference is the structure of the strategy sets. Regardless of this structure, we denote a set of all strategies of player $i \in \mathcal{N}$ by Σ_i and $\Sigma = \Pi_{i \in \mathcal{N}} \Sigma_i$.

Definition 2.1.3 (Best response)**.** *For a strategy profile σ, we define a* best response *of player i to the other players' strategies σ_{-i} as the strategy*

$$\mathsf{br}(\sigma_{-i}) \in \operatorname*{argmax}_{\sigma'_i \in \Delta(\mathcal{A}_i)} u_i(\sigma'_i, \sigma_{-i}). \tag{2.2}$$

One of the most fundamental solution concepts in game theory is the Nash equilibrium (see e.g., [OR94]).

Definition 2.1.4 (Nash equilibrium)**.** *A strategy profile $\sigma^* \in \Sigma$ is an* ϵ-Nash Equilibrium

$$\textit{if } u_i(\sigma_i, \sigma^*_{-i}) - u_i(\sigma^*) \leq \epsilon \; \forall i \in \mathcal{N}, \sigma_i \in \Sigma_i.$$

A strategy profile is an exact Nash equilibrium (NE) if $\epsilon = 0$.

In words, in Nash equilibrium, each player plays the best response to the strategies of the other players. In zero-sum games, a Nash equilibrium is a very appealing strategy to play. It prescribes a (generally randomized) strategy, which is optimal in several aspects. It is a strategy that gains the highest expected reward against its worst opponent, even if the opponent knows the strategy played by the player in advance. Moreover, in the zero-sum setting, even if the opponent does not play rationally, the strategy still guarantees at least the reward it would gain against the rational opponent. This guaranteed reward is termed the *value* of the game. The *support* of a strategy σ_i is the set of pure strategies of player i, which have non-zero probability of being played in σ_i.

A game can have, in general, infinite number of Nash equilibria. A *refinement* of the concept of Nash equilibrium is a subset of the equilibria, which satisfies some additional conditions. The most common NE refinement in perfect-information extensive-form games, which is also applicable in simultaneous-move games is *subgame perfect* Nash equilibrium. A *subgame* is a subtree of the game tree that is rooter in a singleton information set and no information set below this root spans out of this subtree. More formally:

Definition 2.1.5 (Subgame)**.** *A* subgame *of an extensive-form game $\Gamma = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{P}, \mathcal{A}, \mathcal{T}, \mathcal{I}, u_i)$ rooted in a history $h_r \in \mathcal{H}$ is the extensive-form game $\Gamma' = (\mathcal{N}, \mathcal{H}', \mathcal{Z}', \mathcal{P}, \mathcal{A}, \mathcal{T}, \mathcal{I}, u_i)$ generated by a subset of histories $\mathcal{H}' \subseteq \mathcal{H}$, such that*

$$|I(h)| = 1 \;\&\; \forall h \in \mathcal{H}' \; h_r \sqsubseteq h \;\&\; \forall h \in \mathcal{H}' \; h' \in I(h) \Rightarrow h' \in \mathcal{H}'.$$

A subgame perfect NE is then a NE, which is also a NE in each subgame of the game. This generally does not have to be the case, because if some action in the game would never be played by a rational player, the NE strategy below this action can be completely arbitrary. For example in the game in Figure 2.2, action A of the minimizer is always worse than action B regardless of whether the opponent plays a or b. This means that a rational player will never play A and any strategy in the subgame below bA can be part of a Nash equilibrium. However, only the equilibria that prescribe playing e,f and E,F with uniform probability are subgame perfect.

In generic EFGs without further restrictions on information sets, well defined non-trivial subgames are rare. In many games, the complete game is the only subgame. The equilibrium refinements are much more complex in this case. See [VD91] for extensive survey. We have evaluated the empirical performance of various equilibrium refinements in [CBL14]. In this thesis, we will mention only one of the more complex equilibria called *sequential equilibrium* introduced in [KW82]. This equilibrium concept takes into account the possibility that the opponent will make a mistake and prescribe meaningful strategies after this point. A strategy profile is *completely mixed* if it assigns a non-zero probability larger than some $\epsilon > 0$ to playing each action in each information set. A strategy profile σ^* is a sequential equilibrium, if there is a belief distribution B^*, such that $(\sigma^*, B^*) = \lim_{n\to\infty}(\sigma^n, B^{\sigma^n})$ for some sequence of fully mixed σ^n. And also, σ^*_i is the best response to σ^*_{-i} in all information sets, given beliefs B^*.

The distance of a strategies from a Nash equilibrium can be measured in terms of *exploitability* (e.g., [JWBM11]), which is the difference between the value of the games (i.e., the value of the best response to the optimal strategy) and the value of the best response to the evaluated strategy.

Definition 2.1.6 (Exploitability)**.** *The exploitability of strategy σ_i is*

$$expl(\sigma_i) = v_i - u_i(\sigma_i, \mathsf{br}(\sigma_i))$$

where v_i is the value of the game for player i. The exploitability of a strategy profile σ is

$$expl(\sigma) = u_i(\sigma_i, \mathsf{br}(\sigma_i)) - u_i(\mathsf{br}(\sigma_{-i}), \sigma_{-i}).$$

An alternative solution concept, which is often important in analysis of learning algorithms, allows the player to coordinate their action selection based on external events. One of the common ways to define the equilibrium is to assume there is an impartial mediator in the game. A *correlated equilibrium* is a probability distribution over the set of all pure strategy profiles. If the mediator selects a profile based on this distribution and discloses each player only his own action and not the actions of the other players, a rational player plays this action, under the assumption that all the other players also obey their suggestion. Many common learning dynamics generally converge to correlated equilibria of a game rather than Nash equilibria. However, the set of all correlated equilibria of a game is a superset of the set of all Nash equilibria [OR94]. Moreover, in zero-sum games, each correlated equilibrium directly corresponds to a Nash equilibrium, because the strategy suggestions to the players are not correlated.

2.1.5 Computing Nash Equilibria

Nash equilibria in zero-sum games can be computed by a linear program for both normal-form games [SLB08] and extensive-form games [KMvS96]. These algorithms are polynomial in the size of the game, but because the size of the game is, in general, exponential in the parameters that define games in the real world, research has focused on improving scalability of NE computation in large extensive from games.

This research includes our double-oracle alpha-beta algorithm for simultaneous-move games [BLC$^+$13, BLL$^+$14], which we use for computing the basic properties of the small versions of the simultaneous-move games used for evaluation in this thesis. As well as the double-oracle algorithm for generic extensive-form games [BKL$^+$13, BKLP14], we use for computing game values of the small versions of generic extensive-form games. The advantage of the methods described above is that they compute and exact Nash equilibrium (within the precision of LP solvers).

For practical applications (e.g., the Computer Poker competition), it is common to compute a very good approximation of the NE, instead of the precise equilibrium. The most effective recent methods for this problem are the Counterfactual Regret Minimization [ZJBP08] we describe in Section 2.5 and the Excessive Gap Technique [HGPnS10].

2.2 Multi-Armed Bandit Problem

Multi-armed bandit (MAB) problem is one of the most basic models in online learning [BCB12]. In theoretical studies, it is often the basic model for studying fundamental trade-offs between exploration and exploitation in an unknown environment [ACBFS95, ACBF02]. In the practical applications, the algorithms developed for this model has recently been very successfully used in online advertising [PCA07], generic optimization [FKM05], and most importantly for the topic of this thesis in Monte Carlo tree search algorithms [KS06, BPW$^+$12, GS11, TF11].

The multi-armed bandit problem got its name after a simple motivating example concerning slot machines in casinos, also known as one-armed bandits. Assume you have a fixed number of coins n you want to use in a casino with K slot machines. Each slot machine has a hole where you can insert a coin and as a result, the machine will give you some (often zero) reward. Each of the slot machines is generally different and decides on the size of the rewards using a different mechanism. The basic task is to use the n coins sequentially, one by one, to receive the largest possible reward. Intuitively, it is necessary to sufficiently explore the quality of the machines, but not to use too many coins in the machines that are not likely to be good. The following formal definitions use notations form an extensive survey of the field by Bubeck [BCB12].

Definition 2.2.1 (Multi-armed Bandit Problem)**.** *The multi-armed bandit problem is a set of actions (or arms) denoted* $1, \dots, K$*, and a set of sequences* $X_{i,t}$ *for each action* i *and time step* $t = 1, 2, \dots$*. In each time step, an agent selects an action* I_t *and receives the payoff* $X_{I_t,t}$*. In general, the agent does not learn the values* $X_{j,t}$ *for* $j \neq I_t$*.*

Different variants of MAB problem pose various assumption on the reward sequences $X_{i,t}$. The algorithms for solving the MAB problem usually optimize some notion of regret. Intuitively, the algorithms try to minimize the difference between playing the prescribed strategy and playing some baseline strategy, which can possibly use information not available to the agent. For example, the most common notion of regret is the *external regret*, which is the difference between playing according to the prescribed strategy and playing the fixed optimal arm all the time.

Definition 2.2.2 (External Regret). *The* external regret *for playing a sequence of actions $I_1, \dots, I_n$ is defined as*

$$R_n = \max_{i=1,\dots,K} \sum_{t=1}^{n} X_{i,t} - \sum_{t=1}^{n} X_{I_t,t}.$$

The payoff sequences $X_{i,t}$ as well as the agent selections I_t are often assumed to be randomized. In these situations, we are interested in average case regret. There are two slightly different definitions of this notion.

Definition 2.2.3 (Expected Regret). *The* expected regret *is the expected value of regret with respect to the randomization of the reward sequences and agent's decisions:*

$$\mathbb{E}\, R_n = \mathbb{E}\left[\max_{i=1,\dots,K} \sum_{t=1}^{n} X_{i,t} - \sum_{t=1}^{n} X_{I_t,t}\right].$$

The pseudo-regret *is the maximum of the expected difference between the agent's strategy and always choosing the best fixed action all the time:*

$$\bar{R_n} = \max_{i=1,\dots,K} \mathbb{E}\left[\sum_{t=1}^{n} X_{i,t} - \sum_{t=1}^{n} X_{I_t,t}\right].$$

The main difference between the notions is that the expected regret allows a different maximal action in each run of the system considered in the expectation, while the pseudo-regret compares to a single fixed action selection for all runs. These two notions of regret coincide if the rewards are selected independently of the agent's selection, such as, if reward sequences are fixed in advance, before the agent starts selecting actions. For this thesis, we will mainly focus on the pseudo-regret, because of its connection to game-theoretic notions. If it is not specified explicitly, by regret we mean the pseudo-regret.

2.2.1 Stochastic Multi-Armed Bandit Problem

The basic variant of the MAB problem is the stochastic bandit problem, first analyzed by [Rob52, LR85].

Definition 2.2.4 (Stochastic MAB). *The* stochastic *MAB problem is a MAB problem, in which each action corresponds do a fixed probability distribution on $\langle 0, 1\rangle$ and $X_{i,t}$ are assumed to be independent draws from these distributions.*

The distributions of the rewards are assumed to be unknown to the agent. In this setting, if the means of the distributions corresponding to individual actions are μ_i and $\mu^* = \max_{i=1,\dots,K} \mu_i$, then the pseudo-regret can be simplified to

$$\bar{R_n} = n\mu^* - \mathbb{E} \sum_{t=1}^{n} X_{I_t,t}. \tag{2.3}$$

The expectation in the formula above is over the agent's randomized action selection. For a fixed setting and any time, the first term is a constant. As a result,

```
Input: $K$-number of actions;
1: for $n \leftarrow 1, 2, \dots$ do
2:     if $n \leq K$ then
3:         $i \leftarrow n$
4:     else
5:         $i \leftarrow \arg\max_i \bar{x}_i + \sqrt{\frac{2 \ln n}{n_i}}$
6:     $r \leftarrow$ reward for playing action $i$
7:     $\bar{x}_i \leftarrow \frac{n_i \bar{x}_i + r}{n_i + 1}$
8:     $n_i \leftarrow n_i + 1$
```

Figure 2.3: Upper Confidence Bounds algorithm for regret minimization in stochastic bandit setting.

minimizing the regret is equivalent to maximizing the expected reward obtained by the agent.

The fundamental problem behind solving the stochastic MAB problem is finding the optimal trade-off between exploration and exploitation. After certain number of samples, the agent can estimate the means of the distributions behind the actions. If the agent takes only the seemingly optimal action all the time from this point on (i.e., pure exploitation), we would never learn if our initial estimate was wrong and the agent may suffer some positive regret. On the other hand, if the agent explores too much and wastes iterations on probing actions than are not likely to be optimal, he receives smaller rewards and suffers positive regret.

The most popular algorithm for solving the stochastic MAB is the Upper Confidence Bounds (UCB1) algorithm [ACBF02]. We present the algorithm in Figure 2.3. The algorithm maintains the mean of the rewards received for each action $\bar{x}_i$ and the number of times each action has been used n_i. It first uses each of the actions once and then decides what action to use based on the size of the one-sided confidence interval on the reward computed based on the Cheroff-Hoeffding bounds (line 5). After the reward for playing the action is received, it updates the mean reward and the number of action uses are updated (lines 7-8).

It is important to realize that this algorithm is fully deterministic, given a fixed sequence of rewards. We will use this observation later. Auer et al. [ACBF02] has proven the following theorem about UCB.

Theorem 2.2.5. *For any stochastic MAB problem with K actions and arbitrary reward distributions on $\langle 0, 1 \rangle$ with means $\mu_1, \dots, \mu_K$, the pseudo-regret after any number n of time steps is*

$$\bar{R_n} \leq \left[8 \sum_{i:\mu_i<\mu^*} \left(\frac{\ln n}{\Delta_i} \right) \right] + \left(1 + \frac{\pi^2}{3} \right) \left(\sum_{j=1}^{K} \Delta_j \right) \quad \in O(\ln n)$$

where $\Delta_i = \mu^* - \mu_i$.

The theorem says that the pseudo-regret grows logarithmically in the number of trials. As a result the average regret per move $\frac{\bar{R_n}}{n}$ quickly approaches zero. This convergence rate is asymptotically best possible [BCB12].

```
Input: $K$ - number of actions; $\gamma$ - exploration parameter
1: $\forall_i G_i \leftarrow 0$
2: for $t \leftarrow 1, 2, \dots$ do
3:     $\forall_i p_i \leftarrow \frac{exp(\frac{\gamma}{K} G_i)}{\sum_{j=1}^{K} exp(\frac{\gamma}{K} G_j)}$
4:     $p'_i \leftarrow (1-\gamma)p_i + \frac{\gamma}{K}$
5:     Use action $I_t$ from distribution $p'$ and receive reward $r$
6:     $G_{I_t} \leftarrow G_{I_t} + \frac{r}{p'_{I_t}}$
```

Figure 2.4: Exponential-weight algorithm for Exploration and Exploitation (Exp3) algorithm for regret minimization in adversarial bandit setting

2.2.2 Adversarial Multi-Armed Bandit Problem

In the stochastic bandit setting, the sequences of rewards are assumed to come from a fixed probability distribution. As a result, they have a well-defined expected value and it is clear that in the long run, the agent should almost always use the action corresponding to the largest of these expectations. In adversarial bandit problem, there is no stochastic assumption about the rewards and the reward sequences can be completely arbitrary.

Definition 2.2.6 (Adversarial MAB)**.** *The* adversarial *MAB problem is a MAB problem, in which in each time step an adversary selects arbitrary rewards $X_{i,t} \in \langle 0, 1 \rangle$ simultaneously with the agent selecting the action.*

In adversarial bandit setting, it is important to distinguish between the *oblivious* adversary, who is selecting the rewards independently of the agent's previous choices and the *non-oblivious* adversary, who can take the agent's previous choices into account in setting the pay-offs for the next round. Minimizing regret in adversarial bandit setting is substantially more difficult than in the stochastic setting. Let us first consider an agent playing UCB. UCB is a deterministic strategy and as with any other deterministic action selection strategy, the adversary can cause the player to suffer a large regret $\bar{R_n} \geq \frac{K-1}{K} n$. Even the oblivious adversary can simulate the execution of the deterministic strategy. At time t, the adversary can select the reward of 0 for the action I_t, which will be selected by the agent and 1 for all the other actions. Even if all the actions are selected equally often by the agent, both pseudo-regret and expected regret of the agent will be at least $\frac{K-1}{K} n$.

The only way how an agent can guarantee a smaller regret against an adversary is to randomize its action selections. The most popular algorithm for minimizing regret in adversarial bandit setting is the Exponential-weight algorithm for Exploration and Exploitation (Exp3) algorithm proposed in [ACBFS03] and further improved in [Sto05]. The algorithm has many different variants for various modifications of the setting and desired properties. We present a formulation of the algorithm based on [ACBFS03] in Figure 2.4.

Exp3 stores the estimates of cumulative reward of each action over all iterations, even in case the action was not selected. In the pseudo-code in Figure 2.4, we denote this value for action i by G_i. It is initially set to 0 on line 1. In each iteration, a probability distribution p is created proportionally to the exponential of these estimates. The distribution is combined with a uniform distribution with probability γ to ensure sufficient exploration of all actions (line 4). After an action is selected and the reward is received, the estimate for the performed action is updated using *importance sampling* (line 6). I.e., the reward is weighted by one over the probability of using the action. As a result, the expected value of the reward estimated only from the time steps where the agent selected the action is the same as the expected reward over all the time steps.

The exponential weights are suitable for formal analysis of the algorithm, but cause problems in implementations. Already after few thousands time steps, line 3 of the algorithm requires dividing numbers out of the precision of basic data types for floating point numbers. For this reason, we have derived a more numerically stable version of the algorithm.

$$\frac{\exp(\frac{\gamma}{K}G_i)}{\sum_{k=1}^{K}\exp(\frac{\gamma}{K}G_k)} = \frac{\exp(\frac{\gamma}{K}G_i)}{\exp(\frac{\gamma}{K}G_i)(\sum_{k=1}^{K}\exp(\frac{\gamma}{K}(G_k - G_i))} = \frac{1}{1+\sum_{k=1;\, k\neq i}^{K}\exp(\frac{\gamma}{K}(G_k - G_i)} \tag{2.4}$$

Not that this reformulation changes the number of call to computing the expensive exponential function from $2K$ to $K(K-1)$, which causes each round of the algorithm to be substantially more expensive with higher number of actions. However, without this modification, the algorithm was not able to converge to good solutions. Practically the same version of the numerically more stable formulation of Exp3 was also proposed in [CPW12]. The following upper bound on the regret of Exp3 was derived in [ACBFS03].

Theorem 2.2.7. *For adversarial MAB problem with oblivious adversary, K actions and any $\gamma \in (0,1]$*

$$\mathbb{E}\,R_n \leq (e-1)\gamma n + \frac{K\ln K}{\gamma}.$$

In other words, the average regret per move decreases quickly to $(e-1)\gamma$, but it is not guaranteed to go any lower. A direct consequence of the definition of regret is that $\bar{R}_n \leq \mathbb{E}\,R_n$; hence, the same bound holds also for pseudo-regret. It is possible to create variants of Exp3 that gradually decrease the exploration parameter and eventually reach arbitrarily low regret. Most of these algorithms periodically restart the algorithm with smaller exploration parameter and forget all the previously seen rewards. For applications in MCTS, this behavior seems to be wasteful. Alternatively, a version of Exp3 with losses and without the explicit exploration is proven to converge at a better rate in [Bub10]. However, the empirical speed of convergence when used with MCTS varies a lot and there is not clear way to tune it, without the exploration factor. For these reasons, we use the formulation presented in Figure 2.4 in this thesis.

Based on the theorem above, Exp3 reaches low pseudo-regret in expectation. However, it is not necessarily the case for each run of the algorithm. For example,

```
Input: $K$ - number of acitons; $\eta_t$ - non-increasing sequence of real numbers; $\gamma, \beta \in [0,1]$
1: $\forall_i G_i \leftarrow 0$
2: for $t \leftarrow 1, 2, \ldots$ do
3:     $\forall_i p_i \leftarrow (1-\gamma)\frac{\exp(\eta_t G_i)}{\sum_{j=1}^{K}\exp(\eta_t G_j)} + \frac{\gamma}{K}$
4:     Use action $I_t$ from distribution $p$ and receive reward $r$
5:     $G_{I_t} \leftarrow G_{I_t} + \frac{r+\beta}{p_{I_t}}$
```

Figure 2.5: Exponential-weight algorithm for Exploration and Exploitation (Exp3.P) algorithm for minimizing regret with high probability.

the variance of the loss estimated is in the order of $1/p_{i,t}$, which can be arbitrarily large. In order to have regret guarantees with high probability, different variant of the algorithm denoted Exp3.P has been developed [ACBFS03]. We base our presentation of the algorithm in Figure 2.5 on [Bub10]. The main ideas of the algorithm is to (1) bound the variance of the loss/reward estimates by adding non-zero uniform probability γ of playing any action and (2) to add small bonus β to all the reward so that the estimates of the rewards are with a high probability an upper bound on the actual cumulative rewards. While authors in [Bub10] say that the anytime bounds can easily be derived based on the techniques presented in the paper, they provide only the bounds for the case of known number of steps of the game, such as the following.

Theorem 2.2.8. *For any $\delta \in (0,1)$, if Exp3.P is run with*

$$\beta = \sqrt{\frac{\ln(K\delta^{-1})}{nK}}, \eta = 0.95\sqrt{\frac{\ln K}{nK}}, \gamma = 1.05\sqrt{\frac{K \ln K}{n}}$$

then, with probability at least $1-\delta$,

$$R_n \leq 5.15\sqrt{nK \ln(K\delta^{-1})}$$

The presented regret bounds on Exp3 and Exp3.P are almost optimal, because the problem cannot be solved asymptotically better than $\sqrt{nK}$.

Regret matching in MAB problem

An alternative learning algorithm that allows minimizing regret in stochastic bandit setting is Regret Matching [HMC01b], later generalized as *polynomially weighted average forecaster* [CBL$^+$06]. Regret matching corresponds to selection of the parameter $p = 2$. It is a general procedure originally developed for playing known general-sum matrix games in [HMC00]. The algorithm computes, for each action in each step, the regret for not playing another fixed action every time the action has been played in the past. The action to be played in the next round is selected randomly with probability proportional to the positive portion of the regret for not playing the action. This procedure has been shown to converge arbitrarily close to the set of correlated equilibria in the general sum game. As a result, it converges to

```
Input: $K$ - number of actions; $\gamma_t$ - non-increasing sequence of real numbers
 1: $\forall i\ R_i \leftarrow 0$
 2: for $t \leftarrow 1, 2, \ldots$ do
 3:     $\forall i\ R_i^+ \leftarrow \max\{0, R_i\}$
 4:     if $\sum_{j=1}^K R_j^+ = 0$ then
 5:         $\forall i\ p_i \leftarrow 1/K$
 6:     else
 7:         $\forall i\ p_i \leftarrow (1-\gamma_t)\frac{R_i^+}{\sum_{j=1}^K R_j^+} + \frac{\gamma_t}{K}$
 8:     Use action $I_t$ from distribution $p$ and receive reward $r$
 9:     $\forall i\ R_i \leftarrow R_i - r$
10:     $R_{I_t} \leftarrow R_{I_t} + \frac{r}{p_{I_t}}$
```

Figure 2.6: Regret matching variant for regret minimization in adversarial bandit setting.

a Nash equilibrium in a zero-sum game. The regret matching procedure in [HMC00] requires the exact information about all utility values in the game, as well as the action selected by the opponent in each step. In [HMC01b], the authors modify the regret matching procedure and relax these requirements. Instead of computing the exact values for the regrets, the regrets are estimated in a similar way as the cumulative rewards in Exp3. As a result, the modified regret matching procedure is applicable in MAB.

We present the algorithm in Figure 2.6. The algorithm stores the estimates of the regrets for not playing action i in all time steps in the past in variables R_i. On lines 3-7, it computes the strategy for the current time step. If there is no positive regret for any action, a uniform strategy is used (line 5). Otherwise, the strategy is chosen proportionally to the positive part of the regrets (line 7). The uniform exploration with probability γ_t is added to the strategy as in the case of Exp3. It also ensures that the addition on line 10 to be bounded.

Cesa-Bianchi and Lugosi [CBL$^+$06] prove that regret matching eventually achieves zero regret in the adversarial MAB problem, but they provide the exact finite time bound only for the perfect-information case, where the agent learns rewards of all arms.

2.3 Learning in Matrix Games

The classical game theory assumes that all players involved in a game are rational, aware of the exact definition of the games, and know the pay-offs of all players in each outcome of the game. Using this, the players can reason to identify the equilibrium strategy for all players and play accordingly. The theory of learning in games studies an alternative approach to how equilibria in games may occur (e.g., [Fud98]). It has been shown, that many simple learning dynamics in a shared environment converge to Nash or Correlated equilibria, if the game is repeated many times. This holds even with a very limited knowledge of the players about the game. In this thesis, we use these learning principles to design sampling-based algorithms for approximating the optimal strategies in games.

The best known learning rule of this type is the fictitious play introduced by [Bro51]. A player using this rule assumes that the other players play a fixed probability distribution, estimates this distribution from past observations of opponents' actions and plays (pure) best response to this estimation in each iteration. If the empirical frequencies of playing individual actions when all players use this dynamics converge, they form the Nash equilibrium of the game. While this may not happen in some games, for zero-sum normal-form games, fictitious play always converges to the Nash equilibrium [Rob51].

Further research led to improvements of this dynamics to converge in more general classes of games. An important concept in this context is a criterion on regret minimization termed Hannan consistency in [HMC01a, CBL+06]. An algorithm A is *Hannan-consistent*, if in any matrix game, with any (not necessarily stationary) strategy of the opponent, the regret for not playing any single fixed action is almost surely non-positive in the limit. We use the following a little more general term also used for example in [JGGE01].

Definition 2.3.1 (Hannan-consistency)**.** *We say that a player' strategy in a repeated matrix game is* ϵ-Hannan-consistent *if, against any opponent strategy, its external regret is almost surely (i.e., with probability 1) smaller than* ϵ*.*

$$\limsup_{t\to\infty}, R_n \leq \epsilon.$$

An algorithm A is ϵ-Hannan-consistent, if a player who chooses his actions based on A is ϵ-Hannan-consistent. If $\epsilon = 0$, we say the strategy (player) is Hannan-consistent.

There is a clear connection between ϵ-Hannan consistency a ϵ-Nash equilibria. A well-known result (e.g., [CBL+06, Wau09]) says that if two players in a repeated zero-sum game play so that their regret for not playing the best fixed action is smaller than ϵ, the average of the strategies used by the player over all rounds form a 2ϵ-Nash equilibrium.

There are two important sub-classes of Hannan-consistent algorithms, based on the information they use about the matrix of the game. In the *known-game setup*, the algorithms use the exact pay-off matrix of the game. An example of such algorithm is the original version of regret matching from [HMC00]. The algorithm is basically the same as the version presented in Section 2.2.2, but instead of the estimates of regret R_i, it computes their exact values. If the agent played action I_t and the opponent played action j

$$\forall i \; R_i = R_i + u_i(i, j) - u_i(I_t, j) \tag{2.5}$$

In the *unknown-game setup*, the players do not know the pay-off matrix of the game and they are often also assumed not to know the number of actions of the opponent. The only information they have is the number of their actions and after each trial, the reward they have obtained. All of the algorithms that minimize regret in the adversarial multi-armed bandit problem introduced in Section 2.2.2 are ϵ-Hannan-consistent. As a result, using any of the algorithms for playing repeated

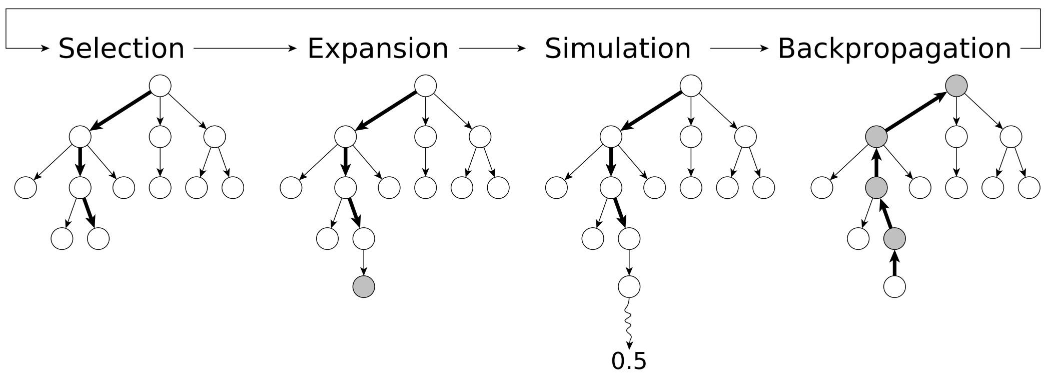


Figure 2.7: General scheme of Monte Carlo Tree Search. The picture is a variation on the well-known schema from [CBSS08].

matrix game leads to eventual convergence of the frequencies of action selection to an approximate Nash equilibrium in two-player zero-sum normal-form games. In Section 3.1.2, we extend this result to simultaneous-move extensive-form games.

2.4 Monte Carlo Tree Search in Perfect-Information Games

Monte Carlo Tree Search (MCTS) [Cou07, KS06] maintains a small part of a search tree close to the initial state. At the beginning, this tree consist of only the root node and then it gradually grows more in the areas that are promising. The standard iteration of a MCTS algorithm consists of four procedures demonstrated in Figure 2.7 that are repeatedly called in the following order:

- **Selection:** It starts in the root note and descends down the already constructed part of the tree based on the statistical information stored in the tree nodes.
- **Expansion:** After a leaf of the part of the tree maintained in memory is reached, it adds a small subtree rooted in the leaf to the tree. Usually, just a single node is added.
- **Simulation:** It chooses one of the leaves of the newly added subtree and runs a simulation of the play following that node using a randomized simulation strategy.
- **Backpropagation:** It returns back to the root of the tree and updates statistics in all visited nodes using the result of the simulation.

The main idea of the algorithm is to use the earlier iterations to create statistics that allow guiding the later iterations to the portions of the search space that are more relevant for the game. The output of the algorithm are the actions in the root node that lead to statistically most favorable results for the agent that runs the algorithm. One of the main advantages of this method in perfect-information games is that it does not require a heuristic evaluation function, as it is in classical minimax-like search. The simulation can be run until the end of the game, where the value of the terminal state can be determined exactly. However, if a good evaluation function is available, it can often improve the results [BLL+14].

MCTS can be instantiated by various *selection functions*, which decide what statistics are stored and how they are used for selecting actions in each iteration. The most common selection function is the Upper Confidence bound applied to trees (UCT)[KS06], which is a minor variation on UCB introduced in Section 2.2.1. Each node in the tree is treated as a multi-armed bandit problem. The action selected in MAB corresponds to a child in the search tree and the reward received in the MAB is the utility of the leaf reached by the simulation.

An extensive survey of MCTS methods with focus on game playing is presented in [BPW+12]. The first algorithm that proposed Monte Carlo tree evaluation with incremental tree building was presented in [Cou07] and evaluated on a small variant of Go. At the same time, another important paper connected Monte Carlo tree search and multi-armed bandit problem [KS06], which lead to first convergence guarantees for MCTS algorithm in perfect-information games. They named their algorithm the Upper Confidence bounds applied to Trees (UCT). This algorithm was immediately adapted by algorithms for computer Go [GWM+06] and along with many other improvements, it lead to an expert level performance [GS11]. The most notable improvement is RAVE [GS07], which combines the ideas from reinforcement learning with UCT and allows learning about multiple actions from a single simulation. The key idea is that in incremental games, such as Go, an action that is good after few moves are played is likely to be good right now as well.

While UCT performs well in many applications and is guaranteed to eventually converge to the optimal strategy, the time required for convergence can be very long in the worst case. Coquelin and Munos [CM+07] show that the cumulative regret of UCT in a tree of depth D can be as high as

$$\Omega(\overbrace{\exp(\exp(\dots\exp(1)\dots))}^{D-1 \text{ times}}) + O(log(n). \tag{2.6}$$

This inspired several novel approaches to Monte Carlo tree search algorithms, such as the once summarized in [Mun14] or presented in [FD13]. These approaches provide substantially better theoretical guarantees then UCT, but they are focused on single agent problems. They aim to minimize the simple regret for eventual decision made based on the sampling algorithms. This is certainly the correct approach for a single agent problem, but it is not clear how to adapt it to imperfect-information games. The approaches in this thesis rely on the properties of average strategies in minimizing cumulative regret, which have a natural connection to Nash equilibrium strategies as it is explained in Section 2.3.

2.5 Counterfactual Regret Minimization

Counterfactual Regret Minimization (CFR) is an adaptation of the regret matching algorithm to extensive-form games introduced in [ZJBP08]. Since then, it is very successfully applied in computer Poker bots [BHRZ13]. The main advantages of the algorithm are the minimal memory requirements, an easy parallelization and the ability to converge to good strategies in game of imperfect recall [WZJ$^+$09, LGBB12]. The algorithm minimizes a special form of regret that can be defined individually for each information set, given a strategy profile of the players.

Let σ be a strategy profile for an extensive-form game and $\sigma_i(I,a)$ be the probability that player i plays action a in information set I. For a history $h \in \mathcal{H}$, we denote $\pi^\sigma(h) = \Pi_{h'a \sqsubset h} \sigma_{P(h')}(I(h'),a)$ the probability of reaching h with strategy profile σ. We denote $\pi_i^\sigma(h) = \Pi_{h'a \sqsubset h, P(h')=i} \sigma_i(I(h'),a)$ the contribution of player i to this probability and therefore $\pi^\sigma(h) = \Pi_{i\in\mathcal{N}} \pi_i^\sigma(h)$. For $h \sqsubset z$, we also use $\pi^\sigma(h,z)$ and $\pi_i^\sigma(h,z)$ as the probability of reaching history z from history h. Let $Z_I = \{(h,z) | z \in \mathcal{Z}, h \in I, h \sqsubset z\}$ be the terminal histories passing though information set I, then the *counterfactual value* of information set I under a strategy profile σ is

$$v_i(I,\sigma) = \sum_{(h,z)\in Z_I} \pi_{-i}^\sigma(h)\pi^\sigma(h,z)u_i(z). \tag{2.7}$$

It is the expected value of player i when information set I is reached multiplied by the probability that the opponent plays to reach the information set. We denote $\sigma_{I\to a}$ a strategy profile identical to σ with the exception that in information set I, action a is played with probability 1.

Definition 2.5.1 (Counterfactual regret)**.** *The counterfactual regret for not playing action a in information set I under the strategy profile σ is*

$$r^\sigma(I,a) = v_{\mathcal{P}(I)}(I,\sigma_{I\to a}) - v_{\mathcal{P}(I)}(I,\sigma) \tag{2.8}$$

The CFR algorithm, similarly to regret matching in normal-form games, iteratively improves the strategies of the players in self-play. Initially, it starts with the uniform probability distribution for each information set σ^0. In each iteration, players play a strategy profile σ^t. In the basic form of the algorithm, the whole tree of the extensive-form game is traversed by a depth-first search in each iteration. The algorithm maintains for each action a and information set I the cumulative regret of not playing action a in all previous iterations: $R^T(I,a) = \sum_{t=1}^T r^{\sigma^t}(I,a)$. Let $R_{sum}^T(I) = \sum_{a\in A(I)} \max(0, R^T(I,a))$. The strategy for the next iteration of the algorithm is obtained based on the cumulative regret by regret matching

$$\sigma^{T+1}(I,a) = \begin{cases} \max(0, R^T(I,a))/R_{sum}^T(I) & \text{if } R_{sum}^T(I) > 0 \\ 1/|A(I)| & \text{otherwise.} \end{cases} \tag{2.9}$$

The algorithm also maintains the average strategy profile used over the iterations

$$\bar{\sigma}^T(I,a) = \frac{\sum_{t=1}^T \pi_{\mathcal{P}(I)}^{\sigma^t}(I)\sigma^t(I,a)}{\sum_{t=1}^T \pi_{\mathcal{P}(I)}^{\sigma^t}(I)}, \tag{2.10}$$

where $\pi_{\mathcal{P}(I)}^{\sigma^t}(I)$ is the contribution to player $\mathcal{P}(I)$ to the probability of reaching information set I with strategy profile σ^t.

Zinkevich et al. [ZJBP08] show that running this procedure will in each information set minimize the immediate counterfactual regret defined as

$$R^T_{i,imm}(I) = \frac{1}{T} \max_{a \in \mathcal{A}(I)} \sum_{t=1}^{T} \pi^{\sigma^t}_{-i}(I) r^{\sigma}(I,a). \tag{2.11}$$

It is the average regret of not playing the best fixed action in information set I over all iterations, weighted by the probability that the opponent will play in the way to allow player i to reach information set I. The rate at which this regret is minimized is

$$R^T_{i,imm}(I) \leq |v_{\max} - v_{\min}| \sqrt{\max_{h \in \mathcal{H}_i} |\mathcal{A}(h)|} / \sqrt{T}. \tag{2.12}$$

The key result from [ZJBP08] is that minimizing the immediate counterfactual regret in all in formation sets minimizes also the average overall regret. Therefore, the average strategy profile converges to an ϵ-Nash equilibrium of the extensive-form game.

Theorem 2.5.2 (CFR Convergence)**.** *Let R^T_i be the average overall regret of playing strategies based on the counterfactual regret minimization algorithm and not the best pure strategy, then*

$$R^T_i \leq \sum_{I \in \mathcal{I}_i} R^T_{i,imm}(I) \leq |v_{\max} - v_{\min}| |\mathcal{I}_i| \sqrt{\max_{h \in \mathcal{H}_i} |\mathcal{A}(h)|} / \sqrt{T}.$$

Recall from Section 2.3 that if both players have average regret for a sequence of strategies lower than ϵ the mean strategies computed from this sequence form a 2ϵ-Nash equilibrium of the game. This fact holds for strategies in the extensive-form games exactly as for strategies in normal-form games discussed in Section 2.3. Recall that there is an equivalent NFG to each EFG.

2.5.1 Monte Carlo Counterfactual Regret Minimization

One of the biggest drawbacks of CFR is the expensive traversal over the whole game tree required in each iteration. In many imperfect-information games, it is possible to create a good estimate of the quality of actions, and therefore of the regret caused by playing a strategy, by traversing only a small portion of the game tree. If the portions of the tree to be updated can be selected well based on background knowledge of the specific game, the convergence process can be substantially faster. Moreover, as we describe later, these fast estimates can then be used to lead the additional computational effort to more important parts of the game, analogically to Monte Carlo tree search.

The Monte Carlo Counterfactual Regret Minimization (MCCFR) algorithm introduced in [LWZB09] is a modification of CFR built on this idea. The set of terminal states of the game is partitioned to a set of *blocks* $\mathcal{Q} = \{Q_1, Q_2, \dots\}$. In each iteration, only the histories that are prefixes of the histories in a single block are traversed and only the regrets in the information sets containing these prefixes are updated. Lanctot et al. [LWZB09] introduce several sampling schemes, but in this thesis, we use only one of them – *outcome sampling*. In outcome sampling, each block corresponds to a single terminal history and only the information sets along a single branch from the game tree root to the leaf are updated. If history $z \in \mathcal{Z}$ is sampled with probability $q(z)$, the *sampled counterfactual value* is defined as

$$\tilde{v}_i(I,\sigma) = \begin{cases} \frac{1}{q(z)}\pi^\sigma_{-i}(h)\pi^\sigma(h,z)u_i(z) & \text{if } (h,z) \in Z_I \\ 0 & \text{otherwise.} \end{cases} \tag{2.13}$$

If each terminal history in $\mathcal{Z}$ is sampled with a non-zero probability, $\tilde{v}_i(I,\sigma)$ is an unbiased estimator of the real conterfactual value $v_i(I,\sigma)$, i.e., $\mathbb{E}\,\tilde{v}_i(I,\sigma) = v_i(I,\sigma)$. The reason is the same importance sampling trick as the one used in Exp3 (see Section 2.2.2). Based on the sampled counterfactual value, we can define the *sampled immediate counterfactual regret* as $\tilde{r}^\sigma(I,a) = \tilde{v}_{\mathcal{P}(I)}(I,\sigma_{I\to a}) - \tilde{v}_{\mathcal{P}(I)}(I,\sigma)$ and the corresponding sampled cumulative regret $\tilde{R}^T(I,a)$. The sampled regrets can be updated in each iteration and the strategies for the following iterations can be computed based on regret matching exactly as for the exact values.

The convergence rate of this algorithms was first proven in [LWZB09] and further improved in [Lan13]. In a little weaker form that does not require defining additional concepts, it is given by the following theorem.

Theorem 2.5.3 (Outcome sampling bound)**.** *For any $p \in (0,1]$, if $q(z) \geq \delta > 0$ or $\pi^\sigma_{-i}(z) = 0$ holds in every time step, the average overall regret suffered by outcome-sampling MCCFR is with probability* $(1-p)$

$$R^T_i \leq \left(1 + \frac{\sqrt{2}}{\sqrt{p}}\right)\left(\frac{1}{\delta}\right)\frac{|v_{\max} - v_{\min}||\mathcal{I}_i|\sqrt{|\mathcal{A}_i|}}{\sqrt{T}}.$$

CFR and MCCFR are algorithms designed for offline equilibrium computation and not as online game playing algorithms. The typical use is to run a parallel version of these algorithms on a large cluster for several weeks on an abstracted form of a game of interest, compute an equilibrium approximation and then use this strategy in form of a look-up table (e.g., [Joh07]). One of the contributions of this thesis is also adaptation of this algorithm to online search setting and evaluation of its performance in this setting. Therefore, pseudocode of two variants of this algorithm will be presented in Sections 3.1.3 and 3.2.4. For a very detailed description of CFR and MCCFR, including many useful implementation suggestions, see [Lan13].

Chapter 3

Descriptions of Algorithms

This chapter presents the existing and novel MCTS algorithms for imperfect-information games. The chapter is split to two sections. In the first, we address the extensive-form games with simultaneous moves, but otherwise perfect information. This section is based on our research presented in [LKLB13, LLW14, BLL$^+$14]. Even this class of games includes some of the challenges present in fully imperfect-information games, such as the need for randomized strategies. We start with the generic template of MCTS for SMGs, which is compatible even with the first related works on MCTS in SMGs [Fin07, TF11]. We further present the existing methods previously used as selection functions in SM-MCTS along with the novel regret matching variant we introduced in [LLW14]. Afterwards, we present the theoretical analysis, first presented in [LKLB13], showing that SM-MCTS can be easily modified to guarantee convergence to an approximate equilibrium with a large class of selection functions. The last algorithm for simultaneous-move games we present is an online version of MCCFR for SMGs, which we first introduced in [LLW14] and further developed and thoroughly evaluated in [BLL$^+$14].

In the second part of this chapter, we present algorithms for generic imperfect-information games. We first discuss the possible search space representations for search in EFGs and use them to categorize the existing work. Afterwards, we briefly explain MCTS on a single-agent information-set tree that generally cannot lead to optimal game-theoretic strategies, but which is often efficient in practice [CF10, LBP12, LBP14], as we show in Chapter 5. Then we focus on MCTS variants on the complete EFG representation of the game. We begin with the most popular IS-MCTS, which we improve in two directions. We propose novel, more efficient, selection functions we originally present in [Lis14] and we describe explicit modelling of belief distributions over the information set, we originally presented in [LPS$^+$12] and further developed in [LBP14]. Finally, we focus on a novel online variant of MCCFR for generic extensive-form games, we have originally introduced in [LLB14], but explain and evaluate in much more details in this thesis.

```
SM-MCTS($h$)
Input: $h$ – current state of the game
1: if $h \in \mathcal{Z}$ then return $u_1(h)$
2: if $h \in \mathcal{C}$ then
3:     Sample $h' \sim \Delta_c(h)$
4:     return SM-MCTS($h'$)
5: if $h \in T$ then
6:     $(a_1, a_2) \leftarrow \underline{Select}(h)$
7:     $h' \leftarrow \mathcal{T}(h, a_1, a_2)$
8:     $v_{h'} \leftarrow$ SM-MCTS($h'$)
9:     $\underline{Update}(h, a_1, a_2, v_{h'})$
10:    return $v_{h'}$
11: else
12:    $T \leftarrow T \cup \{h\}$
13:    $v_h \leftarrow$ Rollout($h$)
14:    return $v_h$
```

Figure 3.1: Simultaneous Move Monte Carlo Tree Search

3.1 Games with Simultaneous Moves and Perfect Information

This section describes the algorithms we have developed for games with simultaneous moves, but otherwise perfect information. We start with the more standard MCTS template with local selection functions and continue with an adaptation of MCCFR to the online search setting.

3.1.1 Simultaneous-Move Monte Carlo Tree Search

We first present a generic template of MCTS algorithms for simultaneous-move games (SM-MCTS) and then explain specific algorithms derived from this template. Figure 3.1 describes a single iteration of SM-MCTS. T represents the incrementally built MCTS tree, in which each state is represented by one node. Every node h maintains algorithm-specific statistics about the iterations that previously used this node. The template can be instantiated by specific implementations of the updates of the statistics on line 9 and the selection based on these statistics on line 6. In the terminal states, the algorithm returns the value of the state for the first player (line 1). In the chance nodes, the algorithm selects one of the possible next stated based on the chance distribution (line 3) and recursively calls the algorithm on this state (line 4). If the current state has a node in the current MCTS tree T, the statistics in the node are used to select an action for each player (line 6). These actions are executed (line 7) and the algorithm is called recursively on the resulting state (line 8). The result of this call is used to update the statistics maintained for state h (line 9). If the current state is not stored in tree T, it is added to the tree (line 12) and its value is estimated using the rollout policy (line 13). Finally the result of the Rollout is returned to higher levels of the tree.

This template can be turned into a working algorithm by selecting a specific selection and update functions. Different algorithms can be the bases for selection functions, but the most successful selection functions are based on the algorithms for multi-armed bandit problem introduced in Section 2.2. We now present the variants of SM-MCTS that were consistently the most successful in previous works, though more variants can be found in [PSPME12, LWWDT13, TLW14].

Decoupled Upper-confidence Bounds applied to Trees

The most common selection function for SM-MCTS is Decoupled Upper-confidence Bound applied to Trees (DUCT) first used in [Fin07]. For selection and updates, it executes the well-known UCT [KS06] algorithm independently for each of the players in each nodes. The statistics stored in search nodes are independently computed for each action of each player. For player $i \in \mathcal{N}$ and action $a_i \in \mathcal{A}_i(h)$ the reward sums x_{a_i} and the number of times the action was used n_{a_i} are maintained. When a joint action needs to be selected by the *Select* function, an action that maximizes the sum of the mean reward and the parameterized UCB bias term is selected for each player independently:

$$a_i = \operatorname*{argmax}_{a_i \in \mathcal{A}_i(h)} \left\{ \bar{x}_{a_i} + C_i \sqrt{\frac{\log n_h}{n_{a_i}}} \right\}, \text{ where } \bar{x}_{a_i} = \frac{x_{a_i}}{n_{a_i}} \text{ and } n_h = \sum_{b_i \in \mathcal{A}_i(h)} n_{b_i}. \quad (3.1)$$

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

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

$$\sigma_i(a_i) = \frac{n_{a_i}}{\sum_{b_i \in \mathcal{A}_i(h)} n_{b_i}}. \quad (3.2)$$

We call the former DUCT(max) and the latter DUCT(mix). Using the first method will certainly not make the algorithm converge to a Nash equilibrium, because the game may require a mixed strategy. The second has been shown not to converge to a Nash equilibrium as well; a counter-example in Rock, Paper, Scissors with biased payoffs is shown in [SSS09].

Note that UCT does not explicitly define which action is selected if multiple actions have identical value in the maximization. It is not relevant in sequential games, but it can have large impact in simultaneous-move games. Consider the following matrix game $\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$. It has only one NE, such that both players play the first action. However, DUCT selecting the first or the last action among the options with the same value will always get only the 0 rewards and the bias term will cause the players to round-robin over the diagonal indefinitely. This is clearly not optimal. DUCT(mix) would explicitly play each action with probability $\frac{1}{2}$ and

DUCT(max) would in half of the runs of the algorithm choose to play the first action and in the other half choose to play the second action, depending on whether it managed to perform even or odd number of iterations. In both variants, the player can improve by playing first action with probability 1. If we choose the action to play randomly among the once with the same probability, DUCT will quickly converge to the optimal solution in this game. Therefore, we use the randomized variant in our implementation.

Even with this modification, we later show that DUCT is not guaranteed to converge to the optimal solution. However, it is often very successful in practice, such as in general game playing [Fin12], or in the game of Tron [PSPME12]. Note that DUCT is different than UCT run on a game that is transformed to perfect information by serialization of the simultaneous moves [TLW14],because in the latter case one player has the advantage of knowing what the other player chose at every stage.

Exponential-weight algorithm for Exploration and Exploitation

The second most common instantiation of SM-MCTS is to use the Exponential-weight for Exploration and Exploitation (Exp3) [ACBFS03] independently for each of the players. In Exp3, each player maintains an estimate of the sum of rewards for each action, denoted G_{a_i}. The joint action produced by *Select* is composed of an action independently selected for each player. The probability of using action a_i is proportional to the exponential of the reward estimates. As we explain in Section 2.2.2, we use the following numerically more stable formula:

$$\sigma_i^t(a_i) = \frac{(1-\gamma)}{\sum_{b_i \in A_i(h)} \exp(\eta(G_{b_i} - G_{a_i}))} + \frac{\gamma}{|\mathcal{A}_i(h)|}, \text{ where } \eta = \frac{\gamma}{|\mathcal{A}_i(h)|}. \tag{3.3}$$

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

$$u_1 \leftarrow \frac{(v_1 - v_{min})}{v_{max} - v_{min}}; \;\; u_2 \leftarrow (1 - u_1) \tag{3.4}$$

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

$$G_{a_i} \leftarrow G_{a_i} + \frac{u_i}{\sigma_i^t(a_i)}. \tag{3.5}$$

Recall that dividing the value by the probability of selecting the corresponding action makes G_{a_i} estimate the sum of rewards over all iterations, not only the once where a_i was selected.

As the final strategy, after all iterations are executed, the algorithm computes the *average strategy* of the Exp3 algorithm over all iterations for each player. After T iterations in the particular node, it is

$$\bar{\sigma}_i^T(a_i) = \frac{1}{T} \sum_{t=1}^{T} \sigma_i^t(a_i). \tag{3.6}$$

In our implementation, we maintain the cumulative sum and normalize it to obtain the average strategy after the iterations. In the previous work (e.g., [ACBFS03, TF11]), empirical frequencies of using individual action are used instead of mean strategy. However, we have shown these two methods to be equivalent in [LKLB13] in limit and in the short time, average strategy produces slightly more stable strategies. Moreover, this approach is more consistent with the methods based on regret matching [ZJBP08].

Previous work [TF11] also suggests first removing the samples caused by the exploration. This modification proved to be useful also in our experiments, so as the resulting final mixed strategy, we often use

$$\bar{\sigma}_i(a_i) \leftarrow \max\left(0, \bar{\sigma}_i(a_i) - \frac{\gamma}{|\mathcal{A}_i(h)|}\right), \tag{3.7}$$

normalized to sum to one. Note that this cannot change the expected payoff of the produced strategy by more than γ.

Decoupled Regret Matching

The decoupled regret matching (DRM) uses in each state an independent regret matching algorithm described in the context of adversarial multi-armed bandit problem in Section 2.2.2. We use it here and in [LKLB13] mainly as an alternative example of a SM-MCTS with decoupled Hannan-consistent selection strategies. The algorithm stores the cumulative regret estimates (R_{a_i}) for each action of each player. The *Select* function chooses for each player an action based on regret matching and uniform exploration with probability γ:

$$\sigma_i^t(a_i) = \begin{cases} \frac{\max(0,R_{a_i})}{R_{sum}} + \frac{\gamma}{|\mathcal{A}_i(h)|} & \text{if } R_{sum} > 0 \\ \frac{1}{|\mathcal{A}_i(h)|} & \text{otherwise.} \end{cases} \tag{3.8}$$

where $R_{sum} = \sum_{b_i \in \mathcal{A}_i(h)} \max(0, R_{b_i})$. *Update* adds the regrets to the estimates using importance sampling. After joint action (a_1, a_2) is selected and utility v_1 is returned from the recursive call of SM-MCTS, assuming $v_2 = -v_1$, the regrets are updated as follows.

$$\forall i \forall b_i \in \mathcal{A}_i(h) \;\; R_{b_i} = \begin{cases} R_{b_i} - v_i & \text{if } b_i \neq a_i \\ R_{b_i} + v_i/\sigma_i(b_i) & \text{otherwise.} \end{cases} \tag{3.9}$$

The final strategy suggested by this method is again the mean strategy over all iterations, as shown in equation 3.6.

Coupled Regret Matching

This variant of the selection function applies the origiginal regret matching (RM) for known matrix games setting [HMC00] to the current estimated matrix game at each stage. The statistics stored by this algorithm in each node are the use count of each joint action ($n_{a_1a_2}$) and the sum of rewards for each joint action ($X_{a_1a_2}$). This algorithm also maintains for each player i a cumulative regret R_{a_i} for having played σ_i^t instead of $a_i \in \mathcal{A}_i(h)$ on iteration t, initially set to 0. The regret values R_{a_i} are maintained separately by each player, as in the decoupled algorithms. However, the updates use values that are a function of the joint actions.

On iteration t, function *Select* builds each player's current strategies exactly the same way as presented in equation 3.8. *Update* adds regret accumulated at the iteration to the regret tables R^i differently. Suppose joint action (a_1, a_2) is selected by the selection policy and utility v_1 is returned from the recursive call of SM-MCTS. Label $reward(b_1, b_2) = \frac{X_{b_1 b_2}}{n_{b_1 b_2}}$ if $(b_1, b_2) \neq (a_1, a_2)$, or v_1 otherwise. Assuming $v_2 = -v_1$, the updates to the regrets are:

$$\forall b_1 \in \mathcal{A}_1(h), \quad R_{b_1} \leftarrow R_{b_1} + (reward(b_1, a_2) - v_1), \tag{3.10}$$

$$\forall b_2 \in \mathcal{A}_2(h), \quad R_{b_2} \leftarrow R_{b_2} - (reward(a_1, b_2) - v_1). \tag{3.11}$$

After all simulations, the strategy to play in state h is defined by the mean strategy used in the corresponding node, as shown in equation 3.6.

3.1.2 Convergence of SM-MCTS to Nash Equilibrium

In spite of the success of MCTS first in perfect-information games and later also in imperfect information, there is a lack of theory analyzing MCTS outside two-player perfect-information sequential games. To the best of our knowledge, no convergence guarantees are known for MCTS in games with simultaneous moves or general EFGs. Therefore, one of the contributions of this thesis is the following analysis of sufficient conditions for convergence of SM-MCTS to ϵ-Nash equilibrium of the game, first presented in [LKLB13]. The detailed proofs are included in the appendix of this paper.

In the following, we work with a slightly modified SM-MCTS algorithm. It does not return the current sample value as in Figure 3.1, but computes means of the values returned on line 8 over all iterations and returns these means on line 10. We were not able to formally prove convergence of the algorithm without this modification. We further call this variant SM-MCTS-A. We show that it converges to $C\epsilon$-Nash equilibrium with any ϵ-Hannan-consistent algorithm, which assures attempting each action infinitely many times, used in the decoupled way (i.e., independently for each player) as the selection function.

We focus on the eventual convergence to approximate NE, which allows us to make an important simplification. We disregard the incremental building of the tree and assume we have built the complete tree. We show that this will eventually happen with probability 1 and that the statistics collected during the tree building phase cannot prevent the eventual convergence.

The main idea of the proof is to show that the algorithm will eventually converge close to the optimal strategy in the leaf nodes and inductively prove that it will converge also in higher levels of the tree. In order to do that, after introducing the necessary notation, we start by analyzing the situation in simple matrix games, which corresponds mainly to the leaf nodes of the tree. In the inner nodes of the tree, the observed payoffs are imprecise because of the stochastic nature of the selection functions and the bias caused by exploration, but the error can be bounded. Hence, we continue with analysis of repeated matrix games with bounded error.

Finally, we compose the matrices with bounded errors in a multi-stage game to prove convergence guarantees of SM-MCTS-A.

Notation and definitions

This proof sketch uses a slightly different notation than the rest of the thesis to keep it consistent with the full proofs of the presented theorems in appendix of [LKLB13] and other related literature, e.g., [ACBFS03]. Consider a repeatedly played matrix game (a_{ij}) where at time s players 1 and 2 choose actions i_s and j_s respectively. We will use the convention $(|\mathcal{A}_1|, |\mathcal{A}_2|) = (m, n)$. Define

$$G(t) = \sum_{s=1}^{t} a_{i_s j_s}, \quad g(t) = \frac{1}{t}G(t), \quad \text{and} \quad G_{max}(t) = \max_{i \in \mathcal{A}_1} \sum_{s=1}^{t} a_{i j_s},$$

where $G(t)$ is the *cumulative payoff*, $g(t)$ is the *average payoff*, and G_{max} is the *maximum cumulative payoff* over all actions, each to player 1 and at time t. We also denote $g_{max}(t) = G_{max}(t)/t$ and by $R(t) = G_{max}(t) - G(t)$ and $r(t) = g_{max}(t) - g(t)$ the *cumulative and average regrets*. For actions i of player 1 and j of player 2, we denote t_i, t_j the number of times these actions were chosen up to the time t and t_{ij} the number of times both of these actions has been chosen at once. By *empirical frequencies* we mean the strategy profile $(\hat{\sigma}_1(t), \hat{\sigma}_2(t)) \in \langle 0,1\rangle^m \times \langle 0,1\rangle^n$ given by the formulas $\hat{\sigma}_1(t,i) = t_i/t$, $\hat{\sigma}_2(t,j) = t_j/t$. By *average strategies*, we mean the strategy profile $(\bar{\sigma}_1(t), \bar{\sigma}_2(t))$ given by the formulas $\bar{\sigma}_1(t,i) = \sum_{s=1}^{t} \sigma_1^s(i)/t$, $\bar{\sigma}_2(t,j) = \sum_{s=1}^{t} \sigma_2^s(j)/t$, where σ_1^s, σ_2^s are the strategies used at time s. Remember that a player is *ϵ-Hannan-consistent* if, for any payoff sequences (e.g., against any opponent strategy), $\limsup_{t\to\infty}, r(t) \le \epsilon$ holds almost surely. An algorithm A is ϵ-Hannan consistent, if a player who chooses his actions based on A is ϵ-Hannan consistent.

Hannan consistency (HC) is a commonly studied property in the context of online learning in repeated (single stage) decisions (e.g., [CBL+06]). In particular, RM and Exp3 algorithms presented in Section 2.2.2 are ϵ-Hannan consistent. In order to ensure that the SM-MCTS-A algorithm will eventually visit each node infinitely many times, we need the selection function to satisfy the following property.

Definition 3.1.1. *We say that A is an* algorithm with guaranteed exploration, *if for players 1 and 2 both using A for action selection $\lim_{t\to\infty} t_{ij} = \infty$ holds almost surely $\forall (i,j) \in \mathcal{A}_1 \times \mathcal{A}_2$.*

Note that many HC algorithms, such as RM and Exp3, guarantee exploration without any modification. If there is an algorithm without this property, it can be adjusted the following way.

Definition 3.1.2. *Let A be an algorithm used for choosing action in a matrix game M. For fixed exploration parameter $\gamma \in (0,1)$ we define a modified algorithm A^* as follows: In each time, with probability $(1-\gamma)$ run one iteration of A and with probability γ choose the action randomly uniformly over available actions, without updating any of the variables belonging to A.*

Repeated matrix games

First we show that the ϵ-Hannan consistency is not lost due to the additional exploration.

Lemma 3.1.3. *Let A be an ϵ-Hannan consistent algorithm. Then A^* is an $(\epsilon+\gamma)$-Hannan consistent algorithm with guaranteed exploration.*

In previous works on learning in games, RM variants generally suggested using the average strategy and Exp3 or UCT variants the empirical frequencies to obtain the strategy to be played. The following lemma says there eventually is no difference between the two.

Lemma 3.1.4. *As t approaches infinity, the empirical frequencies and average strategies are almost surely equal. That is, $\limsup_{t\to\infty} \max_{i\in\mathcal{A}_1} |\hat{\sigma}_1(t,i) - \bar{\sigma}_1(t,i)| = 0$ holds with probability 1.*

The proof is a consequence of the martingale version of Central Limit Theorem. It is well-known that two Hannan consistent players will eventually converge to NE (see [CBL+06] and [BM07]). We prove a similar result for the approximate versions of the notions.

Lemma 3.1.5. *Let $\epsilon > 0$ be a real number. If both players in a matrix game with value v are ϵ-Hannan consistent, then the following inequalities hold for the empirical frequencies almost surely:*

$$\limsup_{t\to\infty} u\left(br, \hat{\sigma}_2(t)\right) \le v + 2\epsilon \quad \textit{and} \quad \liminf_{t\to\infty} u\left(\hat{\sigma}_1(t), br\right) \ge v - 2\epsilon. \tag{3.12}$$

The proof shows that if the value caused by the empirical frequencies was outside of the interval infinitely many times with positive probability, it would be in contradiction with definition of ϵ-HC. The following corollary is than a direct consequence of this lemma.

Corollary 3.1.6. *If both players in a matrix game are ϵ-Hannan consistent, then there almost surely exists $t_0 \in \mathbb{N}$, such that for every $t \ge t_0$ the empirical frequencies and average strategies form $(4\epsilon + \delta)$-Nash equilibrium for arbitrarily small $\delta > 0$.*

The constant 4 is caused by going from a pair of strategies with best responses within 2ϵ of the game value guaranteed by Lemma 3.1.5 to the approximate NE, which multiplies the distance by two.

Repeated matrix games with bounded error

After defining the repeated games with error, we present a variant of Lemma 3.1.5 for these games.

Definition 3.1.7. *We define $M(t) = (a_{ij}(t))$ to be a game, in which if players chose actions i and j, they receive randomized payoffs $a_{ij}\left(t, (i_1, ...i_{t-1}), (j_1, ...j_{t-1})\right)$. We will denote these simply as $a_{ij}(t)$, but in fact they are random variables with values in $\langle 0, 1\rangle$ and their distribution in time t depends on the previous choices of actions.*

We say that $M(t) = (a_{ij}(t))$ is a repeated game with error η*, if there is a matrix game $M = (\tilde{a}_{ij})$ and almost surely exists $t_0 \in \mathbb{N}$, such that $|a_{ij}(t) - \tilde{a}_{ij}| < \eta$ holds for all $t \geq t_0$.*

Symbols v and $u(\cdot,\cdot)$ will still be used with respect to M without errors. The following lemma states that even when receiving rewards with errors, ϵ-HC algorithms still converge to an approximate NE of the game of the exact unperturbed games.

Lemma 3.1.8. *Let $\epsilon > 0$ and $c \geq 0$. If $M(t)$ is a repeated game with error $c\epsilon$ and both players are ϵ-Hannan consistent then the following inequalities hold almost surely:*

$$\limsup_{t\to\infty} u(br, \hat{\sigma}_2) \leq v + 2(c+1)\epsilon, \quad \liminf_{t\to\infty} u(\hat{\sigma}_1, br) \geq v - 2(c+1)\epsilon \tag{3.13}$$

$$\text{and} \quad v - (c+1)\epsilon \leq \liminf_{t\to\infty} g(t) \leq \limsup_{t\to\infty} g(t) \leq v + (c+1)\epsilon. \tag{3.14}$$

The proof is similar to the proof of Lemma 3.1.5. It needs an additional claim that if the algorithm is ϵ-HC with respect to the observed values with errors, it still has a bounded regret with respect to the exact values. In the same way as in the previous subsection, a direct consequence of the lemma is the convergence to an approximate Nash equilibrium.

Theorem 3.1.9. *Let $\epsilon, c > 0$ be real numbers. If $M(t)$ is a repeated game with error $c\epsilon$ and both players are ϵ-Hannan consistent, then for any $\delta > 0$ there almost surely exists $t_0 \in \mathbb{N}$, such that for all $t \geq t_0$ the empirical frequencies form $(4(c+1)\epsilon + \delta)$-equilibrium of game M.*

Extensive-form games with simultaneous moves

Now we have all the necessary components to prove the main theorem.

Theorem 3.1.10. *Let $\left(M^h\right)_{h\in H}$ be a game with perfect information and simultaneous moves with maximal depth D. Then for every ϵ-Hannan consistent algorithm A with guaranteed exploration and arbitrary small $\delta > 0$, there almost surely exists t_0, so that the average strategies $(\hat{\sigma}_1(t), \hat{\sigma}_2(t))$ form a subgame perfect*

$$\left(2D^2 + \delta\right)\epsilon\text{-Nash equilibrium for all } t \geq t_0.$$

Once we have established the convergence of the ϵ-HC algorithms in games with errors, we can proceed by induction. The games in the leaf nodes are simple matrix game so they will eventually converge and they will return the mean reward values in a bounded distance from the actual value of the game (Lemma 3.1.8 with $c = 0$). As a result, in the level just above the leaf nodes, the ϵ-HC algorithms are playing a matrix game with a bounded error and by Lemma 3.1.8, they will also eventually return the mean values within a bounded interval. On level d from the leaf nodes, the errors of returned values will be in the order of $d\epsilon$ and players can gain $2d\epsilon$ by deviating. Summing the possible gain of deviations on each level leads to the bound in the theorem. The subgame perfection of the equilibrium results from the fact that for proving the bound on approximation in the whole game (i.e., in the root of the game tree), a smaller bound on approximation of the equilibrium is proven for all subgames in the induction.

This result tells us that if we use an exact HC algorithm as a selection function, we will converge arbitrarily close to the exact Nash equilibrium in this class of games. However, the situation is less promising with the commonly used ϵ-HC algorithms with constant exploration factor [TLW14, LLW14]. With exploration $\gamma = 0.1$, a game can be constructed so that the algorithm is no less than 0.1-HC. The bound then does not give us practically any guarantees already for a game of depth 7. However, we assume the bound is not tight. We were able to construct example games with equilibrium error in the order of $2D\epsilon$, but not $D^2\epsilon$. Therefore, we are currently working on a tighter bound. On the other hand, from the practical perspective and with a very short time, constant exploration often performs better than lowering the exploration parameter over time. This result only tells us, that we should not expect convergence close to equilibrium using this method. It is not always correlated with the game playing performance, as we show in the experiments.

3.1.3 Online Counterfactual Regret Minimization in Simultaneous Move Games

As we explained before, the simultaneous-move games we analyze in this thesis are a special case of generic extensive-form games. For generic EFGs, we introduced MCCFR in Section 2.5.1. It is a sampling-based algorithm similar to MCTS, which is guaranteed to converge to Nash equilibrium in EFGs. This algorithm was originally designed and used for offline strategy computation with weeks of computation time spent before the game is actually played. In this section, we adapt it to an online search algorithm, while keeping the guarantee of convergence to a Nash equilibrium. We have first introduced this algorithm a joint work with Marc Lanctot and Mark Winands in [LLW14].

The main idea of the algorithm is to combine MCCFR with outcome sampling strategy, the incremental building of the game tree, and random simulations used in MCTS. In short time, MCTS would not be able to converge to reasonable strategies deeper in the tree anyway, hence in online game playing setting, it is pointless to allocate the memory for the whole tree. Good results may be achievable already with keeping the statistics close to the root of the tree, where the decision about the next move needs to be performed.

In the root state of the game, the algorithm runs as in the offline case, with the exception of incremental tree building, which may slow-down the convergence, but saves substantial amount of memory. The complications could occur later in the game, after some actions were performed and we want to use additional thinking time to refine the computed strategy. Running the additional samples from the root of the game as suggested by the offline approach would likely have practically no effect on the quality of the strategy in the specific history the game has reached. Only a small fraction of the samples would actually reach the relevant history in the game. The solution of this problem for simultaneous-move games presented in this section is quite straightforward, but it is substantially more complicated in generic extensive-form games discussed later.

Simultaneous Move Online Outcome Sampling

In simultaneous-move games, the algorithm can be run directly from the current information set and not the root state, because the subtree below the (singleton) information set forms a subgame. When started from a history h, the algorithm runs iterations from this history to the end of the game until it uses the given time limit. Each iteration is performed by the function presented in Figure 3.2, which recursively descends down the tree. In the root of the game, the function is run as SM-OOS$(root, i, 1, 1, 1)$, alternating player $i \in \{1, 2\}$ in each iteration. If the function reaches a terminal history of the game (line 1), it returns the utility of the terminal node for player i, the sampling probability of reaching this node s and 1 for the tail probability. If it reaches a chance node, is recursively continues after a randomly selected action of chance (lines 2-4). If none of the first two conditions holds, the algorithm reaches a state where players make decisions. If this state is already included in the incrementally built tree (line 5), the following state is selected based on the statistics in the tree by regret matching, with additional γ-exploration for player i (line 7). The recursive call on line 11 then updates the player's reach probabilities directly with the strategy from regret matching and the sampling probability with the modified strategy σ'_i for player i. If the reached node is not in the tree (line 12), it a node for the current that is added to the tree (line 13). An action for each player is selected based on the uniform distribution (lines 14-17) and random rollout of the game until a terminal node is initiated on line 18. The rollout is similar to the MCTS case, but in addition, it has to compute the probabilities x and q required by the definition of the sampled counterfactual value. Whether the current node was in the tree or not, the algorithm updates the regret table of player i based on the definition of sampled counterfactual regret (lines 19-23) and the mean strategy of player $-i$ (line 24). Finally, the function returns the correct probabilities to the upper level of the tree.

Progressing in the game

We disregard the chance transitions in the following discussion for better clarity, but they can be included without any complications. When SM-OOS is used for game playing, it is run in the root of the game until the time limit is reached, the action a_i is selected based on the (normalized) mean strategy $\bar{\sigma}_i$ in the root node ($\emptyset$) and based on the action of the opponent (a_{-i}), the game progresses to some $h = \mathcal{T}(\emptyset, a_i, a_{-i})$. In MCTS, we would simply make h the new root of the game and continue sampling. Importantly, all other successors of the root and whole their subtrees can be removed from memory to save computational resources. However, CFR works with reach probabilities computed from the root of the game. In order to start the following samples from node h, we should reflect the reach probabilities in the initial calls of SM-OOS. In node h, we would start the iterations with SM-OOS$(h, \bar{\sigma}_i(\emptyset, a_i), \bar{\sigma}_i(\emptyset, a_i), 1)$. After next game progression, the reach probabilities should be further updated to correctly reflect π_i and π_{-i} in the specific node in the

```
SM-OOS($h, i, \pi_i, \pi_{-i}, s$)
Input: $h$ – current state of the game; $i$ – the regret updating player; $\pi_i$ – player
    $i$'s contribution to the probability of reaching $h$; $\pi_{-i}$ – player $-i$'s and chance's
    contribution to probability of reaching $h$; $s$ – the probability of sampling the
    current history
Output: $(x, q, u)$: $x$ – probability of reaching the sampled leaf from current state
    based on player's strategies ($\pi^\sigma(h, z)$); $q$ – probability of sampling the leaf from
    the root ($q(z)$); $u_i$ – utility of the leaf for player $i$
 1: if $h \in \mathcal{Z}$ then return $(1, s, u_i(h))$
 2: if $h \in \mathcal{C}$ then
 3:     Sample chance outcome $h'$ from $\Delta_c(h)$
 4:     return SM-OOS($h', i, \pi_i, \pi_{-i}\Delta_c(h, h'), s\Delta_c(h, h')$)
 5: if $h \in T$ then
 6:     $\sigma_i \leftarrow$ RegretMatching($R_i(h)$)
 7:     $\forall a \in \mathcal{A}_i(h)\ \sigma'_i(a) \leftarrow (1-\gamma)\sigma_i(a) + \frac{\gamma}{|\mathcal{A}_i(h)|}$
 8:     Sample action $a_i$ from $\sigma'_i$
 9:     $\sigma_{-i} \leftarrow$ RegretMatching($R_{-i}(h)$)
10:     Sample action $a_{-i}$ from $\sigma_{-i}$
11:     $(x, q, u_i) \leftarrow$SM-OOS($\mathcal{T}(h, a_i, a_{-i}), \sigma_i(a_i)\pi_i, \sigma_{-i}(a_{-i})\pi_{-i}, \sigma'_i(a_i)\sigma_{-i}(a_{-i})s$)
12: else
13:     $T \leftarrow T \cup \{h\}$
14:     $\forall a \in \mathcal{A}_i(h)\ \sigma_i(a) \leftarrow \frac{1}{|\mathcal{A}_i(h)|}$
15:     Sample action $a_i$ from $\sigma_i$
16:     $\forall a \in \mathcal{A}_{-i}(h)\ \sigma_{-i}(a) \leftarrow \frac{1}{|\mathcal{A}_{-i}(h)|}$
17:     Sample action $a_{-i}$ from $\sigma_{-i}$
18:     $(x, q, u_i) \leftarrow$ OOS-Rollout($\mathcal{T}(h, a_i, a_{-i}), s\frac{1}{|\mathcal{A}_i(h)|}\frac{1}{|\mathcal{A}_{-i}(h)|}$)
19: $c \leftarrow x \cdot \sigma_{-i}(a_{-i})$
20: $x \leftarrow c \cdot \sigma_i(a_i)$
21: $W \leftarrow u_i\pi_{-i}\ /\ q$
22: $R_i(h, a_i) \leftarrow R_i(h, a_i) + (c - x)W$
23: $\forall a \in \mathcal{A}_i(h) \setminus \{a_i\}\ \ R_i(h, a) \leftarrow R_i(h, a) - xW$
24: $\bar{\sigma}_{-i}(h) \leftarrow \bar{\sigma}_{-i}(h) + \frac{1}{s}\pi_{-i}\sigma_{-i}$
25: return $(x, q, u_i)$
```

Figure 3.2: Online Outcome Sampling for simultanoeus-move games.

tree. The sampling probability can stay 1, as from now on, all the samples will go through the current history.

An alternative solution is to simply consider the current history h to be a new root of the game. As a result, values π_i and π_{-i} in a specific node will be substantially larger in the iterations run from h compared to the initial iterations run from $\emptyset$. As a result, disregarding the higher q which is in both cases compensated by the frequency of visits, W would be larger and the regret updates will be stronger as before. This should not be a problem, since the later iterations are generally more correct due to a larger part of the tree already stored in the memory and better approximations of the optimal strategies later in the tree. We use the approach with setting the new game root in simultaneous-move games.

Convergence guarantees

Both approaches to progress in the game discussed above are, given enough time, guaranteed to converge to the Nash equilibrium of the current subgame. The algorithm run in the subgame eventually builds the complete game tree and then performs the same regret updates as the offline Outcome Sampling MCCFR [LWZB09]. If there are some wrong regret values accumulated during tree building or in the previous stages of the game, these have bounded size. Eventually, their contribution will be negligible compared to the correct MCCFR samples. Therefore, we can state the following theorem.

Theorem 3.1.11. *Let $\bar{\sigma}^t(h)$ be a strategy produced by SM-OOS in a zero-sum extensive-form game with simultaneous moves started from match history h run for t iterations, and with exploration $\gamma > 0$. For any $p \in (0, 1\rangle, \varepsilon > 0$ there exists $t < \infty$ such that with probability $1 - p$ the strategy $\bar{\sigma}^t(h)$ is a ε-Nash equilibrium.*

3.2 Generic Imperfect-Information Games

This section introduces various MCTS algorithms for generic imperfect-information games. We first discuss possible state space representations and the existing algorithms that use them. Afterwards, we present the existing MCTS algorithms, our improvements of these algorithms and a variant of Online Outcome Sampling for generic imperfect-information games.

3.2.1 Search Space of Imperfect-Information Games

As we discuss in Section 2.1.2, an extensive-form game (EFG) can be visualized as a game tree with information sets denoted as subsets of game nodes. A simple example of such game is on top of Figure 3.3. However, until recently, search algorithms for imperfect-information games rarely used this explicit EFG representation as the search space. The reason is possibly that before Monte Carlo tree search algorithms were introduced, the overhead of selecting the best consistent decision over the whole information set was too large and in many domains, unnecessary.

Collection of perfect-information games

One of the search methods popularized after allowing for an expert level computer Bridge player [Gin01] was Perfect Information Monte Carlo sampling (PIMC), originally proposed in [Lev89]. The main idea is to model the current information set of the player during the match, but ignore the information structure of the game in thinking about the future moves. The algorithm randomly selects states from the current information set and runs a perfect-information search algorithm from each of them. The action selected for a player is the one with the best average performance over these perfect-information searches. This approach also performs well, for example, in Scrabble [She02], Hearts [Stu08], Skat [BLFS09], and Phantom Go [BSCU07].

This method is clearly not guaranteed to converge to the optimal solutions in general. A systematic analysis of this fact is provided in [FB98], where the authors show two specific situations that cause this approach to find sub-optimal strategies. (1) *Strategy fusion*, caused by allowing the player to select a different action in the same information set, when it is reached in perfect-information searches from different states in the current information set. (2) *Non-locality*, which is says that the optimal strategy in one information set can depend on the optimal strategy of the opponent in information sets in distant parts of the tree. One specific example is discussed later in Section 3.2.3.

The discrepancy between the good empirical performance in some games and the clear theoretical flaws of this approach was resolved in [LSBF10], where authors empirically analyze a class of artificially constructed games along with few real world games to show the measurable properties of games that predict the ability of PIMC to achieve a good performance in the games.

In this thesis, we focus on algorithms that can possibly be modified to guarantee convergence close to NE in any imperfect-information game. Therefore, we do not evaluate methods based on PIMC.

Information set tree

An alternative search space representation used by search algorithms for imperfect-information games, such as the once presented in [RNK$^+$10, CF10, CPW12, LBP12], is called information-set tree (IST). It represents the decisions of only one of the players and models the effect of the other player's actions using a heuristic opponent model. The information set trees corresponding to the example EFG for both players are presented in the bottom part of Figure 3.3. There are two different kinds of nodes in the information-set tree – *decision nodes* and *observation nodes.* Nodes in a single ply of the tree are of the same kind and the plies of decision and observation nodes are regularly alternating. The decision nodes represent whole information sets, in which the searching player is making a decision. This make search on this data structure avoid strategy fusion, but also unable to deal with non-locality. The children of the decision node are observation nodes, one for each action that is applicable for the player in the information set. In turn-taking games, each action of the searching player in combination with an action of the opponent in the next turn produces a set of possible observations. Each observation corresponds to some subset

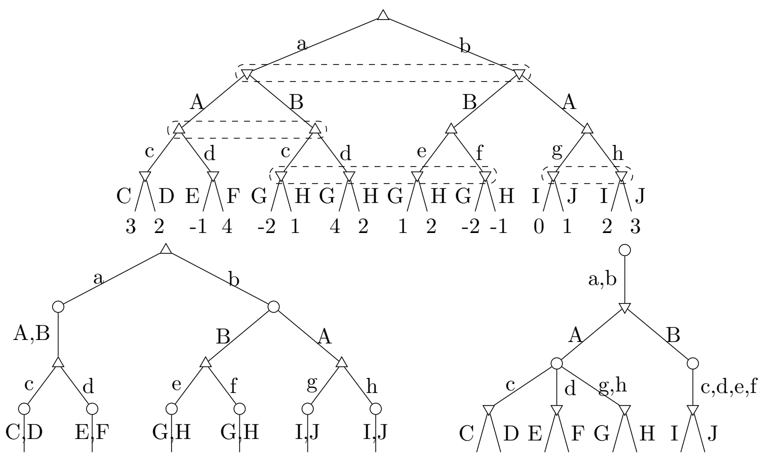


Figure 3.3: Full extensive-form game between △ and ▽, and the corresponding information-set trees for both players, in which the observation nodes are denoted by ○.

of actions (or action sequences) the opponent and chance could have performed between the current and the next information set of the player. The observation node has one child for each such observation. Its children are again the decision nodes corresponding to information sets in the game.

This kind of search space has been used in Kriegspiel [CF10], where the observations correspond to messages about legality of moves and capturing of pieces from the referee. In pursuit-evasion games [RNK$^+$10, LBP12], the observations correspond exactly to the opponent's moves when there is a direct visibility contact between the pursuers and the evader. If the pursuing agents lose the evader from sight, the possible observations resulting from a joint action can either be that the pursuer discovers the evader at some place in the environment, or that the evader will stay hidden.

The leaves in the information-set tree represent sets of terminal states of the game. It is not clear how to aggregate the utility values in the leaves of EFG to the leaves of IST. However, the applications using this search space representation generally use only depth-limited search with heuristic evaluation function. In that case, the mapping between the leaf nodes of EFG and IST does not have to be created explicitly. It is part of the domain-specific expert knowledge.

The problem with mapping between the leaves in the information set tree and the complete EFG can also be solved by maintaining an IST for each player in the game. Any compatible combination of terminal nodes in these trees uniquely determines a leaf node of EFG. We further discuss this approach in Section 3.2.3

3.2.2 Search from Single Player's Perspective

Approximating the optimal strategy for both players is a computationally expensive task. With a very limited time frame, it may be beneficial to rather approximate the best response to some fixed heuristic behavior of the opponent, which is a substantially simpler problem. Moreover, in many realistic situations, the player will play against a human opponent or computationally bounded agent, which means that a heuristic model of the opponent's behavior may even be more precise than the assumption of rationality used by equilibrium solutions.

Computing an optimal strategy with a fixed opponent model is essentially computing the best response to the model of the opponent's strategy. A systematic (not sampling) algorithm computing a strategy based on a fixed opponent model was introduced, e.g., in [PNS06], we present several efficient pruning techniques for this algorithms in [BKLP14], and much more efficient implementations are possible for specific domains [JWBM11]. Computing the pure best response from [PNS06] was later extended by the same authors to computing a uniformly mixed strategy over the actions that belong to the best response [PNS10].

A more interesting question addressed by [PNS06, PNS10] is what heuristic opponent model to use, if we do not have enough historic data to learn from. They define two basic opponent models for their depth-first search algorithm. The *paranoid* model chooses in opponent's information sets an action that minimizes the searching player's utility. Note that this uses the information of the player, not the opponent, to choose the worst opponent's action. This is often worse than the real worst case, which gave the name to the model. The *overconfident* opponent model returns in opponent's information sets the mean value over all actions, assuming that the opponent plays randomly. The authors compare these two opponent models on Kriegspiel and randomly generated games, concluding that the overconfident opponent model performs better in these domains.

In [PNS06, PNS10], the authors propose to run the search on the full extensive-form representation of the game. Without a perfect opponent model, this approach does not provide any guaranteed on the quality of play. A very similar algorithm can be formulated on IST of one player, which is much more efficient in domains that allow fast implementation of the generative model of IST. IST can be orders of magnitude smaller than the complete EFG (e.g, [LPS$^+$12]). If we can generate the IST directly, we can modify the search procedure to select the maximizing action for the searching player as before, but to select the worst possible observation or the mean over all observations in the observation nodes.

This approach was taken by [CF10, RNK$^+$10, LBP12] and it was presented as Single-Observer Information Set Monte Carlo Tree Search with Partially Observable Moves in [CPW12]. It allows the search to go substantially deeper, but it may cause the strategy fusion problem, which causes the paranoid search to be even more paranoid. As we can see in the example in Figure 3.3, multiple observation nodes can correspond to the same information set of the opponent. While in the EFG, the opponent would have to make a consistent worst case decision for whole information set, observations corresponding to different actions can be selected in IST.

One mayor advantage of running paranoid search on IST is its direct equivalence to the minimax algorithm, just on a modified search space. It allows alpha-beta pruning and all other search improvements developed for perfect-information games. For this reason, the full width minimax (or expectimax) search can be substituted by classical perfect-information MCTS without any modifications. For the paranoid opponent model, this MCTS uses a selection function minimizing the searching player's utility in the observation nodes. For the overconfident opponent model, it just picks one of the observations randomly.

The problem of strategy fusion in IST is less important for the overconfident opponent model. If the opponent's actions are selected randomly, splitting the opponent's information set does not have any effect. On the other hand, the model for generating probabilities of individual observations would have to be more complicated on IST, if it is supposed to model the uniform player in EFG. On the other hand, for general non-uniform opponent model, it may be even easier to express it in terms of probabilities of observations than the probabilities of opponent's actions, mainly is the model is learned automatically.

The biggest computational advantages for search from single player's perspective are possible mainly with a domain specific generative representation of IST, which is generally not the case for all EFGs. Therefore, we further discuss and evaluate this algorithm in domain specific case study in Chapter 5.

3.2.3 Information Set Monte Carlo Tree Search

Information Set Monte Carlo Tree Search (IS-MCTS) is a Monte Carlo tree search technique that learns from samples in the full extensive-form tree of the game. Fundamentally very similar algorithms have been previously formulated in two different ways. The formulation that is easier to understand is presented in [PdJL11]. The MCTS-like iterations are performed on the complete game tree of the extensive-form games (i.e., all possible states on the game). However, the statistics for the selection algorithm, such as UCT, are collected for the whole information set. If any of the nodes in the information set is reached, the selection algorithm is used to select the next action and subsequently, the statistics stored by this algorithm are updated by the result of the simulation. In the tree expansion step, authors in [PdJL11] suggests adding a single node to the extensive-form tree of the game.

A very similar algorithm is presented in [LPS^{+}12], in [CPW12] as "multi-observer information set Monte Carlo tree search" (MO-ISMTCS) and [Aug11] as "Multiple Monte Carlo Tree Search". The algorithms perform the exact same computation as [PdJL11], but they use different data structures to represent the tree. Instead of the regular extensive-form game tree, they use two information set trees for individual players. If we run a MCTS iteration from the root state of the game, we can simultaneously descend in the information set trees of both players. When we need an action for a player, we have reached a decision node in his information set tree and we can use the selection function from that node. If an action is selected for one player, an appropriate observation can be generated for the other player to select the right child in his information set tree. Note that the strategy fusion problem for the opponent is not present in the tree anymore. The move is always selected from IST of the player with all information sets represented by a unique node.

Differences between the algorithms presented in [CPW12] and [Aug11] are that the latter focusses only on the offline case without further search during the game play and that the first uses UCT and the second Exp3 algorithm with depth-dependent weighting for selection. In [CPW12], the authors discuss the online case and explicitly say they do not model the likelihood of individual states in the current inner information set and run the search assuming all these states are equally likely. In this section, we adapt an approach based on [LPS$^+$12] and compute non-uniform belief distribution among the states in the game.

Even though the algorithms with single and multiple trees perform the same computation, they can have different applicability trade-offs. The single tree approach, as proposed in [PdJL11], requires storing all states of the game in the partially constructed EFG search tree, while the multiple trees approach requires storing only the ISTs of the players. If the information sets include large number of nodes, the latter can be substantially more memory efficient. In [LPS$^+$12], we show that for the specific game modelling real world problem from network security, the size of the EFG is in the order of 10^20 while the size of bot ISTs combined is only 10^14. One the other hand, computing directly with the information sets, determining what observations can be seen after performing an action in an information set and computing the successor information sets without explicitly storing all its nodes can be expensive. In the case of the visibility tracking game, information set consists of the set of all possible trajectories the unseen agents of the opponent might have followed since they were spotted for the last time. The possible observations are all positions where the opponent's agents may become visible in the next step of the game. Computing these successors in the information set tree can be substantially more time consuming than computing the next states after applying an action in a specific game state, which is required in the regular EFG representation. The belonging to the information sets in case of EFG can be efficiently checked using a hashing approach, as the one we explain later.

Proposed IS-MCTS implementation

In this subsection we propose a variant of MCTS, which combines the lower memory requirements of approaches from [Aug11, LPS$^+$12, CPW12] and the simpler representation of state space from [PdJL11]. Moreover, it allows estimating the belief distribution among individual states in the current information set, which substantially improves performance of the algorithm in our experiments. We first presented this approach in [LBP14].

Pseudo-code for a single iteration of the algorithm is presented in Figure 3.4. It starts in the root node of the game tree and descends the game tree towards a terminal state. However, the tree nodes are never created in the memory and only the statistics for the information sets are stored in a hash table (HT). If the function is called with a terminal state, it just returns its utility for the first player (line 1). If nature selects an action in the current node (line 2), it is selected from the commonly known nature distribution and the algorithm is called recursively on the resulting state (line 4). Otherwise, the statistics for the information set it belongs to are accessed in the hash table (line 5) and used to make the decision about the action to select (line 7). This action is than executed on the game state, producing the following game state, which is used in recursive call of the function (line 8). This process is continued until a state without information set statistics in the hash table is reached (line 9). This is the end of the selection stage and the expansion consists of adding new (empty) information set statistics for the currently reached

```
IS-MCTS(h)
1: if $h$ is terminal then return $u_1(h)$
2: if $h$ is a chance node ($P(h) = c$) then
3:     $a \leftarrow$ random chance action from $\mathcal{A}_c(h)$
4:     return $IS\text{-}MCTS(\mathcal{T}(h, a))$
5: $IS \leftarrow HT(h)$
6: if $IS$ is not null then
7:     $a \leftarrow select(IS)$
8:     $v \leftarrow IS - MCTS(\mathcal{T}(h, a))$
9: else
10:    add new $IS$ for $h$ to $HT$
11:    $a \leftarrow select(IS)$
12:    $v \leftarrow simulate(\mathcal{T}(h, a))$
13: $update(IS, a, v)$
14: return $v$
```

Figure 3.4: Information-set Monte Carlo tree search

information set to the hash table (line 10). At this point, an action is selected (typically randomly) by the selection function and the simulation is executed to estimate the quality of the following position (lines 11-12). The information sets accessed during the iteration are updated by the result of the simulation (line 13) when returning from the recursion and the next iteration can start. Iterations are repeated until a given time budget is spent.

The functions *select* and *update* in the pseudo-code can be implemented by a suitable selection function, such as any algorithm for multi-armed bandit problem form Section 2.2 (with negative of the received value for the opponent). Function *simulate* can be either completely random, or domain dependent simulation, as in perfect-information MCTS.

An important part of the algorithm is the hash table HT. It has to quickly return the statistics of the information set that includes a specified game state. For this, the game state can be used as the key of the hash table, but the functions for computing the hash code and checking the equality have to be modified to use only that part of the information included in the game state that is known to the player making the decision in the requested information set. Since IS-MCTS with UCT selection does not have any guarantees that would rely on perfect recall, domain specific applications can use a custom imperfect recall abstraction within the hash table.

Progressing in the game

MCTS algorithms in perfect-information games usually do not run a new computation from scratch in each step of the game, but reuse a part of the previously build game tree. This process is natural, because in a perfect-information game, the players know exactly the state of the game and the search for the successor of the root node looks exactly the same as if the node is a root node from the beginning. The

```
ISMCTS-progress($a_i$, $o$, $curIS$)
Input: $a_i$ – performed action, $o$ – new observations, $curIS$ - set of possible states
    and their probabilities
Output: $newIS$ – new possible states and their probabilities
1: for $s \in curIS$ do
2:     $s' \leftarrow execute(s, a_i)$
3:     for $a_{-i}$ opponent's action applicable in $s'$ do
4:         if $execute(s', a_{-i})$ generates $o$ then
5:             $s'' \leftarrow execute(s', a_{-i})$
6:             $newIS \leftarrow newIS \cup \{s''\}$
7:             $newIS.p(s'') \leftarrow curIS.p(s) * freq(a_{-i}, HT(s))$
8: normalize probability estimates in $newIS$
9: return $newIS$
```

Figure 3.5: Reusing information after a move in Information-set Monte Carlo tree search

situation is more complicated in imperfect-information games. The player knows only the current information set, which can include hundreds of game states. Furthermore, these states are not equally probable and in many games, the probability of being in a specific game state heavily depends on the strategy deployed by the opponent.

None of the versions of the IS-MCTS algorithm presented in [CPW12] tries to model these probabilities. They assume the uniform probability distribution over all possible states if the match proceeds to a non-singleton information set. This is often a very unrealistic assumption. For example in a visibility-based pursuit-evasion game, consider the evader sees he is being followed by a pursuer unit, which is 3 steps behind him and he turns behind a corner, losing the visibility contact with the pursuer. After two more steps, the evader assumes it is equally likely that the pursuer is just behind the corner as that the evader turned around and moved 2 steps in the opposite direction. The other papers on IS-MCTS ([Aug11, PdJL11]) did not discuss running the algorithm from a non-root information set at all.

We suggest better approximating the likelihood of individual states in the current information set using the approximation of the opponent's optimal strategy computed by the algorithm. We first suggested this method in [LPS$^+$12] and we further developed it in [LBP14].

The algorithm assumes that the player has selected an action a_i to play and before it has to select next action, he is informed about some observations o, which represents the information defining the information set in which he will have to decide next. Without loss of generality, we assume that the searching player's observation is generated by a single action of the opponent. It opponent can perform sequences of moves before the searching player moves again, they can be substituted by individual actions without fundamentally changing the game.

The algorithm is presented in Figure 3.5. During a match, the player maintains the set of all game states in the current information set ($curIS$). For each game state, it sores an approximation of the probability that this state is actually the current state of the game ($curIS.p(s)$). In the root of the game, the players know the exact state of the game with probability 1. In any information set in the game, after the player chooses and performs the action a_i, he applies this action to each state they consider possible (line 2). Furthermore, for each possible state, he applies all actions that are applicable by the opponent (lines 3-7). If the action generates observations different from the once received, it is discarded (line 4). Otherwise, the player can estimate the relative probability of the rational opponent actually using the action based on the frequency of selecting the actions during his search (line 7). At the end, these estimates are normalized to ease further processing (line 8).

In some domains, the size of the current information set can grow too quickly to make its complete representation practical. In that case, the states in the current IS can be sampled (e.g., [PNS10]), but issues with receiving an observation inconsistent with all considered game states can occur. In that case, methods for sampling new states in an information set, such as the one in [RA12] can be used to recover. In our evaluation, this happens very rarely. Therefore, we simply continue with random actions until the end of the match if it happens.

When further IS-MCTS iterations are run from the new information set, the starting state of each iteration is selected from the possible state with frequency proportional to its probability estimate. In order to implement this quickly, we suggest selecting a state from the current information set uniformly, but then run $1+K*curIS.p(s)$ iterations from this information set, where K is a constant selected with respect to the expected number of iterations per move.

The action to be played by player after the time limit for simulations is depleted is for all MAB-based selection functions selected at random from a probability distribution given by normalizations of the use counts of actions in the root information set.

IS-MCTS non-convergence

In this thesis, we focus on convergence of MCTS methods to Nash equilibrium of the game. The following example shows why it is usually not the case with IS-MCTS without approximating the belief distribution in the current information set, as it is introduced in [CPW12]. After a non-root information set is reached during the match, further simulations are run from a random node in this information set with uniform probability. Consider the game in Figure 3.6. Suppose IS-MCTS searches from information set I. Because utilities for actions taken from both states are combined, IS-MCTS will choose left and right action equally often. However, mixing uniformly at I is not part of an equilibrium in this game. It would lead to an expected utility of $\frac{3}{4}$ to the maximizing player for playing right and $\frac{5}{4}$ for playing left. Hence, the uniform strategy in I could be equilibrium only with pure strategy of playing left for the maximizer. This is not an equilibrium, because it the maximizer plays left, the minimizer would be better off playing right, reaching expected reward of 0.5 than keeping the uniform distribution with reward $\frac{5}{4}$.

In this particular example, even our modification with approximating the belief in the current information set does not help the IS-MCTS algorithm to converge to Nash equilibrium. When the algorithm is run from the root, it likely converges close

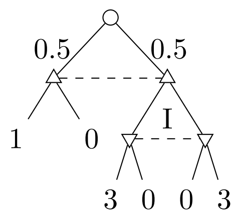


Figure 3.6: An example game with maximizing △, minimizing ▽ and chance ◯ players. The optimal (Nash equilibrium) strategy is for △ to play (0.5,0.5) and for ▽ to play (left,right) with probabilities $(\frac{1}{3},\frac{2}{3})$. Its value is 1.0.

to a Nash equilibrium of the game, learning that player △ should play a uniform probability distribution. After information set I is reached, the belief distribution is correctly uniform and the problem remains. Even if the correct strategy is originally present at I, it will be re-learned to the incorrect uniform strategy. This example also shows that the non-locality problem introduced in [FB98] is not always caused by misrepresentation of the beliefs. Even with correct beliefs, the strong relation between different parts of the game can prevent a search algorithm to reach an optimal decision.

Our modification wit modelling beliefs would; however, help in a slightly modified game, where the left-most leaf node has value 5 instead on 1. In that case, the optimal strategy of player △ is to play left with probability 1. This will be reflected in the belief distribution in information set I, which will say that the left node is much more probable than the right. Hence, after additional search, the right action will be selected for player ▽, giving him the loss of zero in this case and 2.5 overall. The original IS-MCTS would still converge to the uniform distribution with expected loss of 4.

3.2.4 Online Conterfactual Regret Minimization

Similarly to the case of the simultaneous-move games described in Section 3.1.3, we have adapted MCCFR to an online algorithm for playing generic extensive-form game. We have first published the algorithm in [LLB14]. The algorithm runs on the full extensive-form game tree, which is incrementally build as in IS-MCTS. The algorithm is very similar to MCCFR when run from the root of the game; however, the situation is much more complex after players have played some moves in the game. There is no clear notion of a subgame in most imperfect-information games. Strategies in different parts of the game strongly influence each other. Therefore, even after the game progresses deep into the tree, we still have to run samples form the root of the game, biasing the sampling mechanism to refine the strategy more quickly in the current information set of the game, where we need to make the next action selection. The disadvantage of the targeting is possibly a higher variance of value estimates for information sets that include both targeted and untargeted histories, which was proven to have negative impact on speed of convergence in [GLB^{+}12]. The advantage we gain, if we do it properly, is an anytime game playing algorithm with guaranteed convergence to the Nash equilibrium of the game.

Online Outcome Sampling

The structure of the algorithm is similar to SM-OOS described in Section 3.1.3 or MCCFR. The main difference is that the algorithm allows running the standard untargeted γ-on policy sample with probability $(1 - \delta)$ and a targeted (or biased) sample reaching one of a pre-selected subsets of terminal nodes of the game tree ($\mathcal{Z}^T$) with probability δ. Typically, the targeted part of the tree will be the current information set (i.e, $\mathcal{Z}^T = \mathcal{Z}_I$). In the OOS-main procedure in Figure 3.7, targeting is decided first (line 1) and then two iterations, one for each player, are run with this setting.

Unlike in SM-OOS, individual iteration in function OOS keeps two variants of the sampling probability of the current history: the sampling probability in the unbiased case s_u and the sampling probability in the biased case s_b. We do that to be able to eventually compute the overall probability of reaching a particular history in the sampling, but keeping track of the probabilities separately makes it simpler. The return values are the same as in SM-OOS, namely (x, q, u_i) corresponding to the tail probability $\pi^\sigma(h, z)$, the probability of sampling the reached terminal history from the root, and the utility of the terminal history for the regret updating player i. If the algorithm reaches a terminal history (line 1), it returns the utility value in the terminal history and computes the probability of sampling the history, regardless of running the biased or unbiased iteration. If it reaches a chance node, the chance strategy for that node is restricted to the actions that lead to the targeted part of the game $\mathcal{Z}^T$ and normalized to sum to one on line 3. If the current iteration is biased, the chance action is sampled from the restricted distribution (line 5). Otherwise, it is sampled from the original chance distribution (line 7). On line 8, OOS is recursively called on the history after the selected chance action, adding the probability of selecting the chance action to the opponent's reach probability (π_{-i}) and unbiased sampling probability, and adding the biased chance probability to the biased sample probability. If none of the conditions on lines 1 and 2 is satisfied, the algorithm reached a decision history for one of the players. On line 9, the algorithm attempts to fetch the information set for this history from memory. If it is already there, it builds strategy σ by regret matching from the cumulative regrets stored in the information set (line 11). Function "SampleAction" on line 12 samples an action a from (possibly biased) strategy σ, with added γ-exploration if player i decides in the history h. This function also updates the sampling probabilities accordingly. The detailed pseudocode for this function is presented in Figure 3.8 and explained later. Line 13 updates the reach probability for the player deciding in history h and the algorithm is recursively called to the following history with the updated probabilities on line 14. If the information set for history h is not found in memory (line 15), it is created and the next action is sampled based on the uniform probability (lines 14-18). Afterwards, a rollout selecting random actions until the end of the game is initiated on line 19. This rollout is similar to the MCTS case; however, it keeps track of the overall probability of reaching the terminal node from

```
**OOS**$(h, i, \pi_i, \pi_{-i}, s_u, s_b)$
Input: $h$ – current state of the game; $i$ – the regret updating player; $\pi_i$ – player $i$'s contribution to the probability of reaching $h$; $\pi_{-i}$ – player $-i$'s and chance's contribution to probability of reaching $h$; $s_u$ – the probability of sampling the current history in an unbiased iteration; $s_b$ – the probability of sampling the current history in a biased iteration
Output: $(x, q, u)$: $x$ – probability of reaching the sampled leaf from current state based on player's strategies ($\pi^\sigma(h, z)$); $q$ – probability of sampling the leaf from the root ($q(z)$); $u_i$ – utility of the leaf for player $i$
1: if $h \in \mathcal{Z}$ then return $(1, (1-\delta)s_u + \delta s_b, u_i(h))$
2: if $h \in \mathcal{C}$ then
3:    $\sigma_c^b \leftarrow$ Restrict$(h, \sigma_c(h), \mathcal{Z}^T)$
4:    if biased then
5:       Sample chance action $a$ from $\sigma_c^b(h)$
6:    else
7:       Sample chance action $a$ from $\sigma_c(h)$
8:    return OOS$(ha, i, \pi_i, \pi_{-i}\sigma_c(h,a), s_u\sigma_c(h,a), s_b\sigma_c^b(h,a))$
9: $I \leftarrow HT(h)$
10: if $I$ is not null then
11:    $\sigma \leftarrow$ RegretMatching$(R(I))$
12:    $(a, s'_u, s'_b) \leftarrow$ SampleAction$(h, \sigma, i, \gamma, s_u, s_b)$
13:    $\pi'_{\mathcal{P}(h)} \leftarrow \sigma(a)\pi_{\mathcal{P}(h)}$; $\pi'_{-\mathcal{P}(h)} \leftarrow \pi_{-\mathcal{P}(h)}$
14:    $(x, q, u_i) \leftarrow$OOS$(ha, i, \pi'_i, \pi'_{-i}, s'_u, s'_b)$
15: else
16:    Add new $I$ into $HT$
17:    $\sigma \leftarrow$ Uniform$(\mathcal{A}(h))$
18:    Sample $a$ from $\sigma$
19:    $(x, q, u_i) \leftarrow$ OOS-Rollout$(ha, \frac{1}{|\mathcal{A}(h)|}((1-\delta)s_u + \delta s_b))$
20: $c \leftarrow x$
21: $x \leftarrow c \cdot \sigma(a)$
22: if $\mathcal{P}(h) = i$ then
23:    $W \leftarrow u_i\pi_{-i} \;/\; q$
24:    $R(I, a) \leftarrow R(I, a) + (c - x)W$
25:    $\forall b \in \mathcal{A}(h) \setminus \{a\}$ $R(I, b) \leftarrow R(I, b) - xW$
26: else
27:    $\bar{\sigma}_i(I) \leftarrow \bar{\sigma}_i(I) + \frac{1}{(1-\delta)s_u + \delta s_b}\pi_{-i}\sigma$
28: return $(x, q, u_i)$
```

```
**OOS-main**$(root, \delta, L)$
1: while time limit $L$ is not reached do
2:    $biased = TRUE$ with probability $\delta$ and $FALSE$ otherwise
3:    OOS$(root, 1, 1, 1, 1, 1)$
4:    OOS$(root, 2, 1, 1, 1, 1)$
```

Figure 3.7: Online Outcome Sampling algorithm. Description of functions Restrict and SampleAction is provided in Figure 3.8.

```
Restrict$(h, \sigma, \mathcal{Z}^T)$
1: $A' \leftarrow \{a \in \mathcal{A}(h) | \exists z \in \mathcal{Z}^T \; ha \sqsubset z\}$
2: $\forall a \in \mathcal{A}(h) \; \sigma^b(a) \leftarrow 0$
3: $sum \leftarrow \sum_{a \in A'} \sigma(a)$
4: $\forall a \in A' \; \sigma^b(a) \leftarrow \sigma(a)/sum$
5: return $\sigma^b$
```

```
SampleAction$(h, \sigma, i, \gamma, s_u, s_b)$
1: $\sigma^b \leftarrow$ Restrict$(h, \sigma, \mathcal{Z}^T)$
2: if $\mathcal{P}(h) = i$ then
3:     $\forall a \in \mathcal{A}(h) \; \sigma(a) \leftarrow (1-\gamma)\sigma(a) + \frac{\gamma}{|\mathcal{A}(h)|}$
4:     $A' \leftarrow \{a \in \mathcal{A}(h) | \exists z \in \mathcal{Z}^T \; ha \sqsubset z\}$
5:     $\forall a \in A' \; \sigma^b(a) \leftarrow (1-\gamma)\sigma^b(a) + \frac{\gamma}{|A'|}$
6: if biased then
7:     Sample $a$ from $\sigma^b$
8: else
9:     Sample $a$ from $\sigma$
10: return $(a, \sigma(a)s_u, \sigma^b(a)s_b)$
```

Figure 3.8: Helper functions for Online Outcome Sampling.

the root (q) and tail probability of reaching the terminal node from history h due to (uniform) strategies of the players and chance (x). Regardless of whether the current information set has been already in memory, the algorithm follows by updating the regrets and mean strategies in the information set. First, it updates x to include also the probability of selecting the action in the current information set (line 21). If player i decides in the history h, it updates the cumulative regrets in the current information set. The product of terms that are identical in the definition of sampled counterfactual value (equation 2.13 in Section 2.5.1) for the actual strategy and the strategy modified to play a in I is stored to W. Using this, the following two lines update the regrets based on sampled counterfactual regret definition for the sampled action a (line 24) and all the other actions (line 25). If player $-i$ decides in history h (line 26), the mean strategy is updated on line 27. Note that we actually store the weighted sum of the used strategies and have to eventually normalize $\bar{\sigma}_i(I)$ to sum to one to get the actual strategy.

The helper functions used in OOS algorithm are presented in details in Figure 3.8. The "Restrict" function restricts a strategy σ in history h to only those actions that lead to terminal histories in $\mathcal{Z}^T$. First, it creates the set of actions, for which adding these actions to the current history creates a prefix of some history in $\mathcal{Z}^T$ (line 1). The new strategy is initialized by zeros for all actions (line 2). The sum of the probabilities of actions leading to $\mathcal{Z}^T$ is computed and if it is not zero, which should never happen in a biased iteration, the rescaled probabilities of playing these actions is stored to the output strategy.

Function "SampleAction" first restricts strategy σ only to the desired actions on line 1. Then if player i decides in history h, it adds the γ-exploration to both the restricted and the unrestricted version of the strategy (lines 3-5). It samples the action from the corresponding version of the strategy (line 6-9) and returns this actions and sampling probabilities for both the biased and unbiased cases (line 10).

Progressing in the game

After some moves are played in an imperfect-information game, the player who decides about the next move knows the current information set I and typically has some additional computation time available before he selects an action to play. We aim to use the targeting included in the algorithm description to refine the strategy more quickly in the current information set. In other words, we want to bias the search to visit more often the terminal histories from $\mathcal{Z}_I$. However, focusing fully only on these histories may cause problems, since convergence guarantees are lost. Consider again the game in Figure 3.6. If the minimizing player knows it is in the information set I and focuses its entire search only to this information set for a sufficiently long time, she computes the incorrect uniform strategy, in the right part of the game. On the other hand, any fixed non-zero probability of sampling the left chance action will eventually solve the problem. The regrets are multiplied by the reciprocal of the sampling probability; hence, their influence to the strategy in the information set is proportionally stronger if the samples are rare. This solves the problem. If there is no such pathology as in Figure 3.6 in the game played by the algorithm, it benefits from refining the important part of the tree more. If it is present, there is a good chance that the strategy will be at least partially corrected by the untargeted samples in the short time and it will provably converge to the optimal strategy in the long run.

An alternative way of understanding the problem caused by fully targeted sampling is breaking the information structure of the game. Even though the player using the proposed algorithm to decide about its next move knows that it has reached information set I, this information is unknown to his opponent. If it restricts the regret updates only to histories with prefixes in I, even opponent's regrets are updated only based on the outcomes of the game after these histories. As a result, the regrets are updated as if the opponent knew that the game is currently in one of the histories in I, which is generally not the case. Note that this is not a problem in the variant of OOS for the simultaneous-move games introduced in Section 3.1.3, because the information in these games is symmetric.

The impact of progressing in the game to an information set I to the algorithm is changing the targeted set $\mathcal{Z}^T$ to $\mathcal{Z}_I$. In general all the previously created information sets remain in memory. Let's analyze the impact of this change to the algorithm. There is no substantial impact on probabilities π_i, π_{-i}, and s_u after the change of targeting. The main impact will be on s_b. In general, the new set of targeted leaves is a subset of the previously targeted leaves. For the histories that will stay targeted, s_b will slightly increase. For the histories that will not be targeted anymore, s_b will become zero, which will dramatically decrease the overall probability of reaching prefixes of these histories ($(1-\delta)s_u + \delta s_b$) and the overall leaf reach probabilities q. This effect will be the strongest when the actions chosen by the players since the last computation are least probable. However, in theory, in should not influence the convergence dramatically, because the decreased weight of samples caused by higher q should be compensated by much more frequent updates in the more targeted part of the tree and vice versa.

Convergence guarantees

We designed the Online Outcome Sampling algorithm to stay consistent with standard offline Outcome Sampling Monte Carlo Counterfactual Regret Minimization with non-uniform sampling scheme [LWZB09]. As a result, we can almost directly reuse the proof of its convergence to the Nash equilibrium of the game.

Theorem 3.2.1. *Let $\bar{\sigma}^t_m(\delta, h)$ be a strategy produced by OOS in a zero-sum extensive-form game with perfect recall, targeting any subset of terminal nodes $\mathcal{Z}^T$ with probability $\delta < 1$ started from match history h run for t iterations, and with exploration $\gamma > 0$. For any $p \in (0, 1], \varepsilon > 0$ there exists $t < \infty$ such that with probability $1 - p$ the strategy $\bar{\sigma}^t_m(\delta, h)$ is a ε-Nash equilibrium.*

Since $\delta < 1$ and $\gamma > 0$, every terminal history has non-zero probability of being sampled. Therefore, every information set will eventually be contained in memory. From this point, the algorithm becomes Outcome Sampling MCCFR with a non-uniform sampling scheme and possibly wrong initial values in the regret tables. While the regret values are cumulated during the run of the algorithm, any such initial values will eventually become irrelevant compare to newly accumulated correct regret values. Consequently by [LWZB09, Theorem 5] OOS minimizes regret with high probability and converges to Nash equilibrium.

Irrational moves and equilibrium refinements

Above we show that OOS converges to the Nash equilibrium of the game. While playing a Nash equilibrium is optimal against rational opponents and ensures at least the value of the game also against irrational play, in general, it does not exploit opponent's mistakes. The behavior of the players after their opponent performs a move with zero probability in an equilibrium is under-defined. The only requirement is that the strategy has to be sufficiently good to prove the move to be irrational. Moreover, the strategy under player's own irrational moves can be completely arbitrary. For these reasons, researchers study various refinements of the concept of Nash equilibrium, such as the once introduced in Section 2.1.4.

The problems that led to studying Nash equilibrium refinements demonstrate also in OOS. During the match, the game can reach an information set with histories only below irrational moves of the opponent. In that case, even with heavy targeting, the strategies below this information set will not improve anymore. Note that in definition of (sampled) counterfactual value in equation 2.13, we multiply the utility of the sampled leaf by the reach probability of the opponent π_{-i}. This is always zero after an irrational move of the opponent; hence, all the regrets will be zero as well, once the move is identified as irrational.

Note that a possible solution would be to remove the rational actions from the game, which would naturally make some of the remaining actions rational, but this might lead to the same problems as absolute targeting, explained in Section 3.2.4.

In order to avoid this problem and allow further improvement of the strategy, we propose substituting the regret matching on line 11 with ϵ-regret matching, which returns a strategy composed from the standard regret matching strategy with weight $(1-\epsilon)$ and a uniform strategy with weight ϵ. This ensures that no action of any player has zero probability of being played and consequently π_{-i} never to be zero. This completely solves the problem for information sets that are fully below the current information set. All updates in the information set are updated with similar small, but non-zero, values of π_{-i} and relative portions of regret represents the quality of the moves well. In games of perfect recall, acting player's information sets are always fully below the current information set. Having an information set reach out of these histories would mean there is a history in which no action was performed in the current information set. The information sets of the opponent are further updated normally, even below his irrational moves. In iterations updating regrets in his information sets, OOS uses γ-on policy exploration and does not weight the values with its own reach probabilities.

Chapter 4

Evaluation on Abstract Games

In the first set of experiments, we evaluate the strengths and weaknesses of the proposed algorithms on artificial and random games. These games are usually based on (possibly simplified versions of) existing games people play for fun, or specific abstract games that demonstrate some fundamental principles. In the evaluation, we focus on two main setting. First, in smaller versions of the games, we study the convergence properties of the algorithms. Mainly, how quickly individual algorithms are able to converge to the Nash equilibrium of the game. Second, in much larger versions of the games, in which computing the distance from an equilibrium is not feasible anymore, we focus mainly on the actual game playing performance of the algorithms in head-to-head matches. The part of this evaluation concerning simultaneous-move games is largely based on [BLL$^+$14], which compares sampling based and exact algorithms for solving this class of games. Part of the evaluation in the generic imperfect-information games was presented in [Lis14] and [LLB14], but most of it has not been published yet.

In the following, we first introduce the rules and properties of all the games we use in this chapter. Afterwards, we proceed with the evaluation of the algorithms for simultaneous-move games in Section 4.2 and the evaluation of the algorithms for generic extensive-form games in Section 4.3.

4.1 Games Descriptions

Here we describe the rules and basic properties of all games used for evaluation in this chapter. We start with simultaneous-move games and continue with the generic imperfect-information games.

4.1.1 Simultaneous-Move Games

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

In some games, we also evaluate the option of substituting the simulation stage of the MCTS algorithms by a heuristic evaluation function. For these domains, we also present the evaluation function here. Note that we are not seeking the best-performing algorithm for a particular game; hence, we have not aimed for the strongest evaluation functions for each game. We intentionally use evaluation functions of different quality that allow us to evaluate the differences between the algorithms from this perspective as well.

Goofspiel Goofspiel is a card game that appears in many works dedicated to simultaneous-move games (e.g., [Ros71, RB12, SFB12, LLW14, BLC$^+$13]). There are 3 identical decks of cards with values $\{0, \dots, (d-1)\}$ (one for nature and one for each player). Value of d is a parameter of the game. The variant of the game played by people has 13 cards. The deck for the nature is shuffled at the beginning of the game. In each round, nature reveals the top card from its deck. Each player selects any of their remaining cards and places it face down on the table so that the opponent does not see the card. Afterwards, the cards are turned face up and the player with the higher card wins the card revealed by nature. The card is discarded in case of a draw. At the end, the player with higher sum of the nature cards wins the whole game. In the results, we use utilities 1/0/-1 for win/draw/loss and count a draw as half win half lose in win-rates. We follow the assumption made in [SFB12] that both players know the sequence of the nature's cards.

Goofspiel 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 exact backward induction presented in [BLL$^+$14] for Goofspiel with 5 cards:

Depth	0	1	2	3	4
Pure	0	17	334	3354	14400
Mixed	1	8	66	246	0
Mixed/All	1.0	0.32	0.16	0.07	0.0

The depth is measured from the root of the game with depth 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.

The evaluation function used in Goofspiel takes into consideration two components: (1) the difference in the current score for both players weighted by the current round in the game, and (2) the remaining cards in the deck weighted by a chance of winning these cards depending on the remaining cards on hand for each player. Formally, let p_i, r, c_i be player i's score (points), the current round, and the sum of values of player i's remaining cards, respectively:

$$eval(s) = \tanh\left(\frac{p_1 - p_2}{p_1 + p_2} \cdot \frac{r}{d} + \frac{c_1 - c_2}{c_1 + c_2} \cdot \frac{c_c}{0.5 \cdot d(d+1)}\right)$$

$eval(s)$ is equal to 0 in the beginning of the game (i.e., when $p_1 + p_2$ is 0) and to the utility value of the game at the end (i.e., when $c_1 + c_2$ is 0). We use tanh to scale the evaluation function into the interval $\langle -1, 1 \rangle$. Moreover, if the position is clearly winning for one of the players (there is not enough cards to change the current score), the evaluation function is set to 1, or -1.

Oshi-Zumo Oshi-Zumo is a board game analyzed from the perspective of computational game theory in [Bur03]. There are two players in the game, both starting with N coins, and there is a playing board represented as a one-dimensional playing field with $2K + 1$ locations (indexed $0, \dots, 2K$). At the beginning, there is a stone (or a wrestler) located in the center of the playing field (i.e., at position K). During each move, both players simultaneously place their bid from the amount of coins they have (but at least M if they still have some coins). 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 loses the game. If the final position of the wrestler is the center, the game is a draw. Again, the utility values are restricted to $\{-1, 0, 1\}$.

The interesting feature of this game is a rather large branching factor, which corresponds to the number of coins left to the players. Furthermore, different branches in the game can have substantially different lengths, which is not the case in the previous game. Furthermore, many instances of Oshi-Zumo have pure Nash equilibrium. With the increasing number of the coins, all optimal strategies become mixed at some point. However, choosing a move randomly is typically required only at the beginning of the game. The following table depicts the number of states with pure strategies and mixed strategies in a subgame-perfect NE calculated by backward induction for Oshi-Zumo with $N = 10$ coins, $K = 3$, and minimal bid $M = 1$. The table shows that there are very few states where mixed strategies are required, and they are present only at the beginning of the game tree.

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
Mixed/All	0	0.01	0.002	0.001	0.0002	0	0	0	0	0

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

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

where b equals to 1 if $\text{coins}_1 \geq \text{coins}_2$ and $wrestler \geq K$, and at least one of the inequalities is strict; b equals to -1, if $\text{coins}_1 \leq \text{coins}_2$ and $wrestler \leq K$, and at least one of the inequalities is strict. Again, we use tanh to scale the evaluation function into the interval $\langle -1, 1 \rangle$.

Pursuit-Evasion Games Another important class of games is pursuit-evasion games (for example, see [NT13]). There is a single evader and a pursuer that controls 2 pursuing units on a four-connected grid in our pursuit-evasion game. In each turn, all units move simultaneously to an adjacent node on the grid. The evader wins, if she successfully avoids the units of the pursuer for a given number of moves; pursuer wins, if her units successfully capture the evader. The evader is captured if either her position is the same as the position of a pursuing unit, or the evader used the same edge as a pursuing unit (in the opposite direction). The game is win-loss and the players receive utility from set $\{-1, 1\}$. We use 3 different square grid-graphs (with the size of a side 4, 5, and 10 nodes) for the experiments without any obstacles or holes. In the experiments we varied the length of the game d and 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).

Since all units move simultaneously, the game has larger branching factor than Goofspiel (up to 16 actions for the pursuer). An interesting property of the game is its asymmetry. The evader has a much smaller branching factor than the pursuer and based on the starting position, one player is often in a more convenient position than the other. Similarly to Oshi-Zumo, many instances of pursuit-evasion games have a pure Nash equilibrium. However, randomization can be required towards the 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). 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
Mixed/All	0	0	0.19	0.12	0.03

The evaluation function used in pursuit-evasion games takes into consideration the distance between the units of the pursuer and evader (denoted distance_j for the distance in moves of the game between the j^{th} unit of the pursuer and the evader). Formally:

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

where w and l are dimensions of the grid graph.

Random/Synthetic Games 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 and 5 (the results were similar for different branching factors) and we varied the depth of the game tree. We assign the utility values to the terminal states of the game based on the N-games, often used to generate random game trees (e.g., [SN95, PNS10]). We randomly assign either -1, 0, or $+1$ value to each joint action (pair of actions) and the utility value in a leaf is a sum of all values on edges on the path from the root of the game tree to the leaf. This creates more realistic games using the intuition of good and bad moves.

Randomly generated games represent games that require mixed strategies in most of the states. The following table shows the number of states that require randomization in each depth of 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
Mixed/All	1	0.88	0.89	0.84	0.69

The evaluation function used in this case is calculated similarly as the utility value and it is equal to the sum of values on the edges from the root to the current node.

Tron Tron is a two-player simultaneous-move game played on a discrete four connected grid, possibly obstructed by walls [SRL10, PSPME12, LWWDT13]. Each player controls one unit. The units are initially positioned in opposite corners of the grid. At each step, both players move to adjacent cells, and a wall is placed to players' original positions. A player loses the game if he hits the wall or the opponent. The goal of both players is to survive as long as possible. If both players move into a wall, off the board, or into each other at the same turn, the game ends in a draw. In the experiments, we used an empty grid with no obstacles and various sizes of the grid.

Tron is a game with a rather small branching factor (at most 3 per player), but potentially a very deep tree. The number of actions in the longest branch is equal to half the number of squares in the grid. Similarly to pursuit-evasion games, there are many instances of Tron that have pure NE. However, even if mixed strategies are required, they appear in the middle of the game once both players reach the center of the board and compete over the advantage of possibly being able to occupy more squares. Once this is determined, the endgame can be solved in pure strategies since it consists of the single-agent problem of filling the available space missing as little squares as possible. The following table comparing the number of states demonstrates this characteristics of Tron on a grid 5×6:

Depth	0	1	2	3	4	5	...
Pure	1	4	14	100	565	2598	
Mixed	0	0	2	0	9	7	
Mixed/All	0	0	0.13	0	0.02	0.003	

...	6	7	8	9	10	11	12	13
	9508	25964	54304	83624	87009	63642	23296	3127
	51	92	106	121	74	0	0	0
	0.005	0.004	0.002	0.001	0.001	0	0	0

The evaluation function is based on how much space is "owned" by each player, which is a more accurate version of the space estimation heuristic [DTW12] that was used in [LWWDT13]. A cell is owned by player i if it can be reached by player i before the opponent. These values are computed using an efficient flood-fill algorithm whose sources start from the two players' current positions. Formally:

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

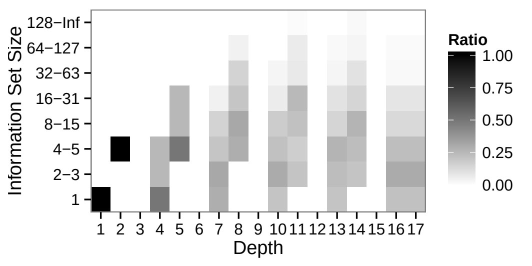
4.1.2 Imperfect-Information Games

We further discuss the imperfect rules and properties of the imperfect-information games we use for evaluation.

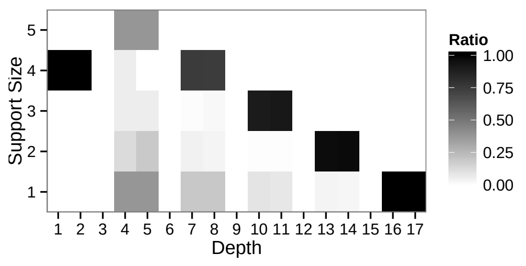
Imperfect-Information Goofspiel Even though Goofspiel is usually played as the simultaneous-move game explained in Section 4.1.1. Lanctot [Lan13] introduced an imperfect-information modification of the game. The players only discover who won or lost a bid, but not the bid cards played. Hence, the game has a different structure of information sets, but it is otherwise identical.

The imperfect information in the game originates only from the unknown actions of the opponent. Figure 4.1a presents the dependence of the size of an information set on the depth of the game with 6 cards in each deck. The darkness of each tile represents the probability that randomly selected information set in the depth depicted on the x-axis will contain the number of states depicted on the y-axis. The players alternate in the game with the chance player playing in depths $0, 3, \ldots,$ player 1 playing in depths $1, 4, \ldots,$ and player 2 in depths $2, 5, \ldots.$ We can clearly see that the information sets of player two are generally larger than the information sets of player 1. This is caused by modelling simultaneous moves in the game as as postponing the information about the move to the second player.

Figure 4.1b presents the dependence of the number of moves played with a non-zero probability in an information set on the depth of the information set on the same game as before. We computed the size of the support based on an equilibrium computed by sequence form linear program [KMvS96]. Because the game is symmetric, each player's support is equally large in each depth. Furthermore, intro-



(a)



(b)

Figure 4.1: The size of the information sets (left) and the size of the support (right) in different depths of Imperfect-Information Goofspiel with 6 cards. The darker color means higher portion of information sets with given property in given depth.

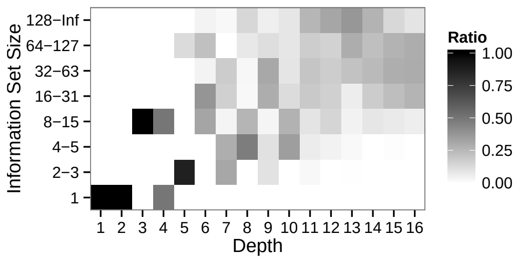


Figure 4.2: The size of the information sets in different depths of Phantom Tic-Tac-Toe, in which the first move of each player is forced to be to the middle field. The darker color means higher portion of information sets with given property in given depth.

ducing the imperfect information in the game increased the need for randomization. In majority of information sets, the players randomize over all remaining cards.

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. The player who achieves this goal first wins 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. 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 ($\approx 10^{10}$ nodes [LGBB12]). 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 player never learns how many attempts the opponent actually performed, a single information set can contain nodes at different depth in the game tree.

Figure 4.2 presents the dependence of the size of the information sets on the depth in the tree in a simplified variant of the game, in which the first move of each player is forced to be to the middle field. We were not able to compute these statistics for the full game in a reasonable time. For measuring the depth, we picked a random history in the information set. We can see that even this simplified version of the game has rather large information sets. Part of the reason is that even after a player discovers the opponent's moves, he never learns the order in which the moves were played, which will still have to be represented by the information sets. This information may easily be irrelevant for the game, but removing it would cause the game to lose perfect recall, which brings many complications to the algorithms and even to the theory defining the solution concepts.

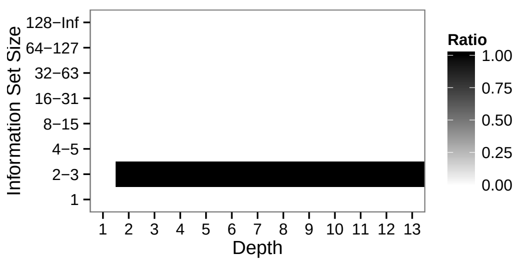
Poker Games Poker is frequently studied in the literature as an example of a large extensive-form game with imperfect information. It is also an example of a game that typically has solutions with large support. We include experiments with a simplified two-player poker game inspired by Leduc Hold'em.

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.

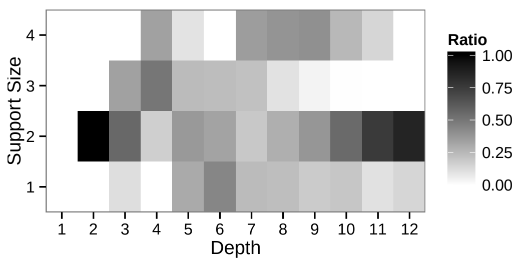
In the experiments we alter the number of types of the cards (from 3 to 4; there are 3 types of cards in Leduc), the number of cards of each type (from 2 to 3; set to 2 in Leduc), the maximum length of sequence of raises in a betting round (ranging from 1 to 4; set to 1 in Leduc), and the number of different 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).

This game is substantially different from the Imperfect-Information Goofspiel in several aspects. Most important aspect is that all actions of the players are public. As a result, all information sets in the game have the same size. In the simple game with 2 bet sizes, at most 2 rounds of rising and 3 cards of each of 3 types, which we use in experiment, the size of all information sets is 3 (see Figure 4.3a). The nodes in the information set correspond to each possible card held by the opponent.

Each player has up to 4 actions in each information set and different situations require mixing among different numbers of actions. Figure 4.3b shows the sizes of the supports in individual information sets computed based on a similar methodology as for before. The first two levels of the tree are decisions of the nature player. Afterwards, the number of actions played a player in an information set if spread quite uniformly.



(a)



(b)

Figure 4.3: The size of the information sets (left) and the size of the support (right) in different depths of Generic Poker with 2 bet sizes, at most 2 rounds of rising and 3 cards of each of 3 types. The darker color means higher portion of information sets with given property in given depth.

4.2 Evaluation on Simultaneous-Move Games

In this section, we first experimentally study the convergence of SM-MCTS and SM-MCTS-A to the Nash equilibrium on very simple games. We show that even though we currently provide the proof only for SM-MCTS-A, even SM-MCTS with a Hannan consistent selection function seems to converge close to a Nash equilibria of the game. Moreover, the empirical convergence rate seems to be faster.

Afterwards, we thoroughly evaluate the performance of SM-MCTS and SM-OOS in one game at a time, providing more general conclusions at the end. For each game, we present three basic experiments. First, we evaluate the speed of convergence of individual algorithms on a smaller version of the game. We run each algorithm from the root of the game for up to 500 seconds and regularly compute the distance form Nash equilibrium. Afterwards, we run a set of matches on the exact same small version of the game and analyze the relation between the speed of convergence and game playing performance. Finally, we run a set of matches on a substantially larger version of the game and study how the results from the small version of the game translate to a more realistic situation. All head-to-head results are means from at least 1000 matches.

4.2.1 Propagating Values Versus Propagating Means

The formal analysis presented in Section 3.1.2 requires the SM-MCTS algorithms to return the mean of all previous samples instead of the value of the current sample. While we can prove convergence to approximate Nash equilibrium for the first, the latter is generally the case in previous works on SM-MCTS [LLW14, TF11, TLW14]. Therefore in our first experiment, we run both variants with the Regret matching selection function on a set on the randomly generated games parameterized by depth and branching factor. In this experiment, the utility values are uniformly randomly selected from interval $\langle 0, 1\rangle$. Each experiment uses 100 random games and 100 runs of the algorithm.

Figure 4.4 presents how the exploitability of the strategies produced by Decoupled regret matching with propagation of the mean (DRMM) and current sample

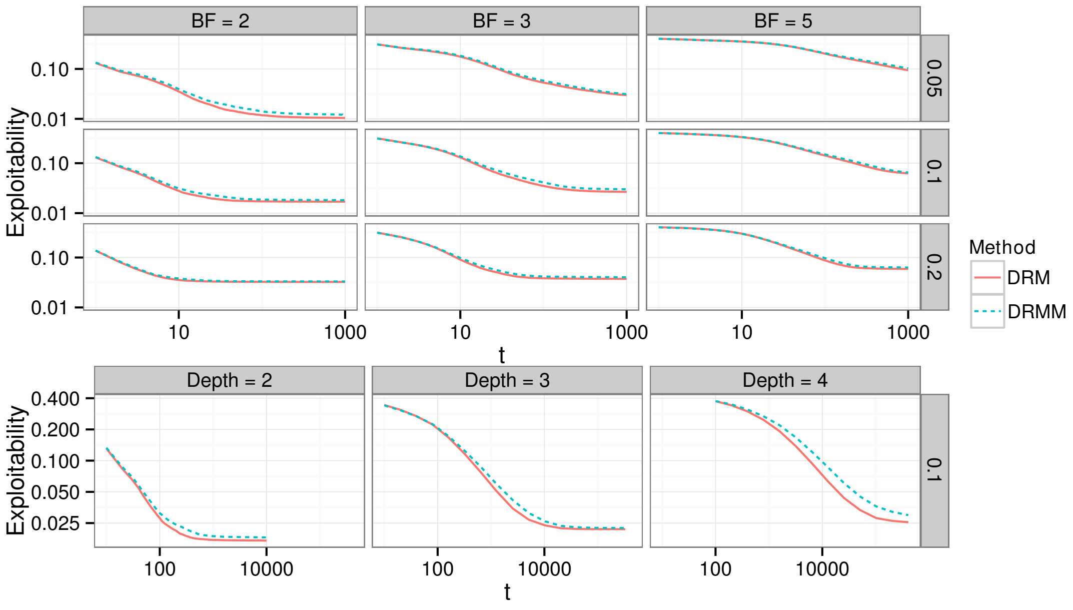


Figure 4.4: Exploitability of strategies given by the empirical frequencies of Regret matching with propagating values (RM) and means (RMM) for various depths and branching factors.

value (DRM) develops with increasing number of iterations. Note that both axes are in logarithmic scale. The top graph is for depth of 2, different branching factors (BF) and $\gamma \in \{0.05, 0.1, 0.2\}$. The bottom one presents different depths for $BF = 2$. The results show that both methods converge to the approximate Nash equilibrium of the game. DRMM converges slower in all cases. The difference is small in shallow games, but becomes more apparent in games with larger depth.

4.2.2 Empirical Worst Case

Although the formal analysis guarantees the convergence to an ϵ-NE of the game, the rate of the convergence is not given. Therefore, we give an empirical analysis of the convergence and specifically focus on the cases that reached the slowest convergence from a set of evaluated games.

We have performed a brute force search through all games of depth 2 with branching factor 2 and utilities from the set $\{0, 0.5, 1\}$. We made 100 runs of DRM and DRMM with exploration set to $\gamma = 0.05$ for 1000 iterations and computed the mean exploitability of the strategy. The games with the highest exploitability for each method are presented in Figure 4.5. These games are not guaranteed to be the exact worst case, because of possible error caused by only 100 runs of the algorithm, but they are representatives of particularly difficult cases for the algorithms. In general, the games that are most difficult for one method are difficult also for the other. Note that we systematically searched also for games in which DRMM performs better than DRM, but this was never the case with a sufficient number of runs of the

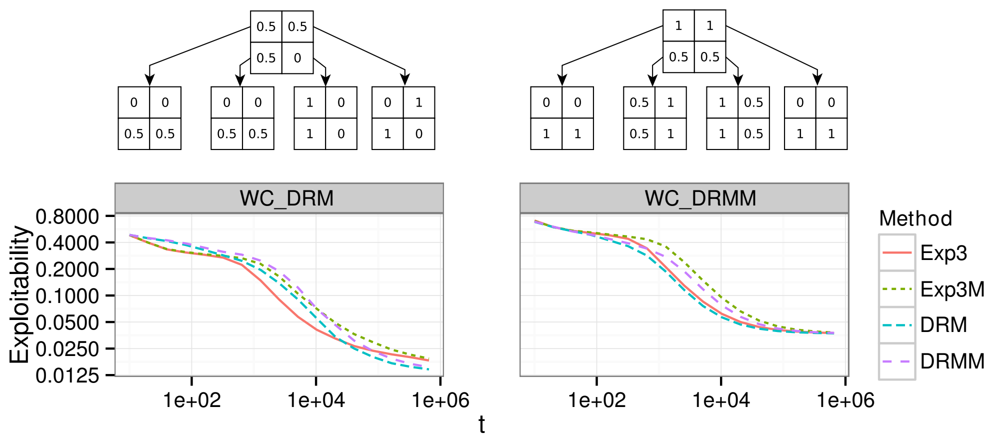


Figure 4.5: The games with maximal exploitability after 1000 iterations with RM (left) and RMM (right) and the corresponding exploitabililty for all evaluated methods.

algorithms. These games are difficult for the sampling algorithms because they are misleading at the beginning of the search. Consider the maximizing (row) player in the second game. He should learn to play the first action. However, any time he tries this action and explores lower in the tree, he receives the worst possible reward of 0. This makes the first action appear worse than it actually is. Every time the agent tries the second action and the opponent explores, he receives a high reward, making the second action looks better. This makes it harder to learn quickly that the first action is always better and should be played with probability 1.

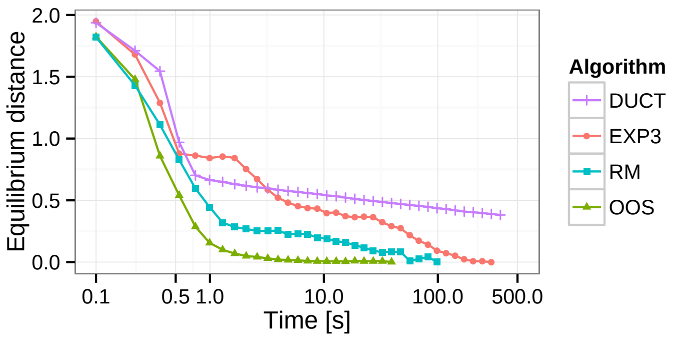
Figure 4.5 shows the convergence of DRM and Exp3 with propagating the current sample values and the mean values (DRMM and Exp3M) on the empirically worst games for the DRM variants. The DRM variants converge to the minimal achievable values (0.0119 and 0.0367) after a million iterations. This values corresponds exactly to the exploitability of the optimal strategy combined with the uniform exploration with probability 0.05. The Exp3 variants most likely converge to the same values; however, they do not make it in the first million iterations in WC_DRM. The convergence rate of all the variants is similar and the variants with propagating means always converge a little slower. Therefore, in the following experiments on wider range of game, we only evaluate the variant with propagating the current sample value, and not the mean.

4.2.3 Goofspiel

We follow with evaluation of the proposed algorithms on a set of games inspired by games that people play for fun. We begin with the game of Goofspiel.

Convergence

First, we focus on the speed of convergence of sampling algorithms in small variant of Goofspiel with 5 cards $(0, 1, \ldots, 4)$ We measure the ability to approximate Nash



0.5s	OOS	DUCT	EXP3	RM
OOS	50.1	49.8	48.9	47.6
DUCT	49.2	50.0	48.6	49.5
EXP3	51.7	49.1	49.9	49.5
RM	50.0	50.0	49.6	51.0

Figure 4.6: Convergence and game playing comparison of different algorithms on Goofspiel with 5 cards. All results in the table are within ±2.0 interval with 95% confidence.

equilibrium strategies of the complete game as the sum of exploitabilities of both players' strategies.

Figure 4.6 depicts the results (note the logarithmic horizontal scale). In all convergence experiments in this section, we compare the SM-MCTS algorithm with three different selection functions: DUCT(various C), Exp3($\gamma = 0.1$), RM($\gamma = 0.1$); and SM-OOS($\gamma = 0.6$). We do not explicitly write the SM-prefix further in this section, in order to save space in the figures, but we always mean the simultaneous-move variants of the algorithms. The convergence graphs are means out of 20 runs of each algorithm. Due to the different selection and update functions, the algorithms differ in the number of iterations per second. RM is the fastest with more than 2.6×10^5 iterations per second, OOS has around 2×10^5 iterations, DUCT 1.9×10^5, and Exp3 only 5.4×10^4 iterations per second.

The results show that OOS converges the fastest from all sampling algorithms during the whole time. MCTS with RM selection function is only slightly slower; however, other two selection functions perform worse. While Exp3 eventually converges close to 0, the exploitability of DUCT decreases rather slowly and it was still over 0.35 at the time limit of 300 seconds. The best exploration constant (C) for DUCT was 5 in this setting. While setting lower constant typically improved slightly the convergence during the first iterations, the final error was always larger. Moreover, when we have not used randomization in DUCT, the algorithm was not able to converge to error lower than 0.5 in the time limit.

Game playing

In the matches on the small game, none of the evaluated algorithms performed significantly differently than the other algorithms (see Figure 4.6). We assume that

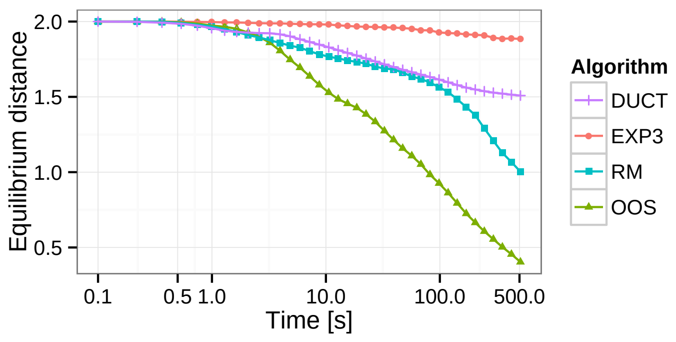
1s	OOS	DUCT	EXP3	RM	RAND
OOS	48.5(3.0)	45.9(3.0)	42.1(2.9)	42.3(2.9)	73.8(2.6)
DUCT	53.9(3.0)	48.3(3.0)	50.6(2.9)	50.7(2.9)	66.3(2.8)
EXP3	57.5(2.9)	48.3(2.9)	51.4(2.4)	42.9(2.6)	66.3(2.8)
RM	**57.8(2.9)**	**49.0(2.9)**	**56.1(2.6)**	**50.0(2.9)**	**66.2(2.8)**
RAND	26.2(2.6)	34.3(2.8)	29.1(2.7)	35.0(2.8)	49.6(3.1)
5s	OOS	DUCT	EXP3	RM	RAND
OOS	51.5(3.0)	48.0(3.0)	41.9(2.9)	47.0(3.0)	70.6(2.8)
DUCT	53.2(3.0)	50.2(3.0)	52.6(2.9)	50.5(3.0)	61.6(3.0)
EXP3	58.5(2.9)	45.6(2.9)	50.6(1.4)	47.5(2.4)	65.4(2.9)
RM	**56.6(3.0)**	**48.0(3.0)**	**55.0(2.4)**	**51.0(2.7)**	**62.4(2.9)**
RAND	28.8(2.8)	38.8(3.0)	30.5(2.8)	38.1(2.9)	51.1(3.1)
* 1s	OOS	DUCT	EXP3	RM	RAND
OOS	48.6(1.4)	47.2(1.3)	44.3(1.3)	42.0(1.3)	80.4(1.0)
DUCT	53.8(1.3)	50.6(1.4)	48.7(1.3)	45.9(1.3)	80.0(1.0)
EXP3	55.7(1.3)	50.6(1.3)	50.3(1.3)	45.2(1.2)	83.5(0.9)
RM	**58.8(1.3)**	**54.7(1.3)**	**54.7(1.2)**	**49.7(1.3)**	**83.6(0.9)**
RAND	19.2(1.0)	19.9(1.0)	16.1(0.9)	16.3(0.9)	49.8(1.4)

Table 4.1: Win-rate (and 95% confidence intervals) in head-to-head matches of Goofspiel(13) with decreasing nature cards with one (top) and five (middle) seconds per move. (bottom) Win-rate in Goofspiel(13) and known random sequences of nature cards.

even though there are significant differences in how quickly the algorithms converge in the whole game, they can all quickly converge to a good strategy in the current state of the game and select a near optimal move.

In the large scale head-to-head comparison, we use Goofspiel with 13 cards, as it is typically played by people. However, for the sake of consistency with the convergence results, we assume a fixed known sequence of the cards of the nature player. In spite of this simplification, the game has still more than 3.8×10^{19} leaf nodes. The win-rates are presented in Table 4.1, where the two top tables show results on an instance of the game with strictly decreasing ordering of nature cards. The bottom table shows the results for 5 random orderings, with 1000 matches played for each ordering. The numbers in the brackets show the 95% confidence interval.

First, we can see that finding a good strategy in Goofspiel is very hard for all algorithms. This is noticeable thanks to the results of random (RAND) player, that performs reasonably well (RAND typically loses almost every match in all the remaining game domains). The differences in the performance of all algorithms are relatively small. Even though OOS converges the fastest (see Figure 4.6), it is losing against all other algorithms with 1 second per move. Its performance improves with 5 seconds per move, but it still does not significantly outperform any other sampling algorithm. With both time settings, RM seems to perform slightly the best. Its performance against other algorithms is very similar to Exp3, but it significantly wins in their mutual matches. This is also seen in the setting across 5 random cards ordering, where RM clearly dominates, which is aligned with our previous results from a completely independent implementation [LLW14].



0.5s	OOS	DUCT	EXP3	RM
OOS	**49.5**	**53.9**	**67.5**	**49.8**
DUCT	46.5	48.9	62.3	44.7
EXP3	32.3	36.0	49.4	33.5
RM	**47.4**	**54.2**	**67.0**	**50.1**

Figure 4.7: Convergence and game playing comparison of different algorithms on Oshi-Zumo game, with 10 coins, $K = 3$, and $M = 1$. All results in the table are within ± 2.1 interval with 95% confidence.

4.2.4 Oshi-Zumo

Convergence

Many instances of the Oshi-Zumo game have Nash equilibria in pure strategies. We show in [BLL$^+$14] that for K equal to 3, the change occurs when the number of coins increase from 11 to 12. Hence, the instance we use for the convergence experiment with 10 coins, K set to 3 and minimum bid set to 1, has an equilibrium in pure strategies. Due to a relatively large branching factor for both players (10 actions at the root node for each player) all sampling algorithms converge slowly. The performance of the algorithms in terms of iterations per second is similar to the previous games, however, OOS is slightly better in this case with 1.9×10^5 iterations per second compared to the second RM with 1.6×10^5 iterations per second. Exp3 is the slowest with 4.8×10^4 iterations per second.

As before, OOS is the best converging algorithm, however, in a given time limit (500 seconds) the reached error was only slightly below 0.5 (0.41). On the other hand, all of the other sampling algorithms perform significantly worse – RM ends with error slightly over 1, DUCT ($C = 2$) with 1.50, and Exp3 with 1.88. This confirms the findings of multi-armed bandit theory (see Section 2.2) and the previous experiments that increasing branching factor slows down the convergence rate. A surprising result is that even in a game with a pure strategy Nash equilibrium, DUCT does not perform very well. This indicates that the ability to model the correct mixed strategy is not the only advantage of OOS. We assume the main reason of better performance is the separate treatment of different player's contributions to the outcome of the iteration.

1s	OOS	DUCT	EXP3	RM	RAND
OOS	53.5(3.0)	52.0(2.1)	56.9(2.0)	31.4(1.9)	97.3(0.7)
DUCT	53.8(2.1)	50.5(3.0)	55.3(2.1)	32.3(1.9)	94.0(1.0)
EXP3	47.3(2.1)	45.1(2.1)	48.9(3.0)	26.0(1.8)	94.2(0.9)
RM	**73.1(1.8)**	**67.8(1.9)**	**75.5(1.8)**	**48.1(3.0)**	**99.0(0.4)**
RAND	3.4(0.7)	5.6(1.0)	5.3(0.9)	0.7(0.3)	48.5(2.9)
5s	OOS	DUCT	EXP3	RM	RAND
OOS	54.9(2.9)	51.3(3.0)	50.0(3.0)	34.9(2.8)	98.2(0.8)
DUCT	54.1(3.0)	51.2(3.0)	51.1(3.0)	35.0(2.8)	97.2(1.0)
EXP3	53.5(3.0)	46.9(3.0)	50.6(2.9)	31.6(2.8)	97.9(0.9)
RM	**72.0(2.6)**	**64.5(2.9)**	**69.2(2.7)**	**51.5(2.9)**	**99.9(0.1)**
RAND	3.2(1.1)	3.6(1.1)	1.6(0.8)	1.0(0.6)	49.6(2.9)
E 1s	OOS	DUCT	EXP3	RM	RAND
OOS	59.5(2.9)	30.9(2.8)	40.0(3.0)	29.4(2.7)	96.6(1.1)
DUCT	**80.9(2.4)**	**46.5(3.0)**	**74.8(2.6)**	**62.0(3.0)**	**97.8(0.9)**
EXP3	63.9(2.9)	25.2(2.6)	52.7(3.0)	43.7(3.0)	96.2(1.2)
RM	80.5(2.4)	43.2(3.0)	56.9(3.0)	48.7(3.0)	99.2(0.5)
RAND	3.0(1.0)	2.6(1.0)	4.6(1.2)	0.5(0.4)	49.0(2.9)
E 5s	OOS	DUCT	EXP3	RM	RAND
OOS	59.4(3.0)	28.2(2.7)	26.6(2.7)	32.8(2.8)	98.4(0.8)
DUCT	**83.6(2.3)**	**48.5(3.0)**	**65.5(2.9)**	**65.8(2.9)**	**99.2(0.6)**
EXP3	77.7(2.5)	32.5(2.8)	48.4(3.0)	64.8(2.9)	99.3(0.5)
RM	77.5(2.5)	32.1(2.8)	35.6(2.9)	44.5(3.0)	99.2(0.5)
RAND	1.6(0.8)	2.0(0.9)	1.8(0.8)	0.6(0.4)	50.5(2.9)

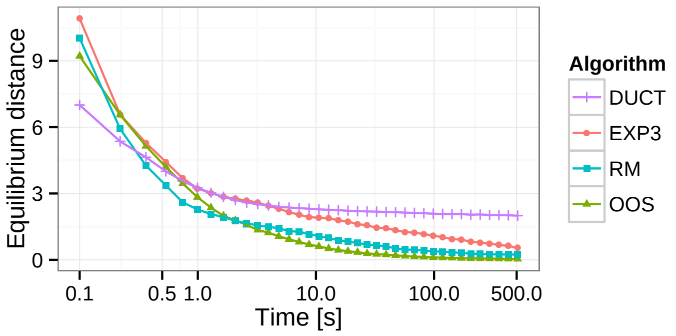
Table 4.2: Win-rate in head-to-head matches of Oshi-Zumo(50,3,1). From top down first 1 and 5 seconds per move without evaluation function, then the same with evaluation function used in sampling algorithms (denoted by "E").

Game playing

The game playing performance with 0.5 seconds of computation time in the small game is presented in Figure 4.7. The results are, to large extend, consistent with the speed of convergence of the algorithms. OOS performs significantly the best, followed closely by RM only based on the mutual matches. DUCT is slightly worse and Exp3 is by far the worst.

In large scale Oshi-Zumo, we use the setting with 50 coins, K=3 fields on each side of the board and minimal bet of 1. We chose this parameters to be consistent with [TLW14] and to have a game with larger branching factor. The size of the game is large with strictly more than 10^{15} leaves (number of choices of a single player in the game; hence, square of this number is the upper bound on the number of leaves).

In the smaller version of the game, none of the algorithms was able to converge anywhere close to the equilibrium in short time (Figure 4.7). In spite of the slow convergence, Table 4.2 shows that all sampling algorithms are able to find reasonably good strategy and strongly win over the random player. With one second per move, RM is clearly the strongest sampling algorithm, having the strongest opponent in DUCT (RM still wins 67.8% of games). OOS and DUCT tie for the second place and Exp3 is the weakest algorithm in this game. With five seconds per move, the differences among the algorithms are a little smaller but the same ranking of the algorithms holds.



0.5s	OOS	DUCT	EXP3	**RM**
OOS	56.0	55.4	51.9	**53.9**
DUCT	54.9	56.7	55.0	**55.2**
EXP3	57.0	53.6	53.9	**52.9**
RM	57.3	55.6	53.9	**52.5**

Figure 4.8: Convergence and game playing comparison of different algorithms on random game with branching factor 4 and depth 5. All results in the table are within ±2.7 interval with 95% confidence.

Because there is a high quality evaluation function available in this game, we performed experiments where the rollout simulation was replaced by the evaluation function. The tables for this variant are labelled with "E" in the tables. The results show that using an evaluation function creates larger differences among the algorithms. DUCT is clearly the best followed by RM with the shorter computation time. Superior performance of DUCT is also obvious in the results for 5 seconds per move. Exp3 is better than RM in this case, even though it performs substantially less iterations.

The main difference between RM, which was the best sampling algorithm without evaluation functions, and the other MCTS algorithms is that RM tries to estimate the quality of each joint action of the players, instead of using a fully decoupled dynamics. These estimates may be misleading when using the evaluation function in Oshi-Zumo, since the evaluation function is not very stable and can change drastically after a single move.

4.2.5 Random Games

Convergence

Figure 4.8 depicts the results for convergence of the sampling algorithms for the random game with correlated utility values, branching factor set to 4 and depth 5. The number of iterations per second is similar to Goofspiel, with Exp3 being the exception. It is able to achieve more than 6.5×10^4 iterations per second, which is still significantly the lowest number of iterations. Interestingly, there is a rather small difference between the performance of the algorithms in this game. Eventually, OOS converges the closest to the equilibrium; however, the performance of RM is very similar and even better during the first second. Again, since the game almost always

1s	OOS	DUCT	EXP3	RM	RAND
OOS	51.1(2.9)	40.2(2.8)	37.1(2.7)	36.0(2.7)	82.8(2.2)
DUCT	57.1(2.9)	52.5(2.9)	46.9(2.8)	47.9(2.8)	85.7(2.1)
EXP3	**65.3(2.7)**	**54.5(2.9)**	**49.2(2.9)**	**51.5(2.9)**	**91.5(1.6)**
RM	**64.8(2.7)**	**55.1(2.9)**	**54.0(2.9)**	**55.0(2.9)**	**87.1(1.9)**
RAND	19.7(2.3)	17.4(2.2)	11.1(1.8)	12.4(1.9)	49.9(2.9)
5s	OOS	DUCT	EXP3	RM	RAND
OOS	50.9(2.9)	40.2(2.8)	35.8(2.8)	36.1(2.8)	83.1(2.2)
DUCT	55.3(2.9)	48.9(2.9)	47.8(2.9)	48.9(2.9)	86.9(2.0)
EXP3	**61.6(2.8)**	**49.9(2.9)**	**47.4(2.8)**	**49.5(2.8)**	**91.0(1.6)**
RM	**59.9(2.8)**	**50.5(2.9)**	**50.9(2.8)**	**48.4(2.8)**	**91.2(1.6)**
RAND	17.1(2.2)	12.2(1.9)	7.6(1.5)	10.1(1.7)	50.1(2.9)

Table 4.3: Win-rate (95% confidence interval) in head-to-head matches of Random games (15,5) with one (top) and five (bottom) seconds per move.

requires mixed strategies, Exp3 outperforms DUCT in the longer run; however, it is the worst sampling algorithm during the first iterations. Exploration constant for DUCT was set to 12 due to larger utility values this setting.

Game playing

In order to compute the win-rates as in other games, we use signum of the utility value defined in Section 4.1. In the small version of the games, the differences between game playing performance of the algorithms are not significant; however, mainly based on the performance in columns, RM performs slightly better than OOS against all opponents besides DUCT, where the difference is very little in favor of OOS. This is consistent with the convergence performance after 0.5 second of computation.

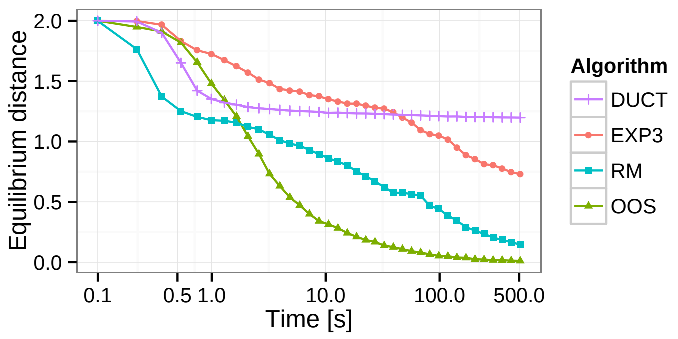
The next set of matches was played on 10 different random games with each player having 5 actions in each stage and depth 15. Hence, the game has more than 9.3×10^{20} leaf nodes. The results are presented in Table 4.3. The performance of all MCTS algorithms is similar as in the small variant of the game. Overall, OOS loses against all MCTS variants. RM and Exp3 are both slightly better than DUCT, with RM performing a little better in the mutual match. The differences between the performance of different algorithms are even smaller with 5 seconds of computation per move. OOS is still the weakest, but the differences between the other algorithms are apparent only against OOS.

We also performed an experiment with larger branching factor (15) and smaller depth (5), but the results were very similar to this setting.

4.2.6 Pursuit-Evasion Games

Convergence

In terms of number of iterations per second, again RM was the fastest and OOS the second fastest with similar performance as in Goofspiel. DUCT achieved slightly less (1.7×10^5 iterations per second), and Exp3 only 2.6×10^4 iterations. The results are depicted in Figure 4.9 for the 4×4 graph and 4 moves for each player (note again the



0.5s	OOS	DUCT	EXP3	RM
OOS	3.9	0.7	13.8	9.0
DUCT	**6.9**	**0.6**	**16.2**	**9.6**
EXP3	4.9	0.4	13.5	7.7
RM	5.1	0.8	11.9	7.8

0.5s	OOS	DUCT	EXP3	RM
OOS	4.0	11.4	9.6	4.4
DUCT	**6.0**	**17.9**	**12.5**	**4.6**
EXP3	4.9	10.7	7.0	3.6
RM	4.0	13.2	9.1	3.3

Figure 4.9: Convergence and game playing comparison of different algorithms on pursuit-evasion game on 4×4 graph, with depth set to 4. The match results are presented for two different starting positions. All results in the tables are within ± 2.4 interval with 95% confidence.

logarithmic horizontal scale). The starting positions were selected such that there does not exist a pure NE strategy in the game. The results again show that OOS is overall the fastest out of all sampling algorithms; however, during the first iterations both DUCT and RM faster reach a smaller exploitability. On the other hand, while OOS is able to keep the convergence rate and reach almost 0 exploitability, DUCT again converges to an exploitable strategy with error 1.16 at best in the time limit of 500 seconds ($C = 2$). The convergence of RM is much better; however, the best final error is 0.14 in the given time. Exp3 converges slowly, but it still outperforms DUCT after 50 seconds.

Game playing

The results on the small variant of the game presented in Figure 4.9 show that many of the games of this size are strongly biased in favor of the second player (pursuer). The two tables present results from two different initial positions of the game and 0.5 second of computation per move. In the first setting, DUCT pursuer was able to capture its opponents in almost all cases. This seems to be a co-incidence, because in the second setting, DUCT was the weakest algorithm on this position. OOS and RM kept more stable performance, which is consistent with lower exploitability. From the evader's position, DUCT wins most often against all other algorithms. Exp3 is consistently among the worst.

For the large version, we compared algorithms on a pursuit-evasion game on an empty 10×10 grid with 15 moves time limit and 10 different randomly selected initial positions of the units. The branching factor is up to 12, causing the number of leaf nodes to be less than 10^{16}. The results in Table 4.4 show that the game is

1s	OOS	DUCT	EXP3	**RM**	RAND
OOS	76.6(3.7)	82.0(0.7)	67.6(0.9)	**58.3(1.0)**	98.1(0.3)
DUCT	93.1(0.5)	91.9(1.7)	86.5(0.7)	**78.0(0.8)**	98.8(0.2)
EXP3	94.2(0.5)	91.7(0.5)	86.6(2.1)	**78.9(0.8)**	99.3(0.2)
RM	94.9(0.4)	93.4(0.5)	90.2(0.6)	**81.3(2.4)**	99.2(0.2)
RAND	30.7(0.9)	38.1(0.9)	8.0(0.5)	**3.9(0.4)**	70.9(2.8)

5s	OOS	DUCT	EXP3	**RM**	RAND
OOS	84.1(2.3)	76.2(2.6)	57.3(3.1)	**58.8(3.1)**	98.5(0.8)
DUCT	93.2(1.6)	85.4(2.2)	79.9(2.5)	**75.3(2.7)**	99.0(0.6)
EXP3	95.0(1.4)	85.9(2.2)	78.8(2.5)	**76.2(2.6)**	99.7(0.3)
RM	94.9(1.4)	87.1(2.1)	78.5(2.5)	**75.2(2.7)**	99.2(0.6)
RAND	28.3(2.8)	30.8(2.9)	1.6(0.8)	**1.5(0.8)**	71.0(2.8)

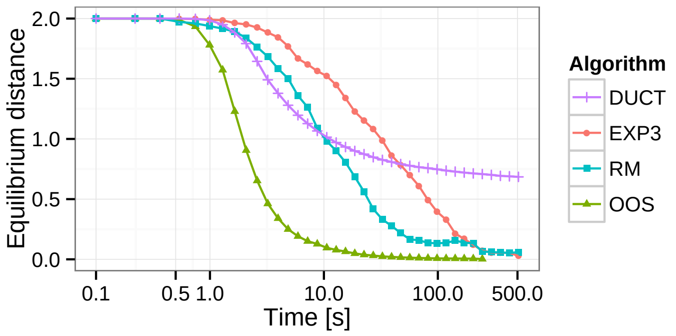
Table 4.4: Win-rate in head-to-head matches of Pursuit evasion game with time limit of 15 moves and 10×10 grid board. One (top) and five (bottom) seconds per move.

strongly biased towards the first player, which is the evader. The self-play results on the diagonal with one second of computation per move show that RM won over 81% matches against itself as the evader. Adding more computation time typically improved the play of the pursuer in self-play. This is caused by more complex optimal strategy of the pursuer. This optimal strategy is more difficult to find due to a larger branching factor (recall that pursuer controls two units) and a higher cost of mistakes. A single move played incorrectly can cause an escape of the evader and can result in losing the game due to the time limit. On both positions, OOS is substantially weaker than the other algorithms, which perform very close to each other when playing for the evader (rows), with RM performing slightly the best. However, if we look at the pursuer's performance in columns, RM is clearly able to pursue the opponents the best. This difference is much more apparent in the shorter time setting. These results are consistent with the convergence results in Figure 4.9, in which in short time, RM converges significantly the fastest.

4.2.7 Tron

Convergence

The size of the game tree in Tron causes slow convergence for all algorithms. This is apparent also in the number of iterations that is slightly lower than in other games. OOS is the fastest performing 1.3×10^5 iterations per second, RM achieves 1.2×10^5 iterations, DUCT only 8×10^4, and Exp3 is again the slowest with 7.8×10^4 iterations per second. Figure 4.10 depicts the results for the grid 5×6. Consistently with the previous results, OOS performs the best and it is able to converge to very close to the exact solution in 300 seconds. Similarly, both RM and Exp3 are again eventually able to converge to a very small error; however, it takes them more time and after 500 seconds, they achieve error 0.05, or 0.02 respectively. Finally, DUCT ($C = 5$) performs reasonably well during the first 10 seconds, where the exploitability is better than both RM and Exp3. This is most likely due to the existence of a pure NE; however, the length of the game tree prohibits DUCT to converge and the best



0.5s	OOS	DUCT	EXP3	RM
OOS	51.2	48.7	53.6	50.2
DUCT	**51.0**	**52.7**	**55.7**	**50.1**
EXP3	47.5	47.0	49.1	47.8
RM	50.1	48.6	53.0	49.2

Figure 4.10: Convergence and game playing comparison of different algorithms on Tron on grid 5×6. All results in the table are within ± 1.8 interval with 95% confidence.

error the algorithm was able to achieve in the time limit was equal to 0.68.

Game playing

Unlike in some of the other games, the matches on the small variant of Tron used for convergence experiments show significant differences in performance of the algorithms (see Figure 4.10). However, these are not consistent with the speed of convergence. The OOS algorithm with the fastest convergence is outperformed by DUCT, which did not manage to converge anywhere close the equilibrium even in 500 seconds. On the other hand, the weakest algorithm is clearly Exp3, which lost significantly to all other algorithms.

The large variant of Tron in our evaluation was played on en empty 13×13 board. The branching factor of this game is up to 4 for each player and its depth is up to 83 moves. Tron is the largest game in our comparison with much more than 10^{21} leaves in the game tree[1]. The results are shown in Table 4.5. RM is a clear winner when the algorithms use the random simulation to evaluate game sates, and this is consistent for both time settings. As before, OOS is the weakest algorithm in a large version of the game. In the setting with 5 seconds, it won only 14.9% matches against RM. Among MCTS algorithms, DUCT performs the worst and loses 60.9% matches against RM.

As in the case of Oshi Zumo, we also run the matches with the evaluation function instead of the random rollout simulation. The usage of the evaluation function improves the performance of OOS, but the relative performance of MCTS algorithms playing against each other does not change very much. Exp3 still outperforms DUCT

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

1s	OOS	DUCT	EXP3	RM	RAND
OOS	50.6(2.8)	23.2(2.2)	18.7(2.1)	15.1(1.9)	96.0(1.0)
DUCT	78.1(2.2)	50.0(2.3)	42.0(2.3)	36.0(2.1)	97.8(0.7)
EXP3	80.9(2.2)	55.0(2.3)	49.6(2.3)	43.5(2.3)	97.8(0.7)
RM	**84.9(2.0)**	**61.6(2.3)**	**54.6(2.3)**	**49.0(2.0)**	**97.7(0.7)**
RAND	5.0(1.2)	3.5(0.8)	1.9(0.6)	2.4(0.7)	51.1(3.2)
5s	OOS	DUCT	EXP3	RM	RAND
OOS	50.3(2.6)	22.8(2.0)	18.3(1.9)	14.9(1.8)	95.8(0.9)
DUCT	77.6(2.1)	50.1(2.1)	42.0(2.1)	36.1(2.0)	97.9(0.6)
EXP3	80.8(2.0)	55.4(2.1)	49.5(2.2)	43.7(2.0)	97.8(0.7)
RM	**85.9(1.6)**	**60.9(1.8)**	**55.6(1.9)**	**49.5(1.7)**	**97.6(0.7)**
RAND	4.6(1.0)	3.5(0.8)	2.1(0.6)	2.4(0.7)	51.4(2.9)
E 1s	OOS	DUCT	EXP3	RM	RAND
OOS	49.8(1.9)	45.0(1.4)	44.2(1.6)	40.9(1.4)	96.1(0.9)
DUCT	56.0(1.8)	50.2(0.5)	44.6(1.1)	43.6(1.2)	97.6(0.7)
EXP3	58.2(1.8)	55.0(1.1)	50.9(0.8)	45.1(1.0)	98.2(0.6)
RM	**62.2(1.7)**	**55.4(1.2)**	**54.4(1.0)**	**49.9(0.4)**	**97.9(0.6)**
RAND	3.3(0.9)	2.2(0.7)	0.8(0.4)	1.7(0.6)	49.8(3.0)
E 5s	OOS	DUCT	EXP3	RM	RAND
OOS	49.8(1.9)	45.0(1.4)	44.2(1.6)	40.9(1.4)	96.1(0.9)
DUCT	56.0(1.8)	50.2(0.5)	44.6(1.1)	43.6(1.2)	97.6(0.7)
EXP3	58.2(1.8)	55.0(1.1)	50.9(0.8)	45.1(1.0)	98.2(0.6)
RM	**62.5(1.6)**	**55.5(1.1)**	**54.6(0.9)**	**50.0(0.4)**	**97.9(0.6)**
RAND	3.3(0.9)	2.2(0.7)	0.8(0.4)	1.7(0.6)	49.8(3.0)

Table 4.5: Win-rate in head-to-head matches of Tron 13. From top down first 1 and 5 seconds per move without evaluation function, then the same with evaluation function used in sampling algorithms (denoted by "E").

in the both time setting. Both Exp3 and DUCT are losing to RM in both time settings regardless of the evaluation function. This makes RM the overall winner in Tron. This is in contrast with the Oshi-Zumo results with the evaluation function, where DUCT performed best. The difference in this case is that the evaluation function provides more consistent game state quality estimates after performing a single move.

4.2.8 Parameter Tuning

In the previous experiments, we kept the exploration parameters of the algorithms fixed to meaningful values with reasonably good performance over all domains. However, the performance of the algorithms can be substantially influenced by setting parameters suitable for a specific domain. In Table 4.6, we present the win-rate of the algorithms with various parameters first against the same algorithm with the default parameter setting and then against a very strong opponent built using the Double Oracle Alpha Beta ($DO\alpha\beta$) algorithm we introduced in [BLL+14]. We chose the Oshi-Zumo game, in which all the presented algorithms perform the worst against $DO\alpha\beta$. Tuning parameters in self-play is the common practice is MCTS research (e.g., [TLW14]). We evaluate also the worst case, because in zero-sum games, the optimal player that approximates the Nash equilibrium in the game should maximize its performance against its strongest opponent. The default exploration parameters

Self-play

Time	OOS					
	0.6	0.5	0.4	0.3	0.2	0.1
1s	53.5(3.0)	50.3(3.1)	54.1(3.1)	58.5(3.1)	55.2(3.1)	57.1(3.1)
5s	54.9(2.9)	53.8(3.0)	56.0(3.0)	59.1(2.9)	60.6(2.9)	59.6(2.9)
	DUCT					
	4	**2**	1.5	1	0.8	0.6
1s	36.2(3.1)	50.5(3.0)	53.6(3.2)	57.8(3.1)	63.0(3.0)	64.7(3.0)
5s	42.2(3.0)	51.2(3.0)	53.4(3.0)	57.4(3.0)	61.0(2.9)	65.8(2.8)
	EXP3					
	0.6	0.5	0.4	0.3	**0.2**	0.1
1s	65.2(2.9)	60.1(3.0)	62.2(3.0)	55.2(3.1)	48.9(3.0)	39.3(3.1)
5s	64.5(1.9)	61.8(2.0)	58.2(2.1)	55.3(2.1)	50.6(2.9)	42.7(2.1)
	RM					
	0.75	0.5	0.3	0.2	**0.1**	0.05
1s	25.4(2.5)	36.7(2.9)	44.2(3.1)	46.8(3.1)	48.1(3.0)	52.8(3.1)
5s	18.1(2.4)	30.0(2.9)	37.6(3.2)	44.9(3.3)	51.5(2.9)	51.4(3.3)

Strongest opponent

Time	OOS					
	0.6	0.5	0.4	0.3	0.2	0.1
1s	12.9(1.4)	15.2(2.2)	14.0(2.1)	15.6(2.2)	15.6(2.2)	16.9(2.3)
5s	10.9(1.4)	9.3(1.8)	11.4(2.0)	10.7(1.9)	14.1(2.1)	13.1(2.1)
	UCT					
	4	**2**	1.5	1	0.8	0.6
1s	7.2(1.6)	11.3(2.0)	13.0(2.1)	13.7(2.1)	12.9(2.0)	11.9(2.0)
5s	8.3(1.7)	11.7(1.4)	7.5(1.6)	7.0(1.6)	7.5(1.6)	8.9(1.8)
	EXP3					
	0.6	0.5	0.4	0.3	**0.2**	0.1
1s	26.1(2.7)	20.1(2.5)	17.9(2.4)	17.5(2.3)	11.8(2.0)	7.2(1.6)
5s	17.4(2.3)	14.2(2.1)	11.3(1.9)	11.7(2.0)	11.0(1.3)	8.1(1.7)
	RM					
	0.75	0.5	0.3	0.2	**0.1**	0.05
1s	21.9(2.5)	25.8(2.7)	21.9(2.6)	17.9(2.4)	16.6(2.3)	18.1(2.4)
5s	15.1(2.5)	14.7(2.7)	10.5(2.2)	9.8(2.2)	14.4(1.5)	12.0(2.3)

Table 4.6: The win-rate of different parameter setting of the algorithms in Oshi-Zumo(50,3,1). First, self-play against the default parameter setting from the head-to-head matches denoted by the bold font. Second, against a strong non-MCTS opponent.

for the algorithms in the complete head-to-head comparison above were hand-tuned and fixed to 0.6 for OOS, 2 for DUCT, 0.2 for Exp3 and 0.1 for RM.

The results show that OOS is least sensitive to the setting of the exploration parameter. Because the game is balanced, the statistically significant winning of the OOS in self-play with the default parameter is just a coincidence. All results corresponding to the default parameter in self-play should be 50%. The results in self-play are consistent over the two time settings: they indicate that OOS could have been better with smaller exploration parameter in this game. The performance of DUCT and Exp3 are more sensitive to modification of the parameter in the evaluated range than OOS. DUCT seems to perform better with smaller exploration parameters and Exp3 with the higher once. RM seems to be very sensitive to the parameter setting and the default parameter seems to be the best.

	Oshi		OshiE		GS		Rand		PE		Tron		TronE	
	$\bar{d}$	Its.	$\bar{d}$	Its.	$\bar{d}$	Its.	$\bar{d}$	Its.	$\bar{d}$	Its.	$\bar{d}$	Its.	$\bar{d}$	Its.
OOS	5.5	187k	5.8	349k	11.2	73k	9.7	93k	8.1	28k	17.7	77k	17.4	265k
RM	7.1	186k	7.6	244k	17.8	92k	12.3	108k	11.0	34k	26.3	75k	25.1	251k
DUCT	4.0	134k	6.5	213k	9.6	74k	8.0	94k	7.0	30k	15.1	72k	16.7	279k
Exp3	3.3	14k	3.7	12k	12.0	50k	8.4	95k	7.6	32k	17.5	80k	18.3	275k

Table 4.7: The number of iterations (in thousands) and the mean depth($\bar{d}$) of the tree constructed by the sampling algorithms in 5 seconds in the root node of the large version of the games.

The situation is different in the tuning against the strong opponent. More time makes the opponent even stronger and the difference between parameter settings smaller. Different parameter settings often do not have statistically significant influence on the performance. The performance of DUCT can likely be slightly improved by setting the parameter to 1.5 instead of 2 for the shorter time setting, but our default value seems to be optimal for the longer time setting. Setting much higher exploration for Exp3 can significantly improve the performance of the algorithm against $DO\alpha\beta$ with one second per move, but the influence of the parameter is weaker with the longer time setting. Similar behavior is apparent also for RM. The algorithm performs better against $DO\alpha\beta$ with higher exploration factor; however, with 1 second in a self-play, RM with exploration factor 0.5 significantly loses against RM with exploration factor 0.1 (i.e., wins only 36.7($\pm$2.9)% of matches).

The conclusion from this parameter tuning is that it can have a significant impact, but it is hard to achieve consistent results among different time setting and different opponents.

4.2.9 Discussion

Several conclusions can be made from the experiments presented above. First, OOS converges significantly the fastest to the exact equilibrium of the game. This was observed consistently in all games. This is most likely caused by the algorithm using the extra information about the samples in the form of the reach probabilities. All the other algorithms use only the value of the sampled terminal node. In the long run, DUCT converges the slowest and it did not manage to converge close to an equilibrium in a reasonable time in any of the other evaluated games.

Second, the fast convergence and low exploitability of OOS in the smaller variants of the games is not a very good predictor of its performance in the online setting. While the matches in the small games were not very conclusive, OOS was consistently one of the worst performing algorithms in the large versions of the games. One possible reason could be that with higher exploration parameter required for faster convergence, the part of the game tree constructed by OOS is too wide and shallow, and does not allow it to find important situations later in the game. However, Table 4.7 shows that while OOS builds a substantially less deep tree than RM, it is often deeper than for other MCTS algorithms, which outperformed OOS in matches. If we set the exploration parameter to smaller numbers, and also deeper in the tree, the importance sampling correction of regret updates causes very high variance of their values, which generally slows-down the convergence to good solutions [GLB+12] and may result in building the tree in less useful directions.

Third, when the sampling algorithms are allowed to use the evaluation functions, their relative performance can change substantially. All MCTS variants, and DUCT in particular, benefit a lot form using evaluation function in Oshi-Zumo. With evaluation function, all these algorithms win over OOS significantly. On the other hand, the situation in Tron is the opposite and OOS seems to benefit from the evaluation function much more than the MCTS algorithms.

Fourth, when the algorithms have more time for finding the move to play, the differences between win-rates of the algorithms are getting smaller. Longer thinking time has also the same effect on different parameters of the algorithms.

Fifth, Exp3 in the form we use is usable only in games with smaller branching factors. The cost of computing the quadratic number of exponential functions in each state causes an order of magnitude smaller number of iterations in Oshi-Zumo, compared to all other sampling algorithms (see Table 4.7).

Finally, RM seems to be consistently the best performing algorithm in game playing for this class of games. Even with similar number of iterations, it builds a substantially deeper tree (see Table 4.7), which indicates the algorithm is able to better target the search to important parts of the tree. Out of all the large games and both time settings, it loses only in Oshi-Zumo with evaluation function, where it is outperformed by DUCT. DUCT seems to be also good in matches in the small games with very short time setting.

4.3 Evaluation on Imperfect-Information Games

In this section, we present similar experiments with the speed of convergence and game playing quality in artificial games with the algorithms for fully imperfect-information games. Here, we focus on the algorithms that work on full extensive-form game representation, because they do not require a domain-specific evaluation function. We evaluate the remaining algorithms in the domain-specific case study in Chapter 5. The default exploration constants for this section were selected based on initial hand-tuning for UCT as $C = 2 * v_{max}$, Exp3 as $\gamma = 0.1$, RM as $\gamma = 0.1$, for OOS exploration constant we use $\gamma = 0.4$ and for ϵ-regret matching in OOS we use $\epsilon = 0.01$.

In the following we first discuss the methodology we use during the experiments, then we present the results on each game separately. In the last section, we summarize the results through all games and draw conclusions.

4.3.1 Evaluating Online Game-Playing Algorithms

First we discuss methodology of evaluating the quality of online game playing algorithms. In generic imperfect-information games, it is a substantially more difficult problem than in simultaneous-move games evaluated in Section 4.2. The main difference is that the situation in the root of the game is completely different from the situations later during the match. During the match, the player knows a set of possible game states the game may be in and he might have some belief distribution estimating the probability that each of the states is the real state of the game. In the root, both players know that the game is in the unique initial state and they do not have to estimate any probabilities. It is possible that strategy quality measures of the algorithm started from the root do not correspond to the same measures when the algorithm is run in an inner information set of the game.

For these reasons, besides the exploitability in the root and the game playing performance in tournaments, we propose additional evaluation methods. They are not dependent on a specific set of opponents as the tournaments and they are more representative of the actual game playing quality than the root exploitability. If we have unlimited computational resources or a very small game, the problem is not difficult. The method we termed *full stitching* runs the search from the root of the game to each information set of a player and stores the strategy computed in the information set to an external "stitched" strategy. After visiting each information set sufficient number of times, the value of the best response to the stitched strategy is exactly the expected value of the online game playing algorithm facing its worst possible opponent.

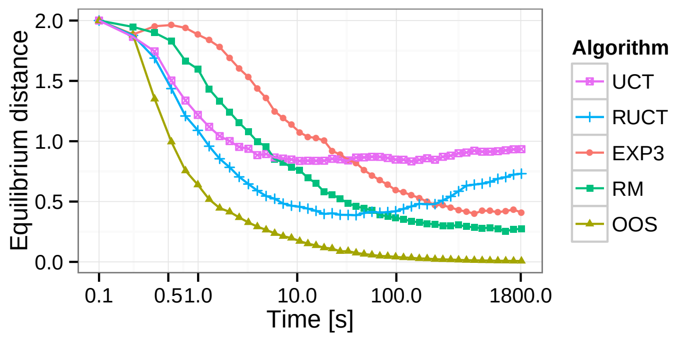
With larger games and limited computation resources, we propose the multi-match *aggregate method* for creating an approximation of the full stitched strategy. We run a larger number of matches of the evaluated algorithm against a random opponent. After each computation of the algorithm, we collect the current strategy in all information sets that can still be reached in the game and add it to the stitched strategy. In order to give more weight to the strategies in the information sets that are evaluated by the algorithm more precisely, we weight the current strategies in the information sets by the number of MCTS samples that visited the information set. After all the matches are played, we normalize the strategy in each information set of the stitched strategy and compute its exploitability.

With really large games, even computing exploitability or explicitly constructing the stitched strategy is not possible. In these cases, we do not know of any better measure of strategy quality than the performance in a tournament.

4.3.2 Imperfect-Information Goofspiel

Convergence

First, we focus on the speed of convergence in a small variant of Imperfect-Information Goofspiel with 6 cards $(0, 1, \ldots, 5)$. We measure the ability of the algorithms to approximate Nash equilibrium strategies of the complete game, as before, by the sum of exploitabilities of both players' strategies. Figure 4.11 depicts the results (note the logarithmic horizontal scale). We compare the IS-MCTS algorithm with four different selection functions (UCT, Randomized UCT, Exp3, and RM), and OOS. UCT denotes the basic variant of UCT always selecting the first action with identical value, Randomized UCT (RUCT) is the same algorithm that breaks ties among the identically-valued actions randomly. The top figure presents the exploitability of the algorithms run from the root state for 30 minutes. The results are means of 20 runs of each algorithm. Due to the different selection and update functions, the algorithms differ in the number of iterations per second. UCT and RM are the fastest with more than 6.4×10^4 and 5.9×10^4 iterations per second, RUCT has around



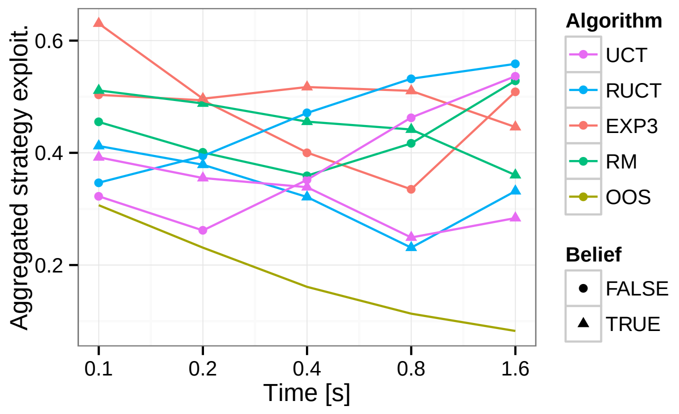


Figure 4.11: Convergence in the game root (top) and overall during the game computed by the aggregate method (bottom) on Imperfect Information version of Goofspiel with 6 cards.

3.9×10^4, EXP3 has around 3.4×10^4 iterations, and OOS 3.3×10^4 iterations per second. The results show that OOS converges the fastest from all sampling algorithms during the whole time. The exploitability of RUCT starts decreasing fairly quickly, but after approximately 20 seconds of computation, it starts increasing again. A similar behavior can be seen also in the convergence of UCT, but it starts diverging sooner and does not reach as good values as RUCT. The lowest error of UCT is 0.84. The variants of the IS-MCTS algorithm with Exp3 and RM converge slowly at the beginning, but eventually achieve smaller error than the UCT variants. After 500 seconds of computation, the error of Exp3 is 0.41 and the error of RM is 0.27. These results confirm that even in the fully imperfect-information games, using randomization in UCT has large positive effect and that OOS is the only one from the algorithms that converges to near exact equilibrium in a reasonable time.

The lower graph in Figure 4.11 presents the overall exploitability of the algorithms in the whole match computed by the aggregate method introduced in Section 4.3.1. Each data point is computed after aggregating the strategy from 500 matches. Note that the horizontal scale is linear in this case and shows substantially shorter times. The trends are consistent with the exploitability in the root. The only algorithm that consistently converges to the equilibrium is OOS, with exploitabil-

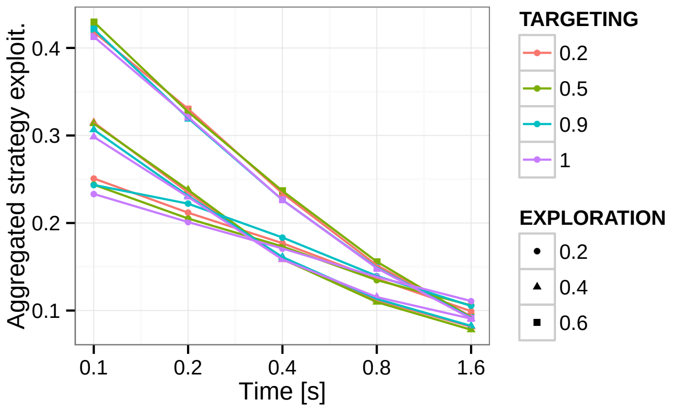


Figure 4.12: Influence of OOS parameters to convergence of aggregated exploitability in Imperfect-Information Goofspiel with 6 cards.

ity below 0.1 with 1.6 second of computation per move. Even with 0.1 second of computation, OOS is still the closest to the equilibrium. The variants of IS-MCTS are depicted both with and without approximating the belief distribution in the current information set during a match. The exploitability of all variants behaves the same way. Without approximating the belief (circles), the exploitability with short computation times is generally lower than with estimating the beliefs (triangles). Imprecise belief estimations are worse than uniform random. With increasing computation time per move, all variants without belief approximation first converge closer to NE, but at some point start to drift further from the equilibrium, eventually reaching highly exploitable strategies. The IS-MCTS variants approximating beliefs generally start with more exploitable strategies in the short times, but then converge to the equilibrium more consistently for a longer time. For UCT and RUCT, which do not converge to the right solution even in matrix games [SSS09, BLL^{+}14], the exploitability eventually starts growing again even when modelling beliefs. However, with larger computation times, the variants modelling beliefs are clearly all less exploitable. The lowest exploitability achieved by IS-MCTS (0.23) is with RUCT selection and modelling belief with 0.8 second of computation per move.

In Figure 4.12, we investigate the influence of the parameters of OOS to its convergence. The amount of exploration has a stronger effect to speed of convergence than the amount of targeting, which indicates that the game is not very misleading in the estimates of the quality of the moves. Additional exploration slows down the convergence mainly with very short computation times. In the longer run, exploration set to 0.4 seems to perform the best in this game. With short computation times, OOS consistently finds better strategies using stronger targeting. However, in the longer time the order of the lines switches and at 1.6 second per move, the strongest targeting is already slightly the worst and the best performance is reached by targeting with probability 0.5. This result was expected, because with very short computation times, it is important to ensure sufficient number of samples in the current information set of the game and the additional time can be useful to converge

Without modelling belief

0.1s	OOS	RUCT	UCT	EXP3	RM	RAND
OOS	**51.4(2.3)**	57.7(2.8)	60.8(2.7)	54.0(2.3)	61.0(2.6)	81.5(2.3)
RUCT	**51.2(2.2)**	62.9(2.6)	65.4(2.5)	55.6(2.1)	62.7(2.5)	84.0(2.1)
UCT	**50.6(1.9)**	64.8(2.7)	67.5(2.4)	60.2(1.8)	67.2(2.3)	85.8(2.0)
EXP3	**52.9(1.8)**	74.5(2.2)	71.2(2.3)	62.8(1.6)	74.8(2.0)	88.0(1.9)
RM	**52.3(1.4)**	70.6(1.7)	71.0(2.3)	60.0(1.6)	73.1(2.1)	87.8(1.4)
RAND	**17.1(2.2)**	15.7(2.2)	15.0(2.1)	8.2(1.6)	10.3(1.8)	49.7(3.0)

Without modelling belief

1s	OOS	RUCT	UCT	EXP3	RM	RAND
OOS	**52.1(2.3)**	59.6(2.7)	58.9(2.8)	54.0(2.3)	59.0(2.6)	82.8(2.2)
RUCT	**52.0(2.2)**	61.5(2.7)	68.8(2.4)	54.0(2.1)	64.5(2.5)	84.2(2.1)
UCT	**51.6(2.0)**	64.2(2.6)	67.0(2.5)	59.0(1.9)	67.7(2.3)	84.0(2.2)
EXP3	**53.0(1.8)**	74.5(2.2)	74.7(2.1)	61.8(1.5)	75.7(2.0)	87.8(1.9)
RM	**52.1(1.9)**	72.8(2.3)	70.5(2.4)	59.2(1.7)	75.2(2.1)	88.8(1.9)
RAND	**19.1(2.3)**	14.9(2.1)	14.5(2.1)	8.7(1.7)	10.6(1.8)	48.7(3.0)

With modelling belief

1s	OOS	RUCT	UCT	EXP3	RM	RAND
OOS	51.6(2.3)	50.9(2.2)	48.6(1.9)	47.9(1.9)	49.0(2.0)	83.0(2.2)
RUCT	49.0(2.2)	47.9(2.1)	49.0(1.5)	49.3(1.7)	50.1(1.7)	82.9(2.2)
UCT	50.5(2.0)	47.7(2.1)	48.2(1.4)	50.3(1.2)	51.5(1.5)	85.8(2.0)
EXP3	53.7(1.8)	51.0(1.7)	50.6(0.8)	49.9(0.3)	50.4(1.0)	89.8(1.8)
RM	52.0(1.9)	49.6(1.8)	49.5(1.1)	50.7(0.9)	50.5(1.3)	89.2(1.8)
RAND	20.1(2.4)	14.1(2.0)	9.3(1.7)	7.2(1.5)	10.4(1.8)	49.4(3.0)

Table 4.8: Win-rates of the row player against different algorithms on Imperfect Information version of Goofspiel with 6 cards. First without modelling beliefs with 0.1 (top) and 1 (middle) second of computation per move. The last table is with modelling beliefs and 1 second per move. The number in brackets indicates 95% confidence interval.

more precisely to the equilibrium, which is ensured only with weaker targeting.

Game playing

After analyzing convergence of the strategies computed by the sampling algorithms, we evaluate how this property translates to actual performance of the algorithms in head to head matches. We first evaluate the small game used for computation of exploitability from the previous section to see the connection to the exploitabilities and then we focus on a substantially larger game to evaluate practical applicability.

Table 4.8 presents the win-rates of the algorithms in mutual matches in Imperfect-Information Goofspiel with 6 cards. The first two tables evaluate the algorithms without modelling beliefs. The table on the top presents the results with 0.1 second per move and the table in the middle with 1 second per move, but they are very similar. The first important observation is that even though the game is symmetric and the first player does not have any advantage, all IS-MCTS variants perform much better as the first player (rows). The reason is the asymmetry of the game model in the form of EFG. Even though in reality the players choose an action simultaneously, the game models this fact as a sequential decision with hidden information. As a result, the second player has substantially larger information sets than the first player (see Figure 4.1). If the search is run from a larger information set, it is more

Without modelling belief

1s	OOS	RUCT	UCT	EXP3	RM	RAND
OOS	49.5(3.1)	25.4(2.7)	23.8(2.6)	20.0(2.4)	**21.2(2.5)**	72.8(2.7)
RUCT	77.2(2.5)	70.0(2.8)	69.7(2.8)	67.7(2.9)	**61.0(3.0)**	90.6(1.8)
UCT	75.8(2.6)	67.4(2.9)	71.2(2.8)	66.3(2.9)	**63.9(3.0)**	92.2(1.6)
EXP3	**83.3(2.3)**	**79.5(2.5)**	**78.2(2.5)**	**63.2(2.8)**	**66.4(2.9)**	**95.6(1.2)**
RM	79.8(2.4)	73.8(2.7)	71.2(2.8)	68.2(2.8)	**67.5(2.9)**	92.8(1.5)
RAND	24.4(2.6)	6.2(1.5)	5.1(1.3)	4.0(1.2)	**4.9(1.3)**	49.0(3.0)

With modelling belief

1s	OOS	RUCT	UCT	EXP3	RM	RAND
OOS	49.0(3.0)	20.1(2.5)	19.4(2.4)	16.0(2.3)	20.1(2.5)	71.5(2.7)
RUCT	**76.8(2.6)**	**62.9(3.0)**	**65.9(2.9)**	**68.0(2.8)**	**64.9(2.9)**	**89.9(1.8)**
UCT	**78.9(2.5)**	**63.2(3.0)**	**65.5(2.9)**	**69.3(2.8)**	**68.3(2.9)**	**90.2(1.8)**
EXP3	84.5(2.2)	52.6(3.0)	51.5(3.0)	61.0(2.5)	59.1(2.8)	94.7(1.3)
RM	80.8(2.4)	59.9(3.0)	59.1(3.0)	63.5(2.7)	66.2(2.8)	94.2(1.4)
RAND	23.2(2.6)	5.5(1.4)	6.0(1.4)	3.5(1.1)	5.4(1.4)	50.7(3.1)

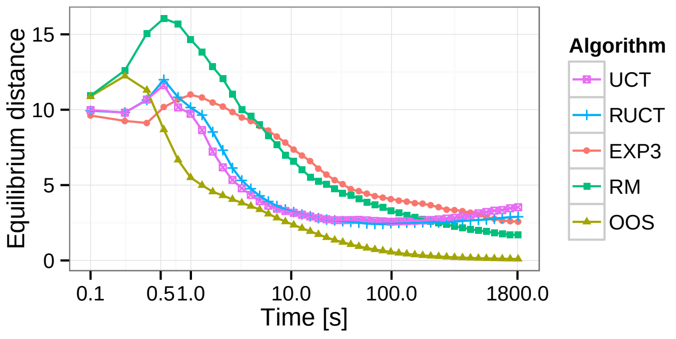
Table 4.9: Win-rate of different algorithms on Imperfect Information version of Goofspiel with 13 cards. The number in brackets indicates 95% confidence interval.

important to have a good approximation of the current belief. The second player is at a large disadvantage here. Recall that IS-MCTS without modelling belief just assumes uniform belief here. This is not a problem in the purely simultaneous moves, as the search is always started from a singleton information set there.

OOS has a much more stable performance. In the first position (row), it is able to win against the weaker opponents playing from the second position. For example, it wins 61% of matches against RM with 0.1 second per move. When OOS is playing on the second position, it is clearly the best algorithm. Only Exp3 and RM are able to win significantly from the first position, but it is only in 53% and 52% matches. Among the IS-MCTS variants, Exp3 wins the most from the first position and loses the least from the second position. This is a surprising result, as Exp3 has the slowest convergence in the root as well as with the aggregated method. Apparently, it quickly reaches a strategy that is exploitable, but performs well against the other opponents in the tournament.

While the added time does not change the table significantly, adding modelling of beliefs to the IS-MCTS algorithm does. The last part of Table 4.8 presents these results. None of the algorithms can exploit any other algorithm and besides the matches with the random player, no result is significantly different from a draw after 1000 matches.

The game playing performance of the algorithms in the large game of Imperfect-Information Goofspiel with 13 cards and one second of computation per move is presented in Table 4.9. In the large game, the performance of IS-MCTS variants is similar to the performance on the smaller version. Exp3 is winning significantly the most against UCR and RUCT, but RM loses the least from the second position. OOS, however, performs poorly on the large game. It wins around 20% to 25% matches against IS-MCTS with various selection functions and loses the most from the second position. Modelling beliefs in the large game helps RUCT, UCT and Exp3 playing for the second player against OOS and does not change the results against the random player significantly. The mutual win-rate are generally closed to



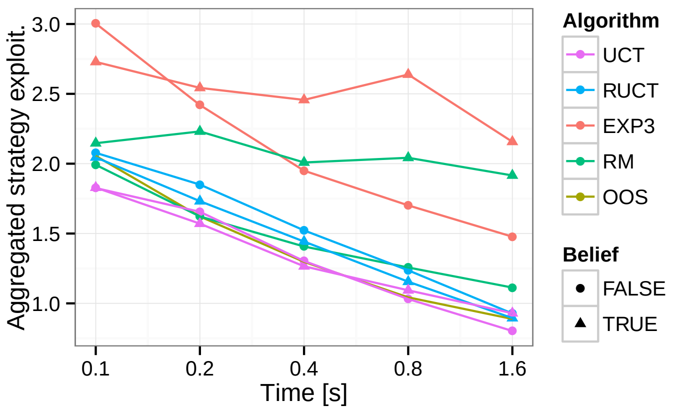


Figure 4.13: Convergence in the game root (top) and overall during the game computed by the aggregate method (bottom) on Generic Poker with 2 bet sizes, at most 2 rounds of rising and 3 cards of each of 3 types.

50%, which indicates that the players play closer to the equilibrium. Exp3 performs the best against OOS, but does not win against UCT and RUCT from the first position. It significantly loses against these opponents from the second position. Among the IS-MCTS variants, UCT and RUCT are winning the most from the first position and losing the least from the second position with the beliefs.

4.3.3 Generic Poker Games

Convergence

Next we analyze the performance of the algorithms in a small Poker game with 2 bet sizes, at most 2 rounds of rising, and 3 cards of each of 3 types. The differences in the number of iterations per second of the algorithms run from the root of the game are similar to Goofspiel. UCT and RM are the fastest with 1.2×10^5 iterations per second, followed by RUCT with 8.2×10^4, OOS with 7.7×10^4 and Exp3 with 6.8×10^4 iterations per second.

The graph on the top of Figure 4.13 presents the exploitability of the algorithms run from the root state for 30 minutes. All of the algorithms have a similar convergence trend in this game. At first, the exploitability of the strategies increases and

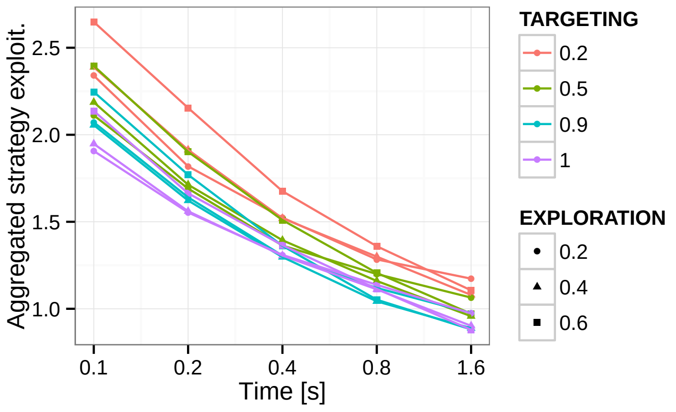


Figure 4.14: Influence of OOS parameters to convergence of aggregated exploitability in Generic Poker with 2 bet sizes, at most 2 rounds of rising and 3 cards of each of 3 types.

it mostly decreases after more than 1 second of computation. With more than 0.5 second of computation, OOS already produces clearly the least exploitable strategies, eventually converging close to the optimum. UCT and RUCT perform almost the same, indicating that the randomization is not helpful in this game. Both these algorithm converge quickly to a reasonably good strategy, but are unable to further improve and slowly make the strategies worse. RM and Exp3 converge slower, but as before, they eventually converge to better strategies.

The results are substantially different for overall in-game exploitability estimated by the aggregated method presented in the bottom of Figure 4.13. The only substantially worse algorithm is Exp3 and RM with estimation of believes. All remaining variants have similar exploitability, indicating that in games with special structure, even UCT and RUCT can converge very close to NE. Even though the algorithms do not converge well deeper in the tree when run from the root, once the relevant information set is reached, the computed strategy is not very exploitable. Furthermore, uniform belief effectively used when the algorithm is not modelling beliefs is a useful approximation of the actual beliefs in this game. The results also show that estimating beliefs generally reduces performance of IS-MCTS with RM and Exp3 evaluation functions in this game. The explanation of this phenomenon can be based on the convergence in the root. The convergence starts from a uniform random strategy. In the first second, the computed strategy for these selection functions gets substantially worse. This means that the uniform belief estimation resulting from the uniform strategy is actually more exact than using the strategy after short computation time.

Figure 4.14 presents the influence of the parameters of OOS to its convergence. In this game, targeting has slightly larger influence on the convergence than exploration, which the opposite of what happens in Goofspiel. The main reason is probably that the important decisions in Goofspiel are made mainly in the beginning of the game. The default ordering of the nature cards in the game are from the highest to the lowest card. On poker, the most important decisions are likely As in the case of Imperfect-Information Goofspiel, more targeting is clearly helpful with short computation times. However, even with large computation times, targeting with probability 1 and 0.9 performs the best. This is likely caused by the special structure of the game that does not include the pathologies that might cause stronger targeting to converge to an incorrect strategy. All actions are public and once they are performed, both players have common knowledge of this fact. This may reduce the negative effects of too strong targeting.

Game playing

The head to head performance results are much harder to interpret, because of much higher variance and the resulting larger confidence intervals. The result of this game is not only win or loss, but a utility value from $\langle -37, +37 \rangle$ in the small and $\langle -121, +121 \rangle$ in the large variant. Even after 1000 matches played for each presented value, the 95% confidence interval is often larger than the actual result. Moreover, the game is not balanced and the first player has a disadvantage of approximately -0.11 in both game size we use in this section. We computed these game values by our exact Double-Oracle Sequence-Form algorithm [BKL$^+$13, BKLP14]. In order to make the results in this section easier to interpret, we subtract the game value form the results. Hence, the numbers represent how much more or less utility the players gain by the quality of their play, as if the game was balanced.

Table 4.10 presents the results from the smaller Generic Poker game with 2 bet sizes, at most 2 rounds of rising and 3 cards of each of 3 types. All the proposed algorithms are able to significantly win over the random player. Most of the results are not statistically significant with 95% confidence, so we highlight those that are significant. With 0.1 second per move, the algorithm that is significantly winning is RM. It was able to significantly win over RUCT and Exp3 in at least one position. RUCT won over UCT at the edge of the confidence interval, but also lost against Exp3 with the same confidence. With one second per move, the results are similar. The only statistically significant results with non-random players are UCT winning over Exp3, Exp3 losing to RM and RM winning over RUCT and UCT. The last table shows the influence of modelling beliefs by the players on the second position. Positions are balanced with respect to the sizes of information sets in this game (see Figure 4.3). The results are again very similar, with little significant results. Adding modeling of beliefs slightly increased variance in the game, which made the only significant result to be the loss of Exp3 against itself form the second position. This is consistent with the aggregate exploitability results in Figure 4.13 showing that modelling beliefs reduces performance of this algorithm.

For the experiment with a larger poker game presented in Table 4.11, we used Generic Poker with 4 bet sizes, at most 4 rounds of rising and 4 cards of each of 6 types. Even though we have removed the natural bias of the game, all algorithms perform consistently better as the second player. A possible explanation of this phenomenon is that the real advantage of the second player is rather small if both players play optimally, but it is much more difficult to come-up with a good strategy for the first player. In the large game, OOS performs the worst. It loses more than

Without beliefs

0.1s	OOS	RUCT	UCT	EXP3	RM	RAND
OOS	0.10(0.23)	-0.07(0.21)	0.13(0.18)	-0.00(0.19)	0.04(0.16)	2.13(0.51)
RUCT	0.19(0.20)	-0.01(0.16)	**0.15(0.15)**	**-0.18(0.18)**	-0.14(0.15)	1.98(0.42)
UCT	0.16(0.18)	0.02(0.18)	0.01(0.23)	0.16(0.23)	-0.19(0.21)	1.66(0.41)
EXP3	0.01(0.37)	0.10(0.34)	0.15(0.32)	0.03(0.30)	**-0.36(0.31)**	2.37(0.48)
RM	0.25(0.26)	**0.38(0.25)**	0.14(0.22)	**0.49(0.25)**	-0.08(0.24)	1.55(0.42)
RAND	-2.13(0.57)	-1.60(0.48)	-1.69(0.45)	-2.23(0.50)	-2.37(0.52)	0.92(0.63)

Without beliefs

1s	OOS	RUCT	UCT	EXP3	RM	RAND
OOS	0.19(0.32)	0.20(0.26)	0.03(0.24)	0.10(0.26)	0.04(0.23)	2.15(0.48)
RUCT	-0.02(0.31)	-0.12(0.23)	0.10(0.22)	0.03(0.24)	-0.20(0.25)	1.37(0.42)
UCT	0.08(0.22)	0.08(0.23)	0.10(0.21)	**0.27(0.22)**	0.04(0.21)	1.85(0.41)
EXP3	-0.09(0.33)	-0.10(0.28)	-0.05(0.29)	0.32(0.33)	**-0.39(0.31)**	2.62(0.47)
RM	0.21(0.24)	**0.37(0.25)**	**0.22(0.22)**	0.13(0.28)	-0.04(0.21)	1.82(0.44)
RAND	-2.34(0.56)	-2.37(0.51)	-1.54(0.46)	-1.80(0.50)	-1.99(0.48)	0.95(0.63)

With beliefs for column players only

1s	RUCT	UCT	EXP3	RM
OOS	0.02(0.27)	-0.05(0.32)	0.06(0.28)	0.13(0.29)
RUCT	-0.03(0.24)	-0.01(0.24)	0.02(0.27)	0.00(0.31)
UCT	0.10(0.23)	0.11(0.26)	0.14(0.25)	0.22(0.24)
EXP3	-0.23(0.33)	0.17(0.36)	**-0.36(0.32)**	0.16(0.33)
RM	0.27(0.29)	0.28(0.29)	0.07(0.27)	-0.07(0.27)
RAND	-2.19(0.48)	-1.81(0.52)	-1.76(0.51)	-1.78(0.53)

Table 4.10: The difference of the average reward of different algorithms and the game value in head to head matches in Generic Poker with 2 bet sizes, at most 2 rounds of rising and 3 cards of each of 3 types. Players have 0.1 (top) and 1 second per move. The number in brackets indicates 95% confidence interval.

the other algorithms as the first player and it is the only algorithm that manages to lose also as the second player. Otherwise, the results without using beliefs are not very consistent. RM as the first player significantly wins over OOS, but loses to RUCT and UCT. RUCT and UCT non-significantly win against OOS, RUCT loses to itself and UCT even more than RM. UCT performs best against itself and RUCT, but seems to lose more than RM and RUCT to Exp3 and RM.

Modeling beliefs does not make the results much more clear. OOS is still the worst and seems to perform slightly better than before. The reason might be that the MCTS algorithms with modeling beliefs play less exploitative strategies. Exp3 also still performs badly, with overall a little less extreme values. The matches among the remaining three algorithms are inconclusive, even with modeling beliefs.

4.3.4 Phantom Tic-Tac-Toe

Convergence

The game of Phantom Tic-Tac-Toe has approximately 10^{10} terminal states, which makes it difficult to compute exploitability of strategies in this game. Therefore, we initially focus on a simpler version of the game with first move enforced to be to the field in the center of the board. The first player has only this action available, the second player can also first play only this action, which reveals the position of

Without beliefs

1s	OOS	RUCT	UCT	EXP3	RM	RAND
OOS	-2.56(1.47)	-3.64(1.33)	-3.79(1.27)	-2.19(1.24)	-2.24(1.13)	10.21(1.72)
RUCT	0.76(1.14)	-1.64(0.98)	-1.36(0.84)	-1.02(0.95)	-0.44(0.80)	8.77(1.48)
UCT	0.74(1.03)	-0.60(0.95)	-0.83(0.92)	-1.38(0.88)	-0.77(0.70)	8.63(1.46)
EXP3	-1.21(1.32)	-2.96(1.10)	-3.54(1.08)	-2.59(0.94)	-3.34(0.94)	11.26(1.55)
RM	1.26(1.19)	-1.23(0.88)	-1.02(0.95)	-0.66(0.83)	-0.25(0.75)	8.35(1.39)
RAND	-4.30(1.27)	-4.03(1.14)	-5.61(1.17)	-4.75(1.19)	-5.56(1.07)	3.77(1.68)

With beliefs

1s	OOS	RUCT	UCT	EXP3	RM	RAND
OOS	-0.98(1.45)	-3.44(1.27)	-3.05(1.23)	-3.90(1.28)	-1.86(1.15)	10.08(1.83)
RUCT	-0.38(1.08)	-0.44(0.93)	-1.54(0.89)	-0.17(0.88)	-1.28(0.79)	8.64(1.40)
UCT	0.17(1.02)	-1.35(0.94)	-1.06(0.86)	-0.41(0.79)	-0.60(0.69)	7.00(1.37)
EXP3	-0.46(1.30)	-3.31(1.11)	-2.29(1.09)	-1.56(0.90)	-1.83(0.86)	11.12(1.62)
RM	1.07(1.10)	-0.72(0.91)	-1.17(0.89)	-0.74(0.81)	-1.41(0.70)	7.83(1.43)
RAND	-4.34(1.23)	-4.23(1.05)	-5.00(1.09)	-4.38(1.10)	-3.48(0.99)	1.30(1.66)

Table 4.11: The difference of the average reward of different algorithms and the game value in head to head matches in Generic Poker with 4 bet sizes, at most 4 rounds of rising and 4 cards of each of 6 types. The number in brackets indicates 95% confidence interval.

the first player's mark and allow to second player to move again. The second move of the second player and all the following moves can then be any legal moves in Phantom Tic-Tac-Toe.

The number of iterations per second in this restricted game is similar to the previous two games. This time, RM makes the most iteration (9.5×10^4), likely because the game has a higher number of applicable actions in many states and RM uses simplest functions to compute the probability of selecting each action. Little less iterations is performed by UCT (8.7×10^4) and OOS (7.6×10^4). RUCT performs significantly less iterations than UCT (5.1×10^4), which is probably caused by actions often having very similar values. RUCT needs to iterate through the actions multiple times to select an action. Exp3 is again the slowest with 3.4×10^4 iterations per second.

The convergence of the algorithms in the root of the game presented in Figure 4.15 shows trends similar to the case of Generic poker. OOS quickly and consistently converges to the Nash equilibrium of the game. UCT and RUCT both first quickly reduce the exploitability of the strategy and then gradually make the strategy more exploitable after more than 100 seconds of computation. Approximately at this point, initially the worst RM becomes the second best algorithm and after the full 30 minutes, it converges to the exploitability of 0.13. Exp3 is clearly the worst algorithm between the first and the hundredth second, possibly because of the slower convergence.

We do not present the results for the aggregate exploitability on Phantom Tic-Tac-Toe, because creating the stitched strategy and computing the exploitability of this strategy is extremely slow even for the game with the enforced first move.

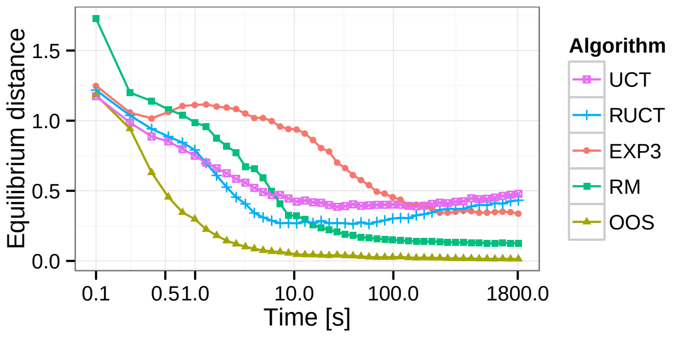


Figure 4.15: Convergence and game playing comparison of different algorithms on Phantom Tic-Tac-Toe, in which the first move of each player is forced to be to the middle field.

With beliefs

0.1s	OOS	RUCT	UCT	EXP3	RM	RAND
OOS	85.3(1.8)	84.5(1.8)	84.3(1.9)	85.2(1.8)	84.2(1.9)	93.9(1.3)
RUCT	86.8(1.6)	85.9(1.7)	84.5(1.7)	84.8(1.7)	82.3(1.8)	95.6(1.1)
UCT	85.0(1.7)	84.5(1.8)	82.9(1.9)	82.8(1.8)	80.0(1.9)	94.0(1.4)
EXP3	86.7(1.6)	87.2(1.6)	85.8(1.7)	88.1(1.5)	85.3(1.6)	96.6(1.0)
RM	**87.8(1.5)**	**87.7(1.6)**	**88.0(1.4)**	**88.1(1.4)**	**85.2(1.5)**	**96.0(1.0)**
RAND	62.4(2.9)	50.5(3.0)	49.3(3.0)	46.1(3.0)	48.0(3.0)	71.4(2.7)

With beliefs

1s	OOS	RUCT	UCT	EXP3	RM	RAND
OOS	86.0(1.7)	85.8(1.8)	85.2(1.9)	85.5(1.8)	84.0(1.9)	92.9(1.4)
RUCT	86.8(1.5)	86.5(1.8)	86.8(1.7)	86.3(1.7)	84.0(1.8)	95.2(1.1)
UCT	86.2(1.6)	88.5(1.6)	87.5(1.7)	86.7(1.7)	84.2(1.8)	94.4(1.3)
EXP3	87.9(1.6)	88.5(1.6)	88.0(1.6)	88.5(1.5)	87.0(1.6)	96.2(1.0)
RM	**88.1(1.4)**	**90.0(1.4)**	**89.0(1.4)**	**87.4(1.5)**	**85.2(1.5)**	**96.3(1.0)**
RAND	63.4(2.8)	52.0(3.0)	52.5(3.0)	48.9(3.0)	46.2(2.9)	70.9(2.7)

Table 4.12: Win-rate of different algorithms on full Phantom Tic-Tac-Toe. The number in brackets indicates 95% confidence interval.

Game playing

Since all the algorithms seems to be performing very well even in the complete game of Phantom Tic-Tac-Toe, we do not present the evaluation on the smaller game with enforced first move here. We assume the differences between the algorithms would be even smaller there.

The results in Table 4.12 show even larger imbalance between the results achieved from the first and the second position in the game. This time, it is not caused only by the disadvantage of larger information sets (see Figure 4.2, but also the fact that the game is not balanced on its own. When both players play an optimal strategy, the player who moves first wins 83% of games. With 0.1 second per move (top), the performance of RM and Exp3 is practically identical from the first position, but RM loses less on the second position, which is consistent with its generally lower exploitability over the experiments. UCT generally performs the worst. With 1 second per move, the performance of all algorithms is more similar to each other. RM still wins most often and loses least often against UCT, but the mutual matches with Exp3 are more balanced. Either of the algorithms wins 87% of matches from the first position against each other. However, when the algorithms play against themselves, RM loses less from the second position, which makes it the more suitable algorithm for this setting. We observed no significant influence of using beliefs in this game.

4.3.5 Discussion

This section summarizes the results over all evaluated games. The novel OOS algorithm was always the algorithm that converged the fastest very close to the actual equilibrium in the game. IS-MCTS algorithm variants, regardless of the selection function, converge slower and do not come close to the equilibrium even with huge computation times. Moreover, the convergence of the most common (R)UCT selection method is in all games non-monotonic and after some point, more computation time makes the computed strategy worse.

We propose the aggregated method for measuring the quality of the convergence to a Nash equilibrium, suitable for online game playing algorithms. These measures allow us to rigorously study the influence of the parameters to the performance of the algorithms without the bias of facing specific set of opponents in a tournament. Thanks to this measure, we show that the quality of convergence of OOS depends more on changes in the exploration parameter than the changes in targeting and we confirm the natural assumption that weaker targeting is beneficial only with longer computation times. Furthermore, we show that this measure is, to certain extent, useful for predicting performance in tournaments. While the order of different algorithms in the tournament did not match the results very well, the less exploitable parameter settings generally perform better. Overall, the results based on measuring exploitability are clearer and easier to interpret than the results obtained from the tournaments.

One of the novel contributions of this thesis is introducing modelling of beliefs to IS-MCTS algorithm. Our experiments clearly show that in the domains where the correct belief is far from uniform, explicitly modelling beliefs produces substantially less exploitable strategies. We show this both in the aggregate exploitability measure as well as in the tournaments, where IS-MCTS with modelling believes loses less often.

Another novel contribution is the use of Regret Matching as a selection function for IS-MCTS. We show that this selection function has several advantages. Unlike with UCT and RUCT, an additional computation time never causes drifting further from an optimal solution. Furthermore, it converges to an equilibrium substantially faster than Exp3 in all experiments. Finally, it often performs the best in the actual head-to-head matches.

We also evaluated the importance of randomization in the UCT selection function in these games. While the randomization seems to have a positive effect on the convergence in the root with short computation times, the performance in the matches did not show RUCT nor UCT to be better than the other.

Finally, head-to-head matches of IS-MCTS confirmed that OOS has a low exploitability in smaller games. It never lost significantly in the smaller games, but it was also not trying to exploit the opponents. This is also often the case with near equilibrium players in the computer Poker tournaments [BHRZ13]. In the large game, OOS performed clearly worse than IS-MCTS.

Chapter 5

Case study: Pursuing Smart Targets

In this chapter, we demonstrate that the advanced MCTS algorithms studied in this thesis can be used to solve complex real-world problems. In particular, we focus on the problem of robotic pursuit-evasion games.

Recent advancements in robotics are noticeable in deploying unmanned vehicles capable of autonomously performing simple tasks in real-world scenarios (e.g. UAVs can navigate through fixed points over an area in order to gather information). These deployments motivate the research of algorithms for more complex tasks that the group of unmanned vehicles can perform. Targeting real robotic systems imposes strong constraints on the computation time required by the algorithms; hence, anytime algorithms are preferable. Important representatives of tasks for a group of autonomous systems are the problem of *visibility tracking* – a group of tracking agents cooperates in order to maintain a direct visibility contact with an evading target as often as possible; or the problem of capturing the evading unit by arriving sufficiently close to the unit.

These problems are of particular interest for the defence or security domains in which the target actively avoids being seen and captured by the tracking agents. Game theory provides theoretic and algorithmic foundations for such situations and a game modeling the described scenarios is defined as a *visibility-based pursuit-evasion game* (PEG) [HS09, BH10, RNK$^+$10] with simultaneous moves — a two-player zero-sum extensive-form game between the *pursuer* (that controls multiple pursuing or tracking agents) and the *evader*. We focus on variants of these games played in an Euclidean environment discretized as a graph, and we assume that both players have a full knowledge of the topology of the environment. However, we consider *imperfect information* in the game; if none of the pursuing agents has a direct visual contact with the evader, the pursuer does not know neither the position nor the actions performed by the evader. He can only infer the set of possible positions of the opponent. The same holds for the evader respectively. In the case of tracking, the main objective of the pursuers is to keep the average size of the set of possible positions of the evader as small as possible. Ideally, the pursuer prefers to have a direct visual contact with the evader as often as possible. The objective of the evader is the opposite. We evaluate multiple alternative formulations of the objective of the pursuer, but the results with respects to all of them are closely correlated.

In this chapter, we propose and evaluate three anytime search-based algorithms for playing this variant of visibility-based pursuit-evasion games from the perspective of both players. The algorithms operate on both types of search spaces introduced in Section 3.2.1. For both search space representations, we propose algorithms based on Monte Carlo Tree Search, but we also compare then to an iterative deepening variant of the minimax algorithm proposed in [RNK$^+$10] for this domain.

The main contributions of this chapter include: (1) Presenting an interesting evaluation domain for game playing in imperfect-information games with a unique set of features. It is modelling a real world problem. It is asymmetric in terms of player's strategies and numbers of available actions. The players choose their actions simultaneously and their reward is cumulative, i.e., it is obtained gradually and cannot be lost, once received. (2) Adaptation of three fundamentally different algorithms for playing imperfect-information games and presenting their domain-specific improvements. (3) Thorough experimental evaluation of the algorithms and their modifications on two scenarios modelling a real-world urban area as well as a complex artificial maze.

The experimental evaluation shows that each of the evaluated search spaces is suitable for a different player in the game. The search based on the single player's IST performs better for the pursuer with substantially more actions available in each move and a larger impact of mistakes, because the evader can become lost and very hard to re-discover. The search based on EFG performs better for the evader with fewer actions per move and less costly mistakes. For both players, the performance of the algorithms operating on EFG was substantially improved by explicitly modelling the belief distributions (i.e., the likelihood of being in individual states consistent with the available observations). Furthermore, consistently with previous work in other domain [CF10], we were able to achieve a better performance using the evaluation function instead of a simulation in the MCTS on a single player's information-set tree.

5.1 Related Work

The problem of the collaborative tracking or surveillance of targets that are aware of being monitored (*evaders*) is formally described by various variants of *pursuit-evasion games*. There are several different characteristics of the studied PEGs. The differences can be found in the environment (it can be a graph, a polygonal environment, or a free space), in the maximal speed of the agents (the speed can be bounded or not, or the speed of the evaders can be superior or not), or in the range of sight (if there is a limited range of visibility, or if there is an angle of the sight (e.g. camera)). In general, we can identify two main tasks that pursuers tackle in these games: (1) search/capture, or (2) target tracking.

The origins of the problem of pursuit-evasion games were given by works [Isa65, BO99] and followed by many others. One of the first works for the task of target tracking is the work by LaValle et al. [LGBBcL97] in which authors solve the problem of maintaining visibility of a *stochastic*, not adversarial, evader in a polygonal environment with obstacles in discrete time. Fabiani et al. [FGBLL02] improve the framework to work under uncertainty in localization of the pursuer. Vidal et al. [VSK$^+$02] pushed the pursuit-evasion games into the real-world experiments with several UAVs and UGVs. Scenarios with an adversarial evader and a group of pursuers were studied by Vieira et al. [VGS08, VSG09], and by Raboin et al. [RNK$^+$10]. Each of these works focus on the algorithmic aspect; the first works solve the problem of capturing the adversarial evader by means of minimax algorithm in [VGS08], and in [VSG09] they scale the scenarios by decomposing the multi-pursuer scenarios. The minimax algorithm, although modified to work under the imperfect-information assumption, is also used in [RNK$^+$10] to solve the problem of tracking. This work is closest to our approach. Our main extensions of this paper are the introduction of the anytime requirement; adaptation of two substantially different MCTS algorithms for solving the problem; development of strategies for both pursuer and evader; and a more realistic evaluation with opponents respecting the imperfect-information restrictions. Our preliminary work on one of the proposed algorithms was reported in [LBP12] and most of the reminder of this chapter is reported in [LBP14].

Besides the solution methods based on the heuristic game-tree search algorithms, there are also exact methods for computing the optimal strategies for the players. For example, Harmati et al. [HS09] formulate the game as a mathematical program and seek a Stackelberg equilibrium for this game. Because we define our game as generic zero-sum imperfect-information game, formulation of the game in form of a mathematical program would also be possible (see Section 2.1.5). The size of this program is, however, polynomial in the size of the complete game tree; hence, it is not solvable for practically large games. Therefore, from the algorithmic perspective we base our approach on the Monte Carlo Tree Search method. A well-studied perfect-information game that is most similar to the game analyzed in this chapter is Ms. Pacman, in which MCTS was successfully applied, e.g., in [NT13, PWL14].

5.2 Problem Definition

We are solving a visibility-based pursuit-evasion game in a discretized environment played in moves. The game definition and the notation is the same as in [RNK$^+$10] after discretization. We assume that there is a single evader and multiple pursuing agents. The long-term goal of the pursuing agents is to collaboratively ensure maintaining visibility contact between at least one of the pursuing agents and the evader. The pursuing agents are therefore seen as resources of a single player controlled by a centralized algorithm. The evader aims for the opposite. The game is therefore an extensive-form zero-sum game with imperfect information.

We use the following notation:

- $P = \{p_1 \dots p_q\}$ is a set of agents controlled by the pursuer;
- e is the agent representing the evader;
- N is a common set of nodes (positions) for all agents;

- $pos_0 : P \cup \{e\} \to N$ are the initial position of the agents, not necessarily known to both players;
- $t : N \to \mathcal{P}(N)$ is the accessibility mapping that assigns each place a set of places an agent can move to from the given position in one time step;
- $v : N \to \mathcal{P}(N)$ is the visibility mapping that assigns each place a set of places in which the agent can see other agents from the given position;
- h is the number of time steps in which the game is played (i.e., the fixed horizon).

Further, we define for brevity the set of all agents $I = P \cup \{e\}$ and we extend the definition of t and v to be applicable also to sets of nodes. For $S \subseteq N$, $t(S) = \bigcup_{n \in S} t(n)$ and $v(S) = \bigcup_{n \in S} v(n)$. We abbreviate $t(t(\dots t(n)))$ by $t^i(n)$.

We define a *trajectory* of agent $a \in I$ as a mapping $\rho_a : \{0 \dots h\} \to N$, such that

$$\rho_a(0) = pos_0(a) \tag{5.1}$$

$$\forall s \in \{1 \dots h\} \;\; \rho_a(s) \in t(a, \rho_a(s-1)) \tag{5.2}$$

We further denote the joined trajectory of a subset of agents $G \subseteq I$ by the same symbol with a set subscript ρ_G and the set of all legal trajectories of agents in G by L_G.

The set of *histories* in this game is the set of all combinations of trajectories of the agents. Let us refer to a portion of a trajectory ρ between steps i and j as $\rho(i \dots j)$, then it is

$$H = \{\rho_I(0 \dots s) : \rho_I \in L_I \;\&\; s \in \{0 \dots h\}\}. \tag{5.3}$$

Note that a history from H corresponds to a node in the game tree, i.e., extensive form of the game. We say that two joint trajectories of the agents are *indistinguishable* by a group of agents in G

$$\begin{aligned} &\rho^1_I(0 \dots s) \sim_G \rho^2_I(0 \dots s) \\ \text{iff } \forall i \in (0 \dots s) \forall a \in I : \;\; &\rho^1_a(i) \neq \rho^2_a(i) \\ &\Rightarrow \left(\forall b \in G;\; \rho^1_a(i) \notin v(\rho^1_b(i)) \& \rho^2_a(i) \notin v(\rho^1_b(i))\right) \end{aligned} \tag{5.4}$$

In our approach, we assume that players can remember their full history; hence, if a group of agents is controlled by a single player, they can distinguish between the trajectories with different histories of actions of their agents even without seeing each other.

The *information set* of a player is the set of histories that are indistinguishable by the agents it controls. The information of the player controlling agents in G about the *possible position* of agent a at time s in case of trajectories ρ_I is:

$$\begin{aligned} PP_G(a, s, \rho_I) = \Big\{\rho'_a(s) : \rho'_a \in L_a \;\&\; \forall i \in \{0 \dots s\} \\ \rho'_a(i) \in \bigcup_{b \in G} v(\rho_b(i)) \Rightarrow \rho'_a(i) = \rho_a(i) \Big\} \end{aligned} \tag{5.5}$$

The main objective of the pursuers is to minimize the average size of the set of possible positions of the evader based on the shared information of the pursuer's agents. The objective of the evader is exactly the opposite. More formally, the game objective is

$$meanSize(\rho_I) = \frac{1}{h} \sum_{s \in \{1,\dots,h\}} |PP_P(e, s, \rho_I)| \tag{5.6}$$

Note, that the evader does not know the optimized value exactly in the states without visual contact with a pursuer's agent. In his reasoning, he has to approximate this value or optimize some correlated value, such as the size of the set of possible positions of the pursuer's agents based on the evader's information.

In some situations, it is important to have visual contact with the target as often as possible (e.g., it may enter an unknown underground passage). In order to evaluate the algorithms for these situations, we consider the number of times the evader has been spotted by a pursuer agent as another performance measure.

$$numSeen(\rho_I) = |\{i \in \{0..h\} : \rho_e(i) \in \bigcup_{p \in P} v(\rho_p(i))\}| \tag{5.7}$$

The last performance measure is the size of the set of possible positions at the end of the game. This measure was used by [RNK$^+$10] and it is included mainly for fair comparison.

$$endSize(\rho_I) = PP_P(e, h, \rho_I) \tag{5.8}$$

The game defined above is a zero-sum extensive-form game with imperfect information. Solution of a game can be under the assumption of the rationality described by various solution concepts; the best known is Nash equilibrium (NE), in which no player is motivated to deviate unilaterally from its strategy. A NE solution not only guarantees the optimal performance against the optimal opponents, but in zero-sum games, it guarantees the predicted quality of the solution even in case the opponent does not play optimally. However, computing an exact NE or its guaranteed approximations is infeasible for imperfect-information games of size we focus on in this chapter. Therefore, similarly to the previous works in pursuit-evasion games [VGS08, VSG09, RNK$^+$10], we focus on the heuristic game tree search algorithms.

5.3 MCTS for Visibility-Based Pursuit-Evasion Games

This section describes the domain specific adaptations and the background knowledge used in our MCTS-based algorithms for solving the problem defined above. Because of large size of the game used for evaluation, with up to 10^{180} leaf nodes, we do not consider Online Outcome Sampling. Perfect Information sampling Monte Carlo is likely also not a suitable method for this problem, because it has been shown to be efficient when the size of information sets is getting smaller during the game [LSBF10], which is not the case for this problem. Therefore, we focus on IS-MCTS as the main solution method for this problem. We evaluate using the explicit belief distributions, because of large size of the information sets in the game. Furthermore, we provide more thorough evaluation of efficiency of MCTS on single player's information set tree, which requires domain specific knowledge and is, therefore, difficult to evaluate in direct comparison with other method on wider range of domains.

5.3.1 MCTS on Single Player's Information Set Tree

Problem-specific knowledge can be added at many places in information set search in both the full-width and MCTS version. In this thesis, we focus only on the most basic options. The main requirement for efficient use of MCTS on single player's information set tree is availability of efficient generative model for IST in the specific domain (see Section 3.2.2). Visibility-based PEG is a suitable domain from this perspective, especially after applying an imperfect recall abstraction.

Imperfect recall abstraction

Recall that a player in an extensive-form game has perfect recall if he remembers all the information he gained during the game, mainly the history of all his moves. This property is necessary to achieve guarantees of all known equilibrium computing algorithms and even existence of some equilibria in extensive-form games. ISS with the practically used heuristic opponent models has no guarantees relaying on the perfect recall. Therefore, we assume the player forgets part of the information to improve efficiency of the algorithm. As a result, some of the information sets in the IST of the game are merged.

The extended information set used by the algorithm at any time consists of the current position of the player's agents and his believe about current possible positions of the opponent's units. When there is direct visibility between the units, this is only one position, but otherwise, it can be a larger collection of possible positions. The player forgets the trajectories used by their agents to reach their positions, but still remember their effect on the set of possible positions of the opponent's agents.

This state representation allows to easily generate the information-set tree directly. The actions available to the searching player are the combinations if the moves available to his units, position of which is explicitly represented by in the state. After an action is applied the observations that can be generated can also be easily computed from the set of possible positions of the opponent's units. Each position that is reachable by the opponent in the next move and visibly by some in the player's units generates a new observation corresponding to the opponent's unit becoming visible at that place. The set of possible position of the opponent unit becomes a singleton in that case. Furthermore, an additional observation is generated for the opponent's unit staying hidden. The set of possible position of the unit typically grows in this case.

Evaluation functions

Heuristic evaluation function is a crucial requirement if we want to apply full-width search techniques to larger game trees. Raboin et al. [RNK$^+$10] compared performance of several evaluation functions for the pursuer and determined that the one they call relaxed lookahead (RLA) performs best in this setting. The heuristic computes the mean size of the region where the evader cannot be spotted under any movement of the pursuers in the following d steps of the game.

$$RLA_P(s,\rho_I) = \tfrac{1}{d+1}\sum_{i=0}^{d}\left|t^i(PP_P(e,s,\rho_I)) \setminus v(t^i(\{\rho_p(s) : p \in P\}))\right| \tag{5.9}$$

An efficient method to compute this value is presented in [RNK+10]. The authors, however, do not define any heuristics for the evader. They assume the worst case behavior of the evader that knows the position of the pursuers all the time in their experiments (E. Raboin 2011, pers. comm. 2 February). In this thesis, we aim to achieve realistic behavior of the evader as well. That is why we define a very similar heuristic for the evading agent, but based on the information available to the evader.

$$RLA_e(s,\rho_I) = \tfrac{1}{d+1}\sum_{i=0}^{d}\left|t^i(\rho_e(s)) \setminus v(t^i(\bigcup_{p\in P} PP_e(p,s,\rho_I)))\right| \tag{5.10}$$

MCTS simulation heuristics

The domain-independent MCTS uses random moves for all players in the simulation phase. In information-set tree, it corresponds to random selection of player's decision as well as the observation received by the player. The most common use of the problem-specific knowledge in MCTS is biasing the simulation towards more probable behavior of the players.

For the pursuer, the basic heuristic rules we added to the simulation are (1) if the evader has an action that will certainly keep him hidden from the pursuers in the next time step, it always selects the action and (2) if the pursuer has actions after which it will certainly see the evader in the next time step, it selects randomly among those actions.

The simulation algorithm for the pursuer based on this strategy is presented in Figure 5.1. It runs for d steps, which is generally until the pre-specified horizon of the game is reached. If the position of the evader is known (line 2), the algorithm computes for each pursuer's agent the set of candidate positions that ensure that the evader is visible also in the next time step (line 4). If the set of these positions is not empty, it selects one of them for the pursuer (line 6). Otherwise, the pursuer moves randomly (line 8). If the position of the evader is unknown (line 9), we assume that it is not likely that we will find candidate positions as above. Therefore, the pursuers move randomly (line 11). After moving the pursuers the simulation continues to an observation node. It computes the set of evader's positions, where it can be seen in the next time step (line 12). If it can be seen at all its possible next positions, its position is selected randomly (line 14). Otherwise, we assume that it is not seen in the next step (line 16). The result is the mean size of the set of possible positions of the evader (line 18).

```
Input: $pos : P \to N$ – positions of the pursuer; $PP_e$ – possible positions of the
    evader; $d$ – length of the simulation
Output: result of the simulation – estimate of $meanSize$
 1: for $i \in \{1 \dots d\}$ do
 2:   if $|PP_e| = 1$ then
 3:     for $p \in P$ do
 4:       $candidates \leftarrow \{n \in t(pos(p)) : PP_e \subseteq v(n)\}$
 5:       if $|candidates| > 0$ then
 6:         $pos(p) \leftarrow random(candidates)$
 7:       else
 8:         $pos(p) \leftarrow random(t(pos(p)))$
 9:   else
10:     for $p \in P$ do
11:       $pos(p) \leftarrow random(t(pos(p)))$
12:   $nextSeen \leftarrow \{n \in t(PP_e) : \exists p \in P\ n \in v(pos(p))\}$
13:   if $nextSeen \leftarrow t(PP_e)$ then
14:     $PP_e \leftarrow \{random(nextSeen)\}$
15:   else
16:     $PP_e \leftarrow t(PP_e) \setminus nextSeen$
17:   $u_P \leftarrow u_P + |PP_e|$
18: return $u_P/d$
```

Figure 5.1: The algorithm of pursuer's heavy simulation in MCTS on the information-set tree.

Unseen pursuer's agents

An important issue for the algorithms controlling the evader is consideration of the pursuers that has never been seen in the game. If the set of their possible positions are all the currently unseen positions of the game, the search on IST cannot be used for the evader. It renders all the strategies of the evader almost equally bad, because the worst-case observation in the case of any action is that the unseen pursuer will appear just in front of the evader. Therefore in our IST implementation, the evader completely ignores the existence of the pursuers it has never seen. Similarly, if the size of the set of possible positions of a pursuer grows over a certain threshold, the set of possible positions of the pursuer stops growing and the evader does not expect it to appear in its search anymore. However, where applicable, the evader still tries to avoid these positions in the evaluation or simulation.

5.3.2 MCTS on Full Extensive-Form Game

The domain specific knowledge for IS-MCTS includes the hash function mapping a game state to the (possibly abstracted) information set it belongs to and the simulation strategy used after the search leaves the portion of the game trees stored in the memory.

Information set hash function

Two search nodes belong to the same abstracted information set if they have identical

- player i deciding in the search node;
- current position of units of player i;
- positions where each of the units of the opponent of i was observed the last time;
- time steps at which each of the units of the opponent of i was observed the last time.

Note that this abstraction is even coarser than the one used for search on information-set tree in Section 5.3.1. Consider that two nodes represent two different trajectories $\rho_a^1 \neq \rho_a^2$ of the same unit a. Than the player controlling a can infer different sets of possible positions of an opponent's unit b ($PP_a(b, s, \rho_a^1) \neq PP_a(b, s, \rho_a^2)$), even if the unit was last observed at the same time in both trajectories. Here, we propose to discard this information, in order to maximize the number of samples used to for a decision in each abstract information set.

Simulation strategy

Domain specific implementations of MCTS often benefit from a non-uniform simulation strategy. In general, the better the simulation strategy approximates the real outcome of the game from the node where it is started the faster the MCTS algorithm converges to the right decision. The only improvement to the uniformly random simulation strategy that proved to be effective in our experiments was preventing the units from returning to their original position:

$$\forall a \in I;\ \rho_a(s+1) \in t(a, \rho_a(s)) \setminus \{\rho_a(s)\}. \tag{5.11}$$

We also evaluate an alternative simulation strategy, which for the pursuer's units follows the shortest path to the last observed position of the evader's unit and for the evader's unit moves to a position with the largest sum of distances from the pursuer's units.

Unseen pursuer's agents

The pursuer's agents not seen for a long time are a little less problematic for this algorithm. The algorithm does not rely on modelling the set of possible positions so they are completely ignored if they can be present at too many locations.

5.4 Experimental Framework

We provide the evaluation of the proposed algorithm in a complex simulation platform developed in a series of projects in our group. Several screenshots from the framework are presented in Figure 5.2. It allows us to test the strategies computed

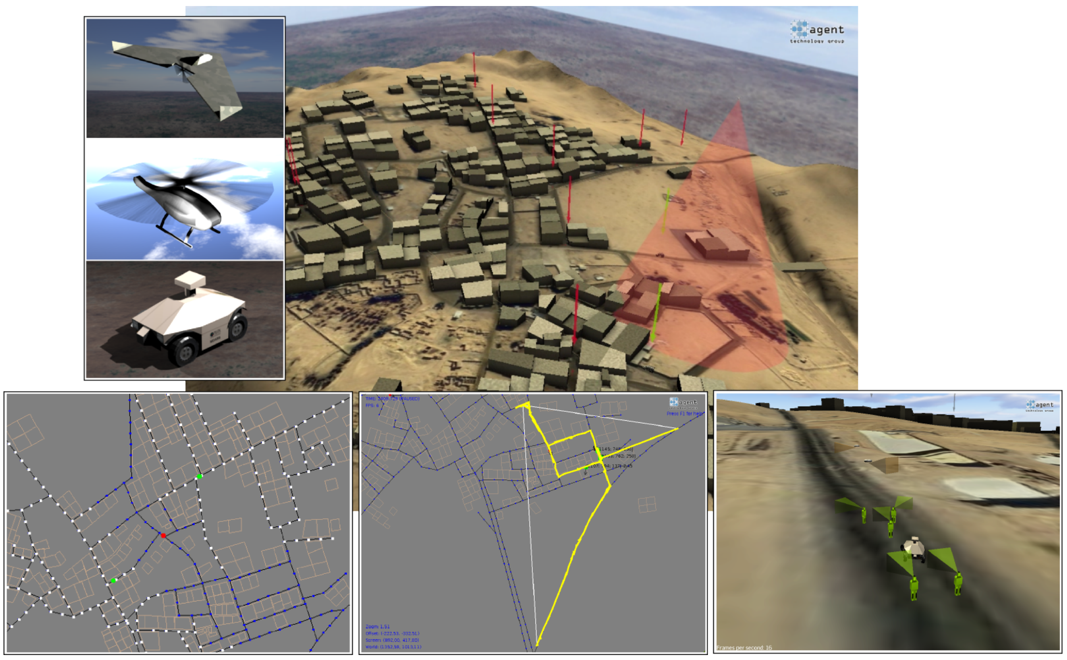


Figure 5.2: Tactical AgentScout simulator facilitates rich models of physical reality and supports physical mobility dynamics of robots, such as ground and aerial vehicles, be it fixed-wing aircraft, helicopters or various rotorcraft.

by our algorithms in a more realistic settings (such as imprecise space discretization) and provides a visually attractive and interactive live-system demonstrator. For example, it allows humans to play the pursuit-evasion game against the algorithms. We presented details on the framework and its capabilities in [NKL+12].

The framework also allows reducing the uncertainty in execution of agent's plans to completely obey the formal model of the problem, in order to provide rigorous evaluation without side effects which are hard to fully replicate. The results in this chapter are all from this fully reproducible setting.

5.5 Experimental Evaluation

We use two main ways to evaluate algorithms for playing zero-sum extensive-form games in the previous chapter. We are not able to compute exploitability in huge games, such as this one. Therefore, we focus only in head-to-head performance here. The exploitability of the players can still be estimated from their performance against the opponent that performs best against the particular player. We further call this player the nemesis opponent. If the set of players in the tournament is sufficiently representative and each pair of the players plays sufficient number of matches, then the player with best minimum performance over all opponents is the best approximation of the optimal (Nash equilibrium) play.

5.5.1 Experimental Settings

In the experiments, we assume two agents of the pursuer are tracking one evader. The implementation of each player uses only one thread and its computation time is limited to one second on Intel(R) Core(TM) i7 CPU @ 2.80GHz. Each of the scenarios runs for 100 time steps.

Algorithms

The first algorithm we use in the evaluation is based on [RNK+10]. It is depth-limited minimax search on the information-set tree with heuristic evaluation function. The authors do not impose the anytime requirement on their algorithm. In order to meet it, we have implemented the algorithm with alpha-beta pruning and iterative deepening, and we denote it *MM*. We start with search depth of three time steps and increase the depth by one with each iteration. The result from the last computation that is completed before the 1 second limit is used. In our experiments, depth over 6 is usually reached; however, if the size of the possible positions of the evader is very large, even the first iteration might not finish in time. In that case, random action is returned. We use two heuristic evaluation functions.

- **RLA** The evaluation functions defined in Section 5.3.1 with depth $d = 10$.
- **URLA** For player i, $URLA_i(s, \rho_I) = RLA_i(s, \rho_I) + u_i(s, \rho_I)$.

The second heuristic is necessary to achieve good results for deeper search. With the original RLA, the pursuers might not go to see the evader immediately if they are certain they can do that later. If the search depth is 10 and the pursuers needs 5 moves to see an evader located anywhere in a small set of possible positions, they do not have an incentive to act immediately. The RLA evaluation is the same even with waiting one move. Adding the utility acquired before reaching the leaf of the tree removes this problem. The threshold for the change of modelling of the unseen opponent (see Section 5.3.1) is 250 possible positions.

The second algorithm is MCTS with UCT selection on the IST of the game. We denote it *SO* consistently with [CPW12]. We run expansion in each iteration of the algorithm and we select the first child generated for simulation without preference ordering. We use five methods for the simulation phase of the algorithm. The first two methods are specific for the pursuer and the rest is applicable for both players

- **Random** In the decision nodes, it takes a random action. In the observation nodes, it randomly decides if the evader stays hidden or not and if not, it randomly selects the node where it appears. This simulation ends at depth 30 form the root.
- **Heavy** Simulation described in Figure 5.1. This simulation ends at depth $h = 30$ from the root.
- **NoSim** No simulation at all. The utility value of the current node is returned as if it was terminal.

- **RLA** Instead of simulation, heuristic value computed using RLA with $d = 10$ is returned.
- **RS** The size of the set (region) of the uncertain positions of the opponent in the current node is returned.

For the Heavy, Random and NoSim simulation strategies, we use the logarithm of the actual utility value (i.e., region size) to stress that the differences between the small region sizes are more important than the differences between large region sizes. For

$$u_P(h, \rho_I) = \log_{10} \frac{1}{h} \sum_{i=1}^{h} |PP_P(e, i, \rho_I)|, \tag{5.12}$$

$$u_e(h, \rho_I) = \log_{10} \frac{1}{h} \sum_{i=1}^{h} \sum_{p \in P} |PP_e(p, i, \rho_I)|. \tag{5.13}$$

Furthermore, this allows us to better scale the UCT parameter and a we tuned the value to $C = 2$ for SO.

The third algorithm is the IS-MCTS simultaneously computing the strategy for both players and we denote it *MO*. We use the estimate of the belief as described in Section 3.2.3 and the maximal number of consequent sample repetitions $K = 100$. The simulations are by default ended after 50 moves form the root for the pursuer and 40 moves form the root for the evader. In the root of the game form the evader's perspective, we completely ignore a pursuer that was never seen or its set of possible positions is larger than 250 nodes. The default simulation strategy only requires the agents not to return to the node they occupied in the previous time step. As for the previous algorithm, the UCT is $C = 2$ and the threshold for ignoring an unseen pursuer is 250.

Map

We use two maps in the experimentation. The first is the exact same map and visibility model as in [RNK$^+$10] for fair comparison with the state-of-the-art algorithm. The topology of the map in form of 50x49 pixels bitmap is presented in Figure 5.3a. White pixels represent possible position of the agents, black pixels are obstacles and agent can move to the adjacent pixels in one time step. Line-of-sight visibility with Euclidean distance limitation of 10 pixels is assumed.

The second map is based on the topology of a small real-world urban area. Figure 5.3b presents the overview of the complete road network and Figure 5.3c is a detail from the center of the map. The road network was discretized as a graph with a node placed every 5 meters, creating 465 nodes. We assume symmetric visibility and the agents can see each other if they are not further than 25 meters from each other and there is no building in their line of sight. Three larger circles represent positions of the agents, the one in the middle being the evader. The white circles are the positions currently visible by the pursuers. The black dots represent the current set of possible positions of the evader, as believed by the pursuers based on their past observations.

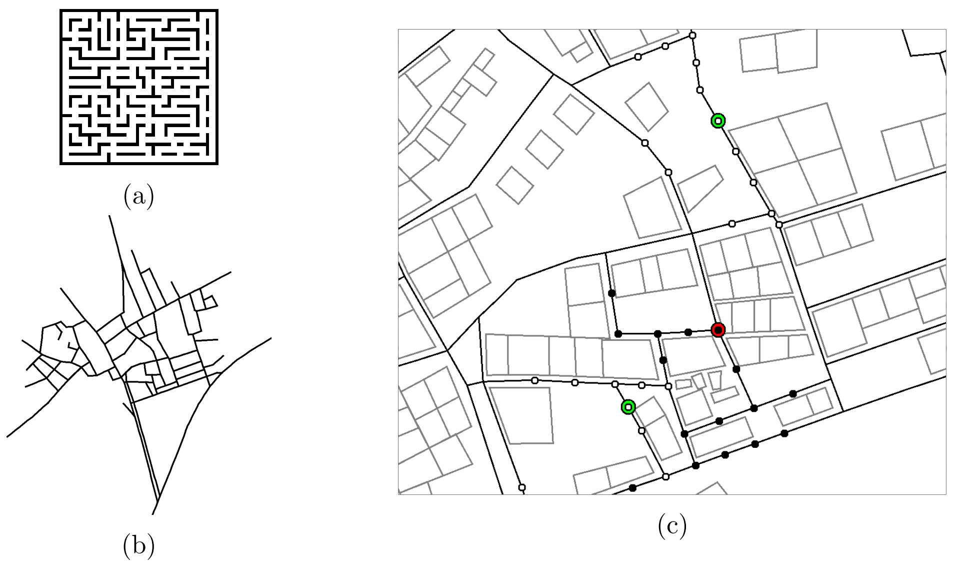


Figure 5.3: The environment maps used for experimentation. (a) the complete road-network map; (b) a detail of the road-network map, the agents are visualized as large circles (the evader is in the middle), the black circles represent the current set of possible positions of the evader, the white circles represent currently visible positions to the pursuer; (c) full maze map used in [RNK$^+$10].

For initial positions of the game, we also follow [RNK$^+$10]. The main idea is to generate random setting where the evader is visible by at least one of the pursuers, but at the same time, it is far enough from the pursuers to make tracking difficult. The first pursuer is placed randomly. The second pursuer is placed randomly, but not closer than 25 moves from the first. The evader is placed randomly to a position that is most distant (in the number of moves) from each of the pursuers and at the same time, visible by at least one of the pursuers (E. Raboin 2011, pers. comm. 31 January). Each reported result is the mean of runs on 100 fixed starting configurations.

5.5.2 Results

We first present analysis of computation times required for using full-width search in our domain to motivate the need for anytime algorithms. Then we present the results of the tournament of the best performing setting of each type of algorithm we study. Afterwards, we explore various setting of the algorithms and their influence on players' performance.

Minimax computation time

If we want to use the state-of-the-art minimax algorithm as it is, we need to set a fixed search depth. However, the time required for computation varies a lot for different situations. Figure 5.4 presents the times required by depth 8 minimax alpha-beta search for the pursuer. It is a histogram of run-times from 10 000 executions of the

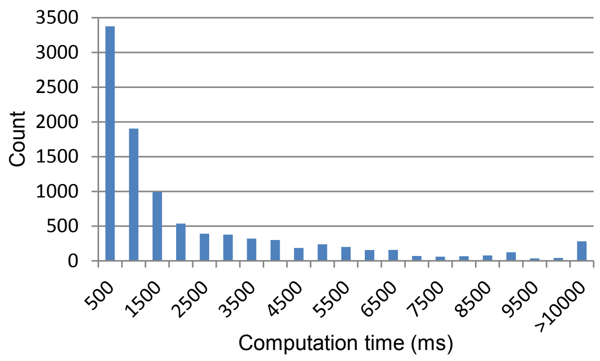


Figure 5.4: The histogram of the computation time needed by the pursuer using the minimax algorithm with alpha-beta pruning and fixed depth of 8.

measure →	numSeen			meanSize			endSize		
pursuers → evader ↓	MM (URLA)	SO (RLA)	MO (h=50)	MM (URLA)	SO (RLA)	MO (h=50)	MM (URLA)	SO (RLA)	MO (h=50)
Route-network Map									
MM(URLA)	-58.6	**-60.7**	-55.7	80.9	**77.4**	78.9	125.4	119.7	**119.4**
SO(NoSim)	-62.3	-68.7	**-68.8**	72.7	**53.2**	56.5	109.5	75.3	**73.0**
MO(h=40)	**-58.6**	**-58.6**	-52.1	78.6	**78.9**	108.1	133.8	**128.7**	164.3
Maze Map									
MM(URLA)	-50.0	**-56.4**	-45.7	79.8	**69.2**	128.3	144.2	**135.0**	232.6
SO(NoSim)	-75.2	**-76.3**	-75.8	47.0	36.3	**34.4**	78.5	63.6	**48.6**
MO(h=40)	**-45.9**	-44.7	-35.7	164.2	**146.6**	252.7	**279.0**	287.1	466.4

Figure 5.5: Comparison of the best settings for both Monte Carlo tree search approaches and the best iterative deepening minimax approach. The pursuer minimizes and the evader maximizes the presented values.

algorithm in 100 different 100 moves long games. More than half of the computations were finished within one second, but there were hundreds of situations that required more than 10 seconds. In a real-world application, the pursuer would not know in time, where to move. If the search depth is lowered, the quality of the produced solutions is decreased.

Methods comparison

Figure 5.5 presents the results of mutual matches of each algorithm on each position. The parameter setting of each algorithm in this table is tuned for the best performance against the worst of the opponents. For example, the parameters of MO on the side of the pursuer in the table are those that perform best against the MO evader.

Let's first focus on the performance of the evader (rows), the first map and the first measure in the upper left part of the table. It is the number of time steps in which the evader was seen by the pursuer. The evader prefers not to be seen so he maximizes the negative of this number. We use the negative numbers in the tables to make consistent that the pursuer tries to maximize the presented values and the evader tries to minimize them. In the table, we always mark the performance of the evader's strategy against its nemesis pursuer's strategy by bold font. For the MM evader, the nemesis pursuer's strategy is SO. The best evader's strategy is the one with the highest value among the bold numbers. It is marked by grey background. For the numSees measure, it is the MO strategy, slightly before the MM strategy. Also with respect to the other two measures, the bold numbers indicate that the MO is the best, followed by MM and SO is the worst. This ordering of game playing quality is even more apparent in the maze map.

On the side of the pursuer (columns), we use the same method of comparison; however, we mark the performance against the nemesis strategy by underlining the value. The minimal value among the underlined values (marked in grey) are reached by both the MM and SO algorithms depending on the map and measure selection with MO performing significantly worse. Based on the meanSize measure that we defined as primary for this paper, SO approach is slightly the best.

An interesting observation is that the SO and MO approaches switch places at being the best and the worst approach for each player. It can be explained by the importance considering of the worst case behavior of the opponent for individual players. If the pursuer makes a mistake and the evader gets out of sight, it is likely that he will be lost completely for the rest of the game. Hence the pursuer should avoid risk and try to approximate the worst case well. On the other hand, if the evader makes a mistake and stays in sight, he will likely have an opportunity to lose the pursuer few moves later. In the information-set tree based SO approach, the pursuer approximates the worst case development of the game even beyond the capabilities of the evader, which creates a very risk-averse strategy. On the other hand, the evader based on SO can become paralyzed when he assumes the worst case observations in the information-set tree. In fact, it can never get completely lost, because the worst case observation is that he is discovered whenever it is possible. On the other hand, the MO approach computes a much more realistic model of the possible positions of the pursuer's units and can decide to take chances against the most likely once. The minimax approach scores in middle on both positions. The reason is likely that for the evader, the guaranteed ways out that might be present are rare and require complex tactical combinations which are more difficult to find by MCTS approaches. As a result, the MM approach gets paralyzed less often.

MO parameters influence

We present the results of MO parameters' tuning in Figure 5.6. The domain specific C parameter for the UCT selection function has moderate influence on the quality of the play of the MO algorithm for the pursuer and quite strong influence on the game playing quality for the evader, for whom $C = 2$ is clearly the best setting. Modifications of the search depth (h) show that too small search depth causes the players not to consider the possible risks and opportunities in more distant future and too large search depth reduces the number of simulation that can be performed in the restricted time. Moreover, it increases the influence of the random simulation part of the iteration relatively to the influence of the actions whose effect the algorithm

C	P	e
0.5	235	31
1	253	43
2	253	147
4	263	115
8	314	76

h	P	e
20	237	126
30	234	157
40	209	147
50	253	171
60	263	160

Figure 5.6: The meanSize measure for the pursuer and the evader after variation of the UCT parameter C and simulation depth h in MO algorithm facing its nemesis algorithm, i.e., SO against the evader and MO against the pursuer.

	numSeen	meanSize	endSize	Iter./s
NoSim	-7.7	625.7	1167	20394
Heavy	-12.2	600.6	1039	570
Random	-12.3	612.8	1109	5271
RS	-20.8	352.9	702	23788
RLA	-44.7	146.6	287	13892

Figure 5.7: Comparison of different simulations for SO pursuer against MO evader

tries to estimate. That further slows down learning of good strategies.

An important modification of our MO algorithm to the algorithm presented in [CPW12] is that we explicitly estimate the belief of the players about the possible current state of the game. If we turn off this feature and assume uniform probability of all the states as assumed in [CPW12] the meanSize for the evader decreases from 146.6 to 89.2 and for the pursuer it increases from 252.9 to 267.6.

The last modification we evaluate is the effect of the knowledge in the simulation. Using a completely random simulation is only slightly worse than using the simulation that disallows moving back to the previous position. The heavy simulation using the last seen opponent's positions described in Section 5.5.1 also did not improve the performance. For example, when facing the MM evader, the meanSize measure increased from 123.4 to 160.1 with the heavy simulation used with 50% probability and the standard simulation otherwise. The average number of simulations per second decreased only slightly from 20867 to 20496 with the heavy simulation; hence, we conjecture it is not a good approximation of the optimal strategy.

SO parameters influence

The best performance of MCTS for the pursuer was surprisingly achieved with an evaluation function and not with simulation. The results for alternative simulations are presented in Figure 5.7. The actual simulations until the end of the game do not perform well, mainly because of their high computational cost. Random simulation was able to make only a little over five thousand and the heavy simulation less than six hundred, because of the one move look-ahead needed to compute the set of candidates on line 4 of the algorithm. Not using any simulation and returning the utility value of the current node leads to the worst performance for the pursuer. It may be caused by high branching factor and insufficient guidance of the search to better parts of the tree.

The clearly best performance is achieved by the RLA evaluation function. Using

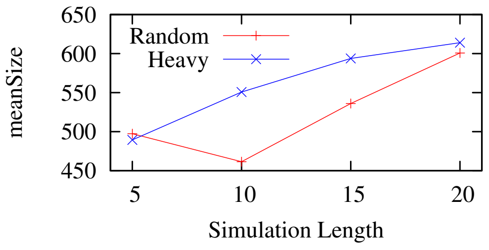


Figure 5.8: The quality of the MCTS pursuer playing against MC(URLA) evader with various lengths of simulation.

URLA is not necessary for the pursuer in MCTS. The reason is that in the progressing iterations, evaluation function is computed for each node on the path to a leaf of the current portion of the tree stored in the memory. Hence, the statistics in the nodes reflect the quality of the path even without adding it explicitly to the evaluation.

Similar results with unsuccessful longer simulations were reported by Ciancarini et al. [CF10] who applied MCTS on the information-set tree in the game of Kriegspiel. They have better results with very short simulations (only a few moves) similar to the quiescence search. Hence, we tried running the simulations only for a few steps after the current leaf of the MCTS tree. The results are presented in Figure 5.8. We confirm that even in our domain, the performance of MCTS decreases with the increasing length of the simulation. Ciancarini et al. argue that it is caused by the instability of the hidden information in Kriegspiel in contrast to, e.g., phantom Go, where the information once learned stays unchanged until the end of the game. Our game has the same property; hence our results also support this hypothesis.

5.6 Discussion

In this chapter, we propose and evaluate three anytime search-based algorithms for playing visibility-based pursuit-evasion games from the perspective of both players, the pursuer and the evader. The algorithms operate on two distinct search spaces: the complete extensive-form game and the information-set tree. In our experiments, each of the search spaces is more suitable for one player of the game.

The pursuer with a higher branching factor and a more critical effect of mistakes in the play performs better when searching on the information-set tree. With the paranoid opponent model, this search space representation is more suitable for finding a strategy robust against the worst-case behavior of the opponent. Both iterative deepening minimax and MCTS search with the evaluation function performed similarly well on this search space for the pursuer, but MCTS with the simulation was substantially less successful. This is consistent with the previously stated hypothesis that in imperfect-information games, in which the information learned can quickly become invalid, longer simulation prevent MCTS from achieving good results.

The performance of the evader with a smaller branching factor and less critical mistakes was much better using MCTS on the full extensive-form representation of the game. The performance of both players using this search space representation was substantially improved by explicitly modelling the likelihood of the current state of the game during the game play. Using these likelihoods and the more exact model of the information available to the opponent in the game, it is easier for the evader to find opportunities to escape the pursuers.

All the presented algorithms allow for endless optimizations and improvements, such as move ordering, transposition tables, progressive bias, etc.; hence, it is likely that each of the methods can be tuned to perform better for the problem. Moreover, the performance of the algorithms can be further improved by alternative methods.

Conceptually, all the proposed adversarial search algorithms perform two separate tasks in parallel:

- **single-agent planning** - finding sequences of actions achieving some partial goals of individual agents, such as coming closer to the opponent or cutting his escape route

- **interactions consideration** - exploring how plans of individual agents interact with each other (in negative but also positive way in case of pursuer's units)

In [LBJP09], we propose a method that substantially reduces the complexity of multi-agent games (i.e., games with players controlling multiple agents) by separating these tasks. Instead of all possible sequences of actions, only the sequences consistent with the satisfaction of some higher level goals of the units are explored by the search.

The knowledge about goal-oriented behavior in multi-agent systems can be generally categorized as *declarative*, which utilizes the formalization of goals and use computationally expensive search-based planning, or *procedural*, where the explicit knowledge of how to achieve goals is captured (typically as algorithms, plan templates, or predefined plans) and agents only decide which goals should be pursued (e.g. in most of BDI architectures). In our approach [LBJP09], we reduce the complexity of the first task by employing the background knowledge in the procedural form and leave only the second task to the search.

Even though we have originally proposed Goal-Based Game-Tree search (GB-GTS) for games with perfect information, it can also be easily extended to games with imperfect information and to both the algorithms that operate on the full extensive-form definition of the games as well as on the single-player information set tree. The algorithms for goal satisfaction used in GB-GTS can also produce the actions to be explored by the search based on information sets, instead of states.

The algorithms for playing visibility-based pursuit-evasion games proposed in this chapter perform very well for smaller teams of pursuers. However, the branching factor in the pursuer's decisions grows exponentially with the number of units it controls. The algorithms would very likely become much weaker if the algorithm controlled more than 5 pursuing units. One possible way to tackle this problem would be the method we proposed in [LBVP10]. Instead of one large search considering all options of all units, we propose computing the strategy by multiple searches, each exploring the options for a subset of units, while the remaining units follow a fixed plan based on past searches for these units. As a result, the complexity of the search grows polynomially with the number of units, but even strategies that require a complex coordination of a larger number of units can still be found by the search. Even this method was originally proposed for perfect-information games, but its extension to imperfect-information games is straightforward.

Chapter 6

Conclusion

The main goal of this thesis was to look at the research on online game playing by means of Monte Carlo Tree Search in two-player zero-sum extensive-form games with imperfect information from the perspective of game theory. To analyze the convergence guarantees of the existing algorithms and propose new Monte Carlo tree search algorithms, that would be better founded in game theory.

All existing MCTS algorithms for this class of games are based on online learning theory developed around the multi-armed bandit problem. In imperfect-information games, this is a meaningful foundation for such algorithms, because of the clear connection between the minimization of cumulative regret, which is the primary optimized criterion in multi-armed bandit problem, and the concept of Nash equilibrium. In this thesis, we argue that in addition to this, game theory in general, and the recent research on (Monte Carlo) Counterfactual Regret Minimization in particular should also be considered as the basis for online Monte Carlo Tree Search algorithms. It provides a solid formal framework that allows creating MCTS algorithms with well-understood asymptotic behavior and further study their behavior in short computation times.

We believe that a methodology that can lead to good algorithms for complex problems is to approach the problem from both the empirical and the theoretical perspective. From the empirical perspective, it is important to try to solve the problem in its whole complexity, invent and empirically evaluate intuitive heuristics that help solving special cases of the problem very efficiently. From the theoretical perspective, it is good to start from well-understood solutions with strict theoretical guarantees, which may initially not be applicable to the problem in its whole complexity. However, good understanding of the fundamental properties of the problem allows making conscious decisions about the trade-offs between losing guarantees in well-defined sub-classes of the problem, the loss in solution quality it can cause and potential benefits for the practical applicability of the solutions.

In this thesis, we tried to make contributions from both these directions, but a larger amount of further research is needed to create well-understood and widely-applicable MCTS algorithms for this class of games.

6.1 Thesis Contributions

In this thesis, we have reviewed the existing MCTS algorithms for two-player zero-sum extensive-form games with imperfect information. We observe that none of the pre-existing algorithms are known to converge to the optimal solution in the game. Therefore, we propose the first MCTS algorithms that do have these guarantees.

First we focus on a simpler subclass of two-player zero-sum extensive-form games with imperfect information – games with simultaneous moves, but otherwise perfect information. On a wide range of games, we confirm the previously-known result showing that the most commonly used MCTS algorithm for this class of games converges to solutions very far from Nash equilibrium. We present a novel algorithm called Simultaneous Moves Online Outcome Sampling and a class of additional MCTS algorithms (SM-MCTS-A) that are guaranteed to converge to a Nash equilibrium in these games. For all of them, we prove their convergence formally as well as demonstrate empirically on various domains. Even though we do not prove it formally, we show that several other existing and novel SM-MCTS algorithms also converge to a NE in our experiments and they do so much faster than SM-MCTS-A. However, SM-OOS is currently the fastest converging MCTS algorithm on all the evaluated domains. Furthermore, we show that even though a Nash equilibrium is the optimal strategy against the worst-case opponent, the speed of convergence to a NE is not a good predictor of the actual performance of algorithms in head-to-head matches in large games. Even though SM-OOS often performed the largest number of iterations and built a comparably deep tree within the time limits, it consistently performed worse than SM-MCTS in these tournaments. In large games, the best empirical performance was achieved by SM-MCTS with our novel Regret Matching selection strategy.

Next we analyze the generic two-player zero-sum extensive-form games with imperfect information. We show that all existing online MCTS algorithms for these games are minor variants of three fundamentally different algorithms that can be described by different search spaces they utilize. We explain that none of them is guaranteed to converge to a Nash equilibrium of the game. We present Online Outcome Sampling, the first online MCTS algorithm that is guaranteed to eventually converge to a NE, given enough time. We prove its convergence formally as well as demonstrate empirically on various domains. We propose novel methods for measuring distance of the strategies computed by online search algorithms from a Nash equilibrium, which are both useful and computable in a reasonable time. Furthermore, we improve IS-MCTS algorithm by approximating the belief distributions and thoroughly experimentally evaluate the convergence properties of the proposed algorithms with various selection functions, one of which has never been used in MCTS before. While even in generic EFGs the distance from NE does not predict the winner of a tournament, the extensive head-to-head tournaments we present show its usefulness in tuning parameters of the algorithms and interpreting results. This thesis provides the first systematic evaluation of convergence and game playing quality of IS-MCTS with various selection functions. The novel regret matching based selection we proposed is often the best-performing selection function.

An important contribution of this thesis are also domain independent reference implementations of the novel algorithms proposed in this thesis, along with implementations of the evaluated existing MCTS algorithms and some of the synthetic domains used for the evaluation. These implementations are a part of a comprehensive framework that includes a range of algorithms for exact and approximate solving of extensive from games, as well as many useful tools for implementing and executing experiments similar to those presented in this thesis. All experiments in Chapter 4 can be easily replicated in the framework. It is publicly available under GNU Lesser General Public License at

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

Finally, we present a case study showing that MCTS algorithms outperform the state-of-the-art algorithm for playing a visibility-based pursuit evasion game. We show that fundamentally different MCTS algorithms are suitable for creating strategies of each of the players and that modelling beliefs in IS-MCTS substantially improves the performance of the algorithm in this realistic domain.

6.2 Future Work

This thesis motivates several lines of future work towards creating well-understood and practical MCTS algorithms for imperfect-information games.

Online Outcome Sampling relaxation The clearly superior speed of convergence of OOS algorithm in both simultaneous-move and fully imperfect-information games has shown that this algorithm is a good start if we want to converge to a NE. However, the practical performance of the algorithms in larger games was a little disappointing. This motivates further improvements of OOS, possibly at the cost of weakening the convergence guarantees. One such direction is to modify the algorithm to start samples only from the current information set. Thanks to the understanding of what exactly makes the algorithm lose the convergence guarantees in this case, we can try to detect those specific situations in the game tree and try to compensate for the error they cause. Another possible direction is trying to reduce the variance of the regret updates in long games by limiting the effective depth of the samples. The probabilities collected for the importance sampling correction could be used only to a limited depth, efficiently solving a slightly different game, in which the very end of the game is assumed to be played randomly in the first moves.

Using additional information about samples Another line of future work should further improve the practical performance of MCTS based on the results of this thesis. One such improvement should be using as much information provided by a sample as possible. The current MCTS algorithms use the information available from the sample very locally. Using additional information about the sample, such as the probability of the sample and its decomposition to the contributions of the individual agents could further improve the performance of current MCTS algorithms. Another novel feature of our algorithms that proved to be useful in simultaneous-move games is using coupled learning dynamics, which primarily tries to approximate the equilibrium in the game and not just play the game competitively and learn the equilibrium as a side effect. Further improvement along this line would be to adapt the coupled dynamics of Excessive Gap Technique with superior asymptotic convergence bounds as the selection function for SM-MCTS.

Confidence on belief distributions The uneven, but strong, effect of using belief distributions in various domains indicates that there are some fundamental properties of the domains that decide if this method is useful. Therefore, it may be interesting to explicitly identify these properties and create methods for their fast automatic detection in a specific domain. This may be based on some form of confidence measure about the current belief, which would indicate whether to use the current belief in the next iterations or rather use the uniform estimate.

Understanding the empirically efficient methods This thesis shows a discrepancy between the ability of the algorithms to converge to game-theoretically optimal solutions and their actual performance in tournaments. This motivates further study of the best performing algorithms from the perspective of game theory. What are the fundamental properties that make them work better than the faster-converging algorithm? Do they remove clearly wrong moves more quickly? Do they find moves that are likely to exploit weak opponents, thanks to their slower convergence to a less exploitable strategy? Formal definition of these notions and a further theoretical and empirical study of the performance guarantees with respect to these notions would be an important future work direction.

Combining online and offline game playing There is a substantial amount of work in offline equilibrium computation in extensive-form games. In this thesis, we summarize the best-performing online game playing methods. However, combinations of the two approaches are still not very well-studied. Substantially new methods need to be developed for complex games, in which an abundant computation time is available before the match is started, but there is still some computation time available during the match. An abstracted version of the large game can be solved in advance and then, during the match, online search on the non-abstracted game can use the abstract solution to initialize the learned strategies and guide the sampling and rollouts. This could create good trade-offs between the memory required to store the abstract strategy and the precision of the decisions made in specific situations in the game.

Bibliography

[ACBF02] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[ACBFS95] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 322–331. IEEE, 1995.

[ACBFS03] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2003.

[Aug11] David Auger. Multiple tree for partially observable monte-carlo tree search. In *Applications of Evolutionary Computation*, pages 53–62. Springer, 2011.

[BCB12] Sébastien Bubeck and Nicolo Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Machine Learning*, 5(1):1–122, 2012.

[BH10] Sourabh Bhattacharya and Seth Hutchinson. On the existence of nash equilibrium for a two-player pursuit-evasion game with visibility constraints. *Int. J. Rob. Res.*, 29(7):831–839, 2010.

[BHRZ13] Nolan Bard, John Alexander Hawkin, Jonathan Rubin, and Martin Zinkevich. The annual computer poker competition. *AI Magazine*, 34(2):112, 2013.

[BKL$^+$13] Branislav Bosansky, Christopher Kiekintveld, Viliam Lisy, Jiri Cermak, and Michal Pechoucek. 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, 2013.

[BKLP14] Branislav Bosansky, Christopher Kiekintveld, Viliam Lisy, and Michal Pechoucek. An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information. *Journal of Artificial Intelligence Research (under review)*, 2014.

[BLC+13] Branislav Bosansky, Viliam Lisy, Jiri Cermak, Roman Vitek, and Michal Pechoucek. 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, 2013.

[BLFS09] Michael Buro, Jeffrey Richard Long, Timothy Furtak, and Nathan R Sturtevant. Improving state evaluation, inference, and search in trick-based card games. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 1407–1413, 2009.

[BLL+14] Branislav Bosansky, Viliam Lisy, Marc Lanctot, Jiri Cermak, and Mark H.M. Winands. Algorithms for Computing Strategies in Two-Player Simultaneous Move Games. *Artificial Intelligence (submitted)*, 2014.

[BM07] Avrim Blum and Yishay Mansour. Learning, regret minimization, and equilibria. In Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, chapter 4. Cambridge University Press, 2007.

[BO99] Tamer Basar and Geert Jan Olsder. Dynamic Noncooperative Game Theory. *SIAM, 2nd edition*, Number 23, 1999.

[BPW+12] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[Bro51] George W Brown. Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, 13(1):374–376, 1951.

[BSCU07] Joris Borsboom, Jahn-Takeshi Saito, Guillaume Chaslot, and JW Uiterwijk. A comparison of monte-carlo methods for phantom go. In *Proc. BeNeLux Conf. Artif. Intell., Utrecht, Netherlands*, pages 57–64, 2007.

[Bub10] Sébastien Bubeck. *Bandits games and clustering foundations.* PhD thesis, PhD thesis, Université Lille 1, 2010.

[Bur03] Michael Buro. Solving the Oshi-Zumo Game. *Advances in Computer Games*, 10:361–366, 2003.

[CBL+06] Nicolo Cesa-Bianchi, Gábor Lugosi, et al. *Prediction, learning, and games*, volume 1. Cambridge University Press Cambridge, 2006.

[CBL14] Jiri Cermak, Branislav Bosansky, and Viliam Lisy. Practical Performance of Refinements of Nash Equilibria in Extensive-Form Zero-Sum

Games. In *Proceedings of European Conference on Artificial Intelligence (ECAI)*, 2014.

[CBSS08] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *AIIDE*, 2008.

[CF10] Paolo Ciancarini and Gian Piero Favini. Monte Carlo tree search in Kriegspiel. *Artificial Intelligence*, 174:670–684, July 2010.

[CM$^+$07] Pierre-Arnaud Coquelin, Rémi Munos, et al. Bandit algorithms for tree search. In *Uncertainty in Artificial Intelligence*, 2007.

[Cou07] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games*, CG'06, pages 72–83. Springer-Verlag, Berlin, Heidelberg, 2007.

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

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

[DTW12] Niek GP Den Teuling and Mark HM Winands. Monte-Carlo Tree Search for the simultaneous move game Tron. In *Proceedings of Computer Games Workshop (ECAI)*, pages 126–141, 2012.

[FB98] Ian Frank and David Basin. Search in games with incomplete information: a case study using bridge card play. *Artificial Intelligence*, 100(1-2):87–123, 1998.

[FD13] Zohar Feldman and Carmel Domshlak. Monte-carlo planning: Theoretically fast convergence meets practical efficiency. In *Proceedings of the Twenty-Ninth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-13)*, pages 212–221, Corvallis, Oregon, 2013. AUAI Press.

[FGBLL02] Patrick Fabiani, H-H González-Baños, J-C Latombe, and David Lin. Tracking an unpredictable target among occluding obstacles under localization uncertainties. *Robotics and Autonomous Systems*, 38(1):31 – 48, 2002.

[Fin07] Hilmar Finnsson. Cadia-player: A general game playing agent. Master's thesis, Reykjavík University, 2007.

[Fin12] Hilmar Finnsson. *Simulation-Based General Game Playing*. PhD thesis, Reykjavík University, 2012.

[FKM05] Abraham D Flaxman, Adam Tauman Kalai, and H Brendan McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 385–394. Society for Industrial and Applied Mathematics, 2005.

[Fud98] Drew Fudenberg. *The theory of learning in games*, volume 2. MIT press, 1998.

[Gin01] Matthew L. Ginsberg. Gib: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, 14(1):303–358, June 2001.

[GLB$^+$12] Richard Gibson, Marc Lanctot, Neil Burch, Duane Szafron, and Michael Bowling. Generalized sampling and variance in counterfactual regret minimization. In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence*, pages 1355–1361, 2012.

[GS07] Sylvain Gelly and David Silver. Combining online and offline knowledge in uct. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 273–280, New York, NY, USA, 2007. ACM.

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

[GWM$^+$06] Sylvain Gelly, Yizao Wang, Rémi Munos, Olivier Teytaud, et al. Modification of uct with patterns in monte-carlo go. *Technical Report RR-6062*, 32:30–56, 2006.

[HGPnS10] Samid Hoda, Andrew Gilpin, Javier Peña, and Tuomas Sandholm. Smoothing techniques for computing nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2):494–512, May 2010.

[HMC00] Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

[HMC01a] Sergiu Hart and Andreu Mas-Colell. A general class of adaptive strategies. *Journal of Economic Theory*, 98(1):26 – 54, 2001.

[HMC01b] Sergiu Hart and Andreu Mas-Colell. *A reinforcement procedure leading to correlated equilibrium.* Springer, 2001.

[HS09] István Harmati and Krzysztof Skrzypczyk. Robot team coordination for target tracking using fuzzy logic controller in game theoretic framework. *Robotics and Autonomous Systems*, 57(1):75 – 86, 2009.

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

[Isa65] Rufus Isaacs. *Differential Games.* John Wiley & Sons, 1965.

[JGGE01] Amir Jafari, Amy Greenwald, David Gondek, and Gunes Ercal. On no-regret learning, fictitious play, and nash equilibrium. In *ICML*, volume 1, pages 226–233, 2001.

[Joh07] Michael Johanson. Robust strategies and counter-strategies: Building a champion level computer poker player. Master's thesis, University of Alberta, 2007.

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

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

[JWBM11] Michael Johanson, Kevin Waugh, Michael Bowling, and Zinkevich Martin. Accelerating best response calculation in large extensive games. In *IJCAI'11: Proceedings of the 22nd international jont conference on Artifical intelligence*, pages 258–265, 2011.

[KE12] Thomas Keller and Patrick Eyerich. Prost: Probabilistic planning based on uct. In *ICAPS*, 2012.

[KMvS96] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2):247–259, 1996.

[KS06] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Frnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer Berlin Heidelberg, 2006.

[KS14] Christian Kroer and Tuomas Sandholm. Extensive-form game abstraction with bounds. In *Proceedings of the fifteenth ACM conference on Economics and computation*, pages 621–638. ACM, 2014.

[Kuh53] Harold W Kuhn. Extensive games and the problem of information. *Contributions to the Theory of Games*, 2(28):193–216, 1953.

[KW82] David M. Kreps and Robert Wilson. Sequential equilibria. *Econometrica*, 1982.

[Lan13] Marc Lanctot. *Monte Carlo Sampling and Regret Minimization for Equilibrium Computation and Decision Making in Large Extensive-Form Games.* PhD thesis, University of Alberta, 2013.

[LBJP09] Viliam Lisy, Branislav Bosansky, Michal Jakob, and Michal Pechoucek. Adversarial search with procedural knowledge heuristic. In *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, 2009.

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

[LBP14] Viliam Lisy, Branislav Bosansky, and Michal Pechoucek. Monte-Carlo Tree Search for Imperfect-Information Pursuit-Evasion Games. In *Transactions on Computational Intelligence and Artificial Intelligence in Games (submitted)*, 2014.

[LBVP10] Viliam Lisy, Branislav Bosansky, Roman Vaculin, and Michal Pechoucek. Agent subset adversarial search for complex non-cooperative domains. In *IEEE Conference on Computational Intelligence and Games*, pages 211–218, 2010.

[Lev89] David NL Levy. *Heuristic Programming in Artificial Intelligence: The First Computer Olympiad.* Ellis Horwood Ltd., 1989.

[LGBB12] Marc Lanctot, Richard Gibson, Neil Burch, and Michael Bowling. No-regret learning in extensive-form games with imperfect recall. In *Proceedings of the Twenty-Ninth International Conference on Machine Learning (ICML 2012)*, 2012.

[LGBBcL97] Steven M. LaValle, Héctor H. González-Baños, Craig Becker, and Jean claude Latombe. Motion strategies for maintaining visibility of a moving target. In *In Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 731–736, 1997.

[Lis14] Viliam Lisy. Alternative selection functions for information set monte carlo tree search. *Acta Polytechnica (to appear)*, 2014.

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

[LLB14] Marc Lanctot, Viliam Lisy, and Michael Bowling. Search in Imperfect Information Games using Online Monte Carlo Counterfactual Regret Minimization. In *AAAI Workshop on Computer Poker and Imperfect Information*, 2014.

[LLW14] Marc Lanctot, Viliam Lisy, and Mark HM Winands. Monte Carlo tree search in simultaneous move games with applications to Goofspiel. In *Computer Games Workshop at IJCAI 2013*, volume 408 of *Communications in Computer and Information Science (CCIS)*, pages 28–43. Springer, 2014.

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

[LR85] Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[LSBF10] Jeffrey Richard Long, Nathan R Sturtevant, Michael Buro, and Timothy Furtak. Understanding the success of perfect information monte carlo sampling in game tree search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 134–140, 2010.

[LWWDT13] Marc Lanctot, Christopher Wittlinger, Mark HM Winands, and Niek GP Den Teuling. 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, 2013.

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

[Mun14] Rémi Munos. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. *Foundations and Trends in Machine Learning*, 7(1):1–129, 2014.

[NKL$^+$12] Peter Novak, Antonin Komenda, Viliam Lisy, Branislav Bosansky, Michal Cap, and Michal Pechoucek. 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.

[NT13] Kien Quang Nguyen and Ruck Thawonmas. 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, 2013.

[OR94] Martin J Osborne and Ariel Rubinstein. *A course in game theory.* MIT press, 1994.

[PCA07] Sandeep Pandey, Deepayan Chakrabarti, and Deepak Agarwal. Multi-armed bandit problems with dependent arms. In *Proceedings of the*

24th International Conference on Machine Learning, ICML '07, pages 721–728, New York, NY, USA, 2007. ACM.

[PdJL11] Marc Ponsen, Steven de Jong, and Marc Lanctot. Computing approximate Nash equilibria and robust best-responses using sampling. *Journal of Artificial Intelligence Research*, 42:575–605, 2011.

[PNS06] Austin Parker, Dana Nau, and VS Subrahmanian. Overconfidence or paranoia? search in imperfect-information games. In *AAAI Conference on Artificial Intelligence*, volume 21, page 1045, 2006.

[PNS10] Austin Parker, Dana Nau, and VS Subrahmanian. Paranoia versus overconfidence in imperfect-information games. In *Heuristics, Probabilities, and Causality: A Tribute to Judea Pearl*, pages 63–87, 2010.

[PSPME12] Pierre Perick, David L. St-Pierre, Francis Maes, and Damien Ernst. Comparison of different selection strategies in monte-carlo tree search for the game of Tron. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, pages 242–249, 2012.

[PWL14] Tom Pepels, Mark H.M. Winands, and Marc Lanctot. Real-Time Monte-Carlo Tree Search in Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*, 2014.

[RA12] Mark Richards and Eyal Amir. Information set generation in partially observable games. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 549–555, 2012.

[RB12] Glenn C. Rhoads and Laurent Bartholdi. Computer solution to the game of pure strategy. *Games*, 3(4):150–156, 2012.

[RN09] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2009.

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

[Rob51] Julia Robinson. An iterative method of solving a game. *Annals of mathematics*, pages 296–301, 1951.

[Rob52] Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58:527–535, 1952.

[Ros71] Sheldon M Ross. Goofspiel — the game of pure strategy. *Journal of Applied Probability*, 8(3):621–625, 1971.

[San10] Tuomas Sandholm. The State of Solving Large Incomplete-Information Games, and Application to Poker . *AI Magazine*, special issue on Algorithmic Game Theory:13–32, Winter 2010.

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

[SFB12] Abdallah Saffidine, Hilmar Finnsson, and Michael Buro. Alpha-beta pruning for games with simultaneous moves. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*, pages 22–26, 2012.

[She02] Brian Sheppard. World-championship-caliber scrabble. *Artificial Intelligence*, 134:241–275, 2002.

[SLB08] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations.* Cambridge University Press, 2008.

[SN95] Stephen JJ Smith and Dana S Nau. An analysis of forward pruning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1386–1386, 1995.

[SRL10] Spyridon Samothrakis, David Robles, and Simon M. Lucas. A UCT agent for Tron: Initial investigations. In *Proceedings of the 2010 IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 365–371, 2010.

[SSS09] Mohammad Shafiei, Nathan Sturtevant, and Jonathan Schaeffer. Comparing UCT versus CFR in Simultaneous Games. In *IJCAI Workshop on General Game Playing*, 2009.

[Sto05] Gilles Stoltz. *Incomplete information and internal regret in prediction of individual sequences.* PhD thesis, PhD thesis, Dept. of Mathematics, University Paris XI, ORSAY, 2005.

[Stu08] Nathan R Sturtevant. An analysis of UCT in multi-player games. *ICGA Journal*, 31(4):195–208, 2008.

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

[TF11] Olivier Teytaud and Sébastien Flory. Upper confidence trees with short term partial information. In *Applications of Evolutionary Computation*, volume 6624 of *Lecture Notes in Computer Science*, pages 153–162. Springer Berlin Heidelberg, 2011.

[TLW14] Mandy JW Tak, Marc Lanctot, and Mark HM Winands. Monte Carlo tree search variants for simultaneous move games. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2014.

[VD91] Eric Van Damme. *Stability and perfection of Nash equilibria.* Springer, 1991.

[VGS08] Marcos Vieira, Ramesh Govindan, and Gaurav S. Sukhatme. Optimal policy in discrete pursuit-evasion games. Technical Report 08-900, Department of Computer Science, University of Southern California, 2008.

[VSG09] Marcos Vieira, Gaurav Sukhatme, and Ramesh Govindan. Scalable and Practical Pursuit-Evasion. *Proceedings of the 2nd International Conference on Robotic Communication and Coordination*, 2009.

[VSK$^+$02] René Vidal, Omid Shakernia, H. Jin Kim, David Hyunchul Shim, and Shankar Sastry. Probabilistic pursuit-evasion games: Theory, implementation and experimental evaluation. *IEEE Transactions on Robotics and Automation*, 18:662–669, 2002.

[Wau09] Kevin Waugh. Abstraction in large extensive games. Master's thesis, University of Alberta, 2009.

[WZJ$^+$09] Kevin Waugh, Martin Zinkevich, Michael Johanson, Morgan Kan, David Schnizlein, and Michael Bowling. A practical use of imperfect recall. In *Proceedings of the 8th Symposium on Abstraction, Reformulation and Approximation (SARA)*, pages 175–182, 2009.

[ZJBP08] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. *Advances in Neural Information Processing Systems (NIPS)*, 20:1729–1736, 2008.

Appendix A

Selected Publications

Publications Related to the Topic of the Thesis

Journals and Book Chapters (5):

- **V. Lisý**. Alternative Selection Functions for Information Set Monte Carlo Tree Search. In *Acta Polytechnica 54(4). 2014 (to appear)* (100%)
- **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)* (80%)
- 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%)
- 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)* (15%)
- **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, Springer Berlin / Heidelberg, 2010. (40%)

In Proceedings (14):

- M. Lanctot, **V. Lisý**, M. Bowling. Search in Imperfect Information Games using Online Monte Carlo Counterfactual Regret Minimization. In *Computer Poker and Imperfect Information Workshop at the AAAI Conference on Artificial Intelligence.* 2014 (45%)
- M. Lanctot, **V. Lisý**, M.H.M. Winands. Monte Carlo Tree Search in Simultaneous Move Games with Applications to Goofspiel. In *Computer Games. Communications in Computer and Information Science.* Volume 408. 2014 (45%)

- J. Čermák, B. Bošanský, and **V. Lisý**. Practical Performance of Refinements of Nash Equilibria in Extensive-Form Zero-Sum Games. In *Proceedings of the 21th European Conference on Artificial Intelligence (ECAI)*. 2014 (10%)

- **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 (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 (30%)

- 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 (17%)

- **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 (35%)

- **V. Lisý**, B.Bošanský, M.Pěchouček: Anytime Algorithms for Multi-agent Visibility-based Pursuit-evasion Games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2012. (75%)

- 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 (20%)

- 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 (15%)

- **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 (45%)

- **V. Lisý**: Adversarial Planning for Large Multi-agent Simulations. In *The 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2009 (100%)

- **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 (40%)

- **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 Intelligence.* 2009 (40%)

Other Publications

Book Chapter (1):

- E.L. van den Broek, **V. Lisý**, J.H.D.M. Westerink, M.H. Schut, K. Tuinenbreijer: Affective Man-Machine Interface: Unveiling Human Emotions through Biosignals. Chapter in *Biomedical Engineering Systems and Technologies. Communications in Computer and Information Science* Volume 52, Springer Berlin / Heidelberg. 2010

In Proceedings (11):

- **V. Lisý**, R. Kessl and T. Pevny: Randomized Operating Point Selection in Adversarial Classification. In *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD), LNCS.* 2014

- **V. Lisý** and R. Píbil: Computing Optimal Attack Strategies Using Unconstrained Influence Diagrams. In *Intelligence and Security Informatics, LNCS.* 2013

- 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.

- 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

- 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

- 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

- B. Bošanský, **V. Lisý** and M. Pěchouček: Towards Cooperation in Adversarial Search with Confidentiality. In *Holonic and Multi-Agent Systems for Manufacturing, LNCS*, 2011
- **V. Lisý**, R. Zivan, K. Sycara and M. Pěchouček: Deception in Networks of Mobile Sensing Agents. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS).* 2010
- **V. Lisý**, M. Jakob, P. Benda, Š. Urban, M. Pěchouček: Towards Cooperative Predictive Data Mining in Competitive Environments. In Agents and Data Mining Interaction, LNCS. 2009
- E.L. van den Broek, **V. Lisý**, J.H.D.M. Westerink, M.H. Schut, K. Tuinenbreijer: Biosignals as an Advanced Man-Machine Interface. *Proceedings of the International Conference on Bio-Inspired Systems and Signal Processing.* 2009
- **V. Lisý**, M. Jakob, J. Tožička, M. Pěchouček: Utility-based Model for Classifying Adversarial Behaviour in Multi-Agent Systems. In *International Multiconference on Computer Science and Information Technology.* 2008

Responses

Follows a list of publications with at least three citations based on Google Scholar as of August 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. 23 citations
- E.L. van den Broek, **V. Lisý**, J.H.D.M. Westerink, M.H. Schut, K. Tuinenbreijer: Affective Man-Machine Interface: Unveiling Human Emotions through Biosignals. *Biomedical Engineering Systems and Technologies. Communications in Computer and Information Science 52.* 2010. 21 citations
- **V. Lisý**, R. Zivan, K. Sycara and M. Pěchouček: Deception in Networks of Mobile Sensing Agents. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS).* 2010. 11 citations
- E.L. van den Broek, **V. Lisý**, J.H.D.M. Westerink, M.H. Schut, K. Tuinenbreijer: Biosignals as an Advanced Man-Machine Interface. *Proceedings of the International Conference on Bio-Inspired Systems and Signal Processing.* 2009. 5 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. 5 citations
- **V. Lisý**, B.Bošanský, M.Pěchouček: Anytime Algorithms for Multi-agent Visibility-based Pursuit-evasion Games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2012. 3 citations