Czech Technical University in Prague
Faculty of Electrical Engineering

Doctoral Thesis

February 2018

Jiří Čermák

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science

Solving Extensive-Form Games Using Imperfect Recall Abstractions

Doctoral Thesis

Jiří Čermák

Prague, February 2018

Ph.D. Programme: Electrical Engineering and Information Technology (P2612)
Branch of study: Information Science and Computer Engineering (2612V025)

Supervisor: prof. Dr. Michal Pěchouček, M.Sc.
Supervisor-Specialist: Mgr. Branislav Bošanský, Ph.D.

Dedicated to my wife Anna and my parents. Thank you for your patience and support.

Acknowledgments

I would like to thank all my colleagues and co-authors that made this thesis possible. Special thanks belong to my supervisor prof. Michal Pěchouček and my supervisor specialist Branislav Bošanský for their support through the years spent as a Ph.D. student.

The work in this thesis was supported by the Czech Science Foundation (grant no. 15-23235S and 18-27483Y), and by the Grant Agency of the Czech Technical University in Prague (grant no. SGS15/205/OHK3/3T/13 and SGS16/235/OHK3/3T/13). Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures".

Abstract

Extensive-form games are an important model of finite sequential interaction between players. The size of the extensive-form representation is, however, often prohibitive and it is the most common cause preventing deployment of game-theoretic solution concepts to real-world scenarios. The state-of-the-art approach to solve this issue is the information abstraction methodology. The information abstraction reduces the size of the original large extensive-form game by removing information available to players; hence merging the information sets which from their decision points. Since the players have to play identical strategy in the merged information sets, the size of the strategy representation in the abstracted game can be significantly smaller than in the original game. The abstracted game is then solved and the small resulting strategies are used in the original game. The majority of existing information abstraction approaches create abstracted games where players remember all their actions and all the information they obtained in the abstracted game – a property denoted as a perfect recall. Remembering all the actions, however, causes the number of decision points of a player (and hence also the size of his strategy) to grow exponentially with the number of actions taken in the past. In this thesis, we focus on imperfect recall information abstractions which allow players to forget and hence have the potential to result in exponentially smaller abstracted games. We provide three major contributions to the state-of-the-art of solving extensive-form games. First, we present a complete picture of the complexity of solving imperfect recall abstractions. Second, we introduce the first family of domain-independent algorithms capable of approximating the strategies with the best worst-case expected value in imperfect recall abstractions. Third, we provide two domain-independent algorithms which can start from arbitrary imperfect recall abstraction of the given extensive-form game and then simultaneously solve and refine this abstraction until guaranteed convergence to the desired approximation of Nash equilibrium of the original game. Finally, we experimentally demonstrate that our algorithms using imperfect recall information abstractions require significantly less memory to solve large extensive-form games compared to the current state-of-the-art algorithms, and hence that they greatly enhance the scalability of solving extensive-form games.

Abstrakt

Hry v extenzivní formě jsou důležitým modelem konečné sekvenční interakce hráčů. Extenzivní reprezentace her je však často příliš velká, což je hlavním důvodem zamezujícím aplikaci herně teoretických konceptů řešení do situací z reálného světa. Nejúspěšnější přístup řešící tento problém je metodologie informačních abstrakcí. Informační abstrakce zmenšují velikost původní extenzivní hry odebíráním informací, které mají hráči k dispozici. To vede ke sjednocování informačních setů tvořících jejich body rozhodnutí. Jelikož hráči musejí hrát identickou strategii ve sjednocených informačních setech, velikost strategie v abstrahované hře může být výrazně menší než v původní hře. Tato abstrahovaná hra je vyřešena a malé strategie získané jejím řešením jsou aplikovány do původní hry. Většina existujících přístupů využívající informační abstrakce vytváří abstrahovanou hru, kde si hráči pamatují všechny své tahy a veškeré informace, které během hry získali – vlastnost nazvaná *perfect recall*. Pamatování si všech akcí však způsobuje exponenciální růst počtu bodů rozhodnutí (a tedy i velikosti strategie) vzhledem k počtu těchto akcí. V této práci se zaměřujeme na imperfect recall informační abstrakce, které umožňují hráčům zapomínat. Tyto abstrakce mají tedy potenciál vést k exponenciálně menším abstrahovaným hrám. V této práci poskytujeme následující tři hlavní kontribuce k řešení her v extenzivní formě. Zaprvé poskytujeme ucelený popis složitosti řešení imperfect recall informačních abstrakcí. Zadruhé představujeme první třídu doménově nezávislých algoritmů schopných aproximovat strategie s nejvyšší očekávanou hodnotou v nejhorším případě v imperfect recall informačních abstrakcích. Zatřetí představujeme dva doménově nezávislé algoritmy, které mohou být inicializovány jakoukoliv imperfect recall abstrakcí řešené hry v extenzivní formě. Tyto algoritmy současně řeší a zlepšují danou abstrakci, dokud není dosaženo garantované konvergence k požadované aproximaci Nashova ekvilibria původní hry. Na závěr experimentálně demonstrujeme, že naše algoritmy využívající imperfect recall informační abstrakce používají mnohem méně paměti na vyřešení obrovských her v extenzivní formě oproti paměti používané současnými nejúspěšnějšími algoritmy pro řešení extenzivních her. Z toho plyne, že námi představené algoritmy umožňují zásadní zvýšení škálovatelnosti řešení extenzivních her.

Contents

Chapter 1

Introduction

The non-cooperative game theory is a tool for modeling and solving strategic interaction of rational players. It provides a notion of optimal behavior in such interaction in the form of solution concepts. The most widely known such solution concept is the *Nash equilibrium* (NE, [46]), which forms a prescription of strategies to players with a guarantee that no player can benefit by deviating from the prescribed strategy. The popularity of NE is caused by its desirable properties in the class of two-player games of pure competition, also known as two-player zero-sum games. The most important of these properties is that the NE strategy guarantees the highest possible payoff against a rational opponent. In games with 3 or more players and games where the utility structure is not competitive (general-sum games), this property no longer holds. Hence, to provide a guaranteed payoff for a player in these classes of games, strategies with the best worst-case payoff (maxmin strategies) need to be used.

Game theory offers two widely used representations for formal modeling of the strategic interaction of players. Simple one-shot scenarios are naturally modeled as *normal-form games* (NFGs) where all players simultaneously choose an action. After that, each player immediately receives utility based on his choice and the choice of the rest of the players. For more complex sequential interactions with stochastic events and partial observability, *extensive-form games* (EFGs) are a more suitable representation. EFGs can model recreational games, such as poker [50], as well as real-world situations in physical security [42], auctions [14], or medicine [51]. Hence, in this thesis, we focus on EFGs. EFGs are visualized as game trees where nodes correspond to states of the game and edges to actions of players. Imperfect information of players is represented by grouping indistinguishable states of a player into information sets, which form the decision point of the player.

There are two approaches to making decisions in EFGs. First, there are online (or game-playing) algorithms which given the observation of the game state compute the action to be played. Second, there are offline algorithms which compute (approximate) the strategy in the whole game and play according to this strategy. The offline algorithms typically provide a better approximation of equilibrium strategies in large games compared to the online algorithms [5]. One exception is the recently introduced continual resolving algorithm used in DeepStack [45], which provides less exploitable strategies than existing

offline algorithms in heads-up no-limit Texas Hold'em poker. The main caveat of this algorithm is that it exploits the specific structure of poker where all actions of players are observable, and its generalization to other games is not straightforward. Another advantage of offline algorithms is that they simplify the deployment in real-world applications since the strategy (a probabilistic distribution over actions in each information set) is precomputed and can be simply stored on any device. It can then be accessed by deployed units such as park rangers (see, e.g., [17]) without the need of large computational resources necessary when using online algorithms. We thus focus on offline algorithms.

Most of the existing offline algorithms [58, 61, 28] require players to remember all the information gained during the game – a property denoted as a *perfect recall*. This requirement comes with a significant disadvantage: The number of decision points, and hence both the memory required during the computation and the memory required to store the resulting strategy, grows exponentially with the number of moves. The reason behind the exponential growth is that the perfect recall allows the player to condition his behavior on all his actions taken in the past (the number of these histories is exponential). For example, the perfect recall extensive-form representation of the heads-up limit Texas hold'em poker has more than 10^{14} decision points [6], storing strategy for a game of this size requires 262 TB of memory [6]. Therefore, a popular approach is to use *information abstractions* [21]. The information abstraction reduces the size of the original large extensive-form game by removing information available to players; hence merging their information sets. Since the players have to play identical strategy in merged information sets, the size of the strategy representation in the abstracted game can be significantly smaller than in the original game. The abstracted game is then solved, and the small resulting strategies are used in the original game. The majority of the existing algorithms (e.g., see [20, 37, 10, 9]) create perfect recall abstractions. However, the requirement of perfect recall in the abstraction severely limits possible memory savings, since the number of decision points in the abstracted game still grows exponentially with the increasing number of moves. To achieve additional memory savings, the assumption of perfect recall may need to be violated in the abstracted game resulting in an *imperfect recall*.

Solving imperfect recall games is known to be a difficult problem [34, 24]. Hence, there is only a limited amount of work that relaxes the perfect recall restriction in abstractions. Very specific imperfect recall abstractions that allow using perfect recall solution techniques are *chance relaxed skew well-formed games* [40, 38] and *normal-form games with sequential strategies* [4, 42]. Chance relaxed skew well-formed games only merge information sets which satisfy strict restrictions on the structure of the game tree above and below them. These restrictions imply that for all possible strategies of the opponent, a strategy which is optimal in one of the merged information sets must have bounded distance from the optimal strategy in the rest of the merged information sets. Even though these restrictions simplify solving of the abstracted game, they prevent us from creating sufficiently small and useful abstracted games and thus fully exploit the possibilities of imperfect recall. In normal-form games with sequential strategies, players cannot observe actions of the opponent at all. As a consequence, only specific scenarios can be modeled as normal-form games with sequential strategies. Existing methods for using imperfect

recall abstractions without severe limitations cannot provide any guarantees of the quality of computed strategies [59].

1.1 Research Goal

The main goal of this thesis is to demonstrate that imperfect recall information abstractions can be used to introduce domain-independent algorithms for solving large EFGs, which require significantly less memory than the current state-of-the-art algorithms. This goal was achieved by taking the following steps:

(1) We provide a complete picture of the complexity of solving imperfect recall games. Most importantly, we provide the analysis of the properties of imperfect recall abstractions which make them hard to solve.

(2) Based on this theoretical analysis we introduce two families of algorithms which use imperfect recall abstractions to reduce the memory required to solve large EFGs and to store the resulting strategies.

The first family of domain-independent algorithms is capable of approximating the strategy with the best guaranteed expected value (maxmin strategy) in the given imperfect recall abstraction. We show that the algorithms are significantly more scalable in the case where we restrict the minimizing player to have a special case of imperfect recall called A-loss recall. We demonstrate that solving this type of abstraction allows us to significantly reduce the memory required to store the resulting strategy.

The second family consists of two domain-independent algorithms which can start with an arbitrary imperfect recall abstraction of the given two-player zero-sum EFG with perfect recall and then simultaneously refine and solve this abstraction until guaranteed convergence to the desired approximation of Nash equilibrium of the original unabstracted EFG. These algorithms have the following differences compared to the first class of algorithms. Both algorithms directly benefit from the reduced size of the solved game also during the computation and not only in the size of the resulting strategies. The algorithms can start from arbitrary imperfect recall abstraction, e.g., provided by domain experts. If no such abstraction is available for a given domain, the algorithms can start from a trivial coarse imperfect recall abstraction (we provide a domain-independent algorithm for constructing such initial abstraction). And finally, the choice of the initial abstraction does not influence the quality of the resulting strategies, since the abstraction is refined during the run of the algorithm to guarantee that the algorithms compute the desired approximation of the Nash equilibrium of the original game.

1.2 Contributions

In this section, we provide a more detailed description of the contributions of this thesis.

1.2.1 Theoretical Properties of Imperfect Recall Abstractions

We provide a complete picture of the complexity of solving imperfect recall games. We demonstrate that the mixed strategy and behavioral strategy representation have different descriptive power in imperfect recall games and show that this difference leads to the loss of the guarantee of the existence of Nash equilibrium in behavioral strategies in imperfect recall extensive-form games. Furthermore, we discuss the numerical representation of Nash equilibrium and maxmin strategies in perfect and imperfect recall games. Next, we present known subsets of imperfect recall games: A-loss recall games and Chance relaxed skew well-formed games. We show that most of the hardness results known for imperfect recall games also extend to A-loss recall games. On the other hand, we provide sufficient and necessary (i.e., if and only if) condition for the existence of Nash equilibrium in A-loss recall games. This result makes A-loss recall games the only subset of imperfect recall games, where such conditions are known. Additionally, we show that A-loss recall property allows us to compute a best response in polynomial time (computing best response is NP-hard in imperfect recall games). Next, we discuss the problems of applying existing algorithms for solving perfect recall extensive-form games to imperfect recall games. Finally, we explain why Chance relaxed skew well-wormed games allow application of perfect recall algorithms and show the relation between A-loss recall games and Chance relaxed skew well-formed games.

1.2.2 Approximating Maxmin Strategies in Imperfect Recall Games Using A-Loss Recall Property

As a part of this work, we provide the following contributions. We provide the first family of algorithms capable of approximating the strategies with the best worst-case expected outcome (maxmin strategies) in imperfect recall games. Additionally, we use the properties of the A-loss recall to significantly improve the scalability of these algorithms. To achieve this result, we require only the minimizing player to have A-loss recall, while the maximizing player is allowed to have imperfect recall. Finally, we experimentally demonstrate that imperfect recall abstractions significantly reduce the size of the resulting strategies and hence greatly simplify their storage.

More specifically, we provide the first approximate algorithm, denoted Imperfect Recall Abstraction Branch-and-Bound algorithm (IRABnB), for computing maxmin strategies in imperfect recall games where the maximizing player has imperfect recall, and the minimizing player has A-loss recall. We base the algorithm on the sequence-form linear program for computing maxmin strategies in perfect recall games [58, 35] extended by bilinear constraints necessary for the correct representation of strategies of the maximizing player in imperfect recall games.

We further extend the IRABnB algorithm by incremental strategy generation technique. The resulting algorithm is denoted Double Oracle Imperfect Recall Abstraction Branch-and-Bound algorithm (DOIRABnB). While such techniques exist for perfect recall games [7], transferring the ideas to imperfect recall games presents a number of challenges that we address in this thesis. The experimental evaluation shows that DOIRABnB

significantly improves the scalability of IRABnB. The algorithm is capable of solving some games with up to $5 \cdot 10^9$ states in approximately 1 hour.

Finally, we experimentally demonstrate the effectiveness of the use of imperfect recall abstractions to reduce the size of strategies to be stored. We show that employing simple abstractions which still allow us to compute the maxmin strategy of the original game can lead to strategies with the relative size as low as 0.03% of the size of the strategy in the original unabstracted game.

This work is presented in Chapter 4 and is based on papers [56, 3, 57, 12].

1.2.3 Algorithms for Constructing and Solving Imperfect Recall Abstractions of Large Extensive-Form Games

In this work, we take a novel approach to imperfect recall information abstractions, which does not require any specific structure of the imperfect recall abstracted game nor does it use computationally complex algorithms to solve it. Instead, we introduce two domain-independent algorithms, which are able to start with an arbitrary imperfect recall abstraction of the solved two-player zero-sum perfect recall EFG. The algorithms simultaneously solve the abstracted game, detect the missing information causing problems and return it to the players. This process is repeated until provable convergence to the desired approximation of the Nash equilibrium of the original game.

The first algorithm is Fictitious Play for Imperfect Recall Abstractions (FPIRA). FPIRA is based on the Fictitious Play (FP, [8]). As a part of the contribution, we discuss the problems of applying FP to the imperfect recall abstraction and how to resolve them. We then demonstrate how to detect the parts of the abstraction that need to be refined to enable convergence to the Nash equilibrium of the original game. We base this detection on the difference between the quality of the strategies expected from running FP directly on the original two-player zero-sum EFG with perfect recall and the result obtained from applying it to the abstraction. Finally, we prove that the guarantee of convergence of FP to the Nash equilibrium of the original two-player zero-sum EFG with perfect recall directly translates to the guarantee of convergence of FPIRA to the Nash equilibrium of this game.

The second algorithm is denoted CFR+ for Imperfect Recall Abstractions (CFR+IRA). CFR+IRA replaces the FP by CFR+ algorithm [54] since CFR+ is known to have significantly faster empirical convergence to the Nash equilibrium in two-player zero-sum EFGs with perfect recall. As a part of the contribution, we describe problems of applying CFR+ directly to the imperfect recall abstraction and how to resolve them. To update the abstraction, we compare the expected theoretical convergence of CFR+ in the original game and the convergence achieved in the abstraction. The abstraction is refined when the observed convergence is slower than the theoretical guarantee provided by CFR+ in the original two-player zero-sum EFG with perfect recall. We prove that CFR+IRA is guaranteed to converge to the Nash equilibrium of the original two-player zero-sum EFG with perfect recall. Finally, we provide an efficient heuristic for the abstraction update and demonstrate that it significantly improves the convergence to the Nash equilibrium of the original EFG.

We assume two possible sources of the initial imperfect recall abstraction for FPIRA and CFR+IRA. First, we assume that the abstraction is provided either by a domain expert or as a result of some heuristic abstraction algorithm. Alternatively, if no such abstraction is available for the solved domain, we provide a domain-independent algorithm, which builds the initial coarse imperfect recall abstraction of the given game.

Both algorithms are conceptually similar to the Double Oracle algorithm (DOEFG, [7]) since they create a smaller version of the original game and repeatedly refine it until the desired approximation of the Nash equilibrium of the original game is found. Our algorithms, however, use imperfect recall information abstractions during the computation, while DOEFG uses a restricted perfect recall game, where the players are allowed to play only a subset of their actions. Hence, the algorithms introduced in this article exploit a completely different type of sparseness than DOEFG.

In the experimental evaluation, we compare the memory requirements and runtime of CFR+IRA, FPIRA, and DOEFG. We demonstrate that CFR+IRA requires at least an order of magnitude less memory than DOEFG and FPIRA to solve a diverse set of domains. Hence it is the most suitable algorithm for solving large domains with prohibitive memory requirements. We show that even if CFR+IRA is initialized with a trivial automatically built abstraction, it requires to build information abstractions with as little as 0.9% of information sets of the original game to find the desired approximation of the NE of the original game. Moreover, the results suggest that the relative size of the abstraction built by CFR+IRA will further decrease as the size of the solved game increases.

This work is presented in Chapter 5 and is based on papers [11, 13].

1.3 Thesis Outline

In Chapter 2 we introduce the EFG representation. In Section 2.1 we provide formal definition of EFGs. In Section 2.2 we discuss the perfect and imperfect recall EFGs and provide an overview of known subclasses of imperfect recall games. As a part of these sections, we describe different representations of strategies in EFGs. We discuss the size of these representations, what strategy representation directly benefits from the reduced number of decision points in the abstracted game and their expressive power in imperfect recall games. In Section 2.3 we formally define the maxmin strategies and NE and discuss the existence of NE in perfect and imperfect recall EFGs.

In Chapter 3 we provide a complete picture of solving EFGs. First, in Section 3.1 we discuss the complexity of computing maxmin and NE strategies in perfect recall and imperfect recall games. In Section 3.2 we follow with the discussion of the numerical representation of Nash equilibrium and maxmin strategies in perfect and imperfect recall games. Next, in Section 3.3 we describe how the perfect and imperfect recall property influences the computation and properties of a best response. In Section 3.4 we provide an overview of the most influential algorithms for computing NE in two-player zero-sum EFGs with perfect recall. In Section 3.5 we discuss the action abstractions and information abstractions. In Section 3.6 we explain why the algorithms designed to solve perfect recall games cannot be applied in imperfect recall games. Finally, in Section 3.6.4 we discuss

a subclass of imperfect recall games specifically designed so that Counterfactual regret minimization (CFR, [61]) is guaranteed to converge to NE.

In Chapter 4 we discuss approximating maxmin strategy in imperfect recall games. First, in Section 4.1 we provide the overview of the relevant state-of-the-art. Next, in Section 4.2.1 we show sufficient and necessary (i.e., if and only if) conditions for the existence of Nash equilibrium in A-loss recall games and show that in imperfect recall games, such conditions are only sufficient. In Section 4.2.2 we discuss the complexity of computing maxmin and Nash equilibrium strategies in A-loss recall games. In Section 4.3 we derive the mixed integer linear program (MILP) for approximating maxmin strategy in imperfect recall game and show that this MILP has size linear to the size of the game if the minimizing player has A-loss recall (its size is exponential in case that the minimizing player has imperfect recall). In Section 4.4 we introduce the algorithms which use the MILP to approximate the maxmin strategies. More specifically, in Section 4.4.1 we introduce the Base algorithm which iteratively solves the MILP until the desired approximation of maxmin strategy is reached. In Section 4.4.2 we describe Imperfect Recall Abstraction Branch-and-Bound algorithm (IRABnB) which uses branch-and-bound search over the linear relaxation of the MILP. In Section 4.4.3 we introduce Double Oracle Imperfect Recall Abstraction Branch-and-Bound algorithm (DOIRABnB) which further extends IRABnB with incremental strategy generation techniques. Finally, in Section 4.5 we provide experimental evaluation of IRABnB and DOIRABnB.

In Chapter 5 we focus on two domain-independent algorithms which start from an arbitrary imperfect recall abstraction of a given game and iteratively solve and refine this abstraction until convergence to the desired approximation of the Nash equilibrium of the original game. First, in Section 5.1 we provide the comparison of the approaches presented in this chapter to the state-of-the-art. In Section 5.2.1 we provide the notation for the abstractions being used during the run of the algorithms. As a part of this section, we present a domain-independent algorithm which creates a coarse imperfect recall abstraction of the given game if no initial abstraction is given. In Section 5.2.2 we introduce the Fictitious Play for Imperfect Recall Abstractions (FPIRA). In Section 5.2.3 we introduce the CFR+ for Imperfect Recall Abstractions (CFR+IRA). Finally in Section 5.3 we compare the memory efficiency and runtime of FPIRA, CFR+IRA, and DOEFG on a set of diverse benchmark domains.

Finally, in Chapter 6 we conclude the thesis and provide directions for future work.

Chapter 2

Introduction to Extensive-Form Games

In this Chapter, we introduce the extensive-form game representation. In Section 2.1 we formally define the extensive-form game and all its components. In Section 2.2 we discuss the perfect and imperfect recall property in extensive-form games and define concepts required for analysis of the effect of imperfect recall. As a part of this section, we discuss the impact of the perfect and imperfect recall on the strategy representation and provide an overview of known subclasses of imperfect recall games from the literature. Finally, in Section 2.3 we introduce maxmin strategies and Nash equilibrium and discuss their properties.

2.1 Extensive-Form Games

A two-player extensive-form game (EFG) is a tuple $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, u, \mathcal{C}, \mathcal{I})$, which is commonly visualized as a game tree (see Figure 2.1).

$\mathcal{N} = \{1, 2\}$ is a set of players, by i we refer to one of the players, and by $-i$ to his opponent. Additionally, the chance player N represents the stochastic environment of the game. $\mathcal{A}$ denotes the set of all actions labeling the edges of the game tree. $\mathcal{H}$ is a finite set of *histories* of actions taken by all players and the chance player from the root of the game. Each history corresponds to a *node* in the game tree; hence, we use the terms history and node interchangeably. $\mathcal{Z} \subseteq \mathcal{H}$ is the set of all *terminal states* of the game corresponding to the leaves of the game tree. For each $z \in \mathcal{Z}$ and $i \in \mathcal{N}$ we define a *utility function* $u_i : \mathcal{Z} \to \mathbb{R}$. If $u_i(z) = -u_{-i}(z)$ for all $z \in \mathcal{Z}$, we say that the game is zero-sum. Chance player selects actions based on a fixed probability distribution known to all players. Function $\mathcal{C} : \mathcal{H} \to [0, 1]$ is the probability of reaching h obtained as the product of probabilities of actions of chance player preceeding h. We further overload $\mathcal{C}$ and use it to denote the probability $\mathcal{C}(a)$ that action a of chance player is taken. Imperfect observation of player i is modeled via *information sets* $\mathcal{I}_i$ that form a partition over $h \in \mathcal{H}$ where i takes action. Player i cannot distinguish between nodes in any $I \in \mathcal{I}_i$. We represent the information sets as nodes connected by dashed lines in the examples. $\mathcal{A}(I)$ denotes

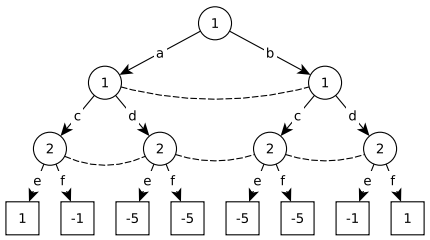

Figure 2.1: An imperfect recall game. Circle nodes represent the states of the game, numbers in the circles show which player acts in that node (player 1, player 2), dashed lines represent indistinguishable states joined to an information set, and box nodes are the terminal states with utility value for player 1 (the game is zero-sum, hence player 1 maximizes the utility, player 2 minimizes it).

actions available in each $h \in I$. The action a uniquely identifies the information set where it is available, i.e., for all distinct $I, I' \in \mathcal{I}$ $\forall a \in \mathcal{A}(I)$ $\forall a' \in \mathcal{A}(I')$ $a \neq a'$. An ordered list of all actions of player i from the root to node h is referred to as a *sequence*, $\sigma_i = seq_i(h)$. Σ_i is a set of all sequences of player i. We use $seq_i(I)$ as a set of all sequences of player i leading to I.

2.1.1 Strategy Representation

There are several representations of strategies in EFGs. A *pure strategy* s_i for player i is a mapping assigning $\forall I_i \in \mathcal{I}_i$ an element of $\mathcal{A}(I_i)$. $\mathcal{S}_i$ is a set of all pure strategies for player i. A *mixed strategy* m_i is a probability distribution over $\mathcal{S}_i$, set of all mixed strategies of i is denoted as $\mathcal{M}_i$. *Behavioral strategy* b_i assigns a probability distribution over $\mathcal{A}(I_i)$ for each $I_i \in \mathcal{I}_i$. $\mathcal{B}_i$ is a set of all behavioral strategies for i, $\mathcal{B}_i^p \subseteq \mathcal{B}_i$ is the set of deterministic behavioral strategies for i. A *strategy profile* is a set of strategies, one strategy for each player. Note that there is a conceptual difference in behavioral and mixed strategies. When using behavioral strategies, players sample the action to be played when reaching an information set. When using mixed strategies, players sample pure strategy before the game starts and follow actions prescribed by this strategy in each information set.

Definition 2.1.1. *A pair of strategies x_i, y_i of player i with arbitrary representation is* realization equivalent *if $\forall z \in \mathcal{Z} : \pi_i^{x_i}(z) = \pi_i^{y_i}(z)$, where $\pi_i^{x_i}(z)$ is a probability that z is reached due to strategy x_i of player i when the rest of the players play to reach z.*

We overload the notation and use u_i as the expected utility of i when the players play according to pure (mixed, behavioral) strategies.

Definition 2.1.2. *We define the* exploitability *of a strategy b_i as*

$$\max_{b'_i \in \mathcal{B}_i} \min_{b'_{-i} \in \mathcal{B}_{-i}} u_i(b'_i, b'_{-i}) - \min_{b'_{-i} \in \mathcal{B}_{-i}} u_i(b_i, b'_{-i}).$$

Informally, the exploitability of a strategy of player i corresponds to the highest loss the player i can suffer for not playing the strategy maximizing his worst case expected outcome.

2.2 Recall

Recall in EFGs corresponds to the memory of players. If the player has *perfect recall*, the player always remembers all the moves he made and all the information he obtained in the past. If the player has *imperfect recall*, some of the information he knew in the past has been lost.

More formally, a game has perfect recall iff $\forall i \in \mathcal{N}\ \forall I_i \in \mathcal{I}_i\ \forall h, h' \in I_i$ holds that $seq_i(h) = seq_i(h')$. If there exists at least one information set where this does not hold (denoted as *imperfect recall information set*), the game has imperfect recall. We use $\mathcal{I}_i^{IR}$ as a set of all imperfect recall information sets of player i.

The perfect recall property was first formalized by Kuhn [39]. In this work, Kuhn showed the equivalence between behavioral strategies and mixed strategies in EFGs with perfect recall. This equivalence closely connected NFGs and EFGs with perfect recall and subsequently allowed to extend all the properties known for NFGs to EFGs with perfect recall. In imperfect recall games, on the other hand, the descriptive power of mixed and behavioral strategies can differ.

Example 2.2.1. *Consider the game depicted in Figure 2.1. This game has 4 pure strategies for player 1:* $\mathcal{S}_1 = \{(a, c), (a, d), (b, c), (b, d)\}$. *A mixed strategy can condition the actions of players on information that the players should no longer have available. For example, a mixed strategy where* (a, c) *and* (b, d) *are played with a uniform probability* 0.5 *allows player 1 to condition playing* c *and* d *on the outcome of his stochastic choice in the root of the game, and thus randomize between the leftmost and the rightmost state in information set of player 2. Note that one cannot model the same behavior using a behavioral strategy that assigns a probability distribution over the actions available in every decision point without conditioning on any previous knowledge. Therefore no additional information can be disclosed to the player.*

Moreover, the size of these representations differs significantly, both in perfect and imperfect recall EFGs. Mixed strategies of player i state probability distribution over $\mathcal{S}_i$, where $|\mathcal{S}_i| \in \mathcal{O}(2^{|\mathcal{Z}|})$, behavioral strategies create probability distribution over the set of actions (therefore, its size is proportional to the number of information sets, which can be exponentially smaller than $|\mathcal{Z}|$). Hence, behavioral strategies are more memory efficient strategy representation. Additionally, when used in combination with information abstractions, behavioral strategies directly benefit from the reduced number of information sets in the abstracted game.

2.2.1 Coarsest Perfect Recall Refinement

To be able to analyze the effect of imperfect recall in any given imperfect recall game G, we need to be able to construct a corresponding perfect recall game G' by adding the minimum

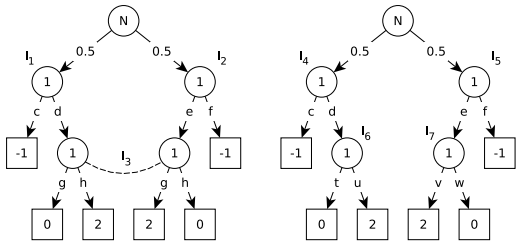

Figure 2.2: (Left) An imperfect recall game. (Right) Its coarsest perfect recall refinement.

amount of information for players to have perfect recall in G. We denote G' as the *coarsest perfect recall refinement* of G. To do this, we first define a partition $H(I_i)$ of states in every information set I_i of some imperfect recall game G to the largest possible subsets, not causing imperfect recall. More formally, let $H(I_i) = \{H_1, ..., H_n\}$ be a disjoint partition of all $h \in I_i$, where $\bigcup_{j=1}^{n} H_j = I_i$ and $\forall H_j \in H(I_i)\ \forall h_k, h_l \in H_j : seq_i(h_k) = seq_i(h_l)$, additionally for all distinct $H_k, H_l \in H(I_i) : seq_i(H_k) \neq seq_i(H_l)$.

Definition 2.2.1. *The coarsest perfect recall refinement G' of the imperfect recall game $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, u, \mathcal{C}, \mathcal{I})$ is a tuple $(\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}', u, \mathcal{C}, \mathcal{I}')$, where $\forall i \in \mathcal{N}\ \forall I_i \in \mathcal{I}_i$, $H(I_i)$ defines the information set partition $\mathcal{I}'$. $\mathcal{A}'$ is a modification of $\mathcal{A}$, which guarantees that $\forall I \in \mathcal{I}'\ \forall h_k, h_l \in I\ \mathcal{A}'(h_k) = \mathcal{A}'(h_l)$, while for all distinct $I^k, I^l \in \mathcal{I}'\ \forall a^k \in \mathcal{A}(I^k)\ \forall a^l \in \mathcal{A}(I^l)\ a^k \neq a^l$. We can limit the coarsest perfect recall refinement to player i and leave the information set structure of $-i$ unchanged.*

We refer to any game created from G' by further splitting its information sets simply as perfect recall refinement of G.

In Figure 2.2 we show an example of an imperfect recall game (left) and its coarsest perfect recall refinement (right). Notice, that in the Definition 2.2.1 we change the labelling of actions described by $\mathcal{A}$ to $\mathcal{A}'$, since we modify the structure of the imperfect information $\mathcal{I}$ to $\mathcal{I}'$ (e.g., actions g, h in Figure 2.2 (left) being relabel to t, u and v, w due to the split of the information set in Figure 2.2 (right)).

2.2.2 Mapping between Games

In this section, we formally introduce functions which allow analysis of a relation between any two games G^x and G where G^x was created from G by merging some information sets.

Let $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, u, \mathcal{C}, \mathcal{I})$ and $G^x = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}^x, u, \mathcal{C}, \mathcal{I}^x)$ such that $\mathcal{I}^x$ is created from $\mathcal{I}$ by joining some of the information sets in $\mathcal{I}$. To formally describe relation between G and G^x we define mappings $\Phi_x : \mathcal{I} \to \mathcal{I}^x$, which for each $I \in \mathcal{I}$ returns the information set containing I in G^x and $\Phi_x^{-1} : \mathcal{I}^x \to \wp(\mathcal{I})$, the inverse of Φ_x. By $\Xi_x : \mathcal{A} \to \mathcal{A}^x$ and $\Xi_x^{-1} : \mathcal{A}^x \to \wp(\mathcal{A})$ we denote the mapping of actions from G to G^x and vice versa.

We say that $I \in \mathcal{I}^x$ is an abstracted information set if $|\Phi_x^{-1}(I)| > 1$. By $\widetilde{\mathcal{I}}^x \subseteq \mathcal{I}^x$ we denote the set of all abstracted information sets in G^x.

Example 2.2.2. *In Figure 2.2 (left) we show an imperfect recall extensive-form game G^x and in Figure 2.2 (right) its coarsest perfect recall refinement G. In this case,*

$$\Phi_x : \{I_4 \to I_1, I_5 \to I_2, I_6 \to I_3, I_7 \to I_3\},$$
$$\Phi_x^{-1} : \{I_1 \to \{I_4\}, I_2 \to \{I_5\}, I_3 \to \{I_6, I_7\}\},$$
$$\Xi_x : \{c \to c, d \to d, e \to e, f \to f, t \to g, u \to h, v \to g, w \to h\},$$
$$\Xi_x^{-1} : \{c \to \{c\}, d \to \{d\}, e \to \{e\}, f \to \{f\}, g \to \{t, v\}, h \to \{u, w\}\},$$
$$\widetilde{\mathcal{I}}^x = \{I_3\}.$$

2.2.3 Relevant Subclasses of Imperfect Recall Games

Here we introduce the relevant subclasses of imperfect recall games introduced in the literature. We start with A-loss recall games [31, 33], where every loss of a memory of a player can be tracked back to forgetting his own action. Next, we present three subclasses of imperfect recall games, Well-formed games, Skew well-formed games and Chance relaxed skew well-formed games. These subclasses are specifically designed as suitable results of imperfect recall information abstractions which allow application of Counterfactual Regret Minimization algorithm (CFR, [61]) with a guarantee of convergence to the NE of the original unabstracted game.

2.2.3.1 A-loss Recall Games

Here we define the subclass of imperfect recall games called A-loss recall games [31, 33].

Definition 2.2.2. *Player i has* A-loss recall *if and only if for every $I \in \mathcal{I}_i$ and nodes $h, h' \in I$ it holds either (1) $seq_i(h) = seq_i(h')$, or (2) $\exists I' \in \mathcal{I}_i$ and two distinct actions $a, a' \in \mathcal{A}_i(I'), a \neq a'$ such that $a \in seq_i(h) \wedge a' \in seq_i(h')$.*

Condition (1) in the definition says that if player i has perfect recall then he also has A-loss recall. Condition (2) can be interpreted as requiring that each loss of memory of A-loss recall player can be traced back to some loss of memory of the player's own previous actions in one information set. Hence, in A-loss games, the use of mixed strategies completely compensates the information hidden to players due to the imperfect recall.

The imperfect recall game in Figure 2.1 has A-loss recall since the first common predecessor of the states in the imperfect recall information set of player 1 is the root state of player 1. Hence, the only information the player 1 forgets is his choice in the root state (player 2 has perfect recall in this game). On the other hand, the imperfect recall game in the Figure 2.2 (left) does not have A-loss recall. This is caused by the fact that the first common predecessor of the states in I_3 is a chance node. Hence player 1 does not only forget information about playing d or e but also the information about the outcome of the stochastic event represented by the chance node. Note that if the states of player 1 directly after the actions of the chance player were merged into an information set, the

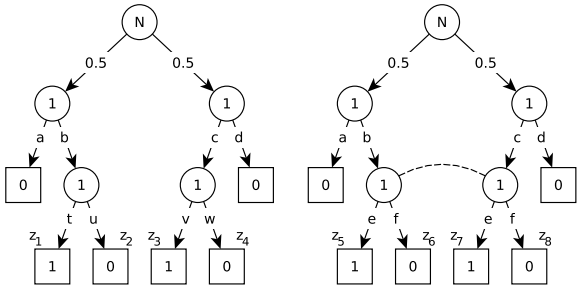

Figure 2.3: (left) A perfect recall EFG (right) Well-formed Game with respect to the perfect recall EFG.

game would have A-loss recall, since the first common predecessor of the states in the imperfect recall information set is the information set of player 1.

2.2.3.2 Well-Formed Games

Well-Formed Games (WFG) are a subset of imperfect recall games defined as suitable results of imperfect recall information abstraction which allow application of perfect recall algorithms, namely CFR, without loosing theoretical guarantees for convergence to NE of the original unabstracted game [40].

Definition 2.2.3. *We say that the imperfect recall game G is well-formed with respect to some perfect recall refinement G' of G if for all $i \in \mathcal{N}$, $I \in \mathcal{I}_i$, I', I'' (where I', I'' are information sets in G' which are unified to I) there exists a bijection $\alpha : \mathcal{Z}_{I'} \to \mathcal{Z}_{I''}$ and constants $k_{I',I''}, l_{I',I''} \in [0, \infty)$ such that for all $z \in \mathcal{Z}_{I'}$:*

1. $u_i(z) = k_{I',I''} u_i(\alpha(z))$,
2. $\mathcal{C}(z) = l_{I',I''} \mathcal{C}(\alpha(z))$,
3. *in G,* $seq_{-i}(z) = seq_{-i}(\alpha(z))$, *and*
4. *in G,* $seq_i(z[I'], z) = seq_i(\alpha(z)[I''], \alpha(z))$,

where $\mathcal{Z}_I$ stands for terminal states reachable from states in I, $z[I]$ denotes the state in I reached when moving from the root of the game to z and $seq_i(h, h')$ is a sequence of actions needed to reach h' from h. We say that G is a well-formed game (WFG) if it is well-formed with respect to some perfect recall refinement.

In Figure 2.3 (right) we present a WFG with respect to the EFG from Figure 2.3 (left) with $\alpha : \{z_1 \to z_5, z_2 \to z_6, z_3 \to z_5, z_4 \to z_6\}$. In Figure 2.4 (right) we present a game which is not WFG with respect to the EFG from 2.4 (left), since there is no mapping of leaves which would satisfy Condition 1 from Definition 2.2.3.

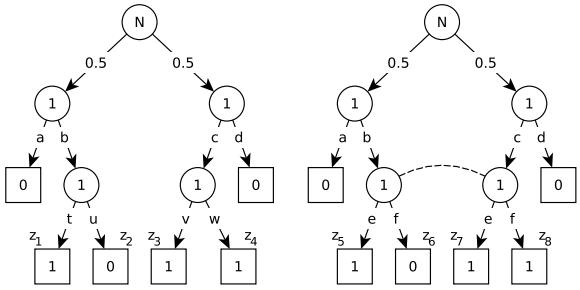

Figure 2.4: (left) A perfect recall EFG (right) Skew Well-formed Game with respect to the perfect recall EFG.

2.2.3.3 Skew Well-Formed Games

Skew Well-Formed games (SWFG) are a generalization of WFG which still allow the CFR to converge to strategies with a fixed distance from NE of the original unabstracted game [40].

Definition 2.2.4. *We say that the imperfect recall game G is skew well-formed with respect to some perfect recall refinement G' of G if for all $i \in \mathcal{N}$, $I \in \mathcal{I}_i$, I', I'' (where I', I'' are information sets in G' which are unified to I) there exists a bijection $\alpha : \mathcal{Z}_{I'} \to \mathcal{Z}_{I''}$ and constants $k_{I',I''}, \delta_{I',I''}, l_{I',I''} \in [0, \infty)$ such that for all $z \in \mathcal{Z}_{I'}$:*

1. $\left|u_i(z) - k_{I',I''} u_i(\alpha(z))\right| \leq \delta_{I',I''}$,
2. $\mathcal{C}(z) = l_{I',I''} \mathcal{C}(\alpha(z))$,
3. *in G,* $seq_{-i}(z) = seq_{-i}(\alpha(z))$, *and*
4. *in G,* $seq_i(z[I'], z) = seq_i(\alpha(z)[I''], \alpha(z))$,

We say that G is skew well-formed game (SWFG) if it is skew well-formed with respect to some perfect recall refinement.

In Figure 2.4 (right) we present a SWFG with respect to the EFG from Figure 2.4 (left) with $\alpha : \{z_1 \to z_5, z_2 \to z_6, z_3 \to z_5, z_4 \to z_6\}$. On the other hand, in Figure 2.5 we present a game where no information set can be merged to create SWFG with respect to this game. The reason is that the leaves z_4 and z_5 cannot be mapped to any of the z_1, z_2, z_3 without breaking the Condition 4 from Definition 2.2.4.

2.2.3.4 Chance Relaxed Skew Well-Formed Games

Finally, Chance Relaxed Skew Well-Formed Games (CRSWFG) further generalize SWFG, while still providing the guarantee of CFR to converge to strategies with a fixed distance from NE of the original unabstracted game [38].

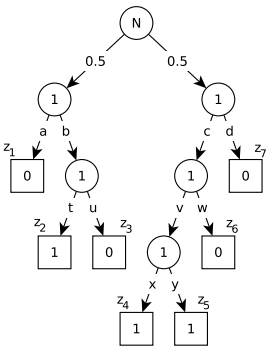

Figure 2.5: A game where no information set can be merged to create SWFG with respect to this game.

Definition 2.2.5. *We say that the imperfect recall game G is chance relaxed skew well-formed with respect to some perfect recall refinement G' of G if for all $i \in \mathcal{N}$, $I \in \mathcal{I}_i$, I', I'' (where I', I'' are information sets in G' which are unified to I) there exists a bijection $\alpha : \mathcal{Z}_{I'} \to \mathcal{Z}_{I''}$ and constants $k_{I',I''}, \delta_{I',I''}, l_{I',I''} \in [0, \infty)$ such that for all $z \in \mathcal{Z}_{I'}$:*

1. *in G, $seq_{-i}(z) = seq_{-i}(\alpha(z))$, and*
2. *in G, $seq_i(z[I'], z) = seq_i(\alpha(z)[I''], \alpha(z))$,*

We say that G is chance relaxed skew well-formed game (CRSWFG) if it is skew well-formed with respect to some perfect recall refinement.

Additionally, following error terms are computed

$$|u_i(z) - k_{I',I''} u_i(\alpha(z))| \leq \varepsilon^R_{I',I''}(z), \tag{2.1}$$

$$|\mathcal{C}(z[I'], z) - \mathcal{C}(\alpha(z)[I''], \alpha(z))| = \varepsilon^C_{I',I''}(z), \tag{2.2}$$

$$\left| \frac{\mathcal{C}(z[I'])}{\mathcal{C}(I')} - \frac{\mathcal{C}(z[I''])}{\mathcal{C}(I'')} \right| = \varepsilon^D_{I',I''}. \tag{2.3}$$

Hence, CRSWFG do not pose any restrictions on the structure of the utility in the reachable leaves, nor the probability that they are reached due to nature. Instead, they keep track of the error terms describing the structure of the utility and nature probability. These error terms are then used to devise the bound on the approximation of NE of G that can be computed in G'.

2.3 Solution Concepts

In this section we introduce maxmin strategies, Nash equilbrium and its approximation ε-Nash equilibrium.

2.3.1 Maxmin Strategy

Maxmin strategy for player i maximizes the expected utility of i assuming that the player $-i$ will play to minimize the expected utility of i, completely disregarding his own expected utility. Hence, the expected value of the maxmin strategy can be seen as the guaranteed expected outcome in the worst case scenario in both zero-sum and general-sum games.

Definition 2.3.1. *We say that* b_i^* *is a* maxmin strategy *iff*

$$b_i^* = \arg\max_{b_i \in \mathcal{B}_i} \min_{b_{-i} \in \mathcal{B}_{-i}} u_i(b_i, b_{-i}).$$

2.3.1.1 ε-Maxmin Strategy

For computational reasons, the exact maxmin strategy might be difficult to obtain. Hence, next we provide the definition of approximation of maxmin strategy.

Definition 2.3.2. *We say that* b_i^* *is a* ε-maxmin strategy *iff*

$$\max_{b_i \in \mathcal{B}_i} \min_{b_{-i} \in \mathcal{B}_{-i}} u_i(b_i, b_{-i}) - \min_{b_{-i} \in \mathcal{B}_{-i}} u_i(b_i^*, b_{-i}) \leq \varepsilon.$$

2.3.2 Nash Equilibrium

A strategy profile is a Nash equilibrium [46] if and only if no player wants to deviate to a different strategy.

Definition 2.3.3. *We say that strategy profile* $b = \{b_i^*, b_{-i}^*\}$ *is a* Nash equilibrium *(NE) in behavioral strategies iff* $\forall i \in \mathcal{N}\ \forall b_i' \in \mathcal{B}_i : u_i(b_i^*, b_{-i}^*) \geq u_i(b_i', b_{-i}^*)$.

The NE using mixed strategies is guaranteed to exist in all finite games [46], and hence the same holds for NE in behavioral strategies in perfect recall EFGs because of the equivalence of mixed and behavioral strategies in perfect recall EFGs [39].

By restricting to two-player zero-sum games, we ensure that NE has a number of desirable properties, which make it a widely used solution concept in this class of games. Namely, all NE have the same expected value for both players (the expected value for player 1 is called the value of the game). Furthermore, NE strategy corresponds to the maxmin strategy. Hence the expected values of players in NE correspond to the best worst case expected utilities the players can obtain and any mistake of their opponent can only increase their expected utility. Finally, the strategies forming NE are interchangeable. I.e., let $b = \{b_i, b_{-i}\}$ and $b' = \{b_i', b_{-i}'\}$ be two NE. Then also $\{b_i, b_{-i}'\}$ and $\{b_i', b_{-i}\}$ form NE of G.

When there are more than two players or when the game is general-sum the NE loses these desirable properties. The expected values of players under different NE strategy profiles can differ, the equivalence between NE strategies and maxmin strategies no longer holds and finally combining strategies from two NE strategy profiles need not yield another NE.

2.3.2.1 ε-Nash Equilibrium

For computational reasons, the exact NE might be difficult to obtain. Hence, next we provide the definition of approximation of NE.

Definition 2.3.4. *We say that strategy profile $b = \{b_i^*, b_{-i}^*\}$ is an* ε-Nash equilibrium *(ε-NE) in behavioral strategies iff $\forall i \in \mathcal{N}\ \forall b_i' \in \mathcal{B}_i : u_i(b_i^*, b_{-i}^*) \geq u_i(b_i', b_{-i}^*) - \varepsilon$.*

Informally, a strategy profile is an ε-NE if and only if no player can gain more than ε by deviating to a different strategy.

2.3.2.2 Nash Equilibrium in Imperfect Recall Games

The guarantee of the existence of NE in behavioral strategies in perfect recall EFGs does not transfer to imperfect recall EFGs, since, as discussed in Section 2.2, the equivalence between mixed strategies and behavioral strategies does not hold in imperfect recall EFGs.

Proposition 2.3.1. *The existence of NE in behavioral strategies is not guaranteed even in two-player zero-sum A-loss recall games.*

Proof. In Figure 2.1 we present the imperfect recall game where there is no NE in behavioral strategies due to Wichardt [60]. This game has A-loss recall since only player 1 has imperfect recall and he forgets only his own choice in the root. This implies that the existence of NE is not guaranteed even in two-player zero-sum A-loss recall games. □

Chapter 3

Solving Extensive-Form Games

In this Chapter, we present a complete picture of computing maxmin strategies and Nash equilibrium in two-player zero-sum EFGs. First, in Section 3.1 and 3.2 we discuss the complexity of computing maxmin and Nash equilibrium strategies and their numerical representation in perfect recall and imperfect recall games. Next, in Section 3.3 we focus on the computation of the best response since best response is one of the main components of the algorithmic game theory. More specifically, we discuss how the perfect and imperfect recall influences the properties of best response and the complexity of its computation. In Section 3.4, we follow with an overview of the most successful algorithms for solving two-player zero-sum extensive-form games with perfect recall. In Section 3.5 we discuss action and information abstractions and give an overview of the problems and successes of these approaches. Next, in Section 3.6 we discuss why algorithms for solving perfect recall games fail in imperfect recall games. Finally, as a part of this section, we present the only known subclasses of imperfect recall games where the Counterfactual regret minimization algorithm [61] is guaranteed to have bounded regret.

3.1 Complexity

Computing the maxmin strategies in two-player perfect recall EFGs is known to be polynomial problem [34]. Hence, from the equivalence of maxmin and NE strategies in zero-sum perfect recall EFGs, also computing NE in two-player zero-sum perfect recall EFGs is a polynomial problem. On the other hand, computing maxmin strategies is NP-hard in imperfect recall EFGs [34], and it is NP-hard to decide whether there exists a NE in behavioral strategies in imperfect recall EFGs [24]. For completeness, we provide both theorems stating the NP-hardness and their proofs below.

Theorem 3.1.1. *The problem of deciding whether player 2 having an imperfect recall can guarantee an expected payoff of at least λ is* NP*-hard even if player 1 has perfect recall, there are no chance moves and the game is zero-sum [34].*

Proof. The proof is made by reduction from 3-SAT problem. The example of the reduction is shown in Figure 3.1. Given n clauses $x_{j,1} \vee x_{j,2} \vee x_{j,3}$ we create a two-player zero-sum

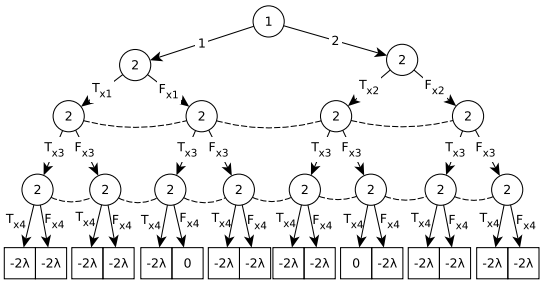

Figure 3.1: An imperfect recall game reduction from Theorem 3.1.1 of 3-SAT problem $x_1 \vee \neg x_3 \vee x_4 \wedge \neg x_2 \vee x_3 \vee \neg x_4$.

game in a following way. In the root of the game player 1 chooses between n actions, each corresponding to one clause. Each action of player 1 leads to a state of player 2. Every such state of player 2 corresponds to the variable $x_{j,1}$ where j is the index of the clause chosen in the root of the game. Every such state has actions $T_{x_{j,1}}, F_{x_{j,1}}$ available, these actions correspond to setting the variable $x_{j,1}$ to true or false respectively. After both $T_{x_{j,1}}, F_{x_{j,1}}$ in $x_{j,1}$ we reach the state representing the assignment to $x_{j,2}$ with the same setup (state representing the assignment to $x_{j,3}$ is reached after that). After the assignment to $x_{j,3}$ we reach the terminal state with utility $-\lambda$ for player 1 if the assignment to $x_{j,1}$, $x_{j,2}$ and $x_{j,3}$ satisfies the clause $x_{j,1} \vee x_{j,2} \vee x_{j,3}$, 0 otherwise. The information sets of player 2 group together all the states corresponding to the assignment to one variable in the original 3-SAT problem (note that we assume that the order of variables in every clause follows some complete ordering on the whole set of variables in the 3-SAT problem).

We will show that player 2 can guarantee the worst case expected value λ if and only if the original 3-SAT problem is satisfiable. First, we show that if the original 3-SAT problem is satisfiable player 2 can guarantee the worst case expected value λ. The worst case expected value λ is achieved when player 2 plays according to the assignment which satisfies the original 3-SAT problem. Next, we show that if player 2 can guarantee the worst case expected value λ, the original 3-SAT problem has to be satisfiable. This holds since if there would be at least one clause not satisfied, player 1 will always choose the action corresponding to this clause, causing the expected value smaller than λ.

The reduction is polynomial since the game has $2^3 n$ leaves. □

Theorem 3.1.2. *It is* NP*-hard to check whether there exists a Nash equilibrium in behavioral strategies in two player imperfect recall games even if the game is zero-sum and there are no chance moves [24].*

Proof. The proof is by reduction from 3-SAT. Given a 3-CNF formula F with n clauses we construct a zero-sum two-player game G as follows. Player 1 (the max-player) starts the game by making two actions, each time choosing one of n clauses of F. We put all corresponding $n + 1$ nodes (the root plus n nodes in the next layer) in one information set. If he fails to choose the same clause twice, he receives a payoff of $-n^3$ and the game

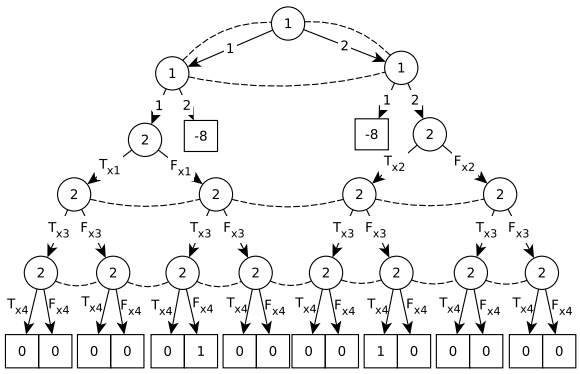

Figure 3.2: An A-loss recall game reduction from Theorem 3.1.2 of 3-SAT problem $x_1 \vee \neg x_3 \vee x_4 \wedge \neg x_2 \vee x_3 \vee \neg x_4$.

stops. Otherwise, the game continues in the same way as in the proof of Theorem 3.1.1. If the choices of player 2 satisfy the clause, player 1 receives payoff 0. If none of them do, player 1 receives payoff 1. An example of the reduction is shown in Figure 3.2.

The proof is now concluded by the following claim: G has an equilibrium in behavior strategies if and only if F is satisfiable. Assume first that F is satisfiable. G then has the following equilibrium (which happens to be pure): player 2 plays according to a satisfying assignment while player 1 uses an arbitrary pure strategy. The payoff is 0 for both players, and no player can modify their behavior to improve this, so we have an equilibrium. Next, assume that G has an equilibrium. We shall argue that F has a satisfying assignment. We first observe that player 1 in equilibrium must have expected payoff at least 0. If not, he could switch to an arbitrary pure strategy and would be guaranteed a payoff of at least 0. Now, look at the two actions (i.e., clauses) that player 1 is most likely to choose. Let clause i be the most likely and let clause j be the second-most likely. If player 1 chooses i and then j, he gets a payoff of $-n^3$. His maximum possible payoff is 1, and his expected payoff is at least 0. Hence, we must have that $-n^3 p_i p_j + 1 \geq 0$. Since $p_i \geq \frac{1}{n}$, we have that $p_j \leq \frac{1}{n^2}$. Since clause j was the second most likely choice, we in fact have that $p_i \geq 1-(n-1)(\frac{1}{n^2}) > 1-\frac{1}{n}$. Thus, there is one clause that player 1 plays with probability above $1-\frac{1}{n}$. Player 2 could then guarantee an expected payoff of less than $\frac{1}{n}$ for player 1 by playing any assignment satisfying this clause. Since we are playing an equilibrium, this would not decrease the payoff of player 1, so player 1 currently has an expected payoff less than $\frac{1}{n}$. Now, look at the assignment defined by the most likely choices of player 2 (i.e., the choices he makes with probability at least 0.5, breaking ties in an arbitrary way). We claim that this assignment satisfies F. Suppose not. Then there is some clause not satisfied by F. If player 1 changes his current strategy to the pure strategy choosing this clause, he obtains an expected payoff of at least $(\frac{1}{2})^3 \geq \frac{1}{n}$ (supposing, wlog, that $n \geq 8$). This contradicts the equilibrium property, and we conclude that the assignment, in fact, does satisfy F. □

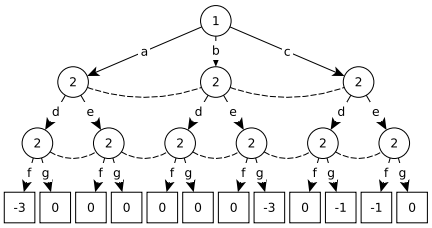

Figure 3.3: An A-loss recall game where all maxmin strategies and NE require irrational numbers [34].

3.2 Representation of Nash Equilibrium and Maxmin Strategies

In this section, we discuss the numerical representation of strategies in perfect recall and imperfect recall games. In two-player perfect recall games with rational payoffs, there always exists a maxmin behavioral strategy which uses only rational probabilities [34]. In imperfect recall games, this no longer holds [34]. Here, we present the example of the imperfect recall game provided in [34], where all maxmin strategies require irrational numbers and show that this game has A-loss recall. Moreover, since the maxmin strategies form a part of all the NE strategies of this game, we extend this result also to NE of A-loss recall games. It follows that computing exact maxmin and NE strategies requires exact representation of irrational numbers even in A-loss recall games.

Theorem 3.2.1. *All the maxmin strategies may require irrational numbers, even in two-player zero-sum A-loss recall game with rational payoffs.*

Proof. The example of the imperfect recall game used in [34] is depicted in Figure 3.3. The maxmin strategy of player 2 is trying to maximize

$$\min\{3b_2(d)b_2(f), 3(1-b_2(d))(1-b_2(f)), b_2(d)(1-b_2(f)) + (1-b_2(d))b_2(f)\}. \tag{3.1}$$

This is maximized when

$$3b_2(d)b_2(f) = 3(1-b_2(d))(1-b_2(f)) = b_2(d)(1-b_2(f)) + (1-b_2(d))b_2(f), \tag{3.2}$$

which leads to $b_2(d) = 0.1(5 \pm \sqrt{5})$.

This game has A-loss recall since the only information player 2 forgets is his own choice in his first information set. □

Theorem 3.2.2. *All the Nash equilibrium strategies may require irrational numbers, even in two-player zero-sum A-loss recall game with rational payoffs.*

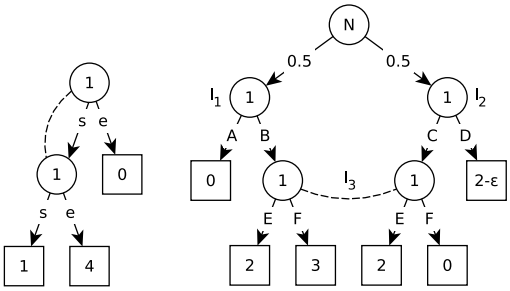

Figure 3.4: (Left) Absentminded Driver. (Right) An EFG without A-loss recall where a time consistent best response (playing the best action in an information set) is not necessarily the ex-ante best response.

Proof. Strategy profiles $b_1(a) = b_1(b) = 0.2, b_2(d) = b_2(g) = 0.1(5 \pm \sqrt{5})$ form all the Nash equilibria of the game in Figure 3.3. This holds since none of the players wants to deviate and the strategies for player 2 are the only solutions of eq. (3.2). Hence, it follows that any other strategy of player 2 has worse expected value against the best responding opponent, and therefore cannot be stable. □

3.3 Best Response Computation

One of the main computational components in algorithmic game theory is the problem of computing a best response since it forms a subproblem of many existing algorithms (e.g., [7, 8]). Formally, a strategy of a player (e.g., pure, mixed, behavioral) is a best response to the given strategy of his opponent if its expected utility is maximal against this strategy compared to all other strategies from the particular class. To denote the best response regardless of the type of the strategy (i.e., regardless whether we consider mixed or behavioral strategies), we use the term *ex-ante best response*.

In perfect recall EFGs, it is sufficient to consider a pure best response. However, this is no longer true in imperfect recall EFGs. Consider a one-player game called the absentminded driver [47] depicted on the left in Figure 3.4. We see that by playing any pure, or even a mixed strategy, the player cannot reach outcome higher than 1. Note that, as described in Section 2.1.1, in a mixed strategy a player samples a pure strategy from the given distribution before the game begins; hence, there is no randomization when the information set is reached, and player 1 always follows the pure strategy sampled from the mixed strategy. When using a behavioral strategy, however, player samples the action from a given distribution independently every time an information set is reached. Hence, the ex-ante optimal strategy is a behavioral strategy $b(s) = \frac{2}{3}$ that reaches the expected value of $\frac{4}{3}$. In general, we need to consider randomized best responses in so-called *absent minded* games (games where there exists a path from the root of the game tree to some leaf such that at least one information set $I_i \in \mathcal{I}_i$ is visited more than once). In the following

text, we will assume that the games are without absentmindedness, where it is sufficient to consider only pure strategies to find an ex-ante best response.

Lemma 3.3.1. *Let G be an imperfect recall game without absentmindedness and b_1 strategy of player 1. There exists an* ex-ante *pure best response of player 2.*

Proof. If the strategy of player 1 is fixed, finding the best response for player 2 corresponds to finding an optimum of a function over a closed convex polytope of all possible behavioral strategies of player 2 created by constraining the strategy by

$$\sum_{a \in A(I)} b_2(a) = 1, \forall I \in \mathcal{I}_2.$$

Vertices of this polytope are formed by pure behavioral strategies. Since each action can be chosen at most once in a game without absentmindedness, the objective function is multilinear in a form

$$\sum_{z \in \mathcal{Z}} C(z) b_1(z) u_2(z) \prod_{a \in seq_2(z)} b_2(a).$$

Notice that, thanks to the assumption of no absentmindedness, the variables $b_2(a)$ in every product always describe behavior at most once for one information set and so the variables are independent. Hence, an optimum must be in one of the vertices of this polytope – that is a pure behavioral strategy. □

The problem of finding an ex-ante best response in imperfect recall games without absentmindedness is still NP-hard (follows from complexity results in [34]) while the problem is easy (polynomial) in perfect recall games. The main difference is that in imperfect recall games an ex-ante best response cannot be found by selecting an action with the highest expected utility to be played in each information set (called a *time consistent strategy* [33]). This is caused by the fact that the belief in an information set of a player is not perfectly determined by the strategy of the opponent and nature, but also by the strategy of the best-responding player.

Example 3.3.1. *Consider the game in Figure 3.4 (right) between player 1 and chance. The ex-ante best response of player 1 in this game is to play B,D,F getting the utility of $\frac{5-\varepsilon}{2}$. Note, however, that since the belief of player* 1 *in his imperfect recall information sets depends on his behavior above the information set, one can reach a time consistent strategy playing B,C,E with the expected utility of* 2*. This strategy is time consistent since when checking every information set separately, there is no deviation of player 1, which could increase his expected value. Note that player 1 does not have A-loss recall in the game in Figure 3.4 (right) since parents of the nodes in the information set I_3 are in two distinct information sets I_1, I_2 and their common predecessor is a chance node.*

Lemma 3.3.2. *Time consistent strategies and ex-ante best responses are guaranteed to be equivalent for player i in a game G if and only if i has A-loss recall in G [33].*

Proof. See proof of Theorem 1 in [33]. □

Consequently, the computation of the best response is in P in A-loss recall games [31, 33]. The following lemmas are a consequence of these facts.

Lemma 3.3.3. *Let G be an imperfect recall game where player i has A-loss recall. Let G' be the coarsest perfect recall refinement of G for player i. Every pure behavioral strategy b'_i of player i from G' has realization equivalent pure behavioral strategy b_i in G and vice versa.*

Proof. First we show how b_i is constructed from b'_i. Consider an information set I of player i in G and corresponding information sets $I^1, \ldots, I^j$ created according to $H(I)$ in the coarsest perfect refinement G' of G.

Let us first show that at most one of information sets $I^1, \ldots, I^j$ can be reached when players play according to strategy profile (b'_i, b_{-i}), for every $b_{-i} \in \mathcal{B}_{-i}$ in G'. Due to A-loss recall of player i, for every pair of nodes h_k, h_l from two different information sets I^k, $I^l \in \{I^1, \ldots, I^j\}$, there exists an information set I' of player i and two distinct actions $a, a' \in \mathcal{A}_i(I'), a \neq a'$ such that $a \in seq_i(h_k) \wedge a' \in seq_i(h_l)$. However, since b'_i is a pure strategy, only one action among the pair of actions a, a' can be played with a non-zero probability and consequently, only one information set in every pair $I^k, I^l \in \{I^1, \ldots, I^j\}$ can be reached.

We use this property to construct b_i. For information set I from G that is divided into information sets $I^1, \ldots, I^j$ in G' we define $b_i(I, a) = 1$ for action $a \in \mathcal{A}(I^k)$, where $b'_i(I^k, a) = 1$ and I^k is the only reachable set from $I^1, \ldots, I^j$ as shown above. If no information set from $I^1, \ldots, I^j$ is reachable, we set b_i in I arbitrarily. For all information sets $I' \in \mathcal{I}'$ that are not split in the coarsest perfect recall refinement (and therefore are the same as in G), we set $b_i(I') = b'_i(I')$. Due to the construction, the realization equivalence between b'_i and b_i follows immediately.

The same construction yields realization equivalent strategy also in the opposite direction. □

Lemma 3.3.4. *Let G be an imperfect recall game where player 2 has A-loss recall and b_1 a strategy of player 1. Let G' be the coarsest perfect recall refinement of G for player 2. Let b'_2 be a pure best response to b_1 in G' and let b_2 be a realization equivalent behavioral strategy to b'_2 in G, then b_2 is a pure best response to b_1 in G.*

Proof. Note that b_1 is a valid strategy in both games since the information set structure for player 1 in G and G' is identical. Since player 2 is not absentminded in G, it is enough to consider pure behavioral strategies (Lemma 3.3.1) as a best response to b_1 in G. Furthermore from Lemma 3.3.3 we know that every pure behavioral strategy $\hat{b}'_2$ from G' has realization equivalent pure behavioral strategy $\hat{b}_2$ in G, hence also the expected utility $u_1(b_1, \hat{b}'_2)$ in G' is equal to $u_1(b_1, \hat{b}_2)$ in G. Since b'_2 is a best response to b_1 in G', it holds that for every pure behavioral strategy $\hat{b}'_2$ in G' and its realization equivalent counterpart $\hat{b}_2$ in G,

$$u_2(b_1, \hat{b}_2) = u_2(b_1, \hat{b}'_2) \leq u_2(b_1, b'_2) = u_2(b_1, b_2).$$

Finally, since also every pure behavioral strategy $\hat{b}_2$ in G has realization equivalent pure behavioral strategy in G' (Lemma 3.3.3), there can be no $\hat{b}_2$ for which $u_2(b_1, \hat{b}_2) > u_2(b_1, b_2)$. □

Lemma 3.3.4 allows us to formulate the concise mathematical program described in Section 4.3 which is used as the core of algorithms discussed in Section 4.4.

3.4 Algorithms Solving Perfect Recall Games

In this section we first present two exact algorithm for finding NE of two-player zero-sum EFGs, namely Sequence-Form LP [35, 58] and Double Oracle algoritm [7]. Next, we turn to approximate algorithms: Fictitious Play [8] and regret minimization based approaches Counterfactual Regret Minimization (CFR, [61]) and its variant CFR+ [54, 55].

3.4.1 Sequence-Form LP

Sequence-form LP [35, 58] is a linear program that allows finding the maxmin strategy in two-player perfect recall EFGs, and hence also a NE strategy in two-player zero-sum perfect recall EFGs.

Let $inf_i(\sigma_i)$ stand for the information set where the last action of σ_i was taken. The sequence form LP has the following formulation:

$$\max_{r,v} v(root) \tag{3.3a}$$

$$s.t. \quad r(\emptyset) = 1 \tag{3.3b}$$

$$0 \leq r(\sigma_i) \leq 1 \qquad \forall \sigma_i \in \Sigma_i \tag{3.3c}$$

$$\sum_{a \in \mathcal{A}(I)} r(\sigma_i a) = r(\sigma_i) \qquad \forall \sigma_i \in \Sigma_i, \forall I \in inf_i(\sigma_i) \tag{3.3d}$$

$$\sum_{\sigma_i \in \Sigma_i} g(\sigma_i, \sigma_{-i} a) r(\sigma_i) + \sum_{I' \in inf_{-i}(\sigma_{-i} a)} v(I') \geq v(I) \qquad \forall I \in \mathcal{I}_{-i}, \forall a \in \mathcal{A}(I), \sigma_{-i} \in seq_{-i}(I) \tag{3.3e}$$

The objective of player i is to find a strategy that maximizes the expected utility against the best responding opponent. The strategy of the maximizing player is represented as a realization plan (variables r) that assigns the probability to a sequence: $r(\sigma_i)$ is the probability that $\sigma_i \in \Sigma_i$ will be played assuming that information sets in which actions of the sequence σ_i are applicable are reached due to player $-i$. The realization plan r must satisfy the network flow Constraints (3.3b)–(3.3d). Finally, a strategy of player i is constrained by the best-responding opponent that selects an action minimizing the expected value of player i in each $I \in \mathcal{I}_{-i}$ (Constraint (3.3e)). These constraints ensure that player $-i$ plays time consistent strategy and hence also an ex-ante best response from Lemma 3.3.2. The expected utility for each action in Constraint (3.3e) is a sum of the expected utility values from immediately reachable information sets I' and from immediately reachable leaves. For the latter we use generalized utility function $g : \Sigma_i \times \Sigma_{-i} \to \mathbb{R}$ defined as $g(\sigma_i, \sigma_{-i}) = \sum_{z \in \mathcal{Z} | seq_i(z) = \sigma_i \wedge seq_{-i}(z) = \sigma_{-i}} u_i(z) \mathcal{C}(z)$.

The sequence-form LP has $|\mathcal{I}_i| + 2|\Sigma_i|$ constraints and $|\Sigma_i| + |\mathcal{I}_i|$ variables. Hence, its size is linear to the size of the solved game, which effectively prevents solving games with more than tens of millions of decision points.

3.4.2 Double Oracle Algorithm

The Double oracle algorithm for solving perfect recall EFGs (DOEFG, [7]) is an adaptation of column/constraint generation techniques for EFGs. The main idea of DOEFG is to create a restricted game where only a subset of actions is allowed to be played by players. The algorithm then incrementally expands this restricted game by allowing new actions. The restricted game is solved as a standard zero-sum extensive-form game using the sequence-form LP. The expansion of the restricted game is performed using best response algorithms that search the original unrestricted game to find new sequences to add to the restricted game for each player. The algorithm terminates when the best response calculated on the unrestricted game provides no improvement to the solution of the restricted game for either of the players.

DOEFG uses two main ideas: (1) the algorithm assumes that players play some pure default strategy outside of the restricted game (e.g., playing the first action in each information set given some ordering), (2) temporary utility values are assigned to leaves in the restricted game that correspond to inner nodes in the original unrestricted game (so-called temporary leaves), which form an upper bound on the expected utility.

Hence, DOEFG aims to increase the scalability of the sequence-form LP by applying it only to the parts of the game relevant to its solution. As shown in [7] this process significantly improves the scalability of sequence-form LP.

3.4.3 Fictitious Play

Fictitious play (FP, [8]) is an iterative algorithm originaly defined for normal-form games. FP aims to improve the scalability of the computation of NE in two-player zero-sum games by approximating it, instead of computing the exact NE. FP keeps track of average mixed strategies $\bar{m}_i^T, \bar{m}_{-i}^T$ of both players. The players take turn updating their average strategy as follows. In iteration T, player i computes $s_i^T \in BR(\bar{m}_{-i}^{T-1})$. He then updates his average strategy $\bar{m}_i^T = \frac{T_i-1}{T_i}\bar{m}_i^{T-1} + \frac{1}{T_i}s_i^T$ (T_i is the number of updates performed by i plus 1). In two-player zero-sum games $\bar{m}_i^T, \bar{m}_{-i}^T$ converge to a NE [48]. Furthermore, there is a long-standing conjecture [32, 15] that the convergence rate of FP is $O(T^{-\frac{1}{2}})$, the same order as the convergence rate of CFR (though the empirical convergence of CFR and CFR+ tends to be better).

When applying FP to behavioral strategies in perfect recall zero-sum EFG G, one must compute the average behavioral strategy $\bar{b}_i^t$ such that it is realization equivalent to $\bar{m}_i^t$ obtained when solving the normal form game corresponding to G for all t and all $i \in \mathcal{N}$ to keep the convergence guarantees. To update the behavioral strategy in such a way we use the following Lemma [27].

Lemma 3.4.1. *Let b_i, b'_i be two behavioral strategies and m_i, m'_i two mixed strategies realization equivalent to b_i, b'_i, and $\lambda_1, \lambda_2 \in [0,1]$, $\lambda_1 + \lambda_2 = 1$. Then*

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

defines a behavioral strategy b''_i realization equivalent to the mixed strategy $m''_i = \lambda_1 m_i + \lambda_2 m'_i$.

3.4.4 Regret Minimization

Another family of algorithms improving the scalability of the computation of NE by its approximation are the algorithms based on regret minimization. In this section we describe the ideas behind external regret, Counterfactual Regret minimization (CFR, [61]) and its variant CFR+ [54, 55].

3.4.4.1 External Regret

Given a sequence of behavioral strategy profiles $b^1, ..., b^T$, the external regret for player i, defined as

$$R_i^T = \max_{b'_i \in \mathcal{B}_i} \sum_{t=1}^{T} (u_i(b'_i, b^t_{-i}) - u_i(b^t_i, b^t_{-i})), \tag{3.4}$$

is the amount of additional expected utility player i could have gained if he played the best possible strategy across all time steps $t \in \{1, ..., T\}$ compared to the expected utility he got from playing b^t_i in every t. An algorithm is a no-regret algorithm for player i, if the average external regret approaches zero; i.e.,

$$\lim_{T \to \infty} \frac{R_i^T}{T} = 0.$$

3.4.4.2 Counterfactual Regret Minimization

Let $b^t_{I \to a}$ be the strategy profile b^t except for I, where $a \in \mathcal{A}(I)$ is played. Let $\pi^b(h)$ be the probability that h will be reached when players play according to the strategy profile b, with $\pi^b_i(h) = \prod_{a \in seq_i(h)} b(a)$ being the contribution of player i and $\pi^b_{-i,c}(h) = \mathcal{C}(h) \prod_{a \in seq_{-i}(h)} b(a)$ the contribution of $-i$ and chance. $z[I]$ stands for the state h which is the predecessor of z in I. Let $\pi^b(h, h')$ be the probability that h' will be reached from h when players play according to b and $\mathcal{Z}_I \subseteq \mathcal{Z}$ a set of leaves reachable from all $h \in I$. Finally, let the counterfactual value of i in the information set I when players play according to the strategy profile b be

$$v_i(b, I) = \sum_{z \in \mathcal{Z}_I} u_i(z) \pi^b_{-i,c}(z[I]) \pi^b(z[I], z).$$

Counterfactual regret is defined for each iteration T, player i, information set $I \in \mathcal{I}_i$ and action $a \in \mathcal{A}(I)$ as

$$R_i^T(I,a) = \sum_{t=1}^{T} \left[v_i(b^t_{I\to a}, I) - v_i(b^t, I) \right]. \tag{3.5}$$

Let $(x)^+$ stand for $\max(x, 0)$. The strategy b_i^T for player i in iteration T in the standard CFR algorithm (sometimes termed vanilla CFR) is computed from counterfactual regrets using the *regret-matching* as follows

$$b_i^T(I,a) = \begin{cases} \frac{\left(R_i^{T-1}(I,a)\right)^+}{\sum_{a'\in\mathcal{A}(I)}\left(R_i^{T-1}(I,a')\right)^+}, \text{ if } \sum_{a'\in\mathcal{A}(I)}\left(R_i^{T-1}(I,a')\right)^+ > 0 \\ \frac{1}{\mathcal{A}(I)}, \text{ otherwise.} \end{cases} \tag{3.6}$$

The immediate counterfactual regret is defined as

$$R^T_{i,imm}(I) = \frac{1}{T} \max_{a\in A(I)} R_i^T(I,a).$$

In games having perfect recall, minimizing the immediate counterfactual regret in every information set minimizes the average external regret. This holds because perfect recall implies that

$$\frac{R_i^T}{T} \le \sum_{I\in\mathcal{I}_i} \left(R^T_{i,imm}(I)\right)^+, \tag{3.7}$$

i.e., the external regret is bounded by the sum of positive parts of immediate counterfactual regrets [61].

Let $\Delta_I = \max_{z\in\mathcal{Z}_I} u_i(z) - \min_{z\in\mathcal{Z}_I} u_i(z)$ and $u_{max} = \max_{i\in\mathcal{N}} \max_{z\in\mathcal{Z}} u_i(z)$. When player i plays according to eq. (3.6) in I during iterations {1, ..., T}, then

$$R^T_{i,imm}(I) \le \frac{\Delta_I \sqrt{|\mathcal{A}(I)|}}{\sqrt{T}}. \tag{3.8}$$

Thus, from eq. (3.7)

$$\frac{R_i^T}{T} \le \frac{\Delta|\mathcal{I}_i|\sqrt{|A_{max}|}}{\sqrt{T}}, \tag{3.9}$$

where $|A_{max}| = \max_{I\in\mathcal{I}_i} |\mathcal{A}(I)|$ and $\Delta = 2u_{max}$ [61].

Furthermore, let $\bar{b}_i^T$ be the average strategy for i defined as

$$\bar{b}_i^T(I,a) = \frac{\sum_{t=1}^T \pi_i^{b_i^t}(I) b_i^t(I,a)}{\sum_{t=1}^T \pi_i^{b_i^t}(I)}, \quad \forall I \in \mathcal{I}_i, \forall a \in \mathcal{A}(I). \tag{3.10}$$

If $\frac{R_i^T}{T} < \varepsilon$ for all $i \in \mathcal{N}$ in a two-player zero-sum EFG, the strategy profile $\bar{b}^T = (\bar{b}_1^T, \bar{b}_2^T)$ forms a 2ε-Nash equilibrium [61].

3.4.4.3 CFR+

The CFR+ [54, 55] replaces the regret-matching shown in equation (3.6) with *regret-matching*$^+$. To do that, the algorithm maintains alternative regret values $Q_i^T(I,a)$ in each iteration T and for each player i defined as

$$Q_i^0(I,a) = 0 \tag{3.11}$$

$$Q_i^T(I,a) = \left(Q_i^{T-1}(I,a) + R_i^T(I,a) - R_i^{T-1}(I,a)\right)^+. \tag{3.12}$$

The update of $Q_i^T(I,a)$ is identical to the update of $R_i^T(I,a)$ in eq. (3.5) except for negative values which are replaced by 0. This prevents negative counterfactual regrets from accumulating. Since the strategy is then computed using the regret-matching$^+$ defined as

$$b_i^T(I,a) = \begin{cases} \frac{Q_i^{T-1}(I,a)}{\sum_{a' \in \mathcal{A}(I)} Q_i^{T-1}(I,a')}, \text{ if } \sum_{a' \in \mathcal{A}(I)} Q_i^{T-1}(I,a') > 0 \\ \frac{1}{\mathcal{A}(I)}, \text{ otherwise,} \end{cases} \tag{3.13}$$

any future positive regret changes will immediatelly affect the resulting strategy instead of canceling out with the accumulated negative regret values.

Additionally, CFR+ uses alternating updates of the regrets, i.e., during one iteration of the algorithm only regrets of one player are updated, and the players take turn.

Finally, as described in [55], the update of the average strategy starts only after a fixed number of iterations denoted as d. The update is then weighted by the current iteration, formally for each $T > d$

$$\bar{b}_i^T(I,a) = \frac{2\sum_{t=d}^{T} t \cdot \pi_i^{b_i^t}(I) \cdot b_i^t(I,a)}{((T-d)^2 + T - d)\sum_{t=d}^{T} \pi_i^{b_i^t}(I)}, \quad \forall I \in \mathcal{I}_i, \forall a \in \mathcal{A}(I). \tag{3.14}$$

Intuitively, the average strategy update puts a higher weight on later strategies, as they are expected to perform better.

The bound on average external regret $\frac{R_i^T}{T}$ from eq. (3.9) also holds when using regret matching$^+$. Hence the average strategy profile computed by CFR+ converges to the Nash equilibrium of the solved two-player zero-sum EFG [55]. Additionally, the empirical convergence of CFR+ is significantly better compared to CFR [55].

3.5 Abstractions

The algorithms described in Section 3.4 either build the entire EFG or require iterative traversal of the game tree. This makes them impractical in many large domains often found in the real world. E.g., heads-up no-limit Texas hold'em poker has more than 10^{160} decision points [29]. A popular approach for solving games with such prohibitive memory requirements is the use of abstractions. The main idea of this approach is to create a smaller game strategically similar to the game being solved (denoted as abstracted game).

The abstracted game is then solved, and the solution is projected back to the original game.

Abstractions in EFGs face many difficulties not present in single agent problems. Most importantly, abstractions in EFGs can be non-monotonic, i.e., strategies computed in a fine-grained abstraction can result in worse performance than strategies computed in strictly coarser abstraction [59]. On the other hand, there has been overwhelming experimental evidence suggesting that in specific types of domains, such as two-player Texas hold'em, finer grained abstractions improve the quality of the resulting strategies [30]. Additionally, to eliminate this problem, a lot of work has focused on constructing abstractions with a bounded error of the resulting strategies [40, 37, 38].

There are two distinct approaches to abstracting an EFG: information abstractions and action abstractions.

The information abstractions reduce the size of the original large extensive-form game by removing information available to players; hence merging their information sets. Since the players have to play identical strategy in the merged information sets, the size of the strategy representation in the abstracted game can be significantly smaller than in the original game. The abstracted game is then solved, and the small resulting strategies are used in the original game. Most of the work on information abstractions was driven by research in the poker domain. Information abstractions were initially created using domain-dependent knowledge [53, 2]. Algorithms for creating information abstractions followed [19, 21] and led to the development of bots with increasing quality of play in poker. This work culminated in lossless information abstractions (i.e., abstractions where the strategies obtained by solving the abstracted game form optimal strategies in the original game) which allowed solving the Rhode Island Hold'em, a game with $3.1 \cdot 10^9$ nodes in the game tree [20]. When moving to larger games, lossless abstractions were found to be too restrictive to offer sufficient reductions in the size of the abstracted game. Hence the focus switched to lossy abstractions. A mathematical framework that can be used to create perfect recall information abstractions with bounds on solution quality was introduced [37]. Additionally, both lossless and lossy imperfect recall abstractions were provided in [40, 38]. The authors show that running the CFR algorithm on this class of imperfect recall abstractions leads to a bounded regret in the full game.

Another form of abstractions are action abstractions. Instead of merging information sets, this abstraction methodology restricts the actions available to the players in the original game. Similarly to information abstractions, most of the work on action abstraction was driven by the poker domain where there is a prohibitive amount of actions available to the players (e.g., when betting the players can choose any value up to the number of chips they have available). The first automated techniques iteratively adjust the bet sizing in no-limit hold'em [25, 26]. The algorithm for simultaneous action abstraction finding and game solving was introduced in [10].

The combination of information abstractions and action abstractions was at the hearth of a number of algorithms for solving heads-up no-limit Texas Hold'em poker (see, e.g., [22]). Furthermore, the algorithm Libratus [9] using both mentioned abstractions led to a superhuman performance in heads-up no-limit Texas Hold'em.

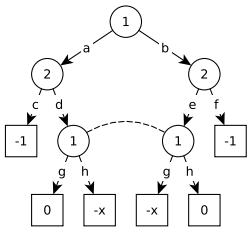

Figure 3.5: An A-loss recall for the demonstration of problems of algorithms for solving perfect recall EFGs in imperfect recall EFGs.

In other domains, abstractions with bounded error were introduced in Patrolling Security Games [1]. Additionally, the first general framework for lossy game abstraction with bounds was provided for stochastic games [52].

3.6 Algorithms for Perfect Recall Games in Imperfect Recall Games

In this section, we discuss the applicability of sequence-form LP, FP and regret minimization based algorithms in two scenarios. First, when we are interested in solving a given imperfect recall game and second when we are applying the algorithm to an imperfect recall abstraction with an aim to solve the corresponding unabstracted two-player perfect recall EFG. Finally, we discuss subclasses of imperfect recall games which are specifically designed as suitable results of imperfect recall information abstractions and explain why the CFR applied to these abstractions has guaranteed convergence to NE of the original unabstracted game.

3.6.1 Sequence-Form Linear Program

Here we discuss the issues of the sequence-form LP applied to an imperfect recall game. More specifically, we discuss the expressive power of realization plans in imperfect recall games and the applicability of constraints ensuring the best response of the minimizing player in the sequence-form LP to imperfect recall games.

3.6.1.1 Realization Plans in Imperfect Recall Games

A realization plan of player i assigns probabilities to $\sigma_i \in \Sigma_i$. This representation, therefore, implicitly allows conditioning of choices of i on all the actions he made in the past. For this reason, the descriptive power of realization plans and mixed strategies is the same in imperfect recall games.

Example 3.6.1. *Consider the game in Figure 3.5. The player 1 has the following set of sequences:* $\Sigma_1 = \{(a), (b), (a, g), (a, h), (b, g), (b, h)\}$. *The realization plan allows conditioning the choice of players on the information that the players should no longer have*

available. E.g., realization plan $r_1(a) = r_1(b) = 0.5, r_1(a,g) = 0.5, r_1(b,h) = 0.5$ *allows player 1 to condition playing* g *and* h *on the outcome of his stochastic choice in the root, and thus randomize between the leafs with utility 0. Additionally, the size of* Σ_1 *is the same whether we join the states after action* d *and* e *to the imperfect recall information set or not, even though joining the states reduces the number of decision points of player 1 from 3 to 2.*

Hence, realization plan allows description of behavior which is inconsistent with the rules of the game. Moreover, when using the realization plan in an imperfect recall abstraction, it does not directly benefit from the reduced size of the abstracted game as it rebuilds some of the removed information.

3.6.1.2 Best Response Constraints in Sequence-Form LP

The constraints (3.3e) in the sequence-from LP applied to a perfect recall EFG ensure that the maximizing player computes a strategy that maximizes his expected utility against the opponent playing time consistent strategy (and therefore also ex-ante best response from Lemma 3.3.2). In imperfect recall games, however, these constraints might not be directly applicable, since the expected utility after an action in imperfect recall information set of the minimizing player does not depend only on the behavior of the maximizing player and the probabilities of nature but also on the previous behavior of the minimizing player.

Example 3.6.2. *Consider the game in Figure 3.5 and let player 1 be the minimizing player. Now, let us try to formalize the constraint* (3.3e) *for the second information set* I *of player 1 and action* g*. Notice that there are two sequences of player 1 leading to* I*, namely* (a) *and* (b)*. And hence, the leaf that will influence the first sum in the constraint is determined by the choice of player 1 in the root. Such dependency is, however, not possible to formalize using linear constraints and so the sequence-from LP is not directly applicable to imperfect recall games.*

Hence, the imperfect recall of the minimizing player prevents direct application of the sequence-form LP, as the best response constraints (3.3e) are not well defined in imperfect recall information sets.

3.6.2 Fictitious Play

The average strategy $\bar{b}_i^t$ for player i in iteration t of FP is computed according to Lemma 3.4.1 from $\bar{b}_i^{t-1}$ with weight $\lambda_1 = \frac{T_i - 1}{T_i}$ and the best response b_i^t of player i against b_{-i}^{t-1} with weight $\lambda_2 = \frac{1}{T_i}$. The guarantee of convergence of FP to NE in two-player zero-sum EFGs with perfect recall [48] is based on the fact that

$$\forall b_{-i} \in \mathcal{B}_{-i}^p u_i(\bar{b}_i^t, b_{-i}) = \frac{T_i - 1}{T_i} u_i(\bar{b}_i^{t-1}, b_{-i}) + \frac{1}{T_i} u_i(b_i^t, b_{-i}). \tag{3.15}$$

However, this property no longer holds in imperfect recall games.

Example 3.6.3. *Consider the imperfect recall game from Figure 3.5. Let* $b'_1(a) = b'_1(g) = 1$ *and* $b''_1(b) = b''_1(h) = 1$ *be two strategies in the game. Let* b_1 *be a strategy computed according to Lemma 3.4.1 from* b'_1 *with* $\lambda_1 = 0.5$ *and* b''_1 *with* $\lambda_2 = 0.5$*, i.e.,* $b_1(a) = b_1(b) = b_1(g) = b_1(h) = 0.5$*. However, for* $b_2(d) = b_2(e) = 1$*,* $u_1(b_1, b_2) = -0.5x$*, while* $\frac{1}{2}u_1(b'_1, b_2) + \frac{1}{2}u_1(b''_1, b_2) = 0$*. And hence the equality in eq.* (3.15) *is not satisfied.*

In Appendix A.1 we experimentally demonstrate that the FP does not converge to reasonable strategies in the game from Figure 3.5.

Hence, FP is not a suitable algorithm for solving a given imperfect recall game. When applied to imperfect recall abstraction, FP directly benefits from the reduced size of the imperfect recall abstraction since it will store the average strategies directly in the information set structure of the abstraction. However, as shown in Appendix A.1, FP does not guarantee to find strategies in the abstracted game with good performance in the original unabstracted game.

3.6.3 Regret Minimization

The no-regret learning cannot work in general in imperfect recall games for similar reasons as FP. The general reason is that the loss function

$$l^t(b_i) = u_i(b_i^t, b_{-i}^t) - u_i(b_i, b_{-i}^t) \tag{3.16}$$

used in equation (3.4) can be non-convex over the probability simplex of behavioral strategies (see Example 3.6.4). The loss function must be convex for no-regret learning to have convergence guarantees [23].

3.6.3.1 CFR and CFR+

In games having perfect recall, minimizing the immediate counterfactual regrets at every information set minimizes the average external regret. This holds because perfect recall implies that the average external regret is bounded by the sum of positive parts of immediate counterfactual regrets as shown in eq. (3.7), and thus the average external regret in bounded as described in eq. (3.9). In imperfect recall games, however, equations (3.7) and (3.9) are not guaranteed to hold due to the non-convexity of the loss function described above.

Example 3.6.4. *Assume we are in the step* T *of a no-regret learning algorithm solving the game from Figure 3.5, and we evaluate the loss from eq.* (3.16) *of some strategy* b_1 *in step* $t < T$*. Let* $b_1^t(a) = b_1^t(g) = 0.5$ *and* $b_2^t(d) = b_2^t(e) = 1$*. Let* $b_1(a) = b_1(g) = 1$*,* $b'_1(b) = b'_1(h) = 1$*, and* $b''_1(a) = b''_1(g) = 0.5$*. The losses of these strategies are* $l^t(b_1) = -0.5x$*,* $l^t(b'_1) = -0.5x$*,* $l^t(b''_1) = 0$*. Since* b''_1 *is a convex combination of* b_1 *and* b'_1 *with uniform weights, it follows that the loss function is non-convex, hence the convergence guarantees used in CFR due to Gordon [23] no longer apply.*

The non-convexity of the loss function shown in Example 3.6.4 will never appear in perfect recall games since the behavior of i after any $a, a' \in \mathcal{A}(I_i)$ is independent $\forall I \in \mathcal{I}_i$ and hence the loss function is linear.

In Appendix A.2 we experimentally demonstrate that the CFR does not converge to reasonable strategies in the game from Figure 3.5.

Hence, CFR and CFR+ are not suitable algorithms for solving a given imperfect recall game. When applied to imperfect recall abstraction, CFR directly benefits from the reduced size of the imperfect recall abstraction, since it will store the average strategies and regrets directly in the information set structure of the abstraction. However, as shown in Appendix A.2, CFR and CFR+ do not guarantee to find strategies in the abstracted game with good performance in the original unabstracted game.

3.6.4 CFR in Chance Relaxed Skew Well-Formed Games

Here, we discuss the WFG, SWFG, and CRSWFG (see Section 2.2.3 for the definition of these subclasses of imperfect recall EFGs) which are specifically designed as suitable results of imperfect recall information abstractions and explain why CFR applied to these abstractions is guaranteed to convergence to NE of the original unabstracted game.

CRSWFG (see Definition 2.2.5) only merge information sets which satisfy strict restrictions on the structure of the game tree above and below them. These restrictions imply that for all possible strategies of the opponent, a strategy which is optimal in one of the merged information sets must have bounded distance from the optimal strategy in the rest of the merged information sets. The distance is bounded by the error terms defined in Definition 2.2.5. This property effectively limits the non-convexity of the loss function from eq. 3.16, which in turn allows derivation of a regret bound on the average external regret of CFR applied to CRSWFG with a constant factor depending on the error terms (see [38] for more details).

SWFG (see Definition 2.2.4) require even more strict bound on the error of the optimal strategies across the merged information sets (defined by the $\delta_{I',I''}$ constants for all pairs I', I'' of merged information sets). Hence, this again allows derivation of regret bound on the average external regret of CFR applied to SWFG with a constant factor depending on the $\delta_{I',I''}$ constants (see [40] for more details).

Finally, in WFG (see Definition 2.2.3) all strategies that are optimal in one of the merged information set have to be optimal in the rest. Similarly to perfect recall games, this restriction leads to linear loss function from eq. 3.16. Hence, the equations (3.7) and (3.9) are guaranteed to hold in WFG. WFG, therefore, directly allow application of CFR with the same guarantees as in perfect recall games.

Chapter 4

Approximating Maxmin Strategies in Imperfect Recall Games Using A-Loss Recall Property

In this Chapter, we discuss theoretical properties of A-loss recall games and use these properties to derive the first family of algorithms for approximating maxmin strategies in imperfect recall abstractions. First, in Section 4.1 we provide the overview of the relevant state-of-the-art. Next, in Section 4.2.1 we show sufficient and necessary (i.e., if and only if) conditions for the existence of Nash equilibrium in A-loss recall games and show that in imperfect recall games, such conditions are only sufficient. In Section 4.2.2 we discuss the complexity of computing maxmin and Nash equilibrium strategies in A-loss recall games. In Section 4.3 we derive the mixed integer linear program (MILP) for approximating maxmin strategy in imperfect recall game and show that this MILP has linear size to the size of the game if the minimizing player has A-loss recall (its size is exponential for minimizing player with imperfect recall). In Section 4.4 we introduce the algorithms which use the MILP to approximate the maxmin strategies. More specifically, in Section 4.4.1 we introduce the Base algorithm which iteratively solves the MILP until the desired approximation of maxmin strategy is reached. In Section 4.4.2 we describe Imperfect Recall Abstraction Branch-and-Bound algorithm (IRABnB) which uses branch-and-bound search over the linear relaxation of the MILP. In Section 4.4.3 we introduce Double Oracle Imperfect Recall Abstraction Branch-and-Bound algorithm (DOIRABnB) which further extends IRABnB with incremental strategy generation techniques. Finally, in Section 4.5 we provide experimental evaluation of IRABnB nad DOIRABnB.

4.1 Comparison to the Current State-of-the-Art

The only known subclasses of imperfect recall games using the imperfect recall to reduce the memory requirements of strategy representation are chance relaxed skew well-formed games (CRSWFG) and their subsets well-formed games (WFG) and skew well-formed (SWFG) [38, 40] (see Section 2.2.3). These classes of games restrict the structure of

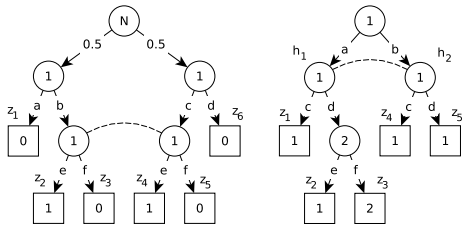

Figure 4.1: (Left) A well-formed game which does not have A-loss recall. (Right) An A-loss recall game which is not chance relaxed skew well-formed.

imperfect recall by requiring similarity of the states included in one imperfect recall information set both in the structure of past and future moves as well as the utilities reachable from the states. As a consequence, the CFR is still guaranteed to converge to (ε-)NE in these games (see Chapter 3 for more details on applying CFR to imperfect recall games and CRSWFG).

While A-loss recall games restrict the structure of the game only above the imperfect recall information set (the requirement of being able to connect any loss of memory to forgetting player's own actions), the WFG, SWFG, and CRSWFG restrict the structure above, below and also the structure of the utilities. In the case of games with no chance, we can formally define the relationship between A-loss recall games and WFG, SWFG, and CRSWFG.

Lemma 4.1.1. *In games with no chance, the WFG, SWFG, and CRSWFG form a strict subset of A-loss recall games.*

Proof. We first prove the Lemma for WFG and then show that it extends to both SWFG and CRSWFG.

The only requirement in A-loss recall games is that players are able to connect any loss of memory to forgetting their own actions, hence the restriction in the information set I always concerns only the part of the tree above I. We, therefore, focus on condition (3) in the Definition 2.2.3 of WFG since it is the only one restricting the upper part of the game tree. Condition (3) requires that for each $h' \in I'$ there must exist $h'' \in I''$ such that $seq_{-i}(h') = seq_{-i}(h'')$, which, combined with the assumption that there is no chance player, implies that there must exist difference in $seq_i(h')$ and $seq_i(h'')$. Furthermore, since $seq_{-i}(h') = seq_{-i}(h'')$ we are sure that there must exist $\bar{I} \in \mathcal{I}_i$ and distinct $a, a' \in \mathcal{A}(\bar{I})$ such that $a \in seq_i(h')$ and $a' \in seq_i(h'')$ which is exactly the condition (2) in the A-loss recall property. The rest of the requirements of the WFG restricts utilities and parts of the tree not restricted in A-loss recall games. Therefore the WFG form a subset of A-loss recall games.

Notice that the Lemma also holds for SWFG and CRSWFG, since they provide relaxations in conditions (1) and (2) of the Definition 2.2.3 only. □

In Figure 4.1 (left) we show a WFG (with $\alpha : \{z_2 \to z_4, z_3 \to z_5\}$) with chance which does not have A-loss recall. The game does not have A-loss recall since player 1 forgets

the information about the move of chance in the root of the game. Finally, in Figure 4.1 (right) we present an A-loss recall game which is not CRSWFG. The game is not CRSWFG since the leaves z_2 and z_3 reachable from h_1 cannot be mapped to any leaf after h_2 without breaking condition 3 in Definition 2.2.3. Notice that when assuming rational player 2, however, it is safe to include h_1 and h_2 to one information set, since the expected utilities after every action available in h_1, h_2 are equal, and so any behavior optimal in h_1 is optimal in h_2. Hence CRSWFG are unnecessarily conservative in the restrictions posed on the game tree.

4.2 NE and Maxmin Strategies in A-Loss Recall Games

To provide a complete picture of the complexity of solving A-loss recall games, we discuss the existence of NE in A-loss recall games and the computational complexity of finding maxmin and NE behavioral strategies in A-loss recall games (for discussion about the numerical representation of NE and maxmin in A-loss recall games see Section 3.2).

4.2.1 Existence of NE in A-loss Recall Games

As we have discussed in Section 2.3.2.2 the guarantee of the existence of NE in mixed strategies does not extend to NE in behavioral strategies in A-loss recall games because of the different descriptive power of mixed and behavioral strategies there [60]. Here, we then provide novel sufficient and necessary (i.e., if and only if) condition for the existence of NE in behavioral strategies in two-player A-loss recall games. Note, that thanks to this result, A-loss recall games are the only subclass of imperfect recall games, for which such condition is known (the only exception are well-formed games, where NE in behavioral strategies always exists, and so the condition is trivial).

Informally, Theorem 4.2.1 states that A-loss recall game G has a NE in behavioral strategies if and only if there exists a behavioral NE in its coarsest perfect recall refinement G' which prescribes the same behavior in every information set which is connected to some imperfect recall information set of G.

Theorem 4.2.1. *An A-loss recall game G has a NE in behavioral strategies if and only if there exists a NE strategy profile b in behavioral strategies of the coarsest perfect recall refinement G' of G, such that $\forall I \in \mathcal{I}$ of G, $\forall H_k, H_l \in H(I) : b(H_k) = b(H_l)$, where $b(H)$ stands for the behavioral strategy in the information set of G' formed by states in H.*

Proof. First, since b is a NE of G' we know that there exists no incentive for any player to deviate to any pure behavioral strategy in G'. From Lemma 3.3.3, it follows that there can exist no pure behavioral strategy in G to which any of the players want to deviate either. Additionally, from Lemma 3.3.1 it is sufficient to consider deviations to pure strategies in G since none of the players is absentminded. This, in combination with the fact, that b prescribes valid strategy in G implies that b is a NE in behavioral strategies of G.

Second, we prove that there exists no NE b' in behavioral strategies of G which is not a NE of G'. Let us assume that such b' exists. This would imply that there is no pure

behavioral strategy in G to which players want to deviate when playing according to b', and therefore no pure behavioral strategy in G' either (Lemma 3.3.3), implying that b' is a NE in G'. This contradicts the assumption and completes the proof. □

Note that in general imperfect recall game this equivalence no longer holds. Consider the game in left subfigure of Figure 2.2. Here the only NE in behavioral strategies is playing d and e deterministically and mixing uniformly between g, h. The only NE of its coarsest perfect recall refinement (shown in right subfigure of Figure 2.2) is, however, playing d, e, h and i deterministically.

Corollary 4.2.1. *An imperfect recall game G (not restricted to A-loss recall) has a Nash equilibrium in behavioral strategies if there exists a Nash equilibrium strategy profile b in behavioral strategies of the coarsest perfect recall refinement G' of G, such that $\forall I \in \mathcal{I}^G \; \forall H_k, H_l \in H(I) : b(H_k) = b(H_l)$, where $b(H)$ stands for the behavioral strategy in the information set of G' formed by states in H, the opposite does not hold.*

Proof. The proof follows from the fact that every pure behavioral strategy of G has realization equivalent pure behavioral strategy in G' since we only merge information sets in G' to obtain G. The merging of information sets eliminates pure behavioral strategies with mutually exclusive behavior in these sets. Since b is a Nash Equilibrium of G', we know that there exists no incentive for any player to deviate to any pure behavioral strategy in G' and therefore also no incentive to deviate to any pure behavioral strategy in G. This, in combination with the fact, that b prescribes valid strategy in G implies that b is a Nash Equilibrium in behavioral strategies of G.

Finally, we provide a counter-example showing that the opposite direction of the implication does not hold. Consider the game in left subfigure of Figure 2.2. Here the only Nash equilibrium in behavioral strategies is playing d and e deterministically and mixing uniformly between g, h. The only Nash equilibrium of its coarsest perfect recall refinement (shown in right subfigure of Figure 2.2) is, however, playing d, e, h and i deterministically. □

4.2.2 Computational Complexity in A-loss Recall Games

Now we turn to the computational complexity of solving imperfect recall games. Computing maxmin strategies is NP-hard in imperfect recall games [34] and it is NP-hard to decide whether there exists a NE in behavioral strategies in imperfect recall games [24] (both theorems and their proofs are presented in Section 3.1). Here, we show that both negative results directly translate to A-loss recall games. Notice that unlike ours, the reduction used in [24] requires a game with absentmindedness. Hence we significantly extend the class of games for which it is known that deciding whether there exists a NE in behavioral strategies is NP-hard.

Theorem 4.2.2. *The problem of deciding whether player 2 having an A-loss recall can guarantee an expected payoff of at least λ is NP-hard even if player 1 has perfect recall, there are no chance moves, and the game is zero-sum.*

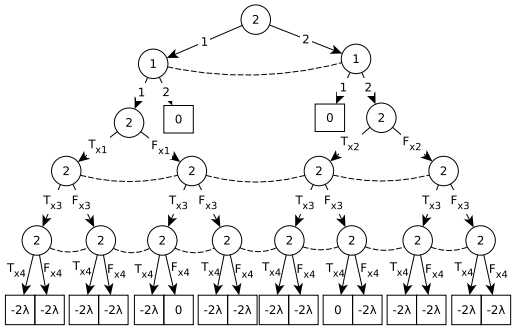

Figure 4.2: An A-loss recall game reduction from Theorem 4.2.2 of 3-SAT problem $x_1 \vee \neg x_3 \vee x_4 \wedge \neg x_2 \vee x_3 \vee \neg x_4$.

Proof. The proof is made by reduction from the 3-SAT problem. It is a modification of the original proof of koller [34] for imperfect recall games. The example of the reduction is given in Figure 4.2. Given n clauses $x_{j,1} \vee x_{j,2} \vee x_{j,3}$ we create a two person zero-sum game in the following way. In the root of the game player 2 chooses between n actions, each corresponding to one clause. Player 1 plays next with no information about the action chosen by player 2. He has again n actions, each corresponding to one clause. In every state of player 1, $n-1$ actions lead directly to a terminal state with utility 0 for player 1 and one action (corresponding to the same clause as the action of player 2 preceding this state) leads to a state of player 2. Every such state of player 2 corresponds to the variable $x_{j,1}$ where j is the index of the clause chosen in the root of the game. Every such state has actions $T_{x_{j,1}}, F_{x_{j,1}}$ available. These actions correspond to setting the variable $x_{j,1}$ to true or false respectively. After both $T_{x_{j,1}}, F_{x_{j,1}}$ in $x_{j,1}$ we reach the state representing the assignment to $x_{j,2}$ with the same setup (state representing the assignment to $x_{j,3}$ is reached after that). After the assignment to $x_{j,3}$ we reach the terminal state with utility $-n\lambda$ for player 1 if the assignment to $x_{j,1}$, $x_{j,2}$ and $x_{j,3}$ satisfies the clause $x_{j,1} \vee x_{j,2} \vee x_{j,3}$, 0 otherwise. The information sets of player 2 group together all the states corresponding to the assignment to one variable in the original 3-SAT problem (note that we assume that the order of variables in every clause follows some complete ordering on the whole set of variables in the 3-SAT problem).

We will show that player 2 can guarantee the worst case expected value λ if and only if the original 3-SAT problem is satisfiable. First, we show that if the original 3-SAT problem is satisfiable player 2 can guarantee the worst case expected value λ. The worst case expected value λ is achieved when player 2 mixes uniformly in the root of the game and plays according to the assignment which satisfies the original 3-SAT problem in the rest of the tree.

Next, we show that if player 2 can guarantee the worst case expected value λ, the original 3-SAT problem has to be satisfiable. There are two cases we need to discuss.

Case 1: Player 2 plays a non-uniform strategy b_2 in the root. In this case player 1 plays action $a \in \mathcal{A}_1$ corresponding to the same clause as the action $a_{min} \in \arg\min_{a \in \mathcal{A}_2(root)} b_2(a)$.

Since b_2 is non-uniform in the root we know that $b_2(a_{min}) < \frac{1}{n}$ and hence the expected value of player 2 must be lower than $\frac{n\lambda}{n}$ no matter what happens in the rest of the game.

Case 2: The last chance to guarantee expected value λ is when player 2 plays a uniform strategy b_2 in the root. Here we show that λ can be guaranteed only when the corresponding 3-SAT problem is satisfiable. If the 3-SAT problem is not satisfiable, that means that there always exists a state h of player 2 after the action of player 1, where $u_2^h(b_2) < n\lambda$, where $u_2^h(b_2)$ stands for the expected value in h when player 2 plays according to b_2. By playing action leading to this state, player 1 guarantees that the expected value for player 2 is lower than $\frac{n\lambda}{n}$. If the 3-SAT is satisfiable on the other hand, the uniform b_2 in the root and playing according to the assignment satisfying the 3-SAT guarantees the expected value λ.

The reduction is polynomial, since the game has $n(n-1) + 2^3 n$ leaves.

The last thing which remains to be shown is that player 2 has A-loss recall. This is satisfied since any loss of information about actions of player 1 can be tracked back to forgetting his own action taken in the root or to setting some of the SAT variables to true or false. □

We leave the question whether the problem stated in Theorem 4.2.2 belongs to NP as an open problem. Even though Theorem 3.2.1 states that the solution to this problem might require irrational numbers, it is not a sufficient argument for showing that this problem does not belong to NP. From this perspective, the problem from Theorem 4.2.2 is similar to *square-root sum* problem, since the square-root sum problem also requires operations with irrational numbers. However, deciding whether square root sum problem belongs to NP is a major open problem [18] and there are known connections of square-root sum problem to other problems in game theory, e.g., computing Nash equilibrium in 3-player games [16].

Theorem 4.2.3. *It is* NP*-hard to check whether there exists a Nash equilibrium in behavioral strategies in two-player A-loss recall games even if player 1 has perfect recall, there are no chance moves, and the game is zero-sum.*

Proof. The proof is made by reduction from the 3-SAT problem. The reduction results in a two-player zero-sum game similar to the one in proof of Theorem 4.2.2. The only change in the game is the substitution of the utility in the leaves directly following actions of player 1 by -1 for player 1 and in the leaves corresponding to satisfying the given clause by -0.5 for player 1. The example of the reduction is shown in Figure 4.3.

We will show that a NE in behavioral strategies exists if and only if the corresponding 3-SAT problem is satisfiable. First, if the 3-SAT problem is satisfiable, this game has a NE where both players mix uniformly in first two levels of the game, and the player 2 plays according to the assignment of variables satisfying this problem.

Next, we show that if the NE exists, the corresponding 3-SAT problem has to be satisfiable. There are two cases we need to discuss.

Case 1: Player 1 plays a non-uniform strategy b_1. In this case, player 2 will always prefer to play action $a \in \mathcal{A}_2(root)$ which corresponds to the same clause as $a_{min} \in$

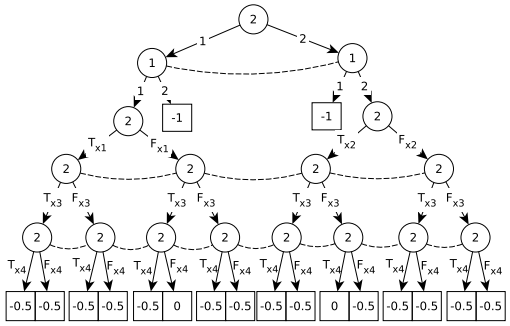

Figure 4.3: An A-loss recall game reduction from Theorem 4.2.3 of 3-SAT problem $x_1 \vee \neg x_3 \vee x_4 \wedge \neg x_2 \vee x_3 \vee \neg x_4$.

$\arg\min_{a \in \mathcal{A}_1} b_1(a)$ deterministically in the root and make the clause corresponding to a_{min} satisfiable. This way player 2 maximizes the probability that the immediate worst possible outcome -1 for player 1 will be reached and minimizes the value player 1 gets when the following state of player 2 is reached. Hence b_1 is not stable against a best responding opponent.

Case 2: The last chance for NE to exist is when player 1 plays a uniform strategy b_1. Here we show that in this case, the NE exists only when the corresponding 3-SAT problem is satisfiable. If the original 3-SAT problem is not satisfiable player 2 will prefer to play a subset of actions $\mathcal{A}'_2 \subset \mathcal{A}_2(root)$ in the root corresponding to a subset of clauses that can be satisfied at the same time, while playing the assignment satisfying these clauses in the rest of the tree. In this case, however, player 1 wants to deviate to playing any distribution over his actions corresponding to the clauses of actions in $\mathcal{A}'_2$. If the 3-SAT is satisfiable on the other hand, the uniform b_1 forms a part of NE when player 2 plays a uniform strategy in the root and according to the assignment satisfying the 3-SAT in the rest of the tree. □

4.3 The Mathematical Program for Approximating Maxmin Strategies in Imperfect Recall Games

In this section, we present a mathematical program approximating maxmin strategies for two player games without absentmindedness where the maximizing player has imperfect recall, first when assuming that the minimizing player has A-loss recall, followed by its generalization where there are no restrictions for the minimizing player. Recall that computing exact maxmin strategy in this class of games requires exact representation of irrational numbers (Theorem 3.2.1) and so approximating maxmin strategies is the only alternative. The main idea behind this formulation is to add bilinear constraints into the sequence-form LP ([58], see Section 3.4.1) to restrict to imperfect recall strategies of the maximizing player. First, we present the exact bilinear program, followed by explanation of Multiparametric Disaggregation Technique (MDT) [36] which will be used for

approximating bilinear terms. Next, we provide the mixed integer linear program (MILP) resulting from the application of the MDT to the bilinear reformulation of the sequence form LP. Finally, we discuss how to use the result of this MILP to construct a strategy with a bounded difference of its expected worst case utility from the maxmin value.

4.3.1 Exact Bilinear Sequence Form Against A-loss Recall Opponent

$$\max_{x,r,v} v(root, \emptyset) \tag{4.1a}$$

$$s.t. \quad r(\emptyset) = 1 \tag{4.1b}$$

$$0 \leq r(\sigma_1) \leq 1 \qquad \forall \sigma_1 \in \Sigma_1 \tag{4.1c}$$

$$\sum_{a \in \mathcal{A}(I)} r(\sigma_1 a) = r(\sigma_1) \qquad \forall \sigma_1 \in \Sigma_1, \forall I \in inf_1(\sigma_1) \tag{4.1d}$$

$$\sum_{a \in \mathcal{A}(I)} x(a) = 1 \qquad \forall I \in \mathcal{I}_1^{IR} \tag{4.1e}$$

$$0 \leq x(a) \leq 1 \qquad \forall I \in \mathcal{I}_1^{IR}, \forall a \in \mathcal{A}(I) \tag{4.1f}$$

$$r(\sigma_1) \cdot x(a) = r(\sigma_1 a) \qquad \forall I \in \mathcal{I}_1^{IR}, \forall a \in \mathcal{A}(I), \forall \sigma_1 \in seq_1(I) \tag{4.1g}$$

$$\sum_{\sigma_1 \in \Sigma_1} g(\sigma_1, \sigma_2 a) r(\sigma_1) + \sum_{I' \in inf_2(\sigma_2 a)} v(I', \sigma_2 a) \geq v(I, \sigma_2) \qquad \forall I \in \mathcal{I}_2, \forall a \in \mathcal{A}(I), \forall \sigma_2 \in seq_2(I) \tag{4.1h}$$

The mathematical program (4.1) is a bilinear reformulation of the sequence-form LP [58] applied to the information set structure of a game G where the player 1 has imperfect recall and the player 2 has A-loss recall. The objective of player 1 is to find a strategy that maximizes the expected utility against the best responding opponent in G. The strategy of the maximizing player is represented as a realization plan (variables r) that assigns the probability to a sequence: $r(\sigma_1)$ is the probability that $\sigma_1 \in \Sigma_1$ will be played assuming that information sets in which actions of the sequence σ_1 are applicable are reached due to player 2. The realization plan r must satisfy the network flow Constraints (4.1b)–(4.1d). Finally, a strategy of player 1 is constrained by the best-responding opponent that selects an action minimizing the expected value of player 1 in each $I \in \mathcal{I}_2$ and for each $\sigma_2 \in seq_2(I)$ that was used to reach I (Constraint (4.1h)). These constraints ensure that the opponent plays the best response in the coarsest perfect recall refinement of G and thus also in G by Lemma 3.3.4. The expected utility for each action in Constraint (4.1h) is a sum of the expected utility values from immediately reachable information sets I' and from immediately reachable leaves. For the latter we use generalized utility function $g : \Sigma_1 \times \Sigma_2 \to \mathbb{R}$ defined as $g(\sigma_1, \sigma_2) = \sum_{z \in \mathcal{Z} | seq_1(z) = \sigma_1 \wedge seq_2(z) = \sigma_2} u_1(z) \mathcal{C}(z)$. In imperfect recall games, multiple σ_i can lead to some imperfect recall information set $I_i \in \mathcal{I}_i^{IR}$; hence, realization plans over these sequences do not have to induce the same behavioral strategy for I_i (see Section 3.6.1. Therefore, for each $I_1 \in \mathcal{I}_1^{IR}$ and each $a \in \mathcal{A}(I_1)$ we define behavioral strategy $x(a)$ (Constraints (4.1e)–(4.1f)). To ensure that

the realization probabilities induce the same behavioral strategy in I_1, we add bilinear constraint $r(\sigma_1 a) = x(a) \cdot r(\sigma_1)$ for every $\sigma_1 \in seq_1(I_1)$ (Constraint (4.1g)).

Lemma 4.3.1. *Let G be a two-player game where the maximizing player has imperfect recall and the minimizing player has A-loss recall. The assignment to r variables satisfies constraints (4.1b)–(4.1g) if and only if there exists a behavioral strategy $b_1 \in \mathcal{B}_1$ in G realization equivalent to r.*

Proof. First, we show that for every $b_1 \in \mathcal{B}_1$ there exists an assignment to r variables that satisfies constraints (4.1b)–(4.1g). For every sequence $\sigma_1 \in \Sigma_1$ we can compute such $r(\sigma_1)$ as

$$r(\sigma_1) = \begin{cases} 1, \text{ if } \sigma_1 = \emptyset \\ \prod_{a \in \sigma_1} b_1(a), \text{ otherwise.} \end{cases} \tag{4.2}$$

The network flow constraints (4.1b)–(4.1d) are satisfied from the construction of r in (4.2). Constraints (4.1e)–(4.1g) are satisfied since b_1 is a probability distribution over actions in information sets of G, and $\forall I \in \mathcal{I}_1^{IR}\ \forall a \in \mathcal{A}(I)\ b_1(a) = x(a)$.

Second, we show that for every assignment to r satisfying constraints (4.1b)–(4.1g) there exists realization equivalent $b_1 \in \mathcal{B}_1$. Such b_1 can be constructed from r in the following way. In each $I \in \mathcal{I}_1 \setminus \mathcal{I}_1^{IR}$, and for each $a \in \mathcal{A}(I)$, $b(a) = r(seq_1(I)a)/r(seq_1(I))$. In case of $I \in \mathcal{I}_1^{IR}$, however, $seq_1(I)$ is no longer a singleton. But from constraints (4.1e)–(4.1g) we know that

$$\forall \sigma_1, \sigma_1' \in seq_1(I) \forall a \in \mathcal{A}(I)\ r(\sigma_1 a)/r(\sigma_1) = r(\sigma_1' a)/r(\sigma_1') = x(a),$$

hence we can use $b_1(a) = x(a)$ for any $I \in \mathcal{I}_1^{IR}$ and $a \in \mathcal{A}(I)$. Finally, from constraints (4.1b)–(4.1d) follows that such b_1 is a probability distribution over actions in information sets of G. □

Lemma 4.3.2. *Let G be a two-player game where the maximizing player has imperfect recall and the minimizing player has A-loss recall. Assume that we fix r variables to arbitrary values in the mathematical program (4.1) applied to G, such that r satisfies constraints (4.1b)–(4.1g). The optimal objective value of such program corresponds to the worst case expected value of player 1 when playing according to r.*

Proof. We need to show that the objective value of the mathematical program (4.1) applied to G with the fixed r corresponds to the expected value of the player 1 playing r against the best responding player 2 minimizing the expected value of player 1.

Let G' be the coarsest perfect recall refinement of G for the minimizing player 2. From Lemma 3.3.4 we know that when searching for a best response to any b_1 in G, it is sufficient to find a pure best response to b_1 in G'. The sequence-form LP applied to G' ensures that the minimizing player plays a best response using constraints

$$\sum_{\sigma_1 \in \Sigma_1} g(\sigma_1, \sigma_2 a) r(\sigma_1) + \sum_{I' \in inf_2(\sigma_2 a)} v(I') \geq v(I) \quad \forall I \in \mathcal{I}_2', \forall a \in \mathcal{A}(I), \tag{4.3}$$

where $\sigma_2 = seq_2(I)$. The fact that player 2 plays a best response in the solution of the mathematical program (4.1) for G is ensured by constraints (4.1h) which are identical to constraints (4.3) applied to G'. This holds since all the pairs $\{(I, \sigma_2) |\ I \in \mathcal{I}_2 \wedge \sigma_2 \in seq_2(I)\}$ for player 2 in G exactly correspond to information sets of player 2 in G' (definition of A-loss recall). Hence, variables v and the quantifiers in (4.1h) for G exactly correspond to variables v and the quantifiers in (4.3) for G'. Finally, also the left sides of (4.1h) for G and (4.3) for G' are equal, since the extended utility function is the same in G and G' and the pairs $(I', \sigma_2 a)$ in the second sum of (4.1h) correspond to I' in the second sum of (4.3). Hence, the objective value corresponds to the expected utility of player 1 playing r against the best responding player 2 minimizing the expected value of player 1 in G' and therefore also in G. □

Theorem 4.3.1. *Let G be a two-player game where the maximizing player has imperfect recall and the minimizing player has A-loss recall. Realization plan r is a part of some optimal solution*[1] *of the mathematical program* (4.1) *if and only if r is a maxmin strategy for the maximizing player in G.*

Proof. Let b_1 be a behavioral strategy realization equivalent to r. We need to show that r is a part of some optimal solution of the mathematical program (4.1) if and only if for b_1 holds that

$$b_1 \in \arg\max_{b_1' \in \mathcal{B}_1} \min_{b_2 \in \mathcal{B}_2} u_1(b_1', b_2). \tag{4.4}$$

First, we prove that if r is a part of an optimal solution of the mathematical program (4.1) then the eq. (4.4) holds for behavioral strategy b_1 realization equivalent to r (such strategy always exists from Lemma 4.3.1) . This follows from the fact that the mathematical program maximizes the worst case expected value of player 1 (Lemma 4.3.2) over all possible strategies of player 1 (Lemma 4.3.1). Hence, b_1 is a maxmin strategy of player 1 in G.

Finally, we prove that if the eq. (4.4) holds for a behavioral strategy b_1, then the realization plan r, realization equivalent to b_1, is a part of the optimal solution of the mathematical program (4.1). Let's assume that such r is not a part of any optimal solution of the mathematical program (4.1). Since r satisfies constraints (4.1b)–(4.1g) (Lemma 4.3.1), it is a part of a valid solution of (4.1). This would imply that the mathematical program (4.1) found r' which guarantees higher worst case expected value (Lemma 4.3.2), and hence b_1 would not be a maxmin strategy for player 1. □

4.3.2 Player 2 without A-Loss Recall.

If player 2 does not have A-loss recall, the mathematical program must use each possible pure best response of player 2 (hence in the worst case each $s_2 \in \mathcal{S}_2$) as a constraint since

[1] r is only a part of the optimal solution since the r variables form a strict subset of all the variables present in (4.1).

the time consistent strategy of player 2, ensured by constraint (4.1h) in the previous case, need not be his ex-ante best response (Lemma 3.3.2). This results in the following bilinear program with size exponential in the size of the solved game.

$$\max_{x,r,v} \; v(root) \tag{4.5a}$$

$$\text{Constraints (4.1b)–(4.1g)}$$

$$\sum_{z \in \mathcal{Z} \mid \pi_2^{s_2}(z)=1} u_1(z)\mathcal{C}(z)r(seq_1(z)) \geq v(root) \qquad \forall s_2 \in \mathcal{S}_2, \tag{4.5b}$$

where $\pi_2^{s_2}(z)$ is 1 if s_2 prescribes all actions in $seq_2(z)$, 0 otherwise. Since the mathematical program (4.5) does not change the parts of the program related to the approximation of strategies of player 1, all the following approximation methods, theorems, and the branch-and-bound algorithm can also be applied to (4.5). However, the scalability would be significantly worse for the mathematical program (4.5), since even the subproblem of computing the best response of the minimizing player is NP-hard (follows from complexity results in [34]), and hence it requires exponential number of constraints (4.5b). The algorithm presented in Section 4.4.2 iteratively solves the mathematical program, hence when using the formulation (4.5) every iteration of the algorithm would require solving exponentially larger mathematical program compared to the case where the minimizing player has A-loss recall.

4.3.3 Approximating Bilinear Terms

The final technical tool that we use to formulate the mathematical program is the approximation of bilinear terms by Multiparametric Disaggregation Technique (MDT) [36]. The main idea of the approximation is to use a digit-wise discretization of one of the variables from a bilinear term. The main advantage of this approximation is a low number of newly introduced integer variables and an experimentally confirmed speed-up over the standard technique of piecewise McCormick envelopes [36].

Let $a = bc$ be a bilinear term. MDT discretizes the variable b and introduces new binary variables $w_{k,\ell}$ that indicate whether the digit k is on ℓ-th position.

$$\sum_{k=0}^{9} w_{k,\ell} = 1 \qquad \ell \in \mathbb{Z} \tag{4.6a}$$

$$w_{k,\ell} \in \{0,1\} \tag{4.6b}$$

$$\sum_{\ell \in \mathbb{Z}} \sum_{k=0}^{9} 10^{\ell} \cdot k \cdot w_{k,\ell} = b \tag{4.6c}$$

$$c^L \cdot w_{k,\ell} \leq \hat{c}_{k,\ell} \leq c^U \cdot w_{k,\ell} \qquad \forall \ell \in \mathbb{Z}, \forall k \in \{0..9\} \tag{4.6d}$$

$$\sum_{k=0}^{9} \hat{c}_{k,\ell} = c \qquad \forall \ell \in \mathbb{Z} \tag{4.6e}$$

$$\sum_{\ell \in \mathbb{Z}} \sum_{k=0}^{9} 10^{\ell} \cdot k \cdot \hat{c}_{k,\ell} = a \tag{4.6f}$$

Constraint (4.6a) ensures that for each position ℓ there is exactly one digit chosen. All digits used according to $w_{k,l}$ variables must sum to b (Constraint (4.6c)). Next, we introduce variables $\hat{c}_{k,\ell}$ that are equal to c for such k and ℓ where $w_{k,l} = 1$, and $\hat{c}_{k,\ell} = 0$ otherwise (eqs. (4.6d), (4.6e)). c^L and c^U are bounds on the value of variable c. The value of a is given by Constraint (4.6f).

This is an exact formulation that requires infinite sums and an infinite number of constraints. By restricting the set of all possible positions ℓ to a finite set $\{P_L, \ldots, P_U\}$ we get a lower bound approximation. Following the approach in [36] we can extend the lower bound formulation to compute an upper bound:

$$\text{Constraints } (4.6a),\ (4.6b), (4.6d), (4.6e)$$

$$\sum_{\ell \in \{P_L,\ldots,P_U\}} \sum_{k=0}^{9} 10^\ell \cdot k \cdot w_{k,\ell} + \Delta b = b \tag{4.7a}$$

$$0 \leq \Delta b \leq 10^{P_L} \tag{4.7b}$$

$$\sum_{\ell \in \{P_L,\ldots,P_U\}} \sum_{k=0}^{9} 10^\ell \cdot k \cdot \hat{c}_{k,\ell} + \Delta a = a \tag{4.7c}$$

$$c^L \cdot \Delta b \leq \Delta a \leq c^U \cdot \Delta b \tag{4.7d}$$

$$\left(c - c^U\right) \cdot 10^{P_L} + c^U \cdot \Delta b \leq \Delta a \tag{4.7e}$$

$$\left(c - c^L\right) \cdot 10^{P_L} + c^L \cdot \Delta b \geq \Delta a \tag{4.7f}$$

Here, Δb is assigned to every discretized variable b allowing it to take up the value between the discretization points (Constraints (4.7a)–(4.7b)). Similarly, we allow the product variable a to be increased with variable $\Delta a = \Delta b \cdot c$. To approximate the product of the delta variables, we use the McCormick envelope defined by Constraints (4.7c)–(4.7f).

4.3.4 Upper Bound MILP Approximation

We are now ready to state the main MILP for computing the upper bound on the optimal value of the bilinear program (4.1) and hence also on the maxmin value of the solved game. The MILP formulation follows the MDT example and uses ideas from Section 4.3.3 to approximate the bilinear term $r(\sigma_1)x(a)$ in Constraint (4.1g). In accord with the MDT, we represent every variable $x(a)$ using a finite number of digits of precision. Since $x(a)$ is a probability, we use $dig(\ell)$ as the function which for every precision $\ell \in \{-P..0\}$ returns the set of digits used to represent $x(a)$, i.e,

$$dig(\ell) = \begin{cases} \{0, 1\}, \text{if } \ell = 0 \\ \{0, ..., 9\} \text{ otherwise.} \end{cases}$$

Binary variables $w_{k,\ell}^{I_1,a}$ correspond to $w_{k,\ell}$ variables from (4.7) and are used for the digit-wise discretization of $x(a)$. Finally, $\hat{r}(\sigma_1)_{k,\ell}^a$ variables correspond to $\hat{c}_{k,\ell}$ variables from (4.7). To allow variable $x(a)$ to attain an arbitrary value from $[0, 1]$ interval using a finite number of digits of precision, we add an additional real variable $0 \leq \Delta x(a) \leq 10^{-P}$ that

can span the gap between two adjacent discretization points. Constraints (4.8d) and (4.8e) describe this loosening. Variables $\Delta x(a)$ also have to be propagated to bilinear terms $r(\sigma_1)\cdot x(a)$ involving $x(a)$. We cannot represent the product $\Delta r(\sigma_1 a) = r(\sigma_1)\cdot\Delta x(a)$ exactly and therefore we give bounds based on the McCormick envelope (Constraints (4.8i)–(4.8j)).

$$\max_{x,r,v} v(root,\emptyset) \tag{4.8a}$$

s.t. Constraints (4.1b) - (4.1f), (4.1h)

$$w^{I,a}_{k,\ell} \in \{0,1\} \qquad \forall I \in \mathcal{I}_1^{IR}, \forall a \in \mathcal{A}(I), \forall \ell \in \{-P..-1\}, \forall k \in dig(\ell) \tag{4.8b}$$

$$\sum_{k\in dig(\ell)} w^{I,a}_{k,\ell} = 1 \qquad \forall I \in \mathcal{I}_1^{IR}, \forall a \in \mathcal{A}(I), \forall \ell \in \{-P..0\} \tag{4.8c}$$

$$\sum_{\ell=-P}^{0} \sum_{k\in dig(\ell)} 10^{\ell}\cdot k \cdot w^{I,a}_{k,\ell} + \Delta x(a) = x(a) \qquad \forall I \in \mathcal{I}_1^{IR}, \forall a \in \mathcal{A}(I) \tag{4.8d}$$

$$0 \le \Delta x(a) \le 10^{-P} \qquad \forall I \in \mathcal{I}_1^{IR}, \forall a \in \mathcal{A}(I) \tag{4.8e}$$

$$0 \le \hat{r}(\sigma)^a_{k,\ell} \le w^{I,a}_{k,\ell} \qquad \forall I \in \mathcal{I}_1^{IR}, \forall a \in \mathcal{A}(I), \forall \sigma \in seq_1(I), \forall \ell \in \{-P..0\}, \forall k \in dig(\ell) \tag{4.8f}$$

$$\sum_{k\in dig(\ell)} \hat{r}(\sigma)^a_{k,\ell} = r(\sigma) \qquad \forall I \in \mathcal{I}_1^{IR}, \forall \sigma \in seq_1(I), \forall \ell \in \{-P..0\} \tag{4.8g}$$

$$\sum_{\ell=-P}^{0} \sum_{k\in dig(\ell)} 10^{\ell}\cdot k \cdot \hat{r}(\sigma)^a_{k,\ell} + \Delta r(\sigma a) = r(\sigma a) \qquad \forall I \in \mathcal{I}_1^{IR}, \forall a \in \mathcal{A}(I), \forall \sigma \in seq_1(I) \tag{4.8h}$$

$$(r(\sigma)-1)\cdot 10^{-P} + \Delta x(a) \le \Delta r(\sigma a) \le 10^{-P}\cdot r(\sigma) \qquad \forall I \in \mathcal{I}_1^{IR}, \forall a \in \mathcal{A}(I), \forall \sigma \in seq_1(I) \tag{4.8i}$$

$$0 \le \Delta r(\sigma a) \le \Delta x(a) \qquad \forall I \in \mathcal{I}_1^{IR}, \forall \sigma \in seq_1(I), \forall a \in \mathcal{A}(I) \tag{4.8j}$$

Note that the MILP has both the number of variables and the number of constraints bounded by $O(|\mathcal{I}|\cdot|\Sigma|\cdot P)$, where $|\Sigma|$ is the number of sequences of both players. The number of binary variables is equal to $10\cdot|\mathcal{I}_1^{IR}|\cdot\mathcal{A}_1^{max}\cdot P$, where $\mathcal{A}_1^{max} = \max_{I\in\mathcal{I}_1^{IR}}|\mathcal{A}_1(I)|$.

4.3.5 Theoretical Analysis of the Upper Bound MILP

Here we show how to use the result of the Upper Bound MILP to construct a strategy with a bounded difference of its expected worst case utility from the maxmin value of the solved game.

The variables $\Delta x(a)$ and $\Delta r(\sigma)$ ensure that the optimal value of the MILP is an upper bound on the value of the bilinear program (4.1) and therefore also on the maxmin value.

The drawback of using $\Delta x(a)$ and $\Delta r(\sigma)$ is that the realization probabilities do not have to induce a valid strategy in the imperfect recall game G, i.e., if σ^1, σ^2 are two sequences leading to an imperfect recall information set $I_1 \in \mathcal{I}_1^{IR}$ where action $a \in \mathcal{A}(I_1)$ can be played, $r(\sigma^1 a)/r(\sigma^1)$ need not equal $r(\sigma^2 a)/r(\sigma^2)$. In the following text we will show how to create a valid corrected strategy in G from r which decreases the expected value against a best responding opponent by at most ε compared to the value of the Upper Bound MILP (4.8), while deriving bound on this ε.

Let $b_1^1(I_1), \ldots, b_1^k(I_1)$ be behavioral strategies in the imperfect recall information set $I_1 \in \mathcal{I}_1^{IR}$ corresponding to realization probabilities of continuations of sequences $\sigma^1, \ldots, \sigma^k \in seq_1(I_1)$ leading to I_1. These behavioral strategies can be obtained from the realization plan as $b_1^j(I_1, a) = r(\sigma^j a)/r(\sigma^j)$ for all $\sigma^j \in seq_1(I_1)$ and $a \in \mathcal{A}(I_1)$. We will omit the information set and use $b_1(a)$ whenever it is clear from the context. Since the imperfect recall is violated in I_1, $b_1^j(a)$ may not be equal to $b_1^l(a)$ for some j, l and action $a \in \mathcal{A}(I_1)$.

Proposition 4.3.1. *Using any of the $b_1^1(I_1), \ldots, b_1^k(I_1)$ as the corrected strategy $b_1(I_1)$ in every $I_1 \in \mathcal{I}_1$ ensures that $\|b_1(I_1) - b_1^j(I_1)\|_1 \leq |\mathcal{A}(I_1)| \cdot 10^{-P}$ for every $b_1^j(I_1) \in \{b_1^1(I_1), \ldots, b_1^k(I_1)\}$, where P is the number of digits used to approximate the bilinear terms.*[2]

Proof. Let us first show that probabilities of playing action a in $b_1^1, \ldots, b_1^k$ can differ by at most 10^{-P}, i.e. $|b_1^j(a) - b_1^l(a)| \leq 10^{-P}$ for every j, l and action $a \in \mathcal{A}(I_1)$. This is based on the MDT we used to discretize the bilinear program.

Let us denote

$$\underline{r}(\sigma_1 a) = \sum_{\ell=-P}^{0} \sum_{k \in dig(\ell)} 10^{\ell} \cdot k \cdot \hat{r}(\sigma_1)_{k,\ell}^{a} \tag{4.9}$$

$$\underline{x}(I_1, a) = \sum_{\ell=-P}^{0} \sum_{k \in dig(\ell)} 10^{\ell} \cdot k \cdot w_{k,\ell}^{I_1,a}. \tag{4.10}$$

as the part of the strategy representation without the Δr and Δx variables. Notice that constraints (4.8f) and (4.8g) ensure that $\underline{r}(\sigma_1 a) = r(\sigma_1) \cdot \underline{x}(I_1, a)$. Hence, the difference in $b^1(I_1), \ldots, b^k(I_1)$ is caused solely by the $\Delta r(\sigma_1 a)$ variables. Furthermore, we know that $\Delta r(\sigma_1 a) \leq 10^{-P} \cdot r(\sigma_1)$ (Constraint (4.8i)) which ensures that the maximum difference in $b_1^1(a), \ldots, b_1^k(a)$ for any a is at most 10^{-P}. Taking any of the behavioral strategies $b_1^1, \ldots, b_1^k$ as the corrected behavioral strategy $b_1(I_1)$, therefore satisfies

$$\|b_1(I_1) - b_1^j(I_1)\|_1 \leq \sum_{a \in \mathcal{A}(I_1)} 10^{-P} = |\mathcal{A}(I_1)| \cdot 10^{-P}.$$

□

We now connect the distance of the corrected strategy $b_1(I_1)$ from the set of behavioral strategies $b_1^1(I_1), \ldots, b_1^k(I_1)$ in $I_1 \in \mathcal{I}_1^{IR}$ to the maximum possible distance in worst case expected values. First, we show this on the level of a single history. Finally, we extend

[2] The L1 norm is taken as $\|x_1 - x_2\|_1 = \sum_{a \in \mathcal{A}(I_1)} |x_1(a) - x_2(a)|$

this result to the distance of the worst case expected value of the corrected strategy b_1 from the maxmin value.

Lemma 4.3.3. *Let $h \in I_1$ be a history and b_1^1, b_1^2 be behavioral strategies prescribing different behavior in I_1 but prescribing the same behavior in all subsequent states $h \sqsubset h'$. Let $v_{max}(h)$ and $v_{min}(h)$ be maximal and minimal utility of player 1 in the subtree of h. Then the following holds:*

$$\max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(b_1^1, b_2^1) - u_1^h(b_1^2, b_2^2)| \leq \frac{v_{max}(h) - v_{min}(h)}{2} \cdot \|b_1^1(I_1) - b_1^2(I_1)\|_1,$$

where $u_1^h(b_1, b_2)$ is the expected utility of player 1, when starting in h and playing according to b_1, b_2.

Proof. When comparing b_1^1 and b_1^2, we can identify two subsets of $\mathcal{A}(I_1)$ — a set of actions A^+ where the probability of playing the action in b_1^2 is higher than in b_1^1 and A^- where the probability in b_1^2 is lower than in b_1^1. Let us denote

$$C^+ = \sum_{a \in A^+} |b_1^1(I_1, a) - b_1^2(I_1, a)| \tag{4.11}$$

$$C^- = \sum_{a \in A^-} |b_1^1(I_1, a) - b_1^2(I_1, a)| \tag{4.12}$$

We know that $C^+ = C^-$, moreover $\|b_1^1(I_1) - b_1^2(I_1)\|_1 = C^+ + C^-$. In the worst case, decreasing the probability of playing action $a \in A^-$ results in the decrease of the expected value

$$v_{max}(h) \cdot \sum_{a \in A^-} |b_1^1(I_1, a) - b_1^2(I_1, a)| = v_{max}(h) \cdot C^-.$$

Similarly the increase of the probabilities of actions in A^+ can add in the worst case $v_{min}(h) \cdot C^+$ to the expected value of the strategy. Hence,

$$\begin{aligned}
\max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(b_1^1, b_2^1) - u_1^h(b_1^2, b_2^2)| &\leq v_{max}(h) \cdot C^- - v_{min}(h) \cdot C^+ \\
&= [v_{max}(h) - v_{min}(h)] \cdot C^+ \\
&= \frac{v_{max}(h) - v_{min}(h)}{2} \cdot 2C^+ \\
&= \frac{v_{max}(h) - v_{min}(h)}{2} \cdot \|b_1^1(I_1) - b_1^2(I_1)\|_1.
\end{aligned}$$

□

Now we are ready to bound the distance of the worst case expected value of the corrected strategy b_1 from the maxmin value.

Theorem 4.3.2. *The distance of the worst case expected value of the corrected strategy b_1 from the maxmin value is bounded by*

$$\varepsilon = 10^{-P} \cdot d \cdot \mathcal{A}_1^{max} \cdot \frac{v_{max}(\emptyset) - v_{min}(\emptyset)}{2},$$

where d is the maximum number of player 1's imperfect recall information sets on any path from the root to a terminal node, $\mathcal{A}_1^{max} = \max_{I_1 \in \mathcal{I}_1^{IR}} |\mathcal{A}(I_1)|$ is the maximal branching factor and $v_{min}(\emptyset)$, $v_{max}(\emptyset)$ are the lowest and highest utilities for player 1 in the whole game, respectively.

Proof. We show an inductive way to compute the upper bound on the distance of the worst case expected value of the corrected strategy b_1 from the maxmin value. Throughout the derivation we assume that all players play to maximize the bound to guarantee that we obtain a valid upper bound. We proceed in a bottom-up fashion over the nodes in the game tree, computing the bound $L(h)$ on the maximum loss player 1 could have accumulated by correcting his behavioral strategy in the subtree of h. The ε is obtained as the value of this bound in the root of the game. The bound $L(h)$ in every $h \in \mathcal{H}$ is guaranteed to be higher or equal to

$$\max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(y_1, b_2^1) - u_1^h(b_1, b_2^2)|, \tag{4.13}$$

where y_1 is created by joining all $b_1^1(I_1), \ldots, b_1^k(I_1)$ from the solution of the Upper Bound MILP for all $I_1 \in \mathcal{I}_1$ (y_1 prescribes behavior only on a level of sequences since $b_1^1(I_1), \ldots, b_1^k(I_1)$ can specify different behavior for every sequence leading to I_1, by $y(\sigma_1, a)$ we denote the probability that a will be played after sequence σ_1), b_1 is the strategy created by correcting y_1 in the whole tree.

The description of the computation of $L(h)$ follows in a case to case manner.

(1) In leaves, $L(h) = 0$ as there is no correction made.

(2) In node h where player 2 or nature acts,

$$L(h) = \max_{a \in \mathcal{A}(h)} L(h \cdot a),$$

since there can be no loss accumulated and in the worst case the direct successor with the highest loss is chosen.

(3) In player 1's node h, which is not a part of an imperfect recall information set, no corrective steps need to be taken. The expected bound at node h is therefore $\sum_{a \in \mathcal{A}(h)} y_1(seq_1(h), a) L(h \cdot a)$. In the worst case player 1's behavioral strategy $y_1(seq_1(h))$ selects deterministically the direct successor with the highest bound, therefore again we use the bound $L(h) = \max_{a \in \mathcal{A}(h)} L(h \cdot a)$.

(4) In player 1's node h, which is a part of an imperfect recall information set, the correction step may have to be taken. Let y_1^{-h} be the strategy created from y_1 by taking corrective steps in all successors of h and let us construct a strategy y_1^h from y_1^{-h} by correcting it also in h. We know that the loss caused by changing y_1 to y_1^{-h} is at most $\max_{a \in \mathcal{A}(h)} L(h \cdot a)$, hence

$$\max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(y_1, b_2^1) - u_1^h(y_1^{-h}, b_2^2)| \leq \max_{a \in \mathcal{A}(h)} L(h \cdot a).$$

Now we have to take the corrective step in the node h and construct strategy y_1^h. When using the corrected strategy from Proposition 4.3.1, we get the following bound

(Lemma 4.3.3):

$$\max_{b_2^1,b_2^2\in\mathcal{B}_2} |u_1^h(y_1^{-h},b_2^1) - u_1^h(y_1^h,b_2^2)| \leq \frac{v_{diff}(h)}{2}\cdot 10^{-P}|\mathcal{A}_1(I_1)|$$
$$\leq \frac{v_{diff}(\emptyset)}{2}\cdot 10^{-P}\mathcal{A}_1^{max}.$$

It follows that

$$\max_{b_2^1,b_2^2\in\mathcal{B}_2} |u_1^h(y_1,b_2^1) - u_1^h(y_1^h,b_2^2)|$$
$$\leq \max_{b_2^1,b_2^2\in\mathcal{B}_2} |u_1^h(y_1^{-h},b_2^1) - u_1^h(y_1^h,b_2^2)| + \max_{b_2^1,b_2^2\in\mathcal{B}_2} |u_1^h(y_1,b_2^1) - u_1^h(y_1^{-h},b_2^2)|$$
$$\leq \frac{v_{diff}(\emptyset)}{2}\cdot 10^{-P}\mathcal{A}_1^{max} + \max_{a\in\mathcal{A}(h)} L(h\cdot a),$$

hence we use

$$L(h) = \frac{v_{diff}(\emptyset)}{2}\cdot 10^{-P}\mathcal{A}_1^{max} + \max_{a\in\mathcal{A}(h)} L(h\cdot a).$$

Finally, we provide a bound on the loss in the root node

$$L(\emptyset) \geq \max_{b_2^1,b_2^2\in\mathcal{B}_2} |u_1(y_1,b_2^1) - u_1(b_1,b_2^2)|. \tag{4.14}$$

We have shown that in order to prove the worst case bound it suffices to consider deterministic choice of action at every node h — this means that a single path in the game tree is pursued during the propagation of the bound. The bound is increased exclusively in nodes which are a part of some imperfect recall information set. We can encounter at most d such nodes on any path from the root. The increase of the bound in each such node is bounded by

$$\frac{v_{max}(\emptyset) - v_{min}(\emptyset)}{2}\cdot 10^{-P}\mathcal{A}_1^{max},$$

therefore the bound in the root is

$$\varepsilon = L(\emptyset) = \frac{v_{max}(\emptyset) - v_{min}(\emptyset)}{2}\cdot d\cdot 10^{-P}\mathcal{A}_1^{max}$$

From eq. (4.14) follows that ε is guaranteed to be higher or equal to the actual difference of worst case expected values of y_1 and b_1, since it forms an upper bound even in the case where we maximize the difference of the expected values over all pairs of player 2's strategies, while in case of the worst case expected value player 2 is restricted to playing a best response. It follows that the worst case expected value of the strategy we have found lies within the interval $[v^* - \varepsilon, v^*]$, where v^* is the worst case expected value of y_1, and therefore the optimal value of the Upper Bound MILP. As v^* is an upper bound on the solution of the original bilinear program and therefore also on the maxmin value, no strategy can have a better worst case expected value than v^*. Hence the strategy b_1 guarantees the ε distance from the maxmin value. □

4.4 Algorithms for Approximating Maxmin Strategies in Imperfect Recall Games

Algorithms for solving perfect recall games are either not applicable to imperfect recall games and A-loss recall games or they do not provide any guarantees on the quality of the obtained solutions (see 3.6 for more details). Hence, new algorithms for solving this class of games are required. In this section we describe a family of algorithms that use the Upper Bound MILP formulation (4.8) introduced in the previous section to approximate the maxmin strategy in two-player games where the maximizing player has imperfect recall and the minimizing player has A-loss recall. First, we describe a simple approach (denoted as Base). Base starts with some initial precision of the representation of bilinear terms and iteratively increases the precision until the distance of the corrected strategy obtained from the solution of the Upper Bound MILP with the current precision from the maxmin value is below a given threshold. Next, to reduce the number of binary variables and hence to improve the scalability of Base we present a branch-and-bound based algorithm (denoted as IRABnB). IRABnB works on a linear relaxation of the Upper Bound MILP and simultaneously searches the possible precision improvements of bilinear terms and the assignment to the relaxed binary variables until the error in the worst case expected value is below a given threshold. Finally, to reduce the size of the mathematical program that needs to be solved, we extend IRABnB with incremental strategy generation technique (the algorithm is denoted as DOIRABnB).

Notice, that the restriction to A-loss recall minimizing player leads to a following properties in all the algorithms.

Proposition 4.4.1. *Let G be a two player game where the maximizing player has imperfect recall and the minimizing player has A-loss recall. Let G' be the coarsest perfect recall refinement of G for the minimizing player with no modifications to the information set structure of the maximizing player. Computing the maxmin strategy in G reduces to computing the maxmin strategy in G'.*

Proof. Follows directly from Lemma 3.3.4. □

Corollary 4.4.1. *Let G be a two-player game where the maximizing player has imperfect recall and the minimizing player has A-loss recall. Let's assume that G is created as an imperfect recall abstraction of some perfect recall game G', such that G' is the coarsest perfect recall refinement of G for the minimizing player. When computing maxmin strategy in G we effectively compute the least exploitable strategy in any game with more refined information set structure of the maximizing player (hence also in G') that can be represented in G.*

Consequently, the maxmin value computed in G gives us the exploitability of the resulting strategy directly in G', hence the value can be used to evaluate the quality of the abstraction (the further the maxmin value of G is from the maxmin value of G', the more exploitable strategies resulting from solving G are).

4.4.1 Iterative Precision Refining MILP

Here we describe the Base algorithm, using the Upper Bound MILP formulation (4.8) to approximate the maxmin strategies in two-player games where the maximizing player has imperfect recall and the minimizing player has A-loss recall.

The distance of the worst case expected value of the corrected strategy b_1 from the maxmin value is a function of P, which is the precision of all approximations of bilinear terms. We design the following algorithm (denoted as Base): (1) start with the precision set to 0 for all bilinear terms, (2) for each approximation of a bilinear term calculate the current error contribution (the difference between $\Delta r(\sigma_1 a)$ and $r(\sigma_1)\Delta x(a)$ multiplied by the expected utility). Choose from the terms which do not yet have maximum allowed precision the term that contributes to the overall error the most and increase the precision of its representation by 1. The algorithm terminates when none of the terms which do not yet have maximum allowed precision contributes to the error.

4.4.2 Branch-and-Bound Algorithm

We now introduce a branch-and-bound search (denoted as IRABnB, Imperfect Recall Abstraction Branch-and-Bound algorithm) for approximating maxmin strategies of two-player games where the maximizing player has imperfect recall and the minimizing player has A-loss recall. We follow the standard practice in solving MILPs and apply the branch-and-bound search to the linear relaxation of the Upper Bound MILP. Recall, that we linearize the $w_{k,\ell}^{I_1,a}$ variables that control digit-wise discretization of $x(a)$. Furthermore, we exploit the following observation in the IRABnB.

Observation 4.4.1. *Even if the current assignment to variables $w_{k,\ell}^{I_1,a}$ is not feasible (they are not set to binary values), we can correct the resulting strategy as described in Section 4.4.2.1 and use it to estimate the lower bound on the maxmin value of player 1 without a complete assignment of all $w_{k,\ell}^{I_1,a}$ variables to either 0 or 1. The lower bound is computed as the expected value of the corrected strategy against a best response to it.*

The IRABnB algorithm starts with the linear relaxation of the Upper Bound MILP with bilinear terms approximated using 0 digits of precision. It builds and searches a branch and bound tree. In every node n of the branch and bound tree, the algorithm solves the LP corresponding to n, heuristically selects the information set I and action a contributing to the current approximation error the most, and creates successors of n by restricting the probability $b_1(I, a)$ that a is played in I. The successors are created by adding new constraints to the LP corresponding to n depending on the value of $b_1(I, a)$ by constraining (and/or introducing new) relaxed binary variables $w_{k,l}^{I_1,a}$. This way, the algorithm simultaneously searches for the optimal approximation of bilinear terms as well as the assignment to binary variables. The algorithm terminates when ε-optimal maxmin strategy is found (using the difference of the global upper bound computed by solving the LP relaxation and the lower bound computed as described in Observation 4.4.1).

Algorithm 1 depicts the complete IRABnB algorithm. The algorithm creates and traverses nodes of the branch and bound tree. Every node n has associated LP with

Algorithm 1: IRABnB algorithm

input : Initial LP relaxation LP_0 of Upper Bound MILP using a $P = 0$ discretization
output : ε-optimal strategy for a player having imperfect recall
parameters: Bound on maximum error ε, precision bounds for $x(a)$ variables $P_{max}(I_1, a)$

1 fringe $\leftarrow$ {`CreateNode`(LP_0)}
2 opt $\leftarrow$ (nil, $-\infty, \infty$)
3 **while** fringe $\neq \varnothing$ **do**
4 $\quad$ (LP, lb, ub) $\leftarrow \arg\max_{n \in \text{fringe}} n.\text{ub}$
5 $\quad$ fringe $\leftarrow$ fringe $\setminus$ (LP, lb, ub)
6 $\quad$ **if** opt.lb $\geq$ ub **then**
7 $\quad\quad$ **return** `ReconstructStrategy`(opt)
8 $\quad$ **if** opt.lb $<$ lb **then**
9 $\quad\quad$ opt $\leftarrow$ (LP, lb, ub)
10 $\quad$ **if** ub $-$ lb $\leq \varepsilon$ **then**
11 $\quad\quad$ **return** `ReconstructStrategy`(opt)
12 $\quad$ **else**
13 $\quad\quad$ $(I_1, a) \leftarrow$ `SelectAction`(LP)
14 $\quad\quad$ $P \leftarrow$ number of digits of precision representing $x(a)$ in LP
15 $\quad\quad$ fringe $\leftarrow$ fringe $\cup$ {`CreateNode`(LP $\cup$ $\{\sum_{k=0}^{\lfloor \frac{a_{ub}+a_{lb}}{2} \rfloor_P} w_{k,P}^{I_1,a} = 1\}$)}
16 $\quad\quad$ fringe $\leftarrow$ fringe $\cup$ {`CreateNode`(LP $\cup$ $\{\sum_{k=\lfloor \frac{a_{ub}+a_{lb}}{2} \rfloor_P}^{9} w_{k,P}^{I_1,a} = 1\}$)}
17 $\quad\quad$ **if** $P < P_{max}(I_1, a)$ **then**
18 $\quad\quad\quad$ fringe $\leftarrow$ fringe $\cup$ {`CreateNode`(LP $\cup$ $\{w_{LP.x(a)_{-P},P}^{I_1,a} = 1$, *introduce vars* $w_{0,P+1}^{I_1,a}, \ldots, w_{9,P+1}^{I_1,a}$ *and corresponding constraints from MDT* $\}$)}
19 **return** `ReconstructStrategy`(opt)
20 **function** `CreateNode`(LP)
21 $\quad$ ub $\leftarrow$ `Solve`(LP)
22 $\quad$ $b_1 \leftarrow$ `ReconstructStrategy`(LP)
23 $\quad$ lb $\leftarrow u_1(b_1,$ `BestResponse`$(b_1))$
24 $\quad$ **return** (LP, lb, ub)

the strategy of player 1 restricted to a certain degree of precision. Additionally, n keeps the lower bound on the overall maxmin value of player 1 and the upper bound on the values of the LPs achievable in the subtree of n. The algorithm starts in the root of the branch and bound tree, where the maxmin strategy is approximated using 0 digits of precision after the decimal point (i.e., precision $P(I_1, a) = 0$ for every variable $x(a)$). The algorithm maintains a set of active branch-and-bound nodes (fringe) and a node opt with the highest guaranteed expected value of player 1 against the best responding opponent that corresponds to the global lower bound on the worst-case guaranteed expected value. In each iteration, the algorithm selects the node with the highest upper bound from fringe (lines 4–5). If there is no potential for improvement in the unexplored parts of the branch and bound tree (i.e., all the nodes in the fringe have upper bound lower than the lower bound in opt), the current best solution is returned (line 7) (upper bounds of the nodes added to the fringe in the future will never be higher than the current upper bound).

Next, the algorithm checks whether the current solution has better lower bound than opt, if yes, the opt is replaced by the current node (line 9). Since the algorithm always selects the most promising node with respect to the upper bound, we are sure that if the lower bound and upper bound have distance at most ε, the algorithm found an ε-optimal solution and it can terminate (line 11) (upper bounds of the nodes added to the fringe in the future will never be higher than the current upper bound). Otherwise, the algorithm heuristically selects an action having the highest effect on the gap between the upper and lower bound in the selected node n (line 13, as described in Section 4.4.2.1). Next, it retrieves the precision used to represent behavioral probability of this action. By default, two successors of the current branch-and-bound node n are added, each with one of the following constraints. $x(a) \leq \lfloor \frac{a_{ub}+a_{lb}}{2} \rfloor_P$ (line 15) and $x(a) \geq \lfloor \frac{a_{ub}+a_{lb}}{2} \rfloor_P$ (line 16), where $\lfloor \cdot \rfloor_p$ is flooring of a number towards p digits of precision and a_{ub} and a_{lb} are the lowest and highest allowed probabilities of playing $x(a)$. This step performs binary halving restricting allowed values of $x(a)$ in the current precision. Additionally, if the current precision is lower than the maximal allowed precision $P_{max}(I_1, a)$ the gap between bounds may be caused by the lack of discretization points; hence, the algorithm adds one more successor with constraint $\lfloor v \rfloor_P \leq x(a) \leq \lceil v \rceil_P$, where v is the current probability of playing a, while increasing the precision used for representing $x(a)$ (line 18) (all the restriction to $x(a)$ in all 3 cases are done via $w_{k,l}^{I_1,a}$ variables).

The function `CreateNode` computes the upper bound on the values achievable in the subtree of the current node by solving the given LP (line 21) and the lower bound on the overall maxmin value of player 1 as described in Observation 1, by using the heuristic construction of a valid strategy b_1 from the solution of the given LP (line 22, as described in Section 4.4.2.1) and computing the expected value of b_1 against a best response to it.

4.4.2.1 LP for Strategy Reconstruction and Action Selection

We provide a linear program that is used as a heuristic to compute a corrected behavioral strategy in a given $I_1 \in \mathcal{I}_1$ and to estimate the contribution of the actions to the overall approximation error. It takes into account the realization probabilities $r(\sigma_1^j)$ of sequences $\sigma_1^j \in seq_1(I_1)$ leading to I_1 as well as errors that can be accumulated in the subtrees of individual histories $h \in I_1$. Let us denote by $\{1, ..., k\}$ the set of indices of all sequences in $seq_1(I_1)$. By b_1^j, for each $j \in \{1, ..., k\}$ we denote the behavioral strategy corresponding to the realization probability of sequence σ_1^j and its continuations. b_1 is the final corrected behavioral strategy.

$$\min_{b,L,\alpha} \sum_{\sigma_1^j \in seq_1(I_1)} r(\sigma_1^j) \cdot L(\sigma_1^j) \tag{4.15a}$$

$$\text{s.t.} \quad L(\sigma_1^j, a) \geq [b_1^j(a) - b_1(a)] \cdot v_{max}(\sigma_1^j \cdot a) \qquad \forall j \in \{1, ..., k\}, \forall a \in \mathcal{A}(I_1) \tag{4.15b}$$

$$L(\sigma_1^j, a) \geq [b_1(a) - b_1^j(a)] \cdot (-v_{min}(\sigma_1^j \cdot a)) \qquad \forall j \in \{1, ..., k\}, \forall a \in \mathcal{A}(I_1) \tag{4.15c}$$

$$L(\sigma_1^j) = \sum_{a \in \mathcal{A}(I_1)} L(\sigma_1^j, a) \qquad \forall \sigma_1^j \in seq_1(I_1) \tag{4.15d}$$

$$b_1(a) = \sum_{j \in \{1,\dots,k\}} \alpha(\sigma_1^j) \cdot b_1^j(a) \qquad \forall a \in \mathcal{A}(I_1) \tag{4.15e}$$

$$0 \leq \alpha(\sigma_1^j) \leq 1 \qquad \forall \sigma_1^j \in seq_1(I_1) \tag{4.15f}$$

$$\sum_{\sigma_1^j \in seq_1(I_1)} \alpha(\sigma_1^j) = 1 \tag{4.15g}$$

The LP finds a strategy minimizing the estimated error in the following way. Constraints (4.15b), (4.15c) compute the maximum cost $L(\sigma_1^j, a)$ of changing the probability that action a is played after σ_1^j, assuming that the worst possible outcome in the subtree following playing $\sigma_1^j a$ is reached. Constraint (4.15d) computes the estimated errors $L(\sigma_1^j)$ for every σ_1^j by summing all the $L(\sigma_1^j, a)$ for all relevant a. The sum of $L(\sigma_1^j)$ weighted by the realization probability of corresponding sequences is minimized in the objective. Constraints (4.15e) to (4.15g) make sure that the result will be a convex combination of all the b_1^j strategies, with the α variables being the coeficients of the convex combination.

The bound from Theorem 4.3.2 on the error of a strategy constructed in this way holds, since we have shown that the L1 distance of any pair of behavioral strategies b_1^i, b_1^j obtained from realization plans in I_1 is at most $10^{-P}|\mathcal{A}_1(I_1)|$ — the distance to their convex combination b_1 cannot be larger. Hence, the algorithm uses this LP to construct a valid strategy b_1 in every imperfect recall information set where the results prescribe inconsistent behavior.

Finally, we use

$$\sum_{\sigma_1^k \in seq_1(I_1)} L(\sigma_1^k, a)$$

as the heuristic estimate of the contribution of action a to the overall approximation error. The function SelectAction returns the action with the highest such estimate over all $I_1 \in \mathcal{I}_1^{IR}$.

4.4.2.2 Theoretical Properties of the IRABnB Algorithm

The IRABnB algorithm takes the error bound ε as the input. First, we provide a method for setting the $P_{max}(I_1, a)$ parameters to guarantee that IRABnB returns ε-maxmin strategy for player 1. Finally, we provide a bound on the number of iterations the algorithm needs to terminate. Notice that the NP-hardness result from Theorem 3.1.1 applies to both settings where IRABnB is applicable.

Theorem 4.4.1. *Let $P_{max}(I_1)$ be the maximum number of digits of precision used for representing variables $x(a), \forall a \in \mathcal{A}(I_1)$ set as*

$$P_{max}(I_1) = \left\lceil \max_{h \in I_1} \log_{10} \frac{|\mathcal{A}(I_1)| \cdot d \cdot v_{max}(h) - v_{min}(h)}{2\varepsilon} \right\rceil,$$

where $v_{min}(\emptyset)$, $v_{max}(\emptyset)$ are the lowest and highest utilities for player 1 in the whole game, respectively. With this setting IRABnB *is guaranteed to return an ε-optimal maxmin strategy for player 1.*

Proof. Let us first show that the limits on the number of refinements $P_{max}(I_1)$ are sufficient to allow representation of ε-maxmin strategy of player 1. The proof is conducted in the same case by case manner as the proof of Theorem 4.3.2. Here we focus only on the case 4 from the proof of Theorem 4.3.2, which handles histories from some imperfect recall information set. In the rest of the cases, we again assume that players play such that the action leading to the child with maximal bound on loss is chosen.

Let $I_1 \in \mathcal{I}_1^{IR}$ and $h \in I_1$. We know that when using $P_{max}(I_1)$ of digits to represent the strategy in I_1, the L1 distance between behavioral strategies in I_1 is at most $10^{-P_{max}(I_1)} \cdot |\mathcal{A}(I_1)|$ (Proposition 4.3.1). This means that the bound in h from case 4 in the proof of Theorem 4.3.2 is modified to:

$$\begin{aligned}
&\max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(y_1, b_2^1) - u_1^h(y_1^h, b_2^2)| \\
&\quad \leq \max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(y_1^{-h}, b_2^1) - u_1^h(y_1^h, b_2^2)| + \max_{b_2^1, b_2^2 \in \mathcal{B}_2} |u_1^h(y_1, b_2^1) - u_1^h(y_1^{-h}, b_2^2)| \\
&\quad \leq \frac{v_{diff}(h)}{2} \cdot 10^{-P_{max}(I_1)} \cdot |\mathcal{A}(I_1)| + \max_{a \in \mathcal{A}(h)} L(h \cdot a) \\
&\quad \leq \frac{v_{diff}(h)}{2} \cdot \frac{|\mathcal{A}(I_1)| \cdot 2\varepsilon}{|\mathcal{A}(I_1)| \cdot d \cdot v_{diff}(h)} + \max_{a \in \mathcal{A}(h)} L(h \cdot a) \\
&\quad = \frac{\varepsilon}{d} + \max_{a \in \mathcal{A}(h)} L(h \cdot a).
\end{aligned}$$

Similarly to the proof of Theorem 4.3.2, it suffices to assume that players choose actions deterministically in every node to obtain the upper bound on the error. The path induced by these choices contains at most d imperfect recall nodes, thus $L(\emptyset) = d \cdot \varepsilon / d = \varepsilon$.

Finally, we show that IRABnB is guaranteed to reach the precision guarantees which result in ε-optimal maxmin strategy. This holds since (1) the upper and lower bound on the best worst case value of player 1's strategy with a given precision restrictions are correct (follows directly from their computation), hence the branch-and-bound search never prunes away the branch with the optimal solution, (2) the IRABnB always retrieves the node with the highest upper bound from the fringe and (3) the algorithm terminates only when an ε-maxmin strategy for player 1 is found. □

Theorem 4.4.2. *When using $P_{max}(I_1)$ from Theorem 4.4.1 for all $I_1 \in \mathcal{I}_1^{IR}$ and all $a \in \mathcal{A}(I_1)$, the number of iterations of the* IRABnB *algorithm needed to find an ε-optimal solution is in*

$$\mathcal{O}\left(\left(\frac{3^{4\log_{10}(S_1)+4}}{\varepsilon^2}\right)^{S_1}\right)$$

where $S_1 = |\mathcal{I}_1^{IR}| \mathcal{A}_1^{max}$.

Proof. We start by proving that there is $n \in \mathcal{O}(3^{4|\mathcal{I}_1|\mathcal{A}_1^{max} P_{max}})$ nodes in the BnB tree, where $\mathcal{A}_1^{max} = \max_{I \in \mathcal{I}^{IR}{}_1} |\mathcal{A}(I)|$ and $P_{max} = \max_{I \in \mathcal{I}^{IR}{}_1} P_{max}(I)$. This holds since in the worst case we branch for every action in every information set, hence $|\mathcal{I}^{IR}{}_1| \mathcal{A}_1^{max}$ branchings. We can bound the number of branchings for a fixed action by $4 \cdot P_{max}$, since there are 10 digits on which we use binary halving and at most P_{max} number of digits of

precision are required. $4|\mathcal{I}_1|\mathcal{A}_1^{max}P_{max}$ is, therefore, the maximum depth of the branch-and-bound tree. Finally, the branching factor of the branch-and-bound tree is at most 3 (we add at most 3 successors in every iteration of Algorithm 1).

By substituting

$$\max_{I_1 \in \mathcal{I}_1} \left\lceil \max_{h \in I_1} \log_{10} \frac{|\mathcal{A}(I_1)| \cdot d \cdot v_{diff}(h)}{2\varepsilon} \right\rceil$$

for P_{max} in the above bound (Theorem 4.4.1), we obtain

$$\begin{aligned}
n &\in \mathcal{O}\left(3^{4S_1 \max_{I_1 \in \mathcal{I}_1} \left\lceil \max_{h \in I_1} \log_{10} \frac{|\mathcal{A}(I_1)| \cdot d \cdot v_{diff}(h)}{2\varepsilon} \right\rceil}\right), \text{where } S_1 = |\mathcal{I}_1|\mathcal{A}_1^{max} \\
&\in \mathcal{O}\left(3^{4S_1 \max_{I_1 \in \mathcal{I}_1} \left\lceil \log_{10} \frac{|\mathcal{A}(I_1)| \cdot d \cdot v_{diff}(\emptyset)}{2\varepsilon} \right\rceil}\right) \\
&\in \mathcal{O}\left(3^{4S_1 \max_{I_1 \in \mathcal{I}_1} \left\lceil \log_{10} \frac{S_1 v_{diff}(\emptyset)}{2\varepsilon} \right\rceil}\right) \\
&\in \mathcal{O}\left(3^{4S_1 \left\lceil \log_{10} \frac{S_1 \cdot v_{diff}(\emptyset)}{2\varepsilon} \right\rceil}\right) \\
&\in \mathcal{O}\left(3^{4S_1 \left(\log_{10} \frac{S_1 \cdot v_{diff}(\emptyset)}{2\varepsilon} + 1 \right)}\right) \\
&\in \mathcal{O}\left(3^{4S_1 (\log_{10}(S_1 \cdot v_{diff}(\emptyset)) - \log_{10}(2\varepsilon) + 1)}\right) \\
&\in \mathcal{O}\left(3^{4S_1 (\log_{10}(S_1 \cdot v_{diff}(\emptyset)) + 1)} 3^{-4S_1 \log_{10}(2\varepsilon)}\right) \\
&\in \mathcal{O}\left(3^{4S_1 (\log_{10}(S_1 \cdot v_{diff}(\emptyset)) + 1)} 3^{-4S_1 \frac{\log_3(2\varepsilon)}{\log_3(10)}}\right) \\
&\in \mathcal{O}\left(3^{4S_1 (\log_{10}(S_1 \cdot v_{diff}(\emptyset)) + 1)} (2\varepsilon)^{\frac{-4S_1}{\log_3(10)}}\right) \\
&\in \mathcal{O}\left(3^{4S_1 (\log_{10}(S_1 \cdot v_{diff}(\emptyset)) + 1)} (2\varepsilon)^{-2S_1}\right) \\
&\in \mathcal{O}\left(3^{4S_1 (\log_{10}(S_1 \cdot v_{diff}(\emptyset)) + 1)} 2^{-2S_1} \varepsilon^{-2S_1}\right) \\
&\in \mathcal{O}\left(3^{4S_1 \log_{10}(S_1 \cdot v_{diff}(\emptyset))} 3^{4S_1} 2^{-2S_1} \varepsilon^{-2S_1}\right) \\
&\in \mathcal{O}\left(3^{4S_1 \log_{10}(S_1 \cdot v_{diff}(\emptyset))} 21^{S_1} \varepsilon^{-2S_1}\right) \\
&\in \mathcal{O}\left(\left(\frac{3^{4 \log_{10}(S_1 \cdot v_{diff}(\emptyset)) + 4}}{\varepsilon^2}\right)^{S_1}\right)
\end{aligned}$$

Finally, we can substitute $v_{diff}(\emptyset)$ by 1, since we can modify the utility structure of the game to have utilities in $[0, 1]$ interval. □

The main disadvantage of IRABnB is that the size of the LP solved in every iteration is linear in the size of the game and the algorithm can refine the precision of bilinear term approximation in parts of the game that may not be relevant for the final solution. To reduce the size of the solved LP and to focus the refinements of the precision of bilinear term approximation to relevant parts of the game, an incremental strategy-generation technique modified for imperfect recall EFGs can be employed.

4.4.3 Double Oracle IRABnB for Imperfect Recall EFGs

In this section, we introduce the DOIRABnB (Double Oracle Imperfect Recall Abstraction Branch-and-Bound) algorithm combining ideas of IRABnB and DOEFG. Adapting the ideas of DOEFG for games with imperfect recall poses several challenges that we need to address. To solve the restricted game means to compute the maxmin strategy for player 1. However, solving the restricted game requires calling IRABnB search that iteratively refines the approximation of bilinear terms instead of solving a single (or a pair of) LPs in DOEFG for perfect recall games. DOIRABnB thus makes an integration of two iterative methods and decides when to expand the restricted game and when to refine the approximation of bilinear terms already in the restricted game.

We first provide the pseudocode of the algorithm with its description, followed by formal definitions of all the necessary components of the algorithm.

In Algorithm 2 we present the DOIRABnB algorithm. Similarly to IRABnB, the algorithm performs a branch and bound search. Every branch and bound node n stores the LP with corresponding precision adjustments to the bilinear term approximation, lower bound on the maxmin value of player 1 and an upper bound on the value achievable in the whole game in the subtree of n. The list of currently active nodes is stored in the fringe. The node with the highest lower bound encountered is stored in opt. There are two differences from IRABnB: (1) through the run, DOIRABnB incrementally builds the restricted game $\bar{G}$, and when solving the LP for any branch and bound node, the LP is always built to solve the current $\bar{G}$. (2) DOIRABnB uses function `Add` to add any node to the fringe. The function `Add` (lines 20 to 29) repeatedly uses the maximizing player oracle (line 24, Section 4.4.3.2) to make sure that before adding the node to the fringe we first update the restricted game so that solving the LP for $\bar{G}$ and current precision restriction gives an upper bound on the value of the LP applied to the original game with the same precision restrictions (see Section 4.4.3.3 for more details). This is required to guarantee the convergence of the algorithm to ε-maxmin strategy for player 1.

Note that DOIRABnB does not simply use the double oracle approach to solve LP in every single node to optimality, instead it applies the oracles of the maximizing and minimizing player separately to avoid increasing the size of the restricted game unnecessarily, while making sure that the algorithm works with valid upper bound on the value in the original game.

The algorithm starts with an empty restricted game $\bar{G}$. Lines 1 to 11 are the same as in the IRABnB algorithm. Additionally, in every iteration, DOIRABnB checks whether the bounds in the current node were computed in some smaller restricted game than the current $\bar{G}$ (line 12). If yes, DOIRABnB recomputes the bounds on the current restricted game, returns the node to the fringe (line 13) and continues with the next iteration. This is done to make sure that DOIRABnB does not make unnecessary precision adjustments due to imprecise bounds. Else, if bounds come from the same game as the current restricted game $\bar{G}$, the algorithm checks whether $\bar{G}$ can be expanded by the minimizing player oracle (line 14, see Section 4.4.3.2). If $\bar{G}$ can be expanded, we expand it, resolve with the current precision restrictions and return the node to the fringe (lines 15, 16). Note that we do not use the maximizing player oracle at this point, because the expansion by maximizing

Algorithm 2: DOIRABnB algorithm

input : Initial LP relaxation LP_0 of Upper Bound MILP, Initial restricted game $\bar{\mathsf{G}}$
output : ε-optimal strategy for the maximizing player
parameters: Bound on maximum error ε, bound P_{max} for bilinear term precision approximation

```
1  fringe ← {(LP_0, −∞, ∞)}
2  opt ← (LP_0, −∞, ∞)
3  while fringe ≠ ∅ do
4      (LP, lb, ub) ← arg max_{n∈fringe} n.ub
5      fringe ← fringe \ (LP, lb, ub)
6      if opt.lb ≥ ub then
7          return ReconstructStrategy(opt)
8      if opt.lb < lb then
9          opt ← (LP, lb, ub)
10     if ub − lb ≤ ε then
11         return ReconstructStrategy(opt)
12     if FromSmallerG(Ḡ, LP) then
13         Ḡ ←Add(Ḡ, LP)
14     else if ExpandableByMinPlayerOracle(Ḡ, LP) then
15         (Ḡ, LP) ← ExpandByMinPlayerOracle(Ḡ, LP)
16         Ḡ ←Add(Ḡ, LP)
17     else
18         (I_1, a) ← SelectAction(LP)
19         AddSuccessors(LP, I_1, a, P_max, Ḡ)
20 function Add(Ḡ, LP)
21     LP_0 ← LP, t ← 1
22     (lb, ub, B̂_2^1) ← Resolve(Ḡ, LP_0)
23     while t = 1 || LP_{t−2} ≠ LP_{t−1} do
24         (Ḡ, LP_t) ← ExpandByMaxPlayerOracle(Ḡ, LP_{t−1}, B̂_2^t)
25         (Ḡ, LP_t) ← UpdateUtilities(Ḡ, LP_t, B̂_2^t)
26         (lb, ub, B_2^{LP_t}) ← Resolve(Ḡ, LP_t)
27         B̂_2^{t+1} ← B̂_2^t ∪ B_2^{LP_t}, t ← t + 1
28     fringe ← fringe ∪ (LP_{t−1}, lb, ub)
29     return Ḡ
```

player oracle is used when adding the node to the fringe in function `Add` (as described in Section 4.4.3.3). Otherwise, if $\bar{G}$ cannot be expanded, the algorithm continues in the same way as IRABnB. It heuristically selects bilinear terms corresponding to action a from the current restricted game $\bar{G}$ (line 18, as described in Section 4.4.2.1). The algorithm then creates new nodes and adds new variables and constraints into the LPs in these nodes that further restrict possible values of $x(a)$. Next, if the maximal allowed precision permits, DOIRABnB creates an additional node with increased precision of representation of $x(a)$. Finally, it adds the new nodes to the fringe (line 19) using the `Add` function.

4.4.3.1 The Restricted Game.

This section formally defines the restricted game $\bar{G} = (\mathcal{N}, \bar{\mathcal{H}}, \bar{\mathcal{Z}}, \bar{\mathcal{A}}, \bar{u}, C, \bar{\mathcal{I}})$ of the original unrestricted game $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, u, C, \mathcal{I})$. The restricted game $\bar{G}$ is built to guarante that solving the LP with a given precision restrictions for $\bar{G}$ gives an upper bound on the solution of the LP for the original game with the same precision restrictions, when the player 2 plays some strategy from a set $\hat{\mathcal{B}}_2$. By $\hat{\mathcal{B}}_2$ we denote a subset of all strategies of player 2 from the restricted game $\bar{G}$ extended by the default strategy (playing first action available in each decision point). In Section 4.4.3.2 we present the oracles used to construct the restricted game to guarantee such bound. In Section 4.4.3.3 we explain how to iteratively build $\hat{\mathcal{B}}_2$ which combined with the use of the oracles guarantees that the solution of the LP gives an upper bound, this time when the player 2 can play any strategy from the original game G.

The restricted game is limited by a set of allowed sequences $\bar{\Phi} \subseteq \Sigma$, that are returned by the oracles. An allowed sequence $\sigma_i \in \bar{\Phi}$ might not be playable to the full length due to missing compatible sequences of the opponent. Therefore, the restricted game is defined using the maximal compatible set of sequences $\bar{\Sigma} \subseteq \bar{\Phi}$. Formally

$$\bar{\Sigma}_i = \{\sigma_i \in \bar{\Phi}_i | \exists \sigma_{-i} \in \bar{\Phi}_{-i}\ \exists h \in \mathcal{H} : seq_i(h) = \sigma_i \wedge seq_{-i}(h) = \sigma_{-i}\}, \forall i \in \mathcal{N}. \tag{4.16}$$

The sets $\bar{\mathcal{H}}$, $\bar{\mathcal{A}}$ are the subsets of $\mathcal{H}$, $\mathcal{A}$ reachable when playing sequences from $\bar{\Sigma}$. $\bar{\mathcal{I}}$ defines the same partition as $\mathcal{I}$ on the reduced set $\bar{\mathcal{H}}$, i.e., for all $h, h' \in \bar{\mathcal{H}}$, holds that $h, h' \in I$ for some $I \in \bar{\mathcal{I}}$ in the restricted game if and only if $h, h' \in I$ for some $I \in \mathcal{I}$ in the original game. The set of leaves in $\bar{G}$ is a union of leaf nodes of G present in $\bar{G}$ and inner nodes from G that do not have a valid continuation in $\bar{\Sigma}$

$$\bar{\mathcal{Z}} = (\mathcal{Z} \cap \bar{\mathcal{H}}) \cup \{h \in \bar{\mathcal{H}} \setminus \mathcal{Z} | \bar{\mathcal{A}}(h) = \emptyset\}. \tag{4.17}$$

We refer to the members of the set $\bar{\mathcal{Z}} \setminus \mathcal{Z}$ as *temporary leaves*. Note that if not stated otherwise, when we operate with a strategy from the restricted game in the whole unrestricted game, we automatically assume that it is extended by a default strategy playing the first action available as in DOEFG.

We define the *temporary utility value* in every $z \in \bar{\mathcal{Z}}$ as $\bar{u}_1(z, \hat{\mathcal{B}}_2)$ so that $\bar{u}_1(z, \hat{\mathcal{B}}_2)$ is an upper bound on the value the player 1 can guarantee in the original game G in z, when the minimizing player plays any strategy from the set $\hat{\mathcal{B}}_2$. Formally, we use

$$\bar{u}_1(z, \hat{\mathcal{B}}_2) = \max_{b_2 \in \hat{\mathcal{B}}_2} \hat{u}_1^z(b_2), \forall z \in \bar{\mathcal{Z}}, \tag{4.18}$$

where $\hat{u}_1^z(b_2)$ stands for the expected value of player 1 in the original game G when starting in z and playing a strategy from the coarsest perfect recall refinement of G maximizing the expected value in z against b_2 (remember that b_2 is extended by the default strategy). We define $\hat{u}_1^z(b_2)$ in such way since the best response of player 1 against b_2 in the coarsest perfect recall refinement is easy to compute as shown in Lemma 3.3.4 (player 1 has imperfect recall in G hence computing the best response there is NP-hard). Furthermore,

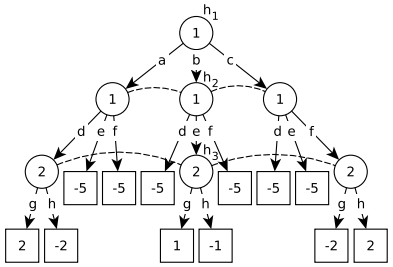

Figure 4.4: An A-loss recall game where the maxmin strategy for player 1 is not a best response to any of the pure best responses of player 2.

since we give more information to player 1 in the coarsest perfect recall refinement, $\hat{u}_1^z(b_2)$ is guaranteed to be an upper bound on the maximal expected value in z achievable by player 1 against $b_2 \in \hat{\mathcal{B}}_2$ in G. The set $\hat{\mathcal{B}}_2$ is built in function `Add` using best responses of player 2 taken from the solution of the LP by finding actions corresponding to active Constraint (4.1h) (see Section 4.4.3.2 for details). Notice that the $\bar{u}_1$ might differ in every iteration of the algorithm, since $\hat{\mathcal{B}}_2$ can change.

4.4.3.2 Updating the Restricted Game.

In this section we dicuss the oracles used in DOIRABnB and the way their results are used to expand the restricted game (lines 15 and 24 in Algorithm 2). Note that the oracle of the maximizing player is given $\hat{\mathcal{B}}_2$ (see Section 4.4.3.3 for details on construction of $\hat{\mathcal{B}}_2$) and expands the restricted game with respect to the strategies in $\hat{\mathcal{B}}_2$.

Minimizing player oracle. The minimizing player plays a best response in the final maxmin solution of the game, hence, similarly to DOEFG we use the best response computation as the oracle of player 2. In every iteration we compute $b_2^{BR} \in BR_2(b_1)$ in the original game G, where b_1 is the strategy of player 1 computed by DOIRABnB in the current node and current restricted game extended by the default strategy. The algorithm extends $\bar{\Phi}_2$ by all the valid continuations of $\sigma_2 \in \bar{\Phi}_2$ by actions in b_2^{BR} and update $\bar{\Sigma}$ accordingly.

Maximizing player oracle. The best response is not a sufficient oracle of the maximizing player 1 since his maxmin strategy does not have to consist of best responses to pure strategies of the minimizing player.

Example 4.4.1. *Consider the game in Figure 4.4. The maxmin strategy for player 1 is playing b and e deterministically, guaranteeing the maxmin value* -1*. Notice, however, that playing b and e is not a best response to any pure strategy of player 2. Since during the run the* DOIRABnB *only computes pure best responses of player 2, the best response oracle for the maximizing player 1 would never add states* h_2 *and* h_3 *and so the* DOIRABnB *would never find the correct solution.*

To fix this, the algorithm keeps track of possible extensions of the restricted game by taking actions in states of the maximizing player 1. To do that, the algorithm uses a set

of pending states

$$\mathcal{H}_p = \{h \in \mathcal{H} \setminus \bar{\mathcal{H}} | \exists h' \in \bar{\mathcal{H}}_1 \exists a \in A(h') : h'a = h\}, \tag{4.19}$$

which contains all the states h not in the restricted game, whose parent h' is in the restricted game and player 1 makes decision in h'. Since we build $\bar{G}$ to find strategy of player 1, which is optimal against the strategies from $\hat{\mathcal{B}}_2$ there is no point in adding $h \in \mathcal{H}_p$ which are not reachable by any $b_2 \in \hat{\mathcal{B}}_2$. Hence, we take a subset $\mathcal{H}'_p \subseteq \mathcal{H}_p$ such that all $h \in \mathcal{H}'_p$ are reachable by some $b_2 \in \hat{\mathcal{B}}_2$. Furthermore, we can exclude pending states which cannot improve the expected value of the player 1 against any $b_2 \in \hat{\mathcal{B}}_2$. Formally, by $\mathcal{H}^*_p$ we denote a subset of $\mathcal{H}'_p$, where for all $h \in \mathcal{H}^*_p$ holds that

$$\bar{u}_1(h, \hat{\mathcal{B}}_2) \geq \min_{b_2 \in \hat{\mathcal{B}}_2} u_1^{h'}(b_1, b_2),$$

where h' is the parent of h, b_1 is the strategy of player 1 from the current LP extended by the default strategy and $u_1^h(b_1, b_2)$ stands for the expected value in state h when players play according to b_1, b_2. When expanding the restricted game, we add to $\bar{\Phi}$ all the sequences leading to all $h \in \mathcal{H}^*_p$.

Example 4.4.1 (continued)**.** *When using the pending states as an oracle of player 1 in the game in Figure 4.4, we always add the states h_2 and h_3. For example when h_2 is not a part of the restricted game, player 1 cannot play action b. h_2 is then added by the maximizing player oracle since all the strategies of player 1 not playing action b can guarantee the expected value of at most -2 in the h_1 against the worst case opponent. On the other hand, $\bar{u}_1(h_2, \hat{\mathcal{B}}_2) \geq -1, \forall \hat{\mathcal{B}}_2$ and so h_2 is added to the restricted game.*

Finally, let us explain the functions used in Algorithm 2. `ExpandableByMinPlayerOracle` checks whether the oracle of the minimizing player suggests any sequence to be added to the restricted game. `ExpandByMinPlayerOracle` and `ExpandableByMaxPlayerOracle` add to the restricted game all the sequences suggested by the minimizing player oracle and the maximizing player oracle respectively.

4.4.3.3 Adding Nodes to the fringe.

There are two requirements $\bar{G}$ needs to fulfill before adding any given node to the fringe in function `Add` (lines 20 to 29) in order to guarantee that solving the LP for $\bar{G}$ and given precision restrictions gives an upper bound on the value of the LP for the original game with the same precision restrictions. (1) The function `Add` needs to make sure that the restricted game is built so that player 1 has no deviation outside of the restricted game, which could increase his expected value. Let us demonstrate this in the following example.

Example 4.4.2. *Consider the game in Figure 4.5 (left). Assume that the restricted game $\bar{G}$ consists of states $\{h_1, h_2, h_3, z_1, z_2, z_3, z_4\}$ and there are no precision restrictions. If we solve the restricted game we obtain the value 0 for player 1; however, maxmin strategy for player 1 is to play x guaranteeing the expected value* 1*.*

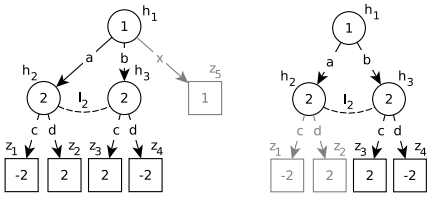

Figure 4.5: Games for demonstration of the necessity of the `Add` function.

(2) The utility $\bar{u}_1$ in all $z \in \bar{\mathcal{Z}}$, which is going to be used to construct the LP must be an upper bound on the worst case utility of player 1 against all the possible best responses $b_2 \in \mathcal{B}_2^{LP}$ corresponding to active Constraints (4.1h) after solving this LP. Hence the utility must be set according to the possible behavior of the minimizing player, which depends on the utility in question. This is required since strategies $b_2 \in \mathcal{B}_2^{LP}$ can define behavior also outside of the restricted game due to information sets of player 2 that can be only partially present in the restricted game. It is, therefore, insufficient to assume that player 2 plays using only default strategy in every information set outside of the restricted game. Since the algorithm sets the utility in the $\bar{\mathcal{Z}} \setminus \mathcal{Z}$ to a fixed value (therefore not reflecting the changing behavior of player 2 in the LP), it needs to make sure that the value is an upper bound against all possible strategies player 1 can face to obtain the required upper bound by solving the LP. Let us demonstrate this in the following example.

Example 4.4.3. *Consider the game in Figure 4.5 (right). Assume that the restricted game $\bar{G}$ consists of states $\{h_1, h_2, h_3, z_3, z_4\}$ and there are no precision restrictions. h_2 is a temporary leaf in $\bar{G}$; hence we need to compute a temporary utility value for it. Let us first discuss what would happen if we do not consider the behavior in the restricted game and use the value from the leaf reachable after the default strategy (playing the first action in every state). The default strategy leads to the terminal state z_1 with utility -2. Solving the restricted game using -2 as the temporary utility value for h_2 leads to strategy with the worst case expected value -2. However, the maxmin value of player 1 in the original game is 0 achievable by playing uniformly in h_1.*

In the function `Add` we iteratively update the restricted game until we are guaranteed to obtain a correct upper bound. To do that, the function `Add` builds in every iteration T a set $\hat{\mathcal{B}}_2^T$ as a union of all the $\mathcal{B}_2^{LP_t}$ obtained by solving the LP_t in every iteration $t \in \{0, ..., T-1\}$ (lines 26, 27) in the current invocation of the function `Add`. In every iteration T the function `Add` expands $\bar{G}$ using the oracle of the maximizing player for the current set $\hat{\mathcal{B}}_2^T$ (line 24) and updates the utilities in $\bar{G}$ again using the current set $\hat{\mathcal{B}}_2^T$ (line 25). The algorithm iterates in function `Add` until the LPs from last two iterations are equal, and only then is the given node added to the fringe.

Example 4.4.2 (continued)**.** *Consider again the game in Figure 4.5 (left). The function* `Add` *will ensure that z_5 is added to $\bar{G}$ since $z_5 \in \mathcal{H}_p^*$ and so it will be added by the maximizing player oracle.*

Example 4.4.3 (continued)**.** *Consider again the game in Figure 4.5 (right). We did not receive a correct upper bound by solving the restricted game, since setting the temporary utility value to* -2 *in* h_2 *is incorrect. Player 2 plays action* d *in the solution of the* LP_0 *to force the resulting value to be* -2 *but the* -2 *in* h_2 *was obtained assuming that player 2 will play* c *as a part of his default strategy (playing* c *and* d *is mutually exclusive). To solve this, the function* `Add` *performs another iteration, where it sets the utility in* h_2 *to* $\bar{u}_1(h_2, \hat{\mathcal{B}}_2^1)$*, where* $\hat{\mathcal{B}}_2^1 = \mathcal{B}_2^{LP_0}$ *is the singleton containing the strategy playing* d *obtained as the best response from the solution of the* LP_0*. In this iteration the algorithm correctly sets* $\bar{u}_1(h_2, \hat{\mathcal{B}}_2) = 2$*. After solving* $\bar{G}$ *we get value* 2*, which is the desired upper bound on the maxmin value in the original game with no precision restrictions.*

4.4.3.4 Theoretical Properties.

Here we demonstrate that if IRABnB is guranteed to find ε-optimal maxmin strategy for some precision parameters P_{max}, DOIRABnB is also guaranteed to find ε-optimal maxmin strategy for the same P_{max}.

Lemma 4.4.1. *Every node* $n \in$ fringe *in every iteration of* DOIRABnB *has a valid lower bound on the maxmin value of player 1 in the original game* G*.*

Proof. The lower bound is valid, since it is computed as $u_1(b_1, b_2)$, where b_1 is the current solution of the LP corresponding to n applied to the current restricted game $\bar{G}$, extended by the default strategy and $b_2 \in BR_2(b_1)$ in the original game G. Since the maxmin strategy b_1^* of player 1 maximizes its expected value assuming the worst case opponent, $u_1(b_1, b_2)$ must be lower or equal to the maxmin value of the game. □

Lemma 4.4.2. *The upper bound in every node* $n \in$ fringe *for the corresponding precision restrictions in every iteration of* DOIRABnB *forms an upper bound on the value of the LP with the same precision restrictions applied to the original game* G*, hence the upper bound in the node* n *is higher or equal to the upper bound in the node used in* IRABnB *applied to* G *with the same precision restrictions.*

Proof. We add all nodes to the fringe using function `Add`. Since we iterate in function `Add` until the LP_T in the current iteration T is equal to the LP_{T-1}, we are sure that there is no deviation of the player 1 outside of $\bar{G}$ which can increase his worst case expected value against any $b_2 \in \mathcal{B}_2^{LP_{T-1}}$, since adding any pending state would cause LP_T to be different from LP_{T-1}. Additionally, since the LP_T and LP_{T-1} are equal, we are sure that none of the $b_2 \in \mathcal{B}_2^{LP_{T-1}}$ change the utility structure created using $\hat{\mathcal{B}}_2^{T-1}$, hence

$$\bar{u}_1(z, \hat{\mathcal{B}}_2^{T-1}) \geq \bar{u}_1(z, \mathcal{B}_2^{LP_{T-1}}), \forall z \in \bar{\mathcal{Z}}. \tag{4.20}$$

Player 1 cannot increase his expected value by playing outside of the restricted game against any $b_2 \in \mathcal{B}_2^{LP_{T-1}}$. Furthermore, every $z \in \bar{\mathcal{Z}}$ has assigned an upper bound on the expected value player 1 can guarantee in z against any $b_2 \in \mathcal{B}_2^{LP_{T-1}}$ in the original game. Hence, the value of the LP_{T-1} is an upper bound on the value of the LP with the same

precision restrictions in G. Finally, since LP_T is equal to LP_{T-1} the same holds for the value of LP_T. □

Theorem 4.4.3. *If the* IRABnB *algorithm is guaranteed to return ε-optimal maxmin strategy for precision parameters P_{max}, the* DOIRABnB *returns ε-optimal maxmin strategy for the same precision parameters P_{max}.*

Proof. When DOIRABnB reaches a node where the upper and lower bound are at most ε distant, we are sure that we have found an ε-optimal solution of the original game. This holds since (1) the node has correct bounds on the value achievable in the original game with given precision restrictions (Lemmas 4.4.1, 4.4.2), (2) the DOIRABnB always retrieves the node with the highest upper bound from the fringe (line 4). Additionally, DOIRABnB is guaranteed to reach this node, since it never prunes away the branch containing the optimal solution in the space of precision restrictions (again from the correctness of the upper and lower bound from Lemmas 4.4.1, 4.4.2).

When DOIRABnB reaches the node with the precision restrictions guaranteeing an ε-optimal solution in the original game, the bounds might be more than ε distant due to the insufficiently built restricted game. This is caused by the temporary leaves $z \in \bar{\mathcal{Z}} \setminus \mathcal{Z}$. The DOIRABnB assigns to every $z \in \bar{\mathcal{Z}} \setminus \mathcal{Z}$ a temporary utility which is an upper bound on the actual utility which player 1 can guarantee against the worst-case opponent in z in the original game. Hence the strategy b_1 computed for the current restricted game can prefer some temporary leaf $z \in \bar{\mathcal{Z}} \setminus \mathcal{Z}$ based on this possibly overestimated utility value. However, when computing the $b_2^{BR} \in BR(b_1)$ where b_1 is extended by the default strategy in the original game, the value $u_1^z(b_1, b_2^{BR})$ can be significantly smaller than the temporary utility value in z. The oracle of the minimizing player, however, expands $\bar{G}$ by actions in b_2^{BR}. Since b_2^{BR} exploits the difference in the temporary utility in z and the actual expected value in z obtained when playing according to b_1, b_2^{BR}, z has to be reached when playing according to these strategies. The restricted game is, therefore, expanded by the action $a \in \mathcal{A}(z)$ played in b_2^{BR} and z is no longer a temporary leaf after the expansion. The expansion of the temporary leaves continues until there is no temporary leaf where player 2 can exploit the overestimated value of the temporary utility (lines 14, 24 in Algorithm 2). Hence the reason for the difference in the bounds directly implies that the expansion of the restricted game on line 14 will occur. The DOIRABnB terminates when the restricted game is built sufficiently to allow the distance of bounds to decrease to at most ε. This must happen after a finite number of steps since in the worst case the algorithm builds the entire original game.

Finally, the fact that there is ε-optimal solution when using given precision parameters P_{max} is guaranteed by the assumption that IRABnB is guaranteed to find the ε-optimal solution for the same parameters in the original game. □

4.5 Experiments

In this section, we provide an experimental evaluation of the performance of DOIRABnB, IRABnB and the Base. Furthermore, we demonstrate the possible space savings

in the size of the strategy representation when employing imperfect recall abstractions and discuss the quality of strategies resulting from solving these abstractions. Since there is no standardized collection of benchmark EFGs, the experiments are conducted on a set of Random games, an imperfect recall search game and an imperfect recall variant of OshiZumo. All algorithms were implemented in Java, each algorithm uses a single thread, 8 GB memory limit and we use IBM ILOG CPLEX 12.6.2 to solve all LPs/MILPs.

Random Games. We use randomly generated games to obtain statistically significant results. We randomly generate a perfect recall game with varying a branching factor and a fixed depth of 6. To control the information set structure, we use observations assigned to every action – for player i, nodes h with the same observations generated by all actions in history belong to the same information set. To obtain imperfect recall games with a non-trivial information set structure, we run a random abstraction algorithm which merges information sets for all $i \in \mathcal{N}$ according to parameter p in the following way. Let $\{\mathcal{I}_i^1, ..., \mathcal{I}_i^n\}$ be the largest possible disjoint subsets of $\mathcal{I}_i$ of the perfect recall game such that

$$\forall \mathcal{I}_i^k \in \{\mathcal{I}_i^1, ..., \mathcal{I}_i^n\} \forall I_i, I_i' \in \mathcal{I}_i^k | seq_i(I_i)| = |seq_i(I_i')| \wedge |A(I_i)| = |A(I_i')|,$$

and $\bigcup_{\mathcal{I}_i^k \in \{\mathcal{I}_i^1, ..., \mathcal{I}_i^n\}} \mathcal{I}_i^k = \mathcal{I}_i$. Each $\mathcal{I}_i^k \in \{\mathcal{I}_i^1, ..., \mathcal{I}_i^n\}$ contains candidates for merging. Let $\mathcal{J} \subseteq \mathcal{I}_i^k$ be a set that initially contains a random element $I \in \mathcal{I}_i^k$. We iterate over all $I' \in \mathcal{I}_i^k \setminus I$, and add I' to $\mathcal{J}$ with probability p. To create the abstraction we iteratively choose the subset $\mathcal{J}$ of $\mathcal{I}_i^k$, create abstracted set containing all elements of $\mathcal{J}$, and remove $\mathcal{J}$ from $\mathcal{I}_i^k$. This procedure repeats until $\mathcal{I}_i^k$ is empty and is performed for all $\mathcal{I}_i^k \in \{\mathcal{I}_i^1, ..., \mathcal{I}_i^n\}$. We further update the abstraction by splitting the information sets of the minimizing player to make sure that he has A-loss recall. We generate a set of experimental instances by varying the branching factor and the parameter p. Our random games are rather difficult to solve since (1) information sets can span multiple levels of the game tree (i.e., the nodes in an information set often have histories with differing sizes) and (2) actions can easily lead to leaves with very different utility values.

OshiZumo. We use a modification of OshiZumo described, e.g., in [5]. The game is played by two players; both start with a given number of coins. At the beginning of a game, a sumo wrestler is positioned at the center of a one-dimensional playing field which consists of 7 positions. In every turn of the game, each player uses some amount of his coins to place a bid. The highest bidder pushes the wrestler one location towards the opponent's side. If the bids are equal, the wrestler does not move. Either way, both players lose all the coins they used to make the bid and the game proceeds until the money runs out or the wrestler is pushed off the field. The players observe only their bid and whether they won or not. The bid of the opponent is never revealed. It is a zero-sum game, where the final position of the wrestler determines the winner: if he is located at the center, the game result is a draw. Otherwise, the player in whose half the wrestler is located gets a negative utility equal to the number of positions between the wrestler and the center of the playing field. In this paper, we create different instances of OshiZumo by changing the number of coins available to players.

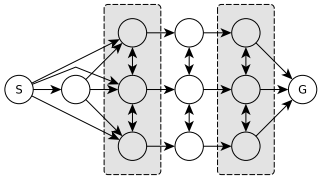

Figure 4.6: Graph for the Search game. The attacker starts in the node S and tries to reach the node G. The defender operates two units, each moving in one of the shaded areas.

To create the imperfect recall abstraction of the OshiZumo, we give the maximizing player only the information about the number of coins he has left, and whether he has won in each of the previous rounds, hence he does not remember the exact bids he had made. The minimizing player remembers both his bids and whether he won in each round. Notice that we do not modify the information set structure of the minimizing player, and so the original game is the coarsest perfect recall refinement of this abstraction for the minimizing player.

Search Game. Our third domain is an instance of search (or pursuit-evasion) game, used, e.g., in [7]. Search games are commonly used for evaluating incremental algorithms [44]. The game is played on a directed graph shown in Figure 4.6 between attacker and defender. The attacker tries to cross from the starting node S to his destination G. The attacker can either move every turn, leaving tracks in each node he visits, or he can move every other turn without leaving any tracks. The defender operates two units, each moving in one of the shaded areas, trying to intercept the attacker by capturing him in a node. The defender observes only the tracks left by the attacker and only in case one of his units steps on the node with the track. The attacker does not have any information about the defender units. The players move simultaneously. It is a zero-sum game, where the attacker obtains utility 1 for reaching his destination and defender obtains utility 1 for intercepting the attacker. If a given number of moves is depleted without either of the events happening, the game is considered a draw and both obtain utility 0. We assume the defender to be the maximizing player.

To create an imperfect recall abstraction of the Search game, we give the defender only information about the tracks he currently observes and the position of both of his units without remembering the history of moves leading there. The attacker knows only the sequence of his actions in the past. Notice that we do not modify the information set structure of the minimizing player, and hence the original game is the coarsest perfect recall refinement of this abstraction for the minimizing player.

4.5.1 Results

The main experiments are divided into 2 parts. (1) We compare the Base, IRABnB and DOIRABnB algorithms to show how the different components used in the algorithms influence the scalability. The results show that DOIRABnB outperforms IRABnB and

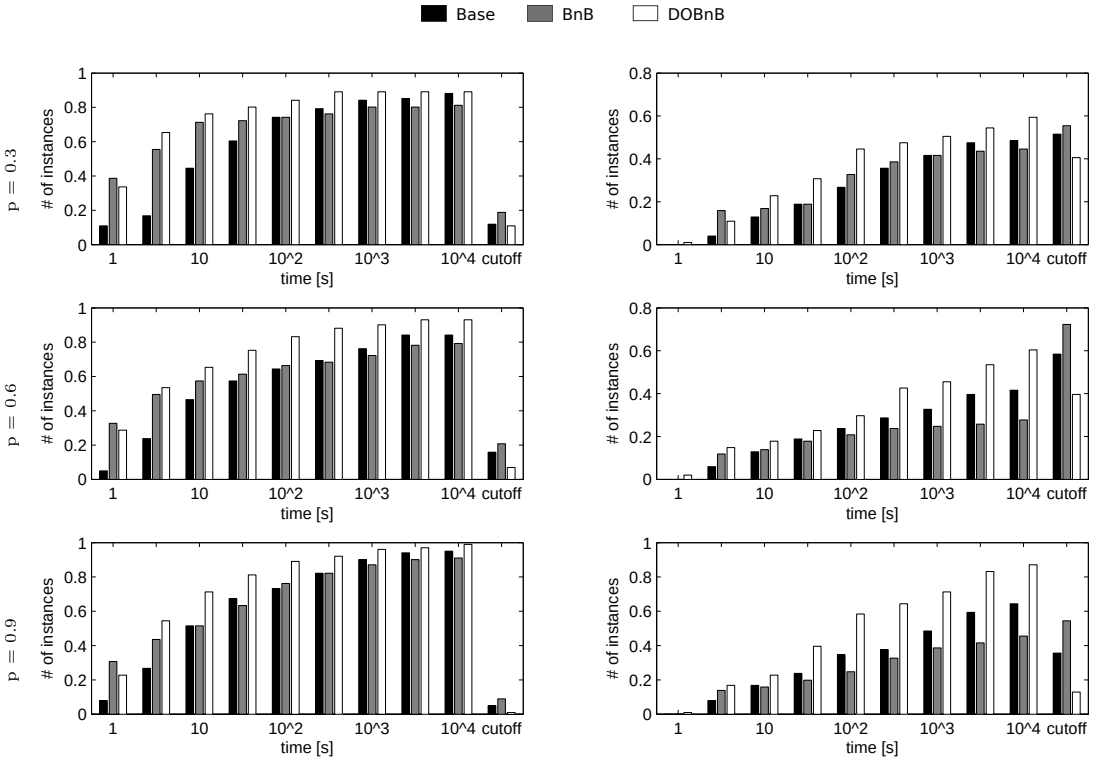

Figure 4.7: Results for random games showing the relative cumulative number of instances (y-axis) solved under a given time limit (x-axis) and the relative amount of instances terminated due to the exceeded runtime in bars labeled *cutoff*. Rows contain results for $p = 0.3, p = 0.6, p = 0.9$, columns show results for branching factor 3 and 4.

Base on smaller domains while providing significantly better scalability than the rest thanks to the fact that the incremental strategy generation keeps the LP being solved small while focusing the precision adjustments to relevant parts of the game tree. (2) We demonstrate the immense space savings in the strategy representation achievable by employing the simple imperfect recall abstractions described above. Additionally, we show that solving these abstractions results in finding the maxmin strategy of the original unabstracted game.

The ε in all the experiments was set to $10^{-4} \cdot u_{max}$, where u_{max} is the maximal utility of the solved game.

4.5.1.1 Runtime comparison

Random Games. In Figure 4.7 we present the runtime results in seconds obtained on random games. Every plot depicts the cumulative relative number of instances (y-axis) solved under a given time limit (logarithmic x-axis). There were 100 instances of random games solved for every setting. The rows contain results for random games with varying p, the first column for branching factor 3, second for branching factor 4. The runtime of

the algorithms was limited to 2 hours on every instance, the relative number of instances terminated after this limit is reported in the bars labeled *cutoff*. The IRABnB usually dominates the other two algorithms in the number of instances solved in the first time interval. As the runtime grows, however, the performance of IRABnB decreases. This is because IRABnB tends to spend a lot of time making adjustments in the irrelevant parts of the game tree. On the other hand, the DOIRABnB outperforms the other two algorithms across all the settings, and we can see a significant decrease in the number of instances not solved in a given 2-hour limit, compared to Base and IRABnB. This is because the DOIRABnB focuses adjustments to approximation precision to the relevant parts of the game tree present in the restricted game while keeping the underlying LP smaller. Note that the random games form an unfavorable scenario for all the presented algorithms since the construction of the abstraction is completely random, which makes conflicting behavior in merged information sets common. As we can see, however, even in these scenarios the DOIRABnB is capable of solving the majority of instances with branching factor 4 which have ~ 3000 nodes in under 2 hours.

Table 4.1: Average relative amount of sequences for maximizing and minimizing player respectively, added to the restricted game by DOIRABnB in random games.

$p \setminus$ b.	3	4
0.3	$46.1\% \pm 2.9\%$, $22.2\% \pm 1.8\%$	$62.5\% \pm 3.1\%$, $17.5\% \pm 2.2\%$
0.6	$58.9\% \pm 2.8\%$, $23.3\% \pm 2.0\%$	$71.7\% \pm 2.8\%$, $17.1\% \pm 2.1\%$
0.9	$68.2\% \pm 2.5\%$, $24.0\% \pm 1.8\%$	$76.6\% \pm 2.9\%$, $18.5\% \pm 1.1\%$

In Table 4.1 we present the average relative amount of sequences for each player needed by DOIRABnB to solve the random games for each setting along with the standard error. The relative amount of sequences needed by the minimizing player is consistently smaller because the restricted game is built to compute maximizing player's robust strategy, while the minimizing player only plays best responses during the computation. Even though the size of the restricted game remains similar across all values of p, we observe an increase in the relative size, since the number of sequences decreases as p increases.

OshiZumo. The DOIRABnB solved the game with 11 coins in 44 minutes using 0.9% sequences for the maximizing player and 0.2% sequences for the minimizing player. The game has $3.5 \cdot 10^6$ states, $2.8 \cdot 10^5$ and $1.4 \cdot 10^6$ sequences for the maximizing and minimizing player respectively. IRABnB and Base were able to solve the game with 9 coins in 20 seconds and 2 hours respectively, however, on the game with 10 coins none of the two algorithms finished in 10 hours.

Search game. In case of Search game, the DOIRABnB was able to solve a game with 10 moves allowed for each player (with $\sim 5 \cdot 10^9$ states, $\sim 2 \cdot 10^4$ sequences for the attacker and $\sim 4 \cdot 10^7$ sequences for the defender) using 0.002% of sequences for the defender and 0.2% sequences for the attacker in 1.2 hours. IRABnB and Base were able to solve the game with 5 moves for every player in 12 seconds and 40 minutes respectively, however, on the game with 6 moves none of the two algorithms finished in 10 hours.

The presented results show that DOIRABnB provides scalability which cannot be achieved by IRABnB and Base because of their requirement to build the entire game. Furthermore, the results on random games show that even on small games, where IRABnB

and Base can be used, DOIRABnB provides the best performance and hence it is the most efficient algorithm.

4.5.1.2 Size of Strategy Representation and Quality of Resulting Strategies

Here, we discuss the size of strategy representation needed in imperfect recall abstractions compared to their perfect recall counterparts. Note that we use the number of information sets of the maximizing player present in a given game for this purpose since in the worst case a strategy needs to define behavior in each of them. Additionally, we provide results showing the quality of the strategy resulting from solving these abstractions in the original unabstracted game.

Table 4.2: The relative amount of information sets of the maximizing player in the imperfect recall abstractions with different number of remembered moves w.r.t. the number of information sets of the maximizing player in the original game for the OshiZumo with 10 and 11 coins (left) and the Search game with depth 6 and 7 (right).

c. \ r. m.	0	1	2
10	3.27%	5.54%	9.30%
11	1.90%	3.27%	5.64%

d. \ r. m.	0	1	2
6	0.10%	0.46%	1.81%
7	0.03%	0.11%	0.42%

In Table 4.2 we present the relative amount of information sets of the maximizing player in a specific abstraction compared to the unabstracted game for OshiZumo (left) and Search game (right). In both domains, we use the perfect recall game and its imperfect recall abstraction with rules described earlier in this section. Moreover, in both domains, we experiment with refining the abstraction by giving the maximizing player information about the last k moves he has made in the past (k is specified in the first row of every column). In the case of OshiZumo, each row of the table represents the setting with the number of coins specified in the first column. In the case of Search game, every row corresponds to a different number of moves allowed for every player specified in the first column. As you can see the number of information sets is dramatically smaller in all the presented settings, showing that the use of imperfect recall abstractions can lead to significant space savings. Additionally, the results suggest that the relative size will further decrease with the increase in the size of the original unabstracted games.

Finally, we provide results showing the actual bounds on the maxmin value computed during the run of DOIRABnB and the quality of the resulting strategies. In the first column of Figure 4.8 we present more detailed results of DOIRABnB in OshiZumo with 11 coins. The plots depict the bounds on the maxmin value in every iteration of the DOIRABnB algorithm, each row for an instance with different number of remembered moves. As we can see DOIRABnB in all the abstractions converges to a strategy with expected value against the minimizing opponent equal to 0. Since the maxmin value of the original game is also 0 and all the assumptions in Corollary 4.4.1 are satisfied, it follows that the maxmin strategy obtained by solving the abstraction is the maxmin strategy of the original game. In the second column of Figure 4.8 we show similar results for Search game with 7 moves for every player. The DOIRABnB in all the abstractions converges to a strategy with expected value against the minimizing opponent equal to $-\frac{1}{3}$ which is

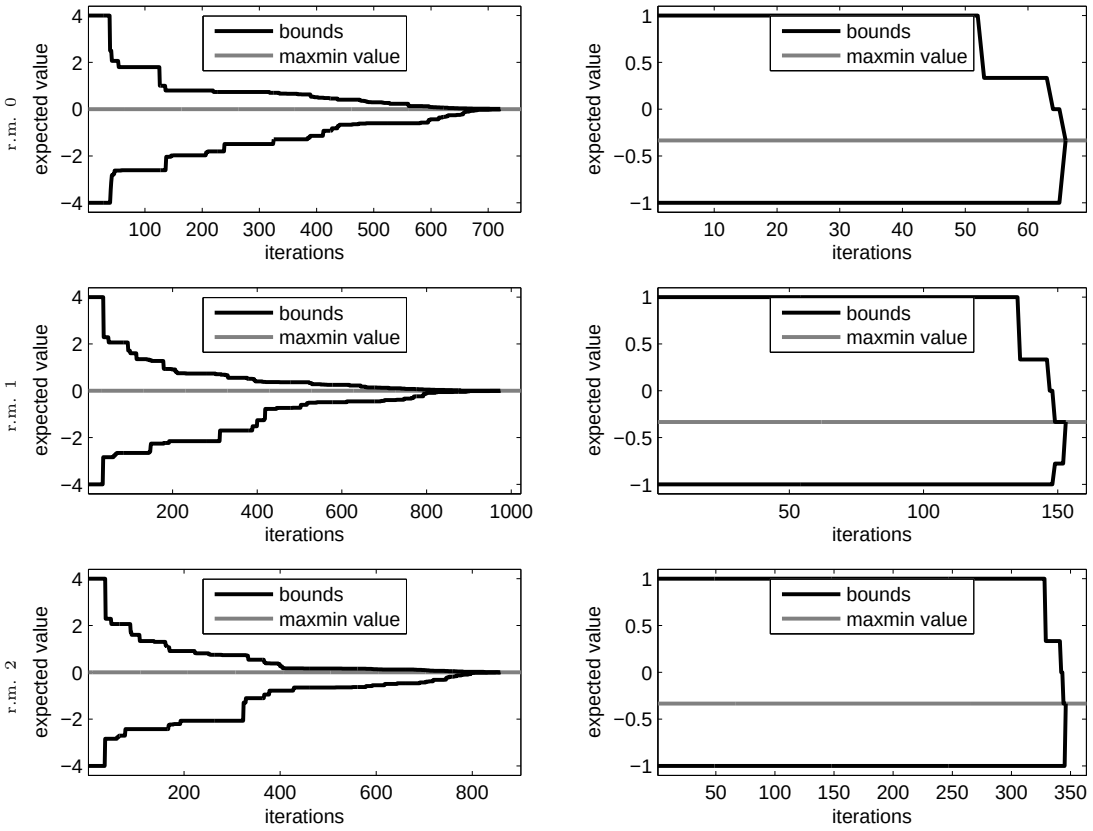

Figure 4.8: The first column shows the convergence of the lower and upper bound in DOIRABnB on OshiZumo with 11 coins as a function of iterations, each row for a different number of remembered moves. Second column shows the convergence of the lower and upper bound in DOIRABnB on Search game with depth 7 as a function of iterations, each row for a different number of remembered moves.

again the maxmin value of the original perfect recall game. Since all the assumptions in Corollary 4.4.1 are satisfied, it follows that the maxmin strategy obtained by solving the abstraction is the maxmin strategy of the original game.

Chapter 5

Constructing Imperfect Recall Abstractions to Solve Large Extensive-Form Games

In this Chapter, we present two domain-independent algorithms which start from an arbitrary imperfect recall abstraction of a given game and iteratively solve and refine this abstraction until provable convergence to the desired approximation of the Nash equilibrium of the original game. In Section 5.1 we provide the comparison of the approaches presented in this chapter to the state-of-the-art. In Section 5.2.1 we provide the notation for the abstractions being used during the run of the algorithms. As a part of this section, we present a domain-independent algorithm which creates a coarse imperfect recall abstraction of the given game if no initial abstraction is given. In Section 5.2.2 we introduce the Fictitious Play for Imperfect Recall Abstractions (FPIRA). In Section 5.2.3 we introduce the CFR+ for Imperfect Recall Abstractions (CFR+IRA). Finally in Section 5.3 we compare the memory efficiency and runtime of FPIRA, CFR+IRA and DOEFG on a set of diverse benchmark domains.

5.1 Comparison to the Current State-of-the-Art

There are several advantages of our algorithms compared to the state-of-the-art in solving EFGs using abstractions described in Section 3.5.

(1) Both algorithms are domain independent, while the majority of the work mentioned in Section 3.5 works specifically for poker.

2) Our algorithms use imperfect recall abstractions. Hence the possible memory savings can be exponentially larger than in the case of approaches restricted to perfect recall.

(3) Our algorithms can be initialized by an arbitrary imperfect recall abstraction since the choice of the initial abstraction does not affect the convergence guarantees of the algorithms. Hence, e.g., in poker, we can use the existing state-of-the-art abstractions used by the top poker bots. Even though these abstractions have no guarantee that they allow solving the original poker to optimality, our algorithms will further refine these ab-

stractions when necessary and provide the desired approximation of the Nash equilibrium in the original game. If there is no suitable abstraction available for the solved game, the algorithms can start with a simple coarse imperfect recall abstraction (we provide a domain independent algorithm for constructing such abstraction) and again update the abstraction until it allows approximation of the Nash equilibrium of the original game to the desired precision.

(4) The strategy from the final abstracted game can be directly applied to the original game without the need to use any translation techniques.

(5) The abstraction is built based on the intermediate results of the algorithm used to solve it. Hence the abstraction is expected to be specifically tailored to the needs of the given algorithm and therefore small.

The approach bearing the highest resemblance to FPIRA and CFR+IRA is the algorithm for simultaneous action abstraction finding and game solving introduced in [10]. This algorithm builds a coarse action abstraction using only a small subset of the actions of the original game. It then iteratively solves this abstraction using CFR and adds actions to the abstraction until convergence to the Nash equilibrium. The algorithm is domain independent with guaranteed convergence to the NE. However, its empirical success is caused by the specific structure of poker where all the actions of players are public. The algorithm iteratively adds parts of the original game denoted as imperfect information subgames, which tend to be small in poker. However, when not restricted to poker, the imperfect information subgame can have the size comparable to the size of the original game. To show that CFR+IRA and FPIRA do not have such drawbacks, in Section 5.3 we empirically demonstrate that both algorithms perform well on a set of domains with a diverse structure.

Additional conceptually similar approach is the DOEFG ([7], see Section 3.4.2) since it also creates a smaller version of the original game and repeatedly refines it until the desired approximation of the Nash equilibrium of the original game is found. Our algorithms, however, use imperfect recall information abstractions during the computation, while DOEFG uses a restricted perfect recall game, where the players are allowed to play only a subset of their actions. Hence, the algorithms introduced in this article exploit a completely different type of sparseness than DOEFG. We provide an experimental comparison of the memory and runtime required by FPIRA, CFR+IRA and DOEFG on a diverse set of domains in Section 5.3.

5.2 Algorithms for Constructing and Solving Imperfect Recall Abstractions

In this section, we present the main algorithmic results of this chapter. We first discuss the initial imperfect recall abstraction of a given two-player zero-sum EFG G which forms a starting point of the algorithms. Note that in this section we focus on the scenario where no initial abstraction is given, and the algorithms need to automatically build the initial coarse imperfect recall abstraction. We follow with the description of the two algorithms which iteratively solve and refine this abstraction until they reach the desired

approximation of the Nash equilibrium of the original unabstracted game G. In Section 5.2.2 we present the FP based approach denoted as Fictitious Play for Imperfect Recall Abstractions (FPIRA). In Section 5.2.3 we show the approach using a modification of CFR+ to iteratively solve and refine this abstraction. We denote the algorithm CFR+ for Imperfect Recall Abstractions (CFR+IRA). We provide proofs of convergence of both algorithms to the NE of the original unabstracted game and the discussion of memory requirements and runtime performance of both algorithms.

5.2.1 Abstraction

As discussed before, the algorithms presented in this section can start from an arbitrary initial imperfect recall abstraction. In Section 5.2.1.1, we demonstrate how to create a coarse imperfect recall abstraction which serves as a starting point of the algorithms if no initial abstraction is given.

We refer to the initial abstraction of the solved two-player zero-sum EFG G as $G^1 = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, u, \mathcal{I}^1, \mathcal{A}^1)$. In every iteration t, the algorithms operate with possibly more refined abstraction with respect to G^1, denoted as $G^t = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, u, \mathcal{I}^t, \mathcal{A}^t)$.

5.2.1.1 Initial Abstraction

Given a game G, the initial abstraction G^1 is created in the following way: Each $I \in \mathcal{I}_i^1$ is formed as the largest set of information sets $\mathcal{I}_I$ of G, so that $\forall I', I'' \in \mathcal{I}_I \; |seq_i(I')| = |seq_i(I'')| \wedge |\mathcal{A}(I')| = |\mathcal{A}(I'')|$. Furthermore, $\bigcup_{I \in \mathcal{I}^1} \mathcal{I}_I = \mathcal{I}$. Informally, for each $i \in \mathcal{N}$, the algorithm groups together information sets of i with the same length of the sequence of i leading there and with the same number of actions available. Additionally, when creating some abstracted information set I by grouping all information sets in $\mathcal{I}_I$, we need to specify the mapping $\Xi_1^{-1}(a)$ for all $a \in \mathcal{A}^1$. When creating the initial abstraction, the algorithm simply uses the order of actions given by the domain description to create $\mathcal{A}^1$ (i.e., the first action available in each $I' \in \mathcal{I}_I$ is mapped to the first action in $\mathcal{A}^1(I)$, etc.).

5.2.2 Fictitious Play for Imperfect Recall Abstractions

Let us now describe the Fictitious Play for Imperfect Recall Abstractions (FPIRA). We first give a high-level idea behind FPIRA. Next, we provide the pseudocode with the description of all steps and prove its convergence in two-player zero-sum EFGs. Finally, we discuss the memory requirements and runtime of FPIRA.

Given a perfect recall game G, FPIRA creates a coarse imperfect recall abstraction G^1 of G as described in Section 5.2.1.1. The algorithm then follows the FP procedure. It keeps track of average strategies of both players in the information set structure of the abstraction and updates the strategies in every iteration based on the best responses to the average strategies. Note that the best responses are computed directly in G (see Section 5.3 for empirical evidence that these best responses are small). To ensure the convergence to Nash equilibrium of G, FPIRA refines the information set structure of the abstraction in every iteration to make sure that the strategy update does not lead to more

exploitable average strategies in the following iterations compared to the strategy update made directly in G.

Algorithm 3: FPIRA algorithm

input : G, ε
output : $\bar{b}_i^T$, $\bar{b}_{-i}^T$, G^T

1 $G^1 \leftarrow$ `InitAbstraction`(G)
2 $\bar{b}_1^0 \leftarrow$ `PureStrat`(G^1), $\bar{b}_2^0 \leftarrow$ `PureStrat`(G^1)
3 $t \leftarrow 1$
4 **while** $u_1(\texttt{BR}(G, \bar{b}_2^{t-1}), \bar{b}_2^{t-1}) - u_1(\bar{b}_1^{t-1}, \texttt{BR}(G, \bar{b}_1^{t-1})) > \varepsilon$ **do**
5 $\quad i \leftarrow$ `ActingPlayer`(t)
6 $\quad b_i^t \leftarrow \texttt{BR}(G, \bar{b}_{-i}^{t-1})$
7 $\quad G^t \leftarrow$ `RefineForBR`$(G^t, b_i^t, \bar{b}_i^t)$
8 $\quad \hat{b}_i^t \leftarrow$ `UpdateStrategy`$(G^t, \bar{b}_i^{t-1}, b_i^t)$
9 $\quad \widetilde{b}_i^t \leftarrow$ `UpdateStrategy`$(G, \bar{b}_i^{t-1}, b_i^t)$
10 $\quad$ **if** `ComputeDelta`$(G, \hat{b}_i^t, \widetilde{b}_i^t) > 0$ **then**
11 $\quad\quad G^{t+1} \leftarrow$ `Refine`(G^t), $\bar{b}_i^t \leftarrow \widetilde{b}_i^t$
12 $\quad$ **else**
13 $\quad\quad G^{t+1} \leftarrow G^t$, $\bar{b}_i^t \leftarrow \hat{b}_i^t$
14 $\quad t \leftarrow t + 1$

In Algorithm 3 we present the pseudocode of FPIRA. FPIRA is given the original perfect recall game $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, u, \mathcal{I}, \mathcal{A})$ and a desired precision of NE approximation ε. FPIRA first creates the coarse imperfect recall abstraction G^1 of G (line 1) as described in Section 5.2.1.1. Next, it initializes the strategies of both players to an arbitrary pure strategy in G^1 (line 2). FPIRA then iterativelly solves and updates G^1 until convergence to $\varepsilon-$Nash equilibrium of G. In every iteration it updates the average strategy of one of the players and if needed the information set structure of the abstraction (the game used in iteration t is denoted as G^t). In every iteration player i first computes the best response b_i^t to $\bar{b}_{-i}^{t-1}$ in G (line 6, Section 5.2.2.1). Since b_i^t is computed in G, FPIRA first needs to make sure that the structure of information sets in G^t allows b_i^t to be played. If not, G^t is updated as described in Section 5.2.2.2, Case 1 (line 7). Next, FPIRA computes $\hat{b}_i^t$ as the strategy resulting from the update in G^t (line 8) and $\widetilde{b}_i^t$ as the strategy resulting from the update in original game G (line 9). FPIRA then checks whether the update in G^t changes the expected values of the pure strategies of $-i$ compared to the update in G using $\hat{b}_i^t$ and $\widetilde{b}_i^t$ (line 10, Section 5.2.2.2 Case 2). If yes, FPIRA refines the information set structure of G^t, creating G^{t+1} such that when updating the average strategies in G^{t+1} no error in expected values of pure strategies of $-i$ is created (Section 5.2.2.2, Case 2). FPIRA then sets $\bar{b}_i^t = \widetilde{b}_i^t$ (line 11), and continues using G^{t+1}. If there is no need to update the structure of G^t, FPIRA sets $G^{t+1} = G^t, \bar{b}_i^t = \hat{b}_i^t$ and continues with the next iteration.

Algorithm 4: Best response computation.

```
1  function BR(G, b̄_{-i})
2      b_i^{BR} ← ∅
3      BR(G.root, G, b̄_{-i}, b_i^{BR})
4      return b_i^{BR}
5  function BR(h, G, b̄_{-i}, b_i^{BR})
6      if π_i^{b̄_{-i}}(h) = 0 then
7          return 0
8      if IsGameEnd(h) then
9          return C(h) · π_i^{b̄_{-i}}(h) · u_i(h)
10     v ← GetValueFromCache(h)
11     if v ≠ null then
12         return v
13     if GetPlayerToMove(h) = i then
14         I ← GetInformationSetFor(h, I)
15         a_max ← arg max_{a∈A(I)} Σ_{h'∈I} BR(h' · a, G, b̄_{-i}, b_i^{BR})
16         For each h' ∈ I store v_{h'} = BR(h' · a_max, G, b̄_{-i}, b_i^{BR}) to cache
17         v ← BR(h · a_max, G, b̄_{-i}, b_i^{BR})
18         b_i^{BR} ← b_i^{BR} ∪ {I → a_max}
19         CleanUnreachable(b_i^{BR})
20     else
21         v ← Σ_{a∈A(h)} BR(h · a, G, b̄_{-i}, b_i^{BR})
22     return v
```

5.2.2.1 Best Response Computation

In Algorithm 4 we present the pseudocode for computing the best response b_i^{BR} of i against $\bar{b}_{-i}$ in G. The algorithm recursivelly traverses the parts of the game tree reachable by $\bar{b}_{-i}$ and computes the best action to be played in each $I \in \mathcal{I}_i$ encountered. More formally, when the best response computation reaches state h, where i plays, it first finds the information set I such that $h \in I$ (line 14). The algorithm then finds the action a_{max} which maximizes the sum of expected values of i, when i plays the best response to $\bar{b}_{-i}$, over all $h \in I$ (line 15). The expected value of playing a_{max} in h is prepared to be propagated up (line 17) and the prescription of playing a_{max} in I is stored to the best response (line 18). Notice, that to eliminate revisiting already traversed parts of the game tree on lines 15 and 17, we use cache. The cache stores the values computed by the BR for each $h \in \mathcal{H}$ visited during the computation (line 16) and provides this value if h is revisited (lines 10 to 12). Finally, since we are searching for a pure best response b_i^{BR}, there is no need to store the behavior in the parts of the tree unreachable when playing b_i^{BR}. Hence to reduce the memory needed to store b_i^{BR}, we delete the prescription in all $I \in \mathcal{I}_i$ which cannot be reached due to player i playing a_{max} (line 19).

In case of nodes of player $-i$ and chance, the algorithm simply propagates up the values of successors (line 21).

Prunning. The implementation of the best response used in FPIRA incorporates pruning based on the lower bound and upper bound in each node h of the game tree. The lower bound represents the lowest value that needs to be achieved in h to make sure that there is a chance that searching the subtree of h influences the resulting best response and the upper bound represents the estimate of how much can the best responding player gain by visiting h. For a more detailed discussion of the pruning see [7], Section 4.2.

5.2.2.2 Updating G^t

There are two reasons for splitting some $I \in \widetilde{\mathcal{I}}_i^t$ in iteration t where player i computes the best response. First, the best response computed in G prescribes more than one action in I. Second, I causes the expected value of some pure strategy of $-i$ to be different against the average strategy of i computed in G^t compared to the expected value against the average strategy computed in G. This can happen since I is an abstracted information set, and hence updating the average strategy in I in G^t changes the behavior in multiple information sets in G (see Example 5.2.1 for more details).

Case 1: Here we discuss the abstraction update which guarantees that the best response b_i^t computed in G is applicable in the resulting abstraction. This abstraction update first detects every $I \in \widetilde{\mathcal{I}}_i^t$ where the b_i^t prescribes more than one action. It then splits each such I by grouping the information sets in $\phi_t^{-1}(I)$ to the largest possible subsets where b_i^t prescribes the same action. One additional information set is created containing all $I' \in \phi_t^{-1}(I)$ which are not reachable when playing b_i^t.

In Algorithm 5 we show the pseudocode for this abstraction update. The algorithm iterates over all abstracted information sets $I \in \widetilde{\mathcal{I}}_i^t$ that can be visited when playing b_i^t (lines 2 and 3). Each such I is first divided to a sets of information sets $\hat{\mathcal{I}}$ and $\mathcal{I}''$. Let $\hat{I}_a \subseteq \Phi_t^{-1}(I)$ be a union of all $I' \in \Phi_t^{-1}(I)$ where $b_i^t(I', a') = 1$ for $\Xi_t(a') = a$ (lines 5 to 11). $\hat{\mathcal{I}}$ is a union of $\hat{I}_a$ for all $a \in \mathcal{A}^t(I)$ (lines 14 to 17). $\mathcal{I}''$ contains all $I' \in \Phi_t^{-1}(I)$ for which b_i^t does not prescribe any action (line 13). If there is more than one element in $\hat{\mathcal{I}}$ (line 18), the average strategy before the strategy update in all information sets which are about to be created is set to the strategy previously played in I (lines 19 to 21). Next, the algorithm removes I from G^t (line 22). Finally, it creates new informations set for each $\hat{I}_a \in \hat{\mathcal{I}}$, that contains all $I' \in \hat{I}_a$ and an information set containing all $I' \in \mathcal{I}''$ (lines 23 to 26).

Case 2: Now we turn to the abstraction update guaranteeing that the expected values of all pure strategies of $-i$ against the average strategy of i computed in G^t are equal to the expected values against the average strategy computed in G. As a part of this abstraction update, FPIRA constructs the average strategy $\hat{b}_i^t$ resulting from the update in G^t and $\widetilde{b}_i^t$ resulting from the update in G. It then checks whether there exists a pure strategy of player $-i$ which has different expected value against $\hat{b}_i^t$ and $\widetilde{b}_i^t$. If there exists such pure strategy, the abstraction is updated to guarantee that the average strategy update in the abstraction results in $\widetilde{b}_i^t$, and hence that there is no difference in the expected values of pure strategies of $-i$.

Algorithm 5: Abstraction update to accomodate best response.

```
1  function RefineForBR(G^t, b_i^t, b̄_i^t)
2      for I ∈ Ĩ_i^t do
3          if π_i^{b_i^t}(I) > 0 then
4              I'' ← ∅
5              for a ∈ A^t(I) do
6                  Î_a ← ∅
7              for I' ∈ φ_t^{-1}(I) do
8                  if ∃a ∈ A(I') such that b_i^t(I, a) = 1 then
9                      a' ← a ∈ A(I') such that b_i^t(I, a) = 1
10                     a ← Ξ_t(a')
11                     Î_a ← Î_a ∪ I'
12                 else
13                     I'' ← I'' ∪ I'
14             Î ← ∅
15             for a ∈ A^t(I) do
16                 if Î_a ≠ ∅ then
17                     Î ← Î ∪ {Î_a}
18             if |Î| > 1 then
19                 for I' ∈ Î ∪ I'' do
20                     b̄_i^t(I') ← b̄_i^t(I)
21                 b̄_i^t ← b̄_i^t \ b̄_i^t(I)
22                 I^t ← I^t \ I
23                 for Î_a ∈ Î do
24                     I^t ← I^t ∪ CreateNewIS(Î_a)
25                 if I'' ≠ ∅ then
26                     I^t ← I^t ∪ CreateNewIS(I'')
27     return G^t
```

More formally, the algorithm first constructs the average behavioral strategy $\hat{b}_i^t$ in G^t (line 8 in Algorithm 3). This is done according to Lemma 3.4.1 from $\bar{b}_i^{t-1}$ with weight $\frac{t_i-1}{t_i}$ and b_i^t with weight $\frac{1}{t_i}$, where t_i is the number of updates performed by i so far, plus 1 for the initial strategy (b_i^t is used with mappings Φ_t and Ξ_t). Next, FPIRA constructs $\widetilde{b}_i^t$ (line 9) in the same way in the information set structure of G ($\bar{b}_i^{t-1}$ is used with mappings Φ_t^{-1} and Ξ_t^{-1}). FPIRA then computes

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

as described below (line 10 in Algorithm 3). If $\Delta_i^t = 0$, all the pure strategies of $-i$ have the same expected value against both $\hat{b}_i^t$ and $\widetilde{b}_i^t$. In this case, FPIRA sets $G^{t+1} = G^t$, $\bar{b}_i^t = \hat{b}_i^t$ (line 13). If $\Delta_i^t > 0$, the expected value of some pure strategy of $-i$ changed when updating the strategy in G^t, compared to the expected value it would get against the strategy updated in G. FPIRA then creates G^{t+1} according to Algorithm 6 in the

Algorithm 6: Abstraction update guaranteeing that the expected values of all pure strategies of $-i$ are the same against the strategy resulting from update in G and G^t.

```
1  function Refine(G^t, b_i^t, b̄_i^t)
2      for I ∈ Ĩ_i^t do
3          if π_i^{b_i^t}(I) > 0 then
4              Î ← {I' ∈ φ_t^{-1}(I) | π_i^{b_i^t}(I') > 0}
5              I'' ← φ_t^{-1}(I) \ Î
6              I^t ← I^t \ I
7              for I' ∈ Î ∪ I'' do
8                  b̄_i^t(I') ← b̄_i^t(I)
9              b̄_i^t ← b̄_i^t \ b̄_i^t(I)
10             if I'' ≠ ∅ then
11                 I^t ← I^t ∪ CreateNewIS(I'')
12             for I' ∈ Î do
13                 I^t ← I^t ∪ CreateNewIS(I')
14     return G^t
```

following way. Every abstracted information set $I \in \widetilde{\mathcal{I}}_i^t$ which is visited when playing b_i^t (lines 2 and 3) is first divided to sets of information sets $\hat{\mathcal{I}} \subseteq \Phi_t^{-1}(I)$ and $\mathcal{I}''$. $\hat{\mathcal{I}}$ contains all the $I' \in \Phi_t^{-1}(I)$ for which $\pi_i^{b_i^t}(I') > 0$ (line 4), $\mathcal{I}''$ contains the rest of $I' \in \Phi_t^{-1}(I)$ (line 5). The average strategy before the strategy update in all information sets which are about to be created is set to the strategy previously played in I (lines 7 to 9). The algorithm then creates new information set for each $\hat{I} \in \hat{\mathcal{I}}$, and an information set containing all $I' \in \mathcal{I}''$ (lines 10 to 13). The strategy resulting from update in G is a valid strategy in G^{t+1} after such update, hence $\bar{b}_i^t = \widetilde{b}_i^t$. Notice that by setting $\bar{b}_i^t = \widetilde{b}_i^t$, we made sure that $\Delta_i^t = 0$ since the update is now equal to the update that would occur in G. This, as we will show in Section 5.2.2.3, is sufficient to guarantee the convergence of $\bar{b}_i^t, \bar{b}_{-i}^t$ to Nash equilibrium of G.

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

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

Let

$$u'_{-i}(z) = \mathcal{C}(z) \left[\pi_i^{\widetilde{b}_i^t}(z) - \pi_i^{\hat{b}_i^t}(z) \right] u_{-i}(z), \quad \forall z \in \mathcal{Z} \tag{5.1}$$

$$u''_{-i}(z) = -\mathcal{C}(z) \left[\pi_i^{\widetilde{b}_i^t}(z) - \pi_i^{\hat{b}_i^t}(z) \right] u_{-i}(z), \quad \forall z \in \mathcal{Z}. \tag{5.2}$$

Δ_i^t can be computed in $\mathcal{O}(|\mathcal{Z}|)$ by ComputeDelta depicted in Algorithm 7. The computation consists of two calls of function ComputePartialDelta similar to the computation of

Algorithm 7: Δ_i^t computation.

```
1  function ComputeDelta(G, b̂_i^t, b̃_i^t)
2      Let u'_{-i} be the function from eq. (5.1)
3      Let u''_{-i} be the function from eq. (5.2)
4      return max(ComputePartialDelta(G.root, u'_{-i}, b̂_i^t, b̃_i^t),
           ComputePartialDelta(G.root, u''_{-i}, b̂_i^t, b̃_i^t))
5  function ComputePartialDelta(h, u'_{-i}, b̂_i^t, b̃_i^t)
6      if π_i^{b̂_i^t}(h) = 0 ∧ π_i^{b̃_i^t}(h) = 0 then
7          return 0
8      if IsGameEnd(h) then
9          return u'_{-i}(h)
10     v ← GetValueFromCache(h)
11     if v ≠ null then
12         return v
13     if GetPlayerToMove(h) = −i then
14         I ← GetInformationSetFor(h, I)
15         a_max ← arg max_{a∈A(I)} Σ_{h'∈I} ComputePartialDelta(h' · a, u'_{-i}, b̂_i^t, b̃_i^t)
16         For each h' ∈ I store v_{h'} = BR(h' · a_max, G, b̄_{-i}, b_i^{BR}) to cache
17         v ← ComputePartialDelta(h · a_max, u'_{-i}, b̂_i^t, b̃_i^t)
18     else
19         v ← Σ_{a∈A(h)} ComputePartialDelta(h · a, u'_{-i}, b̂_i^t, b̃_i^t)
20     return v
```

the best response described in Section 5.2.2.1. The only difference is the use of functions shown in eqs. (5.1) and (5.2) when evaluating the leaves (line 9). ComputePartialDelta using u'_{-i} searches for the largest positive difference in the utility of pure strategies of $-i$, while ComputePartialDelta using u''_{-i} searches for the highest negative difference. The two calls are neccessary, since ComputePartialDelta using u'_{-i} cannot reliably detect negative differences in the utility, since it will always prefer choosing pure strategy with no difference in the utility over the pure strategy with the negative difference. Notice, that similarly to the best response computation, we use cache to eliminate redundant tree traversals caused by line 15.

Example 5.2.1. *Let us demonstrate several iterations of* FPIRA *algorithm. Consider the EFG from Figure 5.1(a) as G and the imperfect recall game from Figure 5.1 (b) as G^1. The function Ξ_1 is $\Xi_1(t) = \Xi_1(v) = c, \Xi_1(u) = \Xi_1(w) = d$, identity otherwise. Note that when we apply strategies from G to G^t and vice versa in iteration t, we assume that it is done with respect to Ξ_t and Ξ_t^{-1}. Lets assume that* FPIRA *first initializes the strategies to $\bar{b}_1^0(b) = \bar{b}_1^0(d) = 1, \bar{b}_2^0(e) = 1$.*

Iteration 1: *The player 1 starts in iteration 1.* FPIRA *computes $b_1^1 \in BR(\bar{b}_2^0)$ in G, resulting in $b_1^1(b) = b_1^1(v) = 1$. Next,* FPIRA *checks whether b_1^1 is playable in G^1. Since there is no information set in G^1 for which b_1^1 assigns more than one action, we do not need to update G^1 in any way. We follow by computing $\hat{b}_1^1$ and $\widetilde{b}_1^1$ according to Lemma*

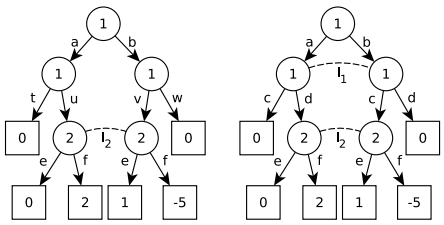

Figure 5.1: (a) G for demonstration of FPIRA iterations (b) G^1 for demonstration of FPIRA.

3.4.1 with $\lambda_1 = \lambda_2 = 0.5$. *In this case* $\hat{b}_1^1(b) = \widetilde{b}_1^1(b) = 1, \hat{b}_1^1(c) = \widetilde{b}_1^1(v) = 0.5$. *Since* $\hat{b}_1^1$ *and* $\widetilde{b}_1^1$ *are equal, w.r.t.* Ξ_1, *we know that* $\Delta_i = 0$. *Hence we let* $G^2 = G^1$, $\bar{b}_1^1 = \hat{b}_1^1$ *and* $\Xi_2 = \Xi_1$.

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

Iteration 3: *The best response in this iteration is* $b_1^3(a) = b_1^3(u) = 1$, *which is again playable in* G^3, *hence we do not need to update* G^3 *at this point.* FPIRA *computes* $\hat{b}_1^3$ *resulting in* $\hat{b}_1^3(a) = \frac{1}{3}, \hat{b}_1^3(d) = \frac{2}{3}$, $\widetilde{b}_1^3$ *is, on the other hand,* $\widetilde{b}_1^3(a) = \frac{1}{3}$, $\widetilde{b}_1^3(u) = 1, \widetilde{b}_1^3(w) = 0.5$ *(both according to Lemma 3.4.1 with* $\lambda_1 = \frac{2}{3}, \lambda_2 = \frac{1}{3}$*). In this case,* $\Delta_1^3 = \frac{1}{3}$ *since by playing* f *player 2 gets* $\frac{2}{3}$ *against* $\hat{b}_1^3$ *compared to* 1 *against* $\widetilde{b}_1^3$. *Hence, the algorithm splits all imperfect recall information sets reachable when playing* b_1^3, *in this case* I_1, *as described in Section 5.2.2.2, Case 2, resulting in* G. *Therefore,* $G^4 = G$, $\bar{b}_1^3 = \widetilde{b}_1^3$ *and* Ξ_4 *is set to identity.*

5.2.2.3 Theoretical Properties

Here, we show that the convergence guarantees of FP in two-player zero-sum perfect recall game G [27] directly apply to FPIRA solving G.

Theorem 5.2.1. *Let* G *be a perfect recall two-player zero-sum EFG. Assume that initial strategies* $\bar{b}_1^0, \bar{b}_2^0$ *in* FPIRA *and initial strategies* $\bar{b'}_1^0, \bar{b'}_2^0$ *in the FP are realization equivalent, additionally assume that the same tie breaking rules are used when more than one best response is available in any iteration. The exploitability of* $\bar{b}_i^t$ *computed by* FPIRA *applied to* G *is exactly equal to the exploitability of* $\bar{b'}_i^t$, *computed by FP applied to* G *in all iterations* t *and for all* $i \in \mathcal{N}$.

Proof. The proof is done by induction. If

$$\forall b_{-i} \in \mathcal{B}_{-i}^p : u_{-i}(\bar{b}_i^t, b_{-i}) = u_{-i}(\bar{b'}_i^t, b_{-i}), \tag{5.3}$$

$$\forall b_i \in \mathcal{B}_i^p : u_i(b_i, \bar{b}_{-i}^t) = u_i(b_i, \bar{b'}_{-i}^t), \tag{5.4}$$

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

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

The initial step trivially holds from the assumption that initial strategies in FPIRA and initial strategies and in FP are realization equivalent. Now let us show that the induction step holds. Let b_i^t be the best response chosen in iteration t in FPIRA and $b'_i{}^t$ be the best response chosen in t in FP. From (5.4) and the use of the same tie breaking rule we know that $b_i^t = b'_i{}^t$. From Lemma 3.4.1 we know that

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

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

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

From (5.3) and from the equality $b_i^t = b'_i{}^t$ follows that

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

and therefore also

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

□

5.2.2.4 Storing the Information Set Map

In this section we discuss the memory requirements for storing the mapping of information sets $\mathcal{I}$ of G to $\mathcal{I}^t$ of G^t in FPIRA.

Initial abstraction. As decribed in Section 5.2.1.1, the mapping between any $I \in \mathcal{I}_i$ and its abstracted information set in G^1 is perfectly defined by $|seq_i(I)|$ and $|\mathcal{A}(I)|$. Hence the mapping can always be determined for any given $I \in \mathcal{I}$ without using any additional memory.

Case 1 Update. When updating the abstraction resulting in G^t according to Case 1 in Section 5.2.2.2, FPIRA can split $I \in \widetilde{\mathcal{I}}_i^{t-1}$ to multiple abstracted information sets. FPIRA is then forced to store the mapping for each newly created abstracted informations set I',

for each $I'' \in \Phi_t^{-1}(I')$. This is neccessary since $|seq_i(I'')|$ and $|\mathcal{A}(I'')|$ is no longer unique identifier of I'. In our implementation, we use a unique integer j to represent the new mapping for I', i.e., the algorithm stores j for each $I'' \in \Phi_t^{-1}(I')$. We use j corresponding to the number of newly created information sets during the run of FPIRA. Finally, let $\mathcal{I}_n^m$ be all the information sets in $\widetilde{\mathcal{I}}_i^t$ such that $\forall I \in \mathcal{I}_n^m | seq_i(I)| = m \wedge |\mathcal{A}(I)| = n$. We can always identify the mapping to one of $\mathcal{I}_n^m$ without using any memory by the sequence length and number of actions, as long as the rest of $I \in \mathcal{I}_n^m$ uses mapping with the unique integer. We keep track of $I \in \mathcal{I}_n^m$ with largest $|\Phi_t^{-1}(I)|$ in each $\mathcal{I}_n^m$ and use the sequence length and number of actions to represent the mapping for all $\Phi_t^{-1}(I')$ to minimize the memory requirements. In Section 5.3 we empirically demonstrate that the memory required to store the mapping is small. Notice that in all $I' \in \mathcal{I}^t \setminus \widetilde{\mathcal{I}}^t$, the mapping is defined by the domain description of G and hence no memory is required.

Case 2 Update. When updating the abstraction resulting in G^t according to Case 2 in Section 5.2.2.2, there is no additional memory required to store the updated mapping compared to the mapping used in G^{t-1}. This holds since this abstraction update only removes information set from G from the abstracted information sets in G^{t-1}. And so we use the same mapping as in G^{t-1} for abstracted information sets in G^t, and the information set structure provided by the domain description in the rest.

5.2.2.5 Memory and Time Efficiency

FPIRA needs to store the average behavioral strategy for every action in every information set of the solved game, hence storing the average strategy in G^t instead of G results in significant memory savings directly proportional to the decrease of information set count. When the algorithm computes $\widetilde{b}_i^t$, it can temporarily refine the information set structure of G^t only in the parts of the tree that can be visited when playing the pure best response b_i^t according to $\mathcal{I}_i$ to avoid representing and storing G. Additional memory used to store the current abstraction mapping is discussed in Section 5.2.2.4.

When computing the best response (see Section 5.2.2.1), the algorithm needs to store the best response strategy and the cache that is used to eliminate additional tree traversals. FPIRA stores the behavior only in the parts of the game reachable due to actions of i in b_i^t (line 19 in Algorithm 4) and due to $\bar{b}_{-i}^t$ (lines 6 and 7 in Algorithm 4). For this reason and since i plays only 1 action in his information sets in b_i^t, there are typically large parts of the game tree where b_i^t does not prescribe any behavior. The cache (used also in the computation of Δ_i^t) stores one number for each state visited during the computation. We show the size of the cache in Section 5.3. If necessary, the memory requirements of the cache can be reduced by limiting its size and hence balancing the memory required and the additional tree traversals performed. Additionally, efficient domain-specific implementations of best response (e.g., on poker [30]) can be employed to further reduce the memory and time requirements.

We empirically demonstrate the size of all the data structures stored during the run of FPIRA in Section 5.3.

The iteration of FPIRA takes approximately three times the time needed to perform one iteration of FP in G, as it now consists of the standard best response computation

in G, two modified best response computations to obtain Δ_i^t and two updates of average behavioral strategies (which are faster than the update in G since the average strategy is smaller).

5.2.3 CFR+ for Imperfect Recall Abstractions

In this section, we describe the CFR+ for Imperfect Recall Abstractions (CFR+IRA). We first provide a high-level idea of CFR+IRA, followed by detailed explanation of all its parts with pseudocodes and proof of its convergence to NE in two-player zero-sum EFGs. Finally, we discuss the memory requirements and runtime of CFR+IRA.

Given a two-player zero-sum perfect recall EFG G, CFR+IRA first creates a coarse imperfect recall abstraction of G as described in Section 5.2.1.1. The algorithm then iteratively solves the abstraction using CFR+. All regrets and average strategies computed as a part of CFR+ are stored in the information set structure of the abstraction. To ensure the convergence to the Nash equilibrium of G, CFR+IRA updates the structure of the abstraction in every iteration based on the differences between the results obtained by CFR+ in the abstraction and the expected behavior of CFR+ in G.

Algorithm 8: CFR+ for Imperfect Recall Abstractions

input : G, k_h, k_b, delay, ε
output: $(\bar{b}_1, \bar{b}_2)$ – ε-Nash equilibrium of G

$G^1 \leftarrow$ `InitAbstraction`(G)
$r \leftarrow$ `InitRegrets`$(\mathcal{I}^1)$
$\bar{b}_1 \leftarrow$ `UniformStrategy`$(\mathcal{I})$, $\bar{b}_2 \leftarrow$ `UniformStrategy`$(\mathcal{I})$
$t \leftarrow 1$, $t_{next} \leftarrow 1$, $t_{last} \leftarrow 0$, $j \leftarrow 0$
$r_b \leftarrow \emptyset$
while $u_1(\texttt{BR}(G, \bar{b}_2), \bar{b}_2) - u_1(\bar{b}_1, \texttt{BR}(G, \bar{b}_1)) > \varepsilon$ **do**
  $\hat{\mathcal{I}}_h \leftarrow$ `SampleInformationSets`$(\widetilde{\mathcal{I}}_i^t, k_h)$
  $r_h \leftarrow$ `InitRegrets`$(\hat{\mathcal{I}}_h)$
  if $t = t_{next}$ **then**
    $\hat{\mathcal{I}}_b \leftarrow$ `SampleInformationSets`$(\widetilde{\mathcal{I}}^t, k_b)$
    $r_b \leftarrow$ `InitRegrets`$(\hat{\mathcal{I}}_b)$
    $t_{last} \leftarrow t$
    $t_{next} \leftarrow t + 2^j$
    $j \leftarrow j + 1$
  `ComputeRegrets`$(\mathcal{I}^t, G^t.root,$ `GetPlayer`$(t), 1, 1, r, r_h, r_b, \hat{\mathcal{I}}_h, \hat{\mathcal{I}}_b)$
  `RemoveNegativeRegrets`(r)
  if $t > delay$ **then**
    `UpdateAverageStrategy`(r, t)
  $G^{t+1} \leftarrow$ `UpdateAbstractionForHeuristic`$(G^t, \hat{\mathcal{I}}_h, r_h, r, \bar{b}_1, \bar{b}_2, t)$
  delete r_h
  if $t \neq t_{last}$ **then**
    $G^{t+1} \leftarrow$ `UpdateAbstractionForBound`$(G^{t+1}, \hat{\mathcal{I}}_b, r, r_b, \bar{b}_1, \bar{b}_2, t, t_{last})$
  $t \leftarrow t + 1$

In Algorithm 8 we provide the pseudocode of the CFR+IRA. CFR+IRA is given the original perfect recall game G, the desired precision of approximation of the NE ε and the limits k_h and k_b on the memory that can be used to update the abstraction. Finally it is given the *delay* which represents the number of initial iterations for which the algorithm does not update the average strategy. The algorithm starts by creating a coarse imperfect recall abstraction G^1 of the given game G (line 1, see Section 5.2.1.1). It stores the regrets r and average strategies $\bar{b}_1, \bar{b}_2$ for each information set of the current abstraction. The algorithm then simultaneously solves the abstraction and updates its structure until the $\bar{b}_1, \bar{b}_2$ form an ε-Nash equilibrium of G (line 6). The players take turn updating their strategies and regrets. In every iteration, the algorithm updates the regrets and the current strategy for the acting player i according to CFR+ update (line 15, see Section 3.4.4.3 and Algorithm 9). As a part of the CFR+ update the algorithm removes the negative regrets (line 16) and updates the average strategy (line 18). Note that we follow the CFR+ as described in Section 3.4.4.3 and so the update of the average strategy starts only after a fixed number of iterations denoted as *delay*. The average strategy is then updated according to eq. (3.14).

The algorithm continues with the update of the current abstraction G^t. There are two procedures for updating the abstraction.

First, the abstraction is updated to guarantee the convergence of the algorithm to the Nash equilibrium (see Section 5.2.3.1 for more details). As a part of this abstraction update, CFR+IRA samples a subset $\hat{\mathcal{I}}_b$ of information sets of G (line 10). It then checks the immediate regret in all $I \in \hat{\mathcal{I}}_b$ for a given number of iterations before again resampling $\hat{\mathcal{I}}_b$. During these iterations, it updates the abstraction so that any $I \in \hat{\mathcal{I}}_b$, where the immediate regret decreases slower than a given function, is removed from its abstracted information set.

Second, the abstraction is updated using a heuristic update which significantly improves the empirical convergence of the algorithm (see Section 5.2.3.2 for more details). The heuristic update samples a subset $\hat{\mathcal{I}}_h$ of information sets of i in G in every iteration of CFR+IRA (line 7). It then keeps track of regrets in all $I \in \hat{\mathcal{I}}_h$ in this iteration. Finally, it uses these regrets to update the abstraction so that only information sets with similar regrets remain grouped.

5.2.3.1 Regret Bound Update

In this section we present more detailed description of the update of the abstraction based on the regret bound.

Let $\mathcal{T}^T = (T_1, ..., T_n)$ be a sequence of iterations, where every $T_{j+1} - T_j = 2^j$ for all $j \in \{1, ..., n-1\}$ (the elements of $\mathcal{T}$ are computed on line 13 in Algorithm 8). As a part of the abstraction update, the algorithm samples a subset $\hat{\mathcal{I}}_b \subseteq \mathcal{I}$ of information sets of G in predetermined iterations specified by elements of $\mathcal{T}$. The subset $\hat{\mathcal{I}}_b$ is sampled on line 10 in Algorithm 8 according to Algorithm 10. The sampling of $\hat{\mathcal{I}}_b$ is done so that $\forall I \in \hat{\mathcal{I}}_b \ \Phi_t(I) \in \widetilde{\mathcal{I}}^t$ and

Algorithm 9: Regret update

```
1  function ComputeRegrets(I, h, i, π1, π2, r, r_h, r_b, Î_h, Î_b)
2      if IsTerminal(h) then
3          return u_i(h)
4      if GetPlayerToMove(h) = c then
5          if i = 1 then
6              return Σ_{a∈A(h)} b_c(a) · ComputeRegrets(I, h·a, i, π1, b_c(a)·π2, r, r_h, r_b, Î_h,
                   Î_b)
7          return Σ_{a∈A(h)} b_c(a) · ComputeRegrets(I, h·a, i, b_c(a)·π1, π2, r, r_h, r_b, Î_h, Î_b)
8      I ← GetInformationSetFor(h, I)
9      r_I ← GetRegretsFor(I, r)
10     b^t ← RegretMatching+(r_I)
11     v_{b^t} ← 0
12     v ← Zeros(|A(h)|)
13     for a ∈ A(h) do
14         if i = 1 then
15             v[a] ← ComputeRegrets(I, h·a, i, b^t(a)·π1, π2, r, r_h, r_b, Î_h, Î_b)
16         else
17             v[a] ← ComputeRegrets(I, h·a, i, π1, b^t(a)·π2, r, r_h, r_b, Î_h, Î_b)
18         v_{b^t} ← v_{b^t} + b^t(a) · v[a]
19     if GetPlayerToMove(h) = i then
20         for a ∈ A(h) do
21             r_I[a] ← r_I[a] + π_{-i} · (v[a] − v_{b^t})
22         if I ∈ Î_h then
23             r^h_I ← GetRegretsFor(I, r_h)
24             for a ∈ A(h) do
25                 r^h_I[a] ← r^h_I[a] + π_{-i} · (v[a] − v_{b^t})
26         if I ∈ Î_b then
27             r^b_I ← GetRegretsFor(I, r_b)
28             for a ∈ A(h) do
29                 r^b_I[a] ← r^b_I[a] + π_{-i} · (v[a] − v_{b^t})
30     return v_{b^t}
```

$$|\hat{\mathcal{I}}_b| = \min\left(k_b, \sum_{I \in \widetilde{\mathcal{I}}^t} |\Phi_t^{-1}(I)|\right).$$

I.e., the size of $\hat{\mathcal{I}}_b$ is limited by the parameter k_b and the actual number of information sets in $\mathcal{I}$ that are still mapped to some abstracted information set in iteration t. Additionally, sampling of information sets on line 6 in Algorithm 10 is performed so that the probability of adding any $I \in \mathcal{I}$ such that $\Phi_t(I) \in \widetilde{\mathcal{I}}^t$ to $\hat{\mathcal{I}}_b$ is equal to

$$\frac{1}{\sum_{I' \in \widetilde{\mathcal{I}}^t} |\Phi_t^{-1}(I')|}. \tag{5.5}$$

Algorithm 10: Sampling of unabstracted sets for unabstracted regret storage.

```
1  function SampleInformationSets(Ĩ, k)
2      Î ← ∅
3      while |Î| < k do
4          if AllSetsSampled(Ĩ, Î) then
5              break
6          I ← GetRandomAbstractedSet(Ĩ)
7          I'' ← Φ_t^{-1}(I)
8          if |I''| + |Î| ≤ k then
9              Î ← Î ∪ I''
10         else
11             Î ← Î ∪ random subset of I'' with size k − |Î|
12     return Î
```

Algorithm 11: Abstraction update for regret bound

```
1  function UpdateAbstractionForBound(G^t, Î_b, r, r_b, b̄_1, b̄_2, t, t_last)
2      for I ∈ Ĩ^t do
3          I'' ← Φ_t^{-1}(I)
4          for I'' ∈ Intersection(I'', Î_b) do
5              r^b_{I''} ← GetRegretsFor(I'', r_b)
6              if max_{a∈A(I'')} r^b_{I''}[a] / (t − t_last) > L^{t_last}_{I''}(t) then
7                  I_new ← CreateNewIS(I'' \ I'')
8                  G^t.I ← G^t.I \ I
9                  G^t.I ← G^t.I ∪ {CreateNewIS(I''), I_new}
10                 r_I ← GetRegretsFor(I, r)
11                 r ← r \ r_I
12                 r_{I_new} ← InitRegret(I_new)
13                 r ← r ∪ r_{I_new}
14                 r_{I''} ← InitRegret(I'')
15                 r ← r ∪ r_{I''}
16                 Î_b ← Î_b \ I''
17                 r_b ← r_b \ r^b_{I''}
18                 if GetPlayer(t) = 1 then
19                     b̄_1 ← b̄_1 \ GetStrategyFor(I, b̄_1)
20                 else
21                     b̄_2 ← b̄_2 \ GetStrategyFor(I, b̄_2)
22     return G^t
```

After sampling $\hat{\mathcal{I}}_b$ in some $T_j \in \mathcal{T}$, the algorithm keeps track of the regrets r_b accumulated in each $I \in \hat{\mathcal{I}}_b$ for $T_{j+1} - T_j$ iterations during the CFR+ update (line 26 to 29 in Algorithm 9) before again resampling the $\hat{\mathcal{I}}_b$.

Let $L_I : \{T_j, ..., T_{j+1}\} \to \mathbb{R}$ be any function for which

$$L_I^{T_j}(t) \leq \frac{\Delta_I \sqrt{|\mathcal{A}(I)|}}{\sqrt{t - T_j}}, \quad \forall t \in \{T_j + 1, ..., T_{j+1}\}.$$

The actual abstraction update in each iteration $T' \in \{T_j + 1, ...T_{j+1}\}$ is done according to Algorithm 11 in the following way. The algorithm iterates over $I \in \widetilde{\mathcal{I}}^t$ in G^t (line 2). For all $I'' \in \Phi_t^{-1}(I) \cap \hat{\mathcal{I}}_h$ the algorithm checks the immediate regret

$$R_{T_j,imm}^{T'}(I) = \frac{1}{T' - T_j} \max_{a \in A(I)} \sum_{t=T_j}^{T'} \left[v_i(b^t_{I \to a}, I) - v_i(b^t, I) \right] = \frac{\max_{a \in A(I)} r_I^b[a]}{T' - T_j}.$$

If $R_{T_j,imm}^{T'}(I) > L_I^{T_j}(T')$ for some $T' \in \{T_j + 1, ..., T_{j+1}\}$ (line 6 in Algorithm 11), the algorithm disconnects I in iteration T' from its abstracted information set in $G^{T'}$, resets its regrets to 0 and removes it from $\hat{\mathcal{I}}_b$ (lines 7 to 21 in Algorithm 11). If I is disconnected, the average strategy in I in all $T > T'$ is computed $\forall a \in \mathcal{A}(I)$ as

$$\bar{b}_i^T(I, a) = \begin{cases} \frac{1}{|\mathcal{A}(I)|}, & \text{if } T = T' + 1, \\ \frac{2 \sum_{t'=T'+1}^{T} t' \cdot \pi_i^{b_i^{t'}}(I) \cdot b_i^{t'}(I,a)}{((T-T'-1)^2 + T - T' - 1) \sum_{t'=T'+1}^{T} \pi_i^{b_i^{t'}}(I)}, & \text{otherwise.} \end{cases} \tag{5.6}$$

The average strategy update in eq. (5.6) corresponds to the average strategy update descibed in Section 3.4.4.3, i.e., it does a weighted average of the b_i^t strategies over iterations $\{T' + 1, ..., T\}$ with weight corresponding to the iteration.

5.2.3.2 Heuristic Update

In this section, we focus on the description of the heuristic update of the abstraction.

Let i be the player who's regrets are updated in iteration t. As a part of the abstraction update, the algorithm samples $\hat{\mathcal{I}}_h \subseteq \mathcal{I}_i$ of information sets of G in t. $\hat{\mathcal{I}}_h$ is sampled on line 7 in Algorithm 8 according to Algorithm 10. In this case, the sampling in Algorithm 10 is done so that $\forall I \in \hat{\mathcal{I}}_h$ $\Phi_t(I) \in \widetilde{\mathcal{I}}_i^t$ and

$$|\hat{\mathcal{I}}_h| = \min \left(k_h, \sum_{I \in \widetilde{\mathcal{I}}_i^t} |\Phi_t^{-1}(I)| \right).$$

I.e., the size of $\hat{\mathcal{I}}_h$ is limited by the parameter k_h and the actual number of information sets in $\mathcal{I}_i$ that are still mapped to some abstracted information set in iteration t. Additionally, sampling of information sets on line 6 in Algorithm 10 is performed so that the probability of adding any $I \in \mathcal{I}$ such that $\Phi_t(I) \in \widetilde{\mathcal{I}}^t$ to $\hat{\mathcal{I}}_h$ is equal to

$$\frac{1}{\sum_{I' \in \widetilde{\mathcal{I}}_i} |\Phi_t^{-1}(I')|}. \tag{5.7}$$

The algorithm keeps track of the regrets r_h in each $I \in \hat{\mathcal{I}}_h$ during the CFR+ update in iteration t (lines 22 to 25 in Algorithm 9).

Algorithm 12: Abstraction update for heuristic

1 **function** `UpdateAbstractionForHeuristic`(G^t, $\hat{\mathcal{I}}_h$, r_h, r, $\bar{b}_1$, $\bar{b}_2$, t)
2 | **for** $I \in \widetilde{\mathcal{I}}_i^t$ **do**
3 | | $\mathcal{I}'' \leftarrow \Phi_t^{-1}(I)$
4 | | **if** `Intersection`($\mathcal{I}''$, $\hat{\mathcal{I}}_h$) $\neq \emptyset$ **then**
5 | | | **for** $I'' \in$ `Intersection`($\mathcal{I}''$, $\hat{\mathcal{I}}_h$) **do**
6 | | | | $r^h_{I''} \leftarrow$ `GetRegretsFor`(I'', r_h)
7 | | | | $\gamma_{I''} \leftarrow$ set of indices of $\{a \in \mathcal{A}(I'') | r^h_{I''}[a] \in \left[\max_{a' \in \mathcal{A}(I'')} \left(r^h_{I''}[a'] - \frac{1}{5\sqrt{t}}, \max_{a' \in \mathcal{A}(I'')} r^h_{I''}[a']\right)\right]\}$
8 | | | Split $\mathcal{I}''$ to subsets $\mathcal{I}''_1, ..., \mathcal{I}''_k$
9 | | | Each $\mathcal{I}''_j \in \{\mathcal{I}''_1, ..., \mathcal{I}''_k\}$ contains all the $I'' \in \mathcal{I}''$ with the same $\gamma_{I''}$
10 | | | $\mathcal{I}''_{max} \leftarrow \arg\max_{\mathcal{I}''_j \in \{\mathcal{I}''_1, ..., \mathcal{I}''_k\}} |\mathcal{I}''_j|$
11 | | | $\mathcal{I}''_f \leftarrow \mathcal{I}''_{max} \cup \mathcal{I}'' \setminus \hat{\mathcal{I}}_h$
12 | | | $G^t.\mathcal{I} \leftarrow G^t.\mathcal{I} \setminus I$
13 | | | $r \leftarrow r \setminus$ `GetRegretsFor`(I, r)
14 | | | **for** $\mathcal{I}''_j \in \left(\{\mathcal{I}''_1, ..., \mathcal{I}''_k\} \cup \mathcal{I}''_f\right) \setminus \mathcal{I}''_{max}$ **do**
15 | | | | $I_{new} \leftarrow$ `CreateNewIS`($\mathcal{I}''_j$)
16 | | | | $G^t.\mathcal{I} \leftarrow G^t.\mathcal{I} \cup I_{new}$
17 | | | | $r_{I_{new}} \leftarrow$ `InitRegret`(I_{new})
18 | | | | $r \leftarrow r \cup r_{I_{new}}$
19 | | | **if** `GetPlayer`(t) = *1* **then**
20 | | | | $\bar{b}_1 \leftarrow \bar{b}_1 \setminus$ `GetStrategyFor`(I, $\bar{b}_1$)
21 | | | **else**
22 | | | | $\bar{b}_2 \leftarrow \bar{b}_2 \setminus$ `GetStrategyFor`(I, $\bar{b}_2$)
23 | **return** G^t

The actual abstraction update is done in a following way (Algorithm 12). The algorithm iterates over $I \in \widetilde{\mathcal{I}}_i^t$ in G^t which contain some of the $I' \in \hat{\mathcal{I}}_h$ (lines 2 and 4). For all $I'' \in \Phi_t^{-1}(I) \cap \hat{\mathcal{I}}_h$ it creates a set of action indices $\gamma_{I''}$ corresponding to actions with regret in r_h at most $\frac{1}{5\sqrt{t}}$ distant from the maximum regret for I'' in r_h (line 7). The abstraction update then splits the set $\Phi_t^{-1}(I) \cap \hat{\mathcal{I}}_h$ to largest subsets $\mathcal{I}''_1, ..., \mathcal{I}''_k$ such that $\forall \mathcal{I}''_j \in \{\mathcal{I}''_1, ..., \mathcal{I}''_k\} \forall I''_1, I''_2 \in \mathcal{I}''_j \ \gamma_{I''_1} = \gamma_{I''_1}$ (lines 8 to 9). Next, the algorithm selects $\mathcal{I}''_{max}$, the largest element of $\{\mathcal{I}''_1, ..., \mathcal{I}''_k\}$ (line 10) and adds all the $I'' \in \Phi_t^{-1}(I)$ which are not in $\hat{\mathcal{I}}_h$ to $\mathcal{I}''_{max}$, creating $\mathcal{I}''_f$ (line 11). This is done to avoid unnecessary split caused by not tracking regrets in $\Phi_t^{-1}(I) \setminus \hat{\mathcal{I}}_h$. Finally, the abstracted set I is replaced in G^t by the set of new information sets $\mathcal{I}^t_n$ created from $\left(\{\mathcal{I}''_1, ..., \mathcal{I}''_k\} \cup \mathcal{I}''_f\right) \setminus \mathcal{I}''_{max}$. The regrets in each $I_n \in \mathcal{I}^t_n$ are set to 0 in r and the average strategies are discarded (lines 12 to 22). Finally, let T be an iteration such that $T > t$. Assuming that $I_n \in \mathcal{I}^t_n$ was not split further during

iterations $\{t+1, ..., T\}$, the average strategy in I_n is computed $\forall a \in \mathcal{A}(I_n)$ as

$$\bar{b}_i^T(I,a) = \begin{cases} \frac{1}{|\mathcal{A}(I)|}, & \text{if } T = t+1, \\ \frac{2\sum_{t'=t+1}^{T} t' \cdot \pi_i^{b_i^{t'}}(I) \cdot b_i^{t'}(I,a)}{((T-t-1)^2+T-t-1)\sum_{t'=t+1}^{T} \pi_i^{b_i^{t'}}(I)}, & \text{otherwise.} \end{cases} \tag{5.8}$$

The average strategy update in eq. (5.8) corresponds to the average strategy update described in Section 3.4.4.3, i.e., it does a weighted average of the b_i^t strategies over iterations $\{t+1, ..., T\}$ with weight corresponding to the iteration.

5.2.3.3 Theoretical Properties

In this section we present the bound on the average extrenal regret of the CFR+IRA algorithm. We first derive the bound for the case where the algorithm uses only the regret bound abstraction update described in Section 5.2.3.1. Since CFR+IRA randomly samples information sets during the abstraction update, we provide a probabilistic bound on the average external regret of the algorithm. We then show that the regret bound still holds when also using the heuristic update described in Section 5.2.3.2. Finally, we discuss why it is insufficient to use only the heuristic abstraction update.

Given iteration T, let $\mathcal{T}^T = (T_1, ..., T_k)$ be a subsequence of $\mathcal{T}$ such that T_k is the largest element in $\mathcal{T}$ for which $T_k < T$. τ^T is the sequence of iteration counts corresponding to $\mathcal{T}^T$. Let $\tau^T_{\bar{R}_i}$ be a sequence containing all the iteration counts $\tau_j \in \tau^T$, where for the corresponding T_j, T_{j+1} holds that

$$L_I^{T_j}(T_{j+1}) < \frac{\bar{R}_i}{|\mathcal{I}_i|}, \quad \forall I \in \mathcal{I}_i, \tag{5.9}$$

for a given regret $\bar{R}_i$. Next, we define $p_r(T, \bar{R}_i)$, as

$$p_r(T, \bar{R}_i) = \begin{cases} 0, \text{if } |\tau^T_{\bar{R}_i}| < |\mathcal{I}|, \\ 1 - \left(1 - \left(\frac{k_b}{|\mathcal{I}|}\right)^{|\mathcal{I}|}\right)^{\binom{|\tau^T_{\bar{R}_i}|}{|\mathcal{I}|}}, \text{otherwise.} \end{cases} \tag{5.10}$$

Lemma 5.2.1. *$p_r(T, \bar{R}_i)$ is the lower bound on the probability, that the abstraction in iteration T allows representation of average strategy with average external regret $\bar{R}_i$ for player i.*

Proof. p_r is computed for the worst case where all the $I \in \mathcal{I}$ are in some abstracted information set in G^1, and where it is necessary to reconstruct the complete original game by removing all $I \in \mathcal{I}$ from their abstracted information sets one by one in a fixed order to allow representation of the average strategy with average external regret $\bar{R}_i$ for player i. The rest of the proof is conducted in the following way: First, we show that the iteration counts in $\tau^T_{\bar{R}_i}$ are large enough to guarantee that if there is an abstracted information set preventing representation of average strategies with average external regret bellow $\bar{R}_i$,

it will be split. We then provide the probability that the information set structure in a given iteration allows CFR+IRA to compute strategies with average external regret $\bar{R}_i$ for player i as a function of the number of iteration counts in $\tau^T_{\bar{R}_i}$.

We know that

$$\bar{R}_i^T \leq \sum_{I \in \mathcal{I}_i} \left(R^T_{i,imm}(I)\right)^+ \leq |\mathcal{I}_i| \max_{I \in \mathcal{I}_i} \left(R^T_{i,imm}(I)\right)^+, \quad \forall T. \tag{5.11}$$

Lets assume that the information set structure of the current abstraction G^T in iteration T does not allow representation of average strategy with average external regret $\bar{R}_i$ and that we do not update G^T any further. Then, from eq. (5.11), there must exist $I \in \mathcal{I}_i$ such that for each iterations T', T'', $T'' > T'$

$$\left(R^{T''}_{T',imm}(I)\right)^+ > \frac{\bar{R}_i}{|\mathcal{I}_i|}. \tag{5.12}$$

To remove an information set I from its abstracted information set as a part of the regret bound abstraction update during the sequence of iterations $(T_j, ..., T_{j+1})$, there must exists $T' \in (T_j, ..., T_{j+1})$ such that $R^{T'}_{T_j,imm}(I) > L_I^{T_j}(T')$. Therefore, from eq. (5.12) follows, that to guarantee that I preventing convergence is removed from its abstracted information set during $(T_j, ..., T_{j+1})$, it needs to hold that $L_I^{T_j}(T_{j+1}) < \frac{\bar{R}_i}{|\mathcal{I}_i|}$. Hence, from eq. (5.9) in definition of $\tau^T_{\bar{R}_i}$ follows that $\tau^T_{\bar{R}_i}$ contains only the iteration counts that guarantee that such information set is removed from its abstracted information set.

Now we turn to the formula in eq. (5.10). The first case of the piecewise function in eq. (5.10) handles the situation where there are not enough iteration counts $\tau_j \in \tau^T_{\bar{R}_i}$ large enough to guarantee the required $|\mathcal{I}|$ splits. The second case computes the probability that given $|\tau^T_{\bar{R}_i}|$ samples we correctly sample the required sequence of the length $|\mathcal{I}|$. The second case has the following intuition. $\binom{|\tau^T_{\bar{R}_i}|}{|\mathcal{I}|}$ is the number of possibilities how to choose a subsequence of the length $|\mathcal{I}|$ from a sequence of length $|\tau^T_{\bar{R}_i}|$. $\left(\frac{k_b}{|\mathcal{I}|}\right)^{|\mathcal{I}|}$ is the probability that a specific sequence of all information sets, i.e., a sequence of length $|\mathcal{I}|$, is sampled when we sample k_b from $|\mathcal{I}|$ elements with a uniform probability. Hence, $\left(1 - \left(\frac{k_b}{|\mathcal{I}|}\right)^{|\mathcal{I}|}\right)^{\binom{|\tau^T_{\bar{R}_i}|}{|\mathcal{I}|}}$ is the probability that the sequence of length $|\mathcal{I}|$ is not sampled in $\binom{|\tau^T_{\bar{R}_i}|}{|\mathcal{I}|}$ attempts.

Since $p_r(T, \bar{R}_i)$ is computed assuming that there is the worst case number of splits necessary and that it takes the maximum possible number of iterations to split each information set, it is a lower bound on the actual probability that the abstraction in iteration T allows representation of average strategy with average external regret $\bar{R}_i$ for player i. □

Lemma 5.2.2. *The average external regret of the CFR+IRA is bounded in the following way*

$$\frac{R_i^T}{T} \leq B_i(T) = \frac{1}{T}\left(\Delta|\mathcal{I}_i|\sqrt{A_{max}}\sqrt{T} + \Delta|\mathcal{I}_i|T_{lim}\right), \tag{5.13}$$

with probability at least $p_r(T_{lim}, B_i(T))$.

Proof. $B_i(T)$ decomposes the bound on the average external regret to two parts. First, in iterations $\{1, ..., T_{lim}\}$ it assumes that the structure of the abstraction prevents the algorithm from convergence. Second, it assumes that the abstraction is updated so that it allows convergence of CFR+ in iterations $\{T_{lim}+1, ..., T\}$. Hence, the regret in all $I \in \mathcal{I}_i$ has the following property:

$$\max_{a \in \mathcal{A}(I)} R_i^T(I, a) \leq \Delta T_{lim} + \Delta\sqrt{|\mathcal{A}(I)|}\sqrt{T}. \tag{5.14}$$

This bound holds since $\Delta\sqrt{|\mathcal{A}(I)|}\sqrt{T}$ is the bound on $\max_{a \in \mathcal{A}(I)} R_i^T(I, a)$ in I provided by regret matching$^+$. Additionally, since the regret matching$^+$ in I in each $t \in \{1, ..., T_{lim}\}$ uses regrets computed for information set $\phi_t(I)$ and not directly for I, it can lead to arbitrarily bad outcomes with respect to the utility structure of the solved game (as we assume that the structure of the abstraction prevents convergence of the algorithm in these iterations). The ΔT_{lim} corresponds to the worst case regret that can be accumulated for each action in I during the first T_{lim} iterations. From eqs. (5.14) and (3.7) follows that

$$\frac{R_i^T}{T} \leq B_i(T) = \frac{1}{T}\left(\Delta|\mathcal{I}_i|\sqrt{A_{max}}\sqrt{T} + \Delta|\mathcal{I}_i|T_{lim}\right).$$

Since we assume that the abstraction in iteration T_{lim} of CFR+IRA allows computing the regret $B_i(T)$, this bound holds with probability at least $p_r(T_{lim}, B_i(T))$ (Lemma 5.2.1). □

Finally, we provide the bound on the external regret of CFR+IRA as a function of the probability that the bound holds.

Theorem 5.2.2. *Let*

$$\alpha(\delta) = \sqrt[3]{2^{\sqrt[|\mathcal{I}|]{|\mathcal{I}|!\log_{1-\left(\frac{k_b}{|\mathcal{I}|}\right)^{|\mathcal{I}|}}(\delta)}+|\mathcal{I}|+3} A_{max}}. \tag{5.15}$$

In each $T \geq (\alpha(\delta)+1)^3$*, the average external regret of CFR+IRA is bounded in the follwing way*

$$\frac{R_i^T}{T} \leq \Delta|\mathcal{I}_i|\frac{\sqrt{A_{max}}}{\sqrt{T}} + \Delta|\mathcal{I}_i|\frac{\alpha(\delta)}{\sqrt[3]{T}} + \Delta|\mathcal{I}_i|\frac{1}{T} \in \mathcal{O}\left(\frac{1}{\sqrt[3]{T}}\right). \tag{5.16}$$

with probability $1-\delta$.

Proof. The bound in eq. (5.16) is created by substituting

$$T_{lim} = \left\lceil \sqrt[3]{2 \sqrt[|\mathcal{I}|]{|\mathcal{I}|! \log_{1-\left(\frac{k_b}{|\mathcal{I}|}\right)^{|\mathcal{I}|}(\delta)} + |\mathcal{I}| + 3}} A_{max} \cdot T^{\frac{2}{3}} \right\rceil, \tag{5.17}$$

to the bound from Lemma 5.2.2. Hence, we need to show that choosing this T_{lim} guarantees that

$$p_r(T_{lim}, B_i(T)) \geq 1 - \delta. \tag{5.18}$$

The proof is conducted in the following way. First, in Lemma 5.2.3 we show the size of $\tau_{B_i(T)}^{T_{lim}}$ sufficient to guarantee that inequality (5.18) holds. In Lemma 5.2.4 we derive the lower bound $T_{B_i(T)}$ on each element of $\tau_{B_i(T)}^{T_{lim}}$. $T_{B_i(T)}$ is the number of iterations sufficient to guarantee that some information set preventing the abstraction from allowing representation of average strategy with average external regret $B_i(T)$ is split during the regret bound abstraction update. Finally, in Lemma 5.2.5 we derive the T_{lim} that implies sufficient number of elements in $\tau_{B_i(T)}^{T_{lim}}$.

Lemma 5.2.3.

$$|\tau_{B_i(T)}^{T_{lim}}| > \sqrt[|\mathcal{I}|]{|\mathcal{I}|! \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}(\delta)} + |\mathcal{I}|} \tag{5.19}$$

guarantees that $p_r(T_{lim}, B_i(T)) \geq 1 - \delta$.

Proof.

$$|\tau_{B_i(T)}^{T_{lim}}| > \sqrt[|\mathcal{I}|]{|\mathcal{I}|! \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}(\delta)} + |\mathcal{I}|} \tag{5.20}$$

$$\implies \left(|\tau_{B_i(T)}^{T_{lim}}| - |\mathcal{I}|\right)^{|\mathcal{I}|} > |\mathcal{I}|! \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}(\delta)} \tag{5.21}$$

$$\implies \prod_{j=|\tau_{B_i(T)}^{T_{lim}}|-|\mathcal{I}|+1}^{|\tau_{B_i(T)}^{T_{lim}}|} j > |\mathcal{I}|! \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}(\delta)} \tag{5.22}$$

$$\implies \frac{\prod_{j=|\tau_{B_i(T)}^{T_{lim}}|-|\mathcal{I}|+1}^{|\tau_{B_i(T)}^{T_{lim}}|} j}{|\mathcal{I}|!} > \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}(\delta)} \tag{5.23}$$

$$\implies \frac{|\tau_{B_i(T)}^{T_{lim}}|!}{(|\tau_{B_i(T)}^{T_{lim}}| - |\mathcal{I}|)!|\mathcal{I}|!} > \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}(\delta)} \tag{5.24}$$

$$\implies \binom{|\tau_{B_i(T)}^{T_{lim}}|}{|\mathcal{I}|} > \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}(\delta)} \tag{5.25}$$

$$\implies \left(1 - \left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}\right)^{\binom{|\tau^{T_{lim}}_{B_i(T)}|}{|\mathcal{I}|}} < \delta \tag{5.26}$$

$$\implies 1 - \left(1 - \left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}\right)^{\binom{|\tau^{T_{lim}}_{B_i(T)}|}{|\mathcal{I}|}} > 1 - \delta \tag{5.27}$$

□

Lemma 5.2.4.

$$T_{B_i(T)} = \left\lceil \left[\frac{T\sqrt{A_{max}}}{T_{lim} + \sqrt{A_{max}}\sqrt{T}} \right]^2 \right\rceil$$

is a sufficient number of iterations to guarantee that the regret bound abstraction update splits some information set preventing the abstraction from allowing representation of average strategy with average external regret $B_i(T)$.

Proof. The regret bound abstraction update removes an information set $I \in \mathcal{I}$ from its abstracted information set during the sequence of iterations $(T_j, ..., T_{j+1})$, when there exists $T' \in (T_j, ..., T_{j+1})$ such that $R^{T'}_{T_j,imm}(I) > L^{T_j}_I(T')$. As dicussed in the proof of Lemma 5.2.1, to guarantee that I is removed from its abstracted information set if it prevents the regret bellow $B_i(T)$, we need to make sure that

$$L^{T_j}_I(T_{j+1}) < \frac{B_i(T)}{|\mathcal{I}_i|}. \tag{5.28}$$

Since

$$L^{T_j}_I(T') \leq \frac{\Delta_I\sqrt{|\mathcal{A}(I)|}}{\sqrt{T' - T_j}}, \quad \forall T' \in \{T_j + 1, ..., T_{j+1}\}, \forall I \in \mathcal{I}_i,$$

it follows that

$$\max_{I \in \mathcal{I}_i} L^{T_j}_I(T_{j+1}) \leq \frac{\Delta\sqrt{A_{max}}}{\sqrt{T_{j+1} - T_j}}. \tag{5.29}$$

And so from eqs. (5.28) and (5.29), it is sufficient for $T_{B_i(T)}$ to satisfy

$$\frac{\Delta\sqrt{A_{max}}}{\sqrt{T_{B_i(T)}}} < \frac{B_i(T)}{|\mathcal{I}_i|}.$$

Smallest $T_{B_i(T)}$ satisfying this inequality is

$$T_{B_i(T)} = \left\lceil \left[\frac{T\sqrt{A_{max}}}{T_{lim} + \sqrt{A_{max}}\sqrt{T}} \right]^2 \right\rceil. \tag{5.30}$$

□

Hence, all elements in $\tau^{T_{lim}}_{B_i(T)}$ must be greater or equal to $T_{B_i(T)}$ to make sure that the structure of the abstraction in iteration T_{lim} allows representation of average strategy with average external regret $B_i(T)$ for player i with sufficient probability.

The number of elements in $\tau^{T_{lim}}$ is at least $\log_2(T_{lim}+1)-1$, since $\forall j \in \{1, ..., |\tau^{T_{lim}}|\}\ \tau_j = 2^{j-1}$. From Lemma 5.2.3 we know that we need the last

$$\left\lceil \sqrt[|\mathcal{I}|]{|\mathcal{I}|! \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta)} + |\mathcal{I}| \right\rceil$$

elements of $\tau^{T_{lim}}$ (which form $\tau^{T_{lim}}_{B_i(T)}$) to be higher or equall to $T_{B_i(T)}$.

Lemma 5.2.5. *When using*

$$T_{lim} = \left\lceil \sqrt[3]{2^{\sqrt[|\mathcal{I}|]{|\mathcal{I}|! \log_{1-\left(\frac{k_b}{|\mathcal{I}|}\right)^{|\mathcal{I}|}}(\delta)}+|\mathcal{I}|+3}} A_{max} \cdot T^{\frac{2}{3}} \right\rceil,$$

the last

$$\left\lceil \sqrt[|\mathcal{I}|]{|\mathcal{I}|! \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta)} + |\mathcal{I}| \right\rceil$$

elements in $\tau^{T_{lim}}$ *are higher or equal to* $T_{B_i(T)}$.

Proof.

$$T_{lim} = \left\lceil \sqrt[3]{2^{\sqrt[|\mathcal{I}|]{|\mathcal{I}|! \log_{1-\left(\frac{k_b}{|\mathcal{I}|}\right)^{|\mathcal{I}|}}(\delta)}+|\mathcal{I}|+3}} A_{max} \cdot T^{\frac{2}{3}} \right\rceil \tag{5.31}$$

$$\implies T_{lim} \geq \sqrt[3]{2^{\sqrt[|\mathcal{I}|]{|\mathcal{I}|! \log_{1-\left(\frac{k_b}{|\mathcal{I}|}\right)^{|\mathcal{I}|}}(\delta)}+|\mathcal{I}|+3}} A_{max} \cdot T^{\frac{2}{3}} \tag{5.32}$$

$$\implies T^3_{lim} \geq 2 \cdot 2^{\sqrt[|\mathcal{I}|]{|\mathcal{I}|! \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|+1}}(\delta)}+|\mathcal{I}|+2} A_{max} T^2 \tag{5.33}$$

$$\implies (T_{lim}+1)\left(T_{lim} + \sqrt{A_{max}}\sqrt{T}\right)^2 \geq \tag{5.34}$$

$$\geq 2^{\sqrt[|\mathcal{I}|]{|\mathcal{I}|! \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta)}+|\mathcal{I}|+2} 2A_{max}T^2 \tag{5.35}$$

$$\implies \frac{T_{lim}+1}{2^{\sqrt[|\mathcal{I}|]{|\mathcal{I}|! \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta)}+|\mathcal{I}|+2}} \geq \frac{2A_{max}T^2}{\left(T_{lim} + \sqrt{A_{max}}\sqrt{T}\right)^2} \tag{5.36}$$

$$\implies \frac{T_{lim}+1}{2^{\sqrt[|\mathcal{I}|]{|\mathcal{I}|! \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta)}+|\mathcal{I}|+2}} \geq \left\lceil \frac{A_{max}T^2}{\left(T_{lim} + \sqrt{A_{max}}\sqrt{T}\right)^2} \right\rceil \tag{5.37}$$

$$\implies \quad 2^{\log_2(T_{lim}+1)-2-\sqrt[|\mathcal{I}|]{|\mathcal{I}|!\log_{1-\left(\frac{k_b}{T}\right)^{|\mathcal{I}|}}(\delta)-|\mathcal{I}|}} \geq T_{B_i(T)} \tag{5.38}$$

$$\implies \quad 2^{\log_2(T_{lim}+1)-1-\left\lceil \sqrt[|\mathcal{I}|]{|\mathcal{I}|!\log_{1-\left(\frac{k_b}{T}\right)^{|\mathcal{I}|}}(\delta)+|\mathcal{I}|} \right\rceil} \geq T_{B_i(T)} \tag{5.39}$$

Eq. (5.39) states that last

$$\left\lceil \sqrt[|\mathcal{I}|]{|\mathcal{I}|!\log_{1-\left(\frac{k_b}{T}\right)^{|\mathcal{I}|}}(\delta)+|\mathcal{I}|} \right\rceil$$

elements of $\boldsymbol{\tau}^{T_{lim}}$ are at least $T_{B_i(T)}$, since the elements in $\boldsymbol{\tau}^{T_{lim}}$ are increasing. □

Hence, using T_{lim} from Lemma 5.2.5 guarantees that $p_r(T_{lim}, B_i(T)) > 1-\delta$ from Lemma 5.2.3. When substituting this T_{lim} to the bound from Lemma 5.2.2, i.e., to

$$\frac{R_i^T}{T} \leq \frac{1}{T}\left(\Delta|\mathcal{I}_i|\sqrt{A_{max}}\sqrt{T} + \Delta|\mathcal{I}_i|T_{lim}\right), \tag{5.40}$$

we get

$$\begin{aligned}
\frac{R_i^T}{T} &\leq \frac{1}{T}\left(\Delta|\mathcal{I}_i|\sqrt{A_{max}}\sqrt{T} + \Delta|\mathcal{I}_i|\left\lceil\alpha(\delta)T^{\frac{2}{3}}\right\rceil\right) && (5.41)\\
&\leq \Delta|\mathcal{I}_i|\frac{\sqrt{A_{max}}}{\sqrt{T}} + \Delta|\mathcal{I}_i|\frac{\alpha(\delta)}{\sqrt[3]{T}} + \Delta|\mathcal{I}_i|\frac{1}{T} && (5.42)\\
&\in \mathcal{O}\left(\frac{1}{\sqrt[3]{T}}\right). && (5.43)
\end{aligned}$$

Finally, we need to show that using $T \geq (\alpha(\delta)+1)^3$ guarantees that $T \geq T_{lim}$:

$$\begin{aligned}
& T \geq (\alpha(\delta)+1)^3 && (5.44)\\
\implies \quad & \sqrt[3]{T} \geq (\alpha(\delta)+1) && (5.45)\\
\implies \quad & T \geq (\alpha(\delta)+1)T^{\frac{2}{3}} && (5.46)\\
\implies \quad & T \geq \left\lceil\alpha(\delta)T^{\frac{2}{3}}\right\rceil && (5.47)\\
\implies \quad & T \geq T_{lim}. && (5.48)
\end{aligned}$$

□

Combining Regret Bound and Heuristic Abstraction Update.

When using the heuristic abstraction update in combination with the regret bound update, Theorem 5.2.2 still holds, since in the worst case the heuristic update does not

Table 5.1: Expected utilities after actions in I' and I''

	I'	I''
c	0	0
d	1	10

	I'	I''
c	10	1
d	0	0

perform any splits and all the information set splits need to be performed by the regret bound update. Adding the heuristic abstraction update can only reduce the number of iterations needed by the algorithm.

Counterexample for Heuristic Abstraction Update.

Here we show the intuition why using only the heuristic abstraction update does not guarantee convergence of CFR+IRA to the NE of the solved game G.

Let I be an information set in G^t such that $\phi_t^{-1}(I) = \{I', I''\}$. Let $\mathcal{A}^t(I) = \{c, d\}$ and $\mathcal{A}(I') = \{x, y\}$, $\mathcal{A}(I'') = \{v, w\}$. Lets assume that the expected values for actions c and d in I' and I'' oscilate between values depicted in Table 5.1 (left) and (right). Hence, when computing the regret $r^h_{\hat{I}}(a) = v_i(b^t_{\hat{I}\to a}, \hat{I}) - v_i(b^t, \hat{I})$ for $\hat{I} \in \{I', I''\}$ and $a \in \mathcal{A}(\hat{I})$ during the heuristic abstraction update in iteration t where the expected utilities from Table 5.1(left) occur, we get

$$r^h_{I'}(c) = -b^t_i(I, d) \tag{5.49}$$
$$r^h_{I'}(d) = 1 - b^t_i(I, d) \tag{5.50}$$
$$r^h_{I''}(c) = -10b^t_i(I, d) \tag{5.51}$$
$$r^h_{I''}(d) = 10 - 10b^t_i(I, d). \tag{5.52}$$

When computing the regret r^h for heuristic update in iteration t' where the expected utilities from Table 5.1 (right) occur, we get

$$r^h_{I'}(c) = 10 - 10b^{t'}_i(I, c) \tag{5.53}$$
$$r^h_{I'}(d) = -10b^{t'}_i(I, c) \tag{5.54}$$
$$r^h_{I''}(c) = 1 - b^{t'}_i(I, c) \tag{5.55}$$
$$r^h_{I''}(d) = -b^{t'}_i(I, c). \tag{5.56}$$

The actions corresponding to d are always prefered in both I' and I'' when the utilties are given by Table 5.1 (left) by the same margin as the actions corresponding to c are always prefer in both I' and I'' when the utilties are given by Table 5.1 (right). Hence, $\gamma_{I'} = \gamma_{I''}$ (i.e., the inidices of actions with the largest regret are always equal) in all iterations t'' for any $b^{t''}_i$. Therefore, I is never split during the heuristic update. However, CFR+IRA in I converges to a uniform startegy with the average expected value 2.75 in both I' and I'' since the regret updates lead to equal regrets for both c and d, while CFR+ converges to strategy $b_i(I', x) = 1$ and $b_i(I'', w) = 1$ with the average expected value 5 in both I' and I''.

5.2.3.4 Storing the Information Set Map

In this section we discuss the details of storing the mapping of information sets $\mathcal{I}$ of G to $\mathcal{I}^t$ of G^t in CFR+IRA.

Initial Abstraction Storage. The initial abstraction is identical to the initial abstraction used by FPIRA, hence the mapping is stored without using any additional memory as described in Section 5.2.2.4.

Regret Bound Update. The regret bound update resulting in G^t only removes information sets from G from the abstracted information sets in G^{t-1}. Hence similarly to Section 5.2.2.4 Case 2, we use the mapping used in G^{t-1} in all information sets of G mapped to abstracted information sets in G^t, and the information set structure provided by the domain description in the rest. Therefore, no additional memory is needed to store the mapping for G^t compared to the mapping required in G^{t-1}.

Heuristic Update. The heuristic update resulting in G^t, can split existing abstracted information set I in G^{t-1} to a set of information sets $\mathcal{I}'$ such that some $I' \in \mathcal{I}'$ are still abstracted information sets. Hence in this case we need to store the new mapping as described in Section 5.2.2.4 Case 1.

5.2.3.5 Memory and Time Efficiency

The CFR+ requires storing the regret and average strategy for each $I \in \mathcal{I}$ and $a \in \mathcal{A}(I)$ in G. Hence, when storing the regrets and average strategy in G^t in iteration t in CFR+IRA, the algorithm achieves memory savings directly proportional to the reduction in the number of information sets in G^t compared to the number of information sets in G. Additional memory used to store the current abstraction mapping is discussed in Section 5.2.3.4. Finally, CFR+IRA stores additional regrets in $k_b + k_h$ information sets of G in every iteration for the purposes of the abstraction update. Since k_b and k_h are parameters of the CFR+IRA, this memory can be adjusted as necessary. In Section 5.3 we empirically demonstrate that memory used to store the information set mapping is small and so it does not substantially affect the memory efficiency of CFR+IRA. Additionally, we show how the different k_h and k_b affect the convergence of CFR+IRA.

The iteration of CFR+IRA consists of a tree traversal in function ComputeRegrets (Algorithm 9) and the two updates of the abstraction. The update of regrets r as a part of ComputeRegrets takes the same time as standard CFR+ iteration applied to G. Additionally, in ComputeRegrets, CFR+IRA updates r_h and r_b. Since there is no additional tree traversal neccessary to obtain the values for the update of r_h and r_b, the update takes time proportional to the number of states in $\hat{\mathcal{I}}_b \cup \hat{\mathcal{I}}_h$. In the worst case, the abstraction updates take time proportional to $|\mathcal{I}|$. In Section 5.3 we provide experimental evaluation of the runtime of CFR+IRA.

5.3 Experiments

In this section we present the experimental evaluation of FPIRA and CFR+IRA. First, we introduce domains used for the experimental evaluation. Next, we demonstrate the

convergence of CFR+IRA compared to CFR+ and explain our choice of values of the k_h and k_b parameters in the rest of the experimental evaluation (these parameters control the memory used by CFR+IRA to update the abstraction, see Sections 5.2.3.1 and 5.2.3.2 for more details). We follow with the comparison of the memory requirements of FPIRA, CFR+IRA and the DOEFG (see Section 5.1 for the explanation why we compare to DOEFG) as a function of the exploitability of the resulting strategies. Finally, we provide the comparison of the runtime of FPIRA, CFR+IRA, and DOEFG. When reporting the results for CFR+IRA, BxHyCFR+IRA stands for CFR+IRA where $k_b = x$ and $k_h = y$.

5.3.1 Experimental Settings

The experiments were performed using domain independent implementation of all algorithms in Java[1]. DOEFG uses IBM CPLEX 12.6. to solve the underlying linear programs. Both FPIRA and CFR+IRA used the initial abstraction built as described in Section 5.2.1.1. Both CFR+ and CFR+IRA used the delay of 100 iterations during the average strategy update, i.e., the average strategies are not computed for the first 100 iterations, and the average strategies in iteration $T > 100$ are computed only from current strategies in iterations $\{101, ..., T\}$. And finally, the $L_I^{T_j}$ functions in CFR+IRA, used during the regret bound update (see Section 5.2.3.1), were set to

$$L_I^{T_j}(T') = \frac{\Delta_I \sqrt{\mathcal{A}(I)}}{100\sqrt{T'}}, \forall I \in \mathcal{I}, \forall T_j, T_{j+1}, \forall T' \in \{T_j + 1, ..., T_{j+1}\}.$$

5.3.2 Domains

Here we introduce the zero-sum domains used in the experimental evaluation. These domains were chosen for their diverse structure both in the utility and the source of imperfect information. The players perfectly observe the actions of their opponent in poker, only the actions of the chance player at the start of the game (corresponding to card deal) are hidden. In II-Goofspiel and Graph pursuit, on the other hand, the imperfect information is created by partial observability of the moves of the opponent through the whole game. Furthermore, in II-Goofspiel and poker, the utility of players cumulates during the playthrough (the chips and the value of the cards won), while in Graph pursuit the utility depends solely on whether the attacker is intercepted or not or whether he reaches his goal.

5.3.2.1 Poker

As a first domain, we use a two-player poker, which is commonly used as a benchmark in imperfect-information game solving [50]. We use a version of poker with a deck of cards with 4 card types 3 cards per type. There are two rounds. In the first round, each player

[1]The implementation is available at http://jones.felk.cvut.cz/repo/gtlibrary. CFR+IRA and FPIRA are available in package /src/cz/agents/gtlibrary/experimental/imperfectrecall/automatedabstractions/memeff.

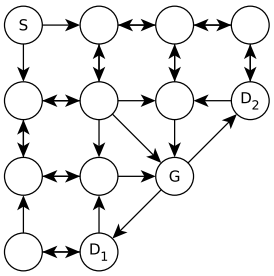

Figure 5.2: Graph used in the Graph pursuit domain.

places an ante of 1 chip in the pot and receives a single private card. A round of betting follows. Every player can bet from a limited set of allowed values or check. After a bet, the other player can raise, again choosing the value from a limited set, call or forfeit the game by folding. The number of consecutive raises is limited. A shared card is dealt after one of the players calls or after both players check. Another round of betting takes place with identical rules. The player with the highest pair wins. If none of the players has a pair, the player with the highest card wins. We create different poker domains by varying the number of bets b, the number of raises r and the number of consecutive raises allowed c. We refer to these instances as Pbrc, e.g., P234 stand for a poker which uses two possible values of bets, 3 values of raises and allows 4 consecutive raises.

5.3.2.2 II-Goofspiel

II-Goofspiel is a modification of the Goofspiel game [49] which is commonly used as a benchmark domain (see, e.g., [43, 41]). Similarly to Goofspiel, II-Goofspiel is a card game with three identical packs of cards, two for players and one randomly shuffled and placed in the middle. In our variant, both players know the order of the cards in the middle pack. The game proceeds in rounds. Every round starts by revealing the top card of the middle pack. Both players proceed to simultaneously bet on this card using their own cards. The cards used to bet are discarded, and the player with the higher value of the card used to bet wins the middle card. After the end of the game, each player gets utility equal to the difference between the points collected by him and the number of points collected by his opponent. The players do not observe the bet of their opponent. Instead, they learn whether they have won, lost, or if there was a tie caused by both players using cards with equal value. We change the number of cards in all 3 decks, by GSx, we refer to the II-Goofspiel where each deck has x cards.

5.3.2.3 Graph Pursuit

Graph pursuit is a game played between the defender and the attacker on the graph depicted in Figure 5.2. The attacker starts in the node labeled S and tries to reach the node labeled G. The defender controls two units which start in nodes D_1 and D_2. The players move simultaneously and are forced to move their units each round. Both the

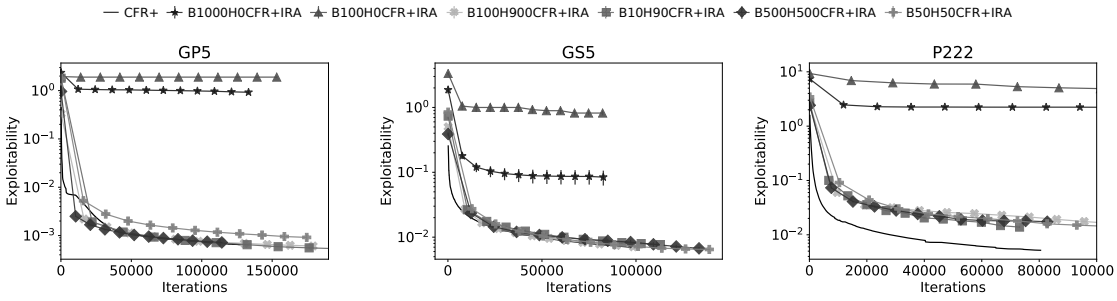

Figure 5.3: The plots showing the sum of exploitabilities of the resulting strategies for player 1 and 2 (log y-axis) as a function of iterations (x-axis) for GP5, GS5, P222.

attacker and defender only observe the content of the nodes with distance less or equal to 2 from the current node occupied by any of their units. The attacker gets utility 2 for reaching the goal G. If the attacker is caught by crossing the same edge as any of the units of the defender or by moving to a node occupied by the defender he obtains the utility -1 and the game ends. If a given number of moves occurs without any of the previous events, the game is a tie, and both players get 0. We create different versions of Graph pursuit by changing the limit on the number of moves. By GPx we denote Graph pursuit where there are x moves of each player allowed.

5.3.3 Convergence of CFR+IRA

In this section, we provide the experiments showing the convergence of CFR+IRA with varying k_h and k_b parameters compared to CFR+ applied directly to the unabstracted game. Additionally, we justify our choice of values of the k_h and k_b parameters in the rest of the experimental evaluation.

In Figure 5.3 we present the sum of exploitabilities of the resulting strategies of player 1 and 2 computed by CFR+IRA with various settings compared to CFR+ for GP5, GS5, and P222 as a function of the number of iterations. We depict the results for CFR+IRA as averages and standard error over 10 runs of the algorithm (the standard error is usually too small to be visible). Each run uses a different seed to randomly sample the information sets for regret bound and heuristic abstraction updates. We show 3 settings of CFR+IRA where $k_h + k_b = 100$ and 3 settings where $k_h + k_b = 1000$. The convergence of CFR+IRA with $k_h = 0$ is significantly worse compared to the rest of the settings across all domains even when we increase $k_h + k_b$ from 100 to 1000. For this reason, we focus only on settings where $k_h > 0$ in the following experimental evaluation. In GP5 and GS5 all CFR+IRA parametrizations with $k_h > 0$ converge similarly to CFR+. Note that CFR+IRA can converge faster than CFR+ since the final abstracted game which allows convergence can have significantly less information sets and hence tighter bound on the average external regret. On the other hand, in P222 CFR+ converges faster than CFR+IRA. Additionally, there is very little difference between the convergence speed of the different settings of CFR+IRA where $k_h > 0$.

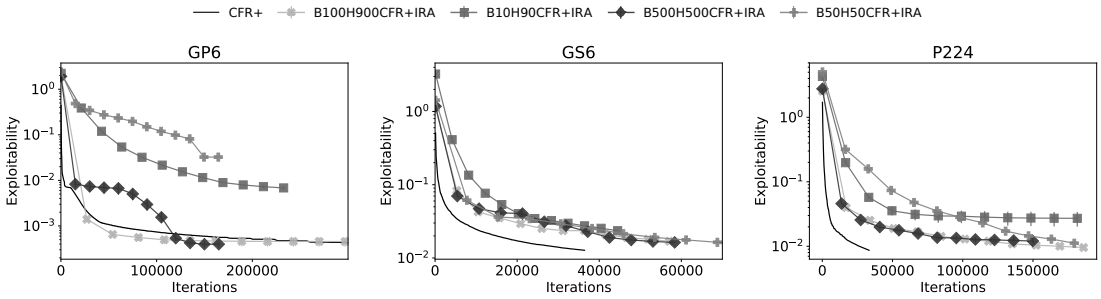

Figure 5.4: The plots showing the sum of exploitabilities of the resulting strategies of player 1 and 2 (log y-axis) as a function of iterations (x-axis) for GP6, GS6, P224.

In Figure 5.4 we show the same results for GP6, GS6 and P224. The plots show slower convergence of CFR+IRA with $k_h + k_b = 100$ compared to $k_h + k_b = 1000$. Additionally, except for GP6, all versions of CFR+IRA converge slower than the CFR+. This is expected, since using $k_h + k_b = 100$ and $k_h + k_b = 1000$ means that the algorithm uses less than 0.1% and 1% of information sets for abstraction update in all 3 domains. Hence it takes longer to refine the abstraction to allow strategies with a smaller exploitability.

The results above suggest that all the evaluated settings of CFR+IRA with fixed $k_h + k_b$ and $k_h > 0$ perform similarly. Additionally, we have observed that this similarity also holds for the memory required by CFR+IRA. On the other hand, increasing the sum of $k_h + k_b$ from 100 to 1000 improved the convergence speed of CFR+IRA in the larger domains. Hence, for clarity, in the following experiments we report only two settings of CFR+IRA, namely B10H90CFR+IRA and B100H900CFR+IRA.

5.3.4 Memory Requirements of Algorithms

In this section, we discuss the memory requirements of FPIRA, CFR+IRA, and DOEFG as a function of the exploitability of the resulting strategies on 2 sets of domains: GP5, GS5, P222 with approx. 10^4 information sets and GP6, GS6, P224 with approx. 10^5 information sets. Since the actual memory usage is implementation dependent, we analyze the size of the abstractions built by CFR+IRA and FPIRA and the size of the rest of the data structures required by these algorithms. We compare these results with the size of the restricted game built by DOEFG and other data structures stored by this algorithm. Next, we compare the number of 32-bit words stored during the run of FPIRA, CFR+IRA, and DOEFG. We conclude this section by more detailed analysis of the scalability of CFR+IRA in the poker domain and compare this scalability with DOEFG and FPIRA.

5.3.4.1 GP5, GS5 and P222

In Figure 5.5 we show the number of information sets in the abstraction (or restricted game) built by the algorithms as a function of the sum of exploitabilities of the resulting strategies of player 1 and 2 for GP5, GS5, and P222. We report these results for DOEFG,

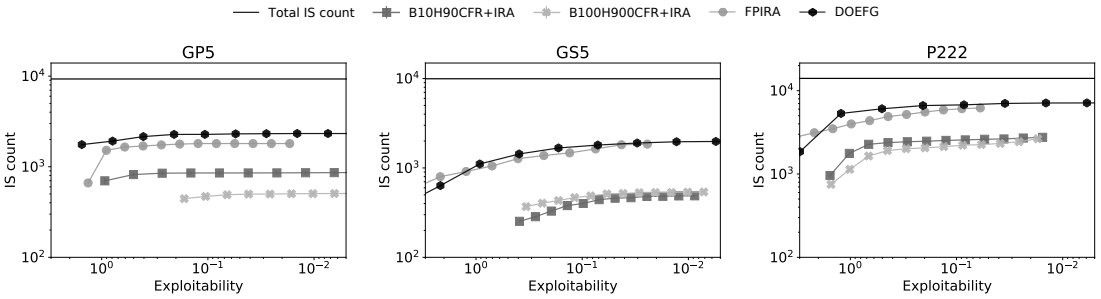

Figure 5.5: The plots depicting the number of information sets (log y-axis) used by algorithms to compute strategies with the exploitability depicted on the log x-axis for GP5, GS5, P222.

FPIRA and two settings of CFR+IRA. The lines for CFR+IRA depict the average number of information sets over 10 runs of CFR+IRA. Each run uses a different seed to randomly sample the information sets for regret bound and heuristic abstraction updates. Note that the standard error is too small to be visible in these plots. FPIRA and DOEFG require a similar number of information sets to reach strategies with equal exploitability. The CFR+IRA, on the other hand, requires significantly less information sets than FPIRA and DOEFG for these domains. For exploitability 0.05, the B10H90CFR+IRA uses on average 9.5%, 3.1%, 14.9% of information sets of the total information set count of GP5, GS5 and P222 respectivelly, while FPIRA uses 19.6%, 15.6%, 42.9% and DOEFG 24.0%, 17.2%, 47.8%.

In Figure 5.6 we show the size of data structures stored during the run of FPIRA, CFR+IRA, and DOEFG for GP5, GS5, and P222.

FPIRA: The results for FPIRA are presented in the plots in the first row of the Figure. We report the number of information sets of the abstraction, the size of the information set mapping, the size of the cache used during the best response computation, the size of the best response and the size of the cache used during the Δ computation as a function of the sum of exploitabilities of the resulting strategies of player 1 and 2. Note that the cache sizes and best response strategy size are reported as the maximum size encountered until FPIRA reached strategies with the corresponding exploitability. The results show that the size of the abstraction mapping remains small. On the other hand, the cache used during the best response computation and the Δ computation can require prohibitive memory when applied to large games (the cache stores value for each state encountered during the best response computation, and hence its size can be significantly larger than the number of information sets). For example, for exploitability 0.05 the size of the information set mapping was 20.5%, 18.2%, 43.4% of the total information set count of GP5, GS5, and P222, while the size of the cache in the best response computation was 109.6%, 23.8%, 106.1%.

CFR+IRA: The plots in the second row show that both the average mapping size and the average number of information sets in the abstraction in CFR+IRA remain small: B10H90CFR+IRA stored the mapping on average only for 10.7%, 3.9%, 22.1% of the total

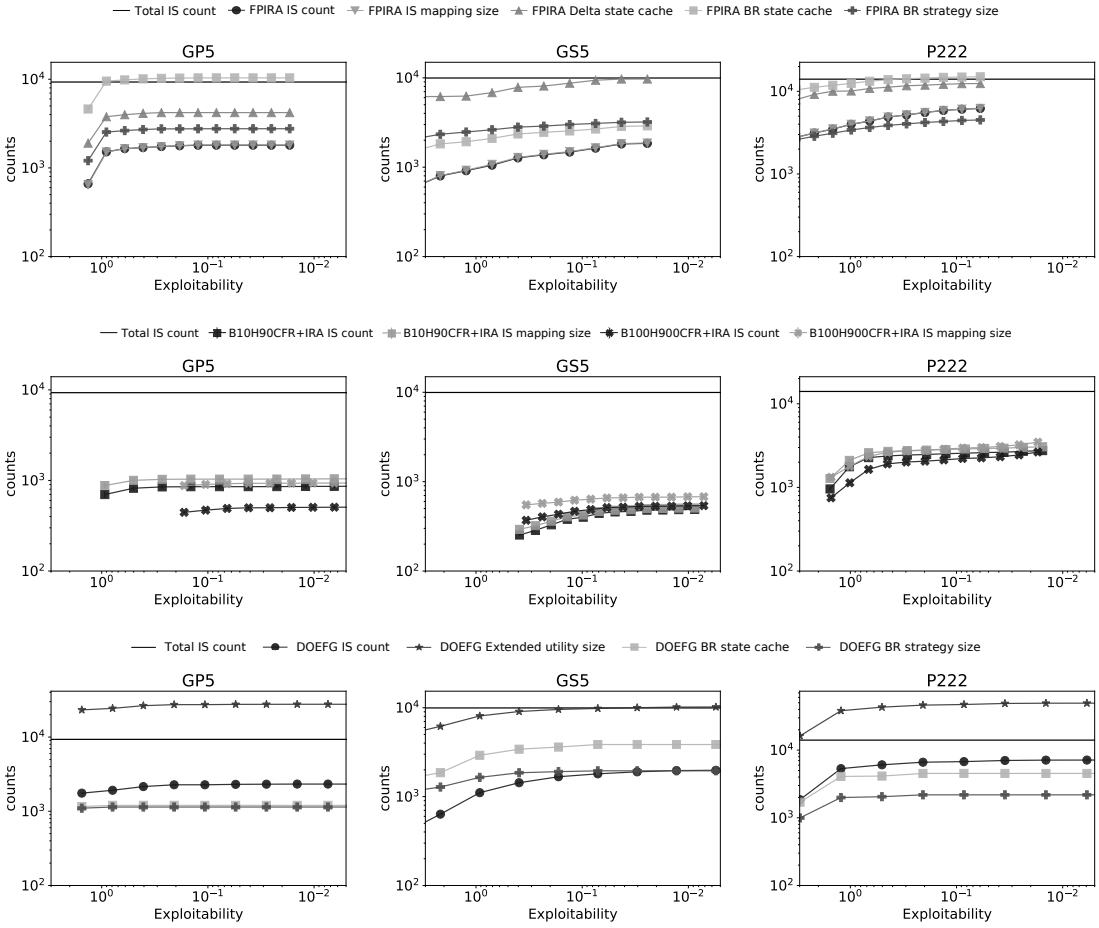

Figure 5.6: Plots showing size of data stored during the run of FPIRA in the first row, CFR+IRA in the second row and DOEFG in the third row (log y-axis) as a function of the sum of exploitabilities of the resulting strategies of player 1 and 2 (log x-axis) for GP5, GS5 and P222.

information sets for GP5, GS5, and P222 respectively. Additionally, in case of GP5, GS5 and P222 $k_h + k_b = 100$ corresponds to storing regrets required for the abstraction update only in $1.1\%, 1.0\%, 0.7\%$ of informations sets of the whole game respectivelly.

DOEFG: Finally in the third row we show the data structures required by the DOEFG. Similarly to FPIRA we report the maximum size of the cache used in the best response and the maximum size of the best response strategy since DOEFG uses the best response computation with the cache as oracle extending the restricted game (see [7], Section 4.2). Additionally, we report the size of the extended utility, which is required to construct the sequence-form LP for the current restricted game. The extended utility stores one value for each combination of sequences leading to some leaf or temporary leaf (see [7] for more details). Finally, we report the number of information sets in the

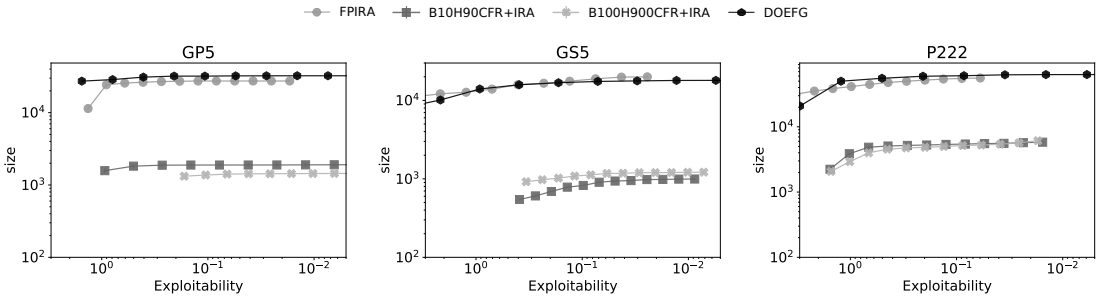

Figure 5.7: The number of 32 bit words the algorithms store (log y-axis) as a function of the exploitability of the resulting strategies (log x-axis) for GP5, GS5 and P222.

restricted game. Note that it is not clear whether these are all the data required by the DOEFG since DOEFG was never implemented or described with emphasis on memory efficiency. However, we believe that these data form a necessary subset of the data required by the DOEFG. The results show that storing the extended utility requires prohibitive memory as its size can be significantly larger than the number of information sets. For example, for exploitability 0.05 the size of the extended utility was 297%, 102% and 352% of the total information set count of GP5, GS5, and P222.

Finally, in Figure 5.7 we present the total number of 32-bit words (integers and floats) the algorithms need to store in GP5, GS5, and P222 as a function of the exploitability of the resulting strategies. These values were computed from the data depicted in Figure 5.6. Furthermore, to properly reflect the data stored by FPIRA in the abstraction, we replace the abstraction size by the number of floats that are used to represent the average strategy stored in the current abstraction. Similarly, for CFR+IRA we replace the average abstraction size by the average number of floats that are required to represent the regrets and average strategies stored in the current abstraction. These results show that the number of words stored by FPIRA and DOEFG is comparable. On the other hand, CFR+IRA in both settings requires an order of magnitude smaller memory than FPIRA and DOEFG.

5.3.4.2 GP6, GS6 and P224

In Figure 5.8 we present the results showing the abstraction size for GP6, GS6 and P224. We depict the results for CFR+IRA as averages with the standard error over 5 runs with different seeds (the standard error is again too small to be visible). The CFR+IRA is capable of solving the games using abstractions with significantly less information sets than the rest of the algorithms. For exploitability of the resulting strategies 0.05, the B100H900CFR+IRA uses on average 2.0%, 2.4%, 0.9% of information sets of GP6, GS6 and P224, while DOEFG uses 13.3%, 12.0%, 4.0%. A slow runtime prevented FPIRA from convergence to strategies with exploitability 0.05 in the given time (see Section 5.3.5 for runtime analysis). Additionally, in case of GP6, GS6 and P224, $k_h + k_b = 100$ corre-

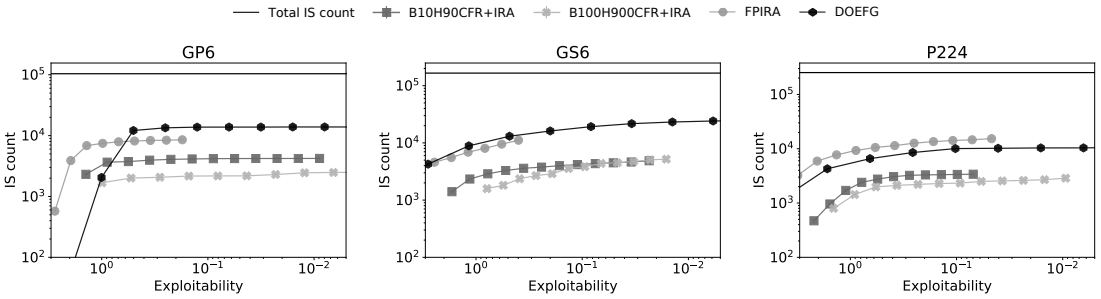

Figure 5.8: The plots depicting the number of information sets (log y-axis) used by algorithms to compute strategies with the sum of their exploitabilities depicted on the log x-axis for GP6, GS6 and P224.

sponds to storing regrets required for the abstraction update only in $0.09\%, 0.06\%, 0.03\%$ of informations sets of the whole game respectivelly.

In Figure 5.9 we present the size of data structures stored during the run of CFR+IRA (first row) and DOEFG (second row) for GP6, GS6, and P224. These results confirm that CFR+IRA requires small memory even in larger domains, while the data stored by DOEFG remain large.

Finally, in Figure 5.10 we again depict the number of 32-bit words stored by the algorithms for GP6, GS6, and P224. These results further confirm that CFR+IRA requires at least an order of magnitude less memory than FPIRA and DOEFG.

5.3.4.3 Relative Size of CFR+IRA Abstractions

Next, we compare how the size of the abstraction of the CFR+IRA scales with the size of the solved domain. In Figure 5.11 we provide the relative size of the abstraction required by CFR+IRA to compute strategies with exploitability 0.05 on P111, P222, and P224 as a function of the total information set count of the solved domains. The relative sizes reported are computed with respect to the total number of information set of the solved domains (left plot), the size of the restricted game required by DOEFG (middle plot) and the size of the abstraction required by FPIRA (right plot). The plots show that the relative size of the abstraction required by CFR+IRA decreases in all 3 settings. This suggests that for larger domains the relative size of the abstraction built by CFR+IRA will further decrease not only compared to the total information set count but also compared to the restricted game required by DOEFG and the abstraction size required by FPIRA.

5.3.5 Runtime

In this section, we provide a comparison of the runtime of the algorithms. The plots in Figure 5.12 show the runtime comparison in seconds of CFR+IRA, FPIRA and CFR+ for GP5, GS5, and P222. We again depict the results for CFR+IRA as averages with the standard error over 10 runs with different seeds. The runtime of CFR+IRA in all version

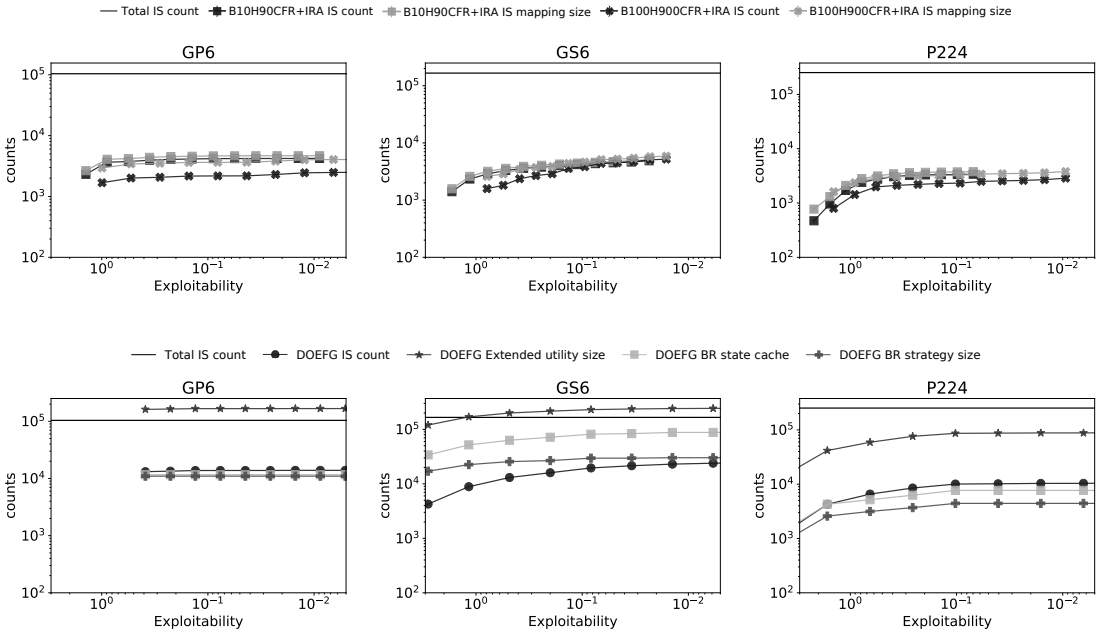

Figure 5.9: Plots showing size of data stored during the run of CFR+IRA in the first row and DOEFG in the second row (log y-axis) as a function of the sum of exploitabilities of the resulting strategies of player 1 and 2 (log x-axis) for GP6, GS6 and P224.

is consistently better than FPIRA and except for P222 is comparable to the runtime of CFR+. We omitted the DOEFG from this comparison since all CFR+IRA, FPIRA and CFR+ use a domain-independent implementation in Java, while our implementation of DOEFG uses efficient IBM CPLEX LP solver. Furthermore, as discussed above, DOEFG was never implemented with emphasis on memory efficiency. And so it heavily exploits additional caches for the restricted game not reported in the previous section which significantly improve its runtime while increasing its memory requirements. Hence, e.g., in case of GP5, the DOEFG took 19 seconds to find strategies with exploitability 0.01, while B100H900CFR+IRA took 38 seconds and the B10H90CFR+IRA 200 seconds. On the other hand, DOEFG required 1 GB of memory, while both settings of CFR+IRA used 38 MB.

The plots in Figure 5.13 show the runtime comparison in seconds of CFR+IRA, FPIRA and CFR+ for GP6, GS6 and P224. These plots further confirm the runtime dominance of CFR+IRA over FPIRA. The high runtime of FPIRA is the cause for omitting the results of FPIRA for smaller exploitabilies of the resulting strategies, as the time required to compute them becomes prohibitive. Furthermore, the results show that the runtime is worse for CFR+IRA where $k_b + k_h = 100$ compared to the case with $k_b + k_h = 1000$. Additionally, there is a more profound difference between the CFR+IRA runtime and the runtime of CFR+. Both observations are expected since using $k_h + k_b = 100$ and $k_h + k_b = 1000$ means that the algorithm uses less than 0.1% and 1% of information sets

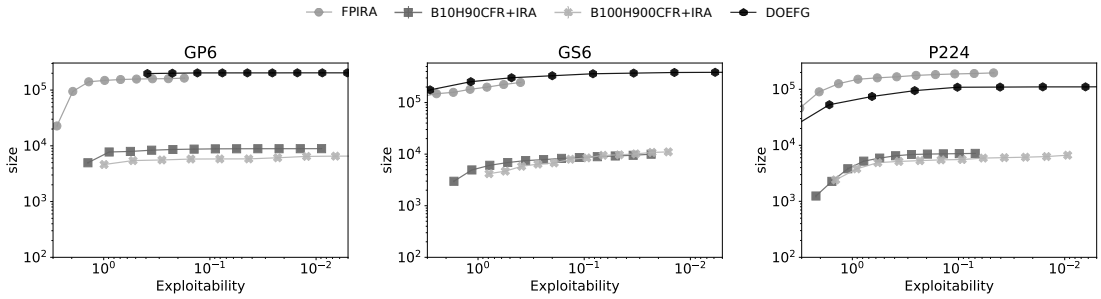

Figure 5.10: The number of 32 bit words the algorithms store (log y-axis) as a function of the exploitability of the resulting strategies (log x-axis) for GP6, GS6 and P224.

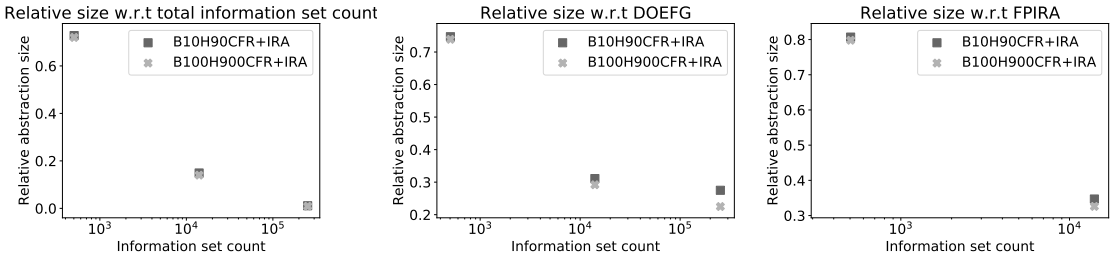

Figure 5.11: Plots showing the relative size of the abstractions used by CFR+IRA (y-axis) compared to the total information set count, the size of the restricted game used by DOEFG and the size of the abstraction used by FPIRA to compute resulting strategies with the sum of exploitabilities 0.05 as a function of the information set count of different poker instances (log x-axis).

for the abstraction update in all 3 domains. Hence it takes longer to refine the abstraction to allow strategies with a smaller exploitability.

5.3.6 Experiment Summary

We have shown that CFR+IRA requires at least an order of magnitude less memory to find strategies with a given exploitability compared to the memory required by FPIRA and DOEFG. Furthermore, the results suggest that when increasing the size of the solved domains, the relative memory requirements of CFR+IRA will further decrease not only compared to the total information set count of the solved domain but also compared to the memory requirements of FPIRA and DOEFG. Additionally, we have shown that the heuristic abstraction update provides a good indication of the parts of the abstraction that need to be updated and hence significantly enhances the convergence speed of CFR+IRA.

From the runtime perspective, the DOEFG is the most efficient algorithm. However, we show that its good performance comes at the cost of high memory requirements. Hence,

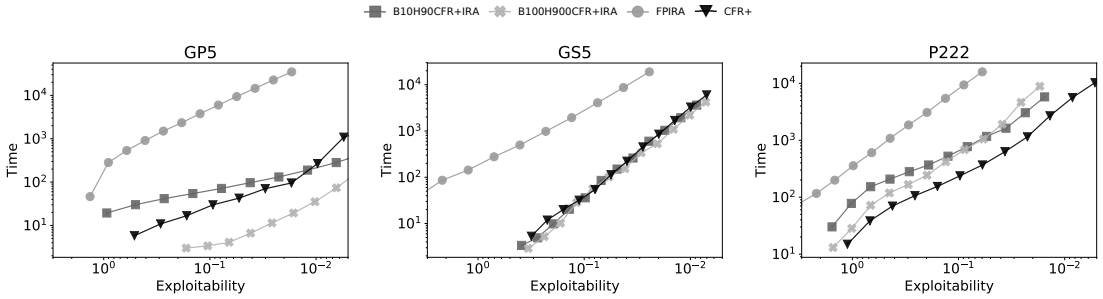

Figure 5.12: The plots showing runtime of FPIRA and CFR+IRA in seconds (log y-axis) required to reach the given sum of exploitabilities of resulting strategies of player 1 and 2 (log x-axis) for GP5, GS5, and P222 respectively.

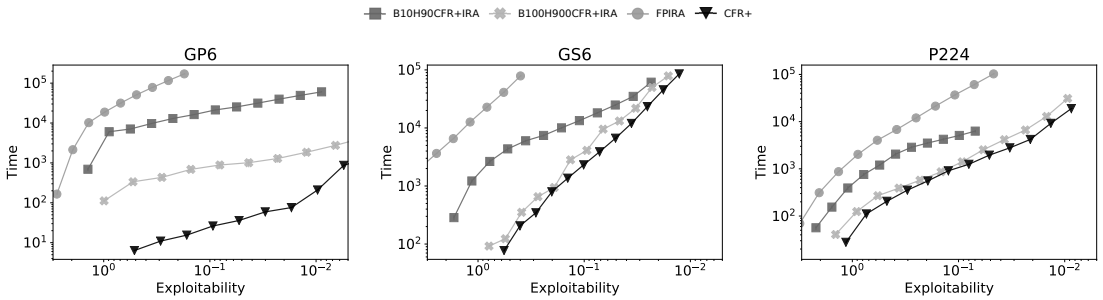

Figure 5.13: The plots showing runtime of FPIRA and CFR+IRA in seconds (log y-axis) required to reach the given sum of exploitabilities of resulting strategies of player 1 and 2 (log x-axis) for GP5, GS5 and P222 respectivelly.

CFR+IRA proves useful, as there are domains where the memory requirements of DOEFG are too large.

Chapter 6

Conclusion and Future Work

Extensive-form games are an important model of finite sequential strategic interaction between players. The size of the extensive-form representation is, however, often prohibitive and it is the most common issue preventing deployment of game-theoretic solution concepts to real-world scenarios. In this thesis, we focus on imperfect recall information abstractions with an aim to increase the scalability of solving extensive-form games. We provide the following contributions. First, we provide a complete picture of the complexity of solving imperfect recall abstractions. Second, we introduce two families of domain-independent algorithms capable of using imperfect recall abstractions to solve extensive-form games.

6.1 Thesis Contributions

6.1.1 Theoretical Properties of Imperfect Recall Abstractions

We provide a complete picture of the complexity of solving imperfect recall games. We demonstrate that the mixed strategy and behavioral strategy representation have different descriptive power in imperfect recall games and show that this difference leads to the loss of the guarantee of the existence of Nash equilibrium in behavioral strategies in imperfect recall extensive-form games. Furthermore, we discuss the numerical representation of Nash equilibrium and maxmin strategies in perfect and imperfect recall games. Next, we present known subsets of imperfect recall games: A-loss recall games and Chance relaxed skew well-formed games. We show that most of the hardness results known for imperfect recall games also extend to A-loss recall games. On the other hand, we provide sufficient and necessary (i.e., if and only if) condition for the existence of Nash equilibrium in A-loss recall games. This result makes A-loss recall games the only subset of imperfect recall games, where such conditions are known. Additionally, we show that A-loss recall property allows us to compute a best response in polynomial time (computing best response is NP-hard in imperfect recall games). Next, we discuss the problems of applying existing algorithms for solving perfect recall extensive-form games to imperfect recall games. Finally, we explain why Chance relaxed skew well-wormed games allow application of perfect recall algorithms and show the relation between A-loss recall games and Chance relaxed skew well-formed games.

6.1.2 Algorithms using Imperfect Recall Abstractions

From the algorithmic perspective, we introduce two families of algorithms.

The first family consists of domain-independent algorithms capable of approximating maxmin strategy in the given imperfect recall abstraction. We show that the algorithms are significantly more scalable in the case where we restrict the minimizing player to have a special case of imperfect recall called A-loss recall. We demonstrate that this approach allows us to significantly reduce the memory required to store the resulting strategy.

The second family consists of two domain-independent algorithms which can start with an arbitrary imperfect recall abstraction of the given two-player zero-sum EFG with perfect recall and then simultaneously refine and solve this abstraction until guaranteed convergence to the desired approximation of Nash equilibrium of the original unabstracted EFG. These algorithms have the following differences compared to the first class of algorithms. Both algorithms directly benefit from the reduced size of the solved game also during the computation and not only in the size of the resulting strategies. The algorithms can start from arbitrary imperfect recall abstraction, e.g., provided by domain experts. If no such abstraction is available for a given domain, the algorithms can start from a trivial coarse imperfect recall abstraction (we provide a domain-independent algorithm for constructing such initial abstraction). And finally, the choice of the initial abstraction does not influence the quality of the resulting strategies, since the abstraction is refined during the run of the algorithm to guarantee that the algorithms compute the desired approximation of the Nash equilibrium of the original game.

6.1.3 Implementation

All the algorithms and domains used in this thesis were implemented as a part of a publicly available library of game theoretic algorithms[1].

6.2 Future Work

The algorithms presented in this work open several possible directions for future work.

6.2.1 Better Automatically Built Initial Abstractions

Currently, we use a mapping of actions defined by the domain description when merging the information sets as a part of the construction of the initial abstraction for FPIRA and CFR+IRA (see Section 5.2.1.1). We believe that a mapping of actions, that takes into account their similarity, could greatly reduce the number of refinements of the initial abstraction performed by CFR+IRA and FPIRA to find the desired approximation of the Nash equilibrium of the original game. Such mapping could be based, e.g., on a fast heuristic search determining the similarity of the parts of the game reachable by taking the corresponding actions.

[1] http://jones.felk.cvut.cz/repo/gtlibrary

6.2.2 Domain Dependent Implementation of CFR+IRA

CFR based approaches are known to perform extremely well in poker domains [45, 55, 9]. This is caused by the specific structure of poker, where all the actions of the players are perfectly observable. Since as a part of CFR+IRA we use a domain-independent implementation of CFR+, we were not able to reach the full performance potential of CFR+IRA. We believe that domain-specific implementation of CFR+IRA, even outside of poker domain, would greatly improve its performance and would allow further significant scale-up of the algorithm.

6.2.3 Domain Specific Initial Abstractions for FPIRA and CFR+IRA

During the evaluation of CFR+IRA and FPIRA, we focused on automatically built initial abstractions. We believe that applying CFR+IRA to existing abstractions commonly used, e.g., in poker would greatly improve the convergence speed of CFR+IRA as these abstractions are built by domain experts and are known to perform well in practice. Applying CFR+IRA to these abstractions would have two benefits: (1) CFR+IRA would suggest further improvements to the abstractions in places where they are too coarse. (2) CFR+IRA would provide an arbitrarily precise approximation of the Nash equilibrium of the original domain when initialized by these abstractions, while the current solvers applied to these abstractions are limited by the quality of these abstractions. This, combined with the domain-specific implementation could lead to a significant improvement of the quality of the strategies currently used in these domains, as there are typically no guarantees that the used abstractions allow computation of sufficient approximation of the Nash equilibrium.

6.2.4 Parallelization of CFR+IRA

As shown in [6] the CFR based algorithms are suitable for massive parallelization. Hence the scalability of CFR+IRA could be further enhanced by an implementation which would allow such parallelization.

6.2.5 NP-Completeness of Computation of Maxmin Strategies and NE in A-loss Recall Games

We have left the question whether the problem stated in Theorem 4.2.2 belongs to NP as an open problem. Even though Theorem 3.2.1 states that the solution to this problem might require irrational numbers, it is not a sufficient argument for showing that it does not belong to NP. From this perspective, the problem from Theorem 4.2.2 is similar to *square-root sum* problem, since the square-root sum problem also requires operations with irrational numbers. However, deciding whether square-root sum problem belongs to NP is a major open problem [18] and there are known connections of square-root sum problem to other problems in game theory, e.g., computing Nash equilibrium in 3-player games [16]. It would be interesting to decide whether the problem from Theorem 4.2.2 belongs to NP or not, or at least find a reduction of the square-root sum problem to this problem.

Bibliography

[1] Nicola Basilico and Nicola Gatti. Automated abstractions for patrolling security games. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[2] Darse Billings, Neil Burch, Aaron Davidson, Robert Holte, Jonathan Schaeffer, Terence Schauenberg, and Duane Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 661–668, 2003.

[3] Branislav Bosansky, Jiri Cermak, Karel Horak, and Michal Pechoucek. Computing Maxmin Strategies in Extensive-Form Zero-Sum Games with Imperfect Recall. In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence*, 2017.

[4] Branislav Bošanský, Albert Xin Jiang, Milind Tambe, and Christopher Kiekintveld. Combining compact representation and incremental generation in large games with sequential strategies. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 812–818, 2015.

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

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

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

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

[9] Noam Brown and Tuomas Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, page eaao1733, 2017.

[10] Noam Brown and Tuomas W Sandholm. Simultaneous abstraction and equilibrium finding in games. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[11] J. Cermak, B. Bosansky, and Lisy V. An algorithm for constructing and solving imperfect recall abstractions of large extensive-form games. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017.

[12] Jiří Čermák, Branislav Bošanský, Karel Horák, Viliam Lisý, and Michal Pěchouček. Approximating maxmin strategies in imperfect recall games using a-loss recall property. *International Journal of Approximate Reasoning*, 93:290–326, 2018.

[13] Jiří Čermák, Branislav Bošanský, and Viliam Lisý. Constructing imperfect recall abstractions to solve large extensive-form games. *submitted to Artificial Intelligence Journal*, 2018.

[14] George Christodoulou, Annamária Kovács, and Michael Schapira. Bayesian combinatorial auctions. *Automata, Languages and Programming*, pages 820–832, 2008.

[15] Constantinos Daskalakis and Qinxuan Pan. A Counter-example to Karlin's Strong Conjecture for Fictitious Play. In *Annual Symposium on Foundations of Computer Science*, pages 11–20. IEEE, 2014.

[16] Kousha Etessami and Mihalis Yannakakis. On the complexity of nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010.

[17] Fei Fang, Thanh H Nguyen, Rob Pickles, Wai Y Lam, Gopalasamy R Clements, Bo An, Amandeep Singh, Brian C Schwedock, Milind Tambe, and Andrew Lemieux. Paws—a deployed game-theoretic application to combat poaching. *AI Magazine*, 2017.

[18] Michael R Garey, Ronald L Graham, and David S Johnson. Some np-complete geometric problems. In *Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 10–22. ACM, 1976.

[19] Andrew Gilpin and Tuomas Sandholm. A competitive texas hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1007. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[20] Andrew Gilpin and Tuomas Sandholm. Lossless abstraction of imperfect information games. *Journal of the ACM (JACM)*, 54(5):25, 2007.

[21] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of texas hold'em poker. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 50, 2007.

[22] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen. A heads-up no-limit texas hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 911–918. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[23] Geoffrey J Gordon. No-regret algorithms for online convex programs. In *Advances in Neural Information Processing Systems*, pages 489–496, 2006.

[24] Kristoffer Arnsfelt Hansen, Peter Bro Miltersen, and Troels Bjerre Sørensen. Finding Equilibria in Games of no Chance. In *Computing and Combinatorics*, pages 274–284. Springer, 2007.

[25] John Alexander Hawkin, Robert Holte, and Duane Szafron. Automated action abstraction of imperfect information extensive-form games. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[26] John Alexander Hawkin, Robert Holte, and Duane Szafron. Using sliding windows to generate action abstractions in extensive-form games. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[27] Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 805–813, 2015.

[28] 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.

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

[30] Michael Johanson, Kevin Waugh, Michael Bowling, and Martin Zinkevich. Accelerating best response calculation in large extensive games. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, volume 11, pages 258–265, 2011.

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

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

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

[34] Daphne Koller and Nimrod Megiddo. The Complexity of Two-person Zero-sum Games in Extensive Form. *Games and Economic Behavior*, 4:528–552, 1992.

[35] 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.

[36] Scott Kolodziej, Pedro M. Castro, and Ignacio E. Grossmann. Global optimization of bilinear programs with a multiparametric disaggregation technique. *Journal of Global Optimization*, 57(4):1039–1063, 2013.

[37] 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.

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

[39] Harold W. Kuhn. Extensive Games and the Problem of Information. *Annals of Mathematics Studies*, 1953.

[40] Marc Lanctot, Neil Burch, Martin Zinkevich, Michael Bowling, and Richard G Gibson. No-regret learning in extensive-form games with imperfect recall. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 65–72, 2012.

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

[42] Viliam Lisý, Trevor Davis, and Michael Bowling. Counterfactual Regret Minimization in Sequential Security Games. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2016.

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

[44] H. Brendan McMahan, Geoffrey J. Gordon, and Avrim Blum. Planning in the Presence of Cost Functions Controlled by an Adversary. In *Proceedings of the International Conference on Machine Learning*, pages 536–543, 2003.

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

[46] John F Nash et al. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

[47] Michele Piccione and Ariel Rubinstein. On the interpretation of decision problems with imperfect recall. *Games and Economic Behavior*, 20(1):3–24, 1997.

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

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

[50] Jonathan Rubin and Ian Watson. Computer poker: A review. *Artificial intelligence*, 175(5-6):958–987, 2011.

[51] Tuomas Sandholm. Steering evolution strategically: computational game theory and opponent exploitation for treatment planning, drug design, and synthetic biology. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 4057–4061. AAAI Press, 2015.

[52] Tuomas Sandholm and Satinder Singh. Lossy stochastic game abstraction with bounds. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 880–897. ACM, 2012.

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

[54] Oskari Tammelin. Cfr+. *CoRR, abs/1407.5042*, 2014.

[55] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. Solving heads-up limit texas hold'em. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, pages 645–652, 2015.

[56] Jiří Čermák and Branislav Bošanský. Towards Solving Imperfect Recall Games. In *Proceedings of AAAI Computer Poker Workshop*, 2017.

[57] Jiří Čermák, Branislav Bošanský, and Michal Pěchouček. Combining Incremental Strategy Generation and Branch and Bound Search for Computing Maxmin Strategies in Imperfect Recall Games. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2017.

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

[59] Kevin Waugh, Martin Zinkevich, Michael Johanson, Morgan Kan, David Schnizlein, and Michael H Bowling. A Practical Use of Imperfect Recall. In *SARA*. Citeseer, 2009.

[60] Philipp C Wichardt. Existence of Nash Equilibria in Finite Extensive Form Games with Imperfect Recall: A Counterexample. *Games and Economic Behavior*, 63(1):366–369, 2008.

[61] Martin Zinkevich, Michael Johanson, Michael H Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems*, pages 1729–1736, 2007.

Appendix A

Supplementary Material for Chapter 3

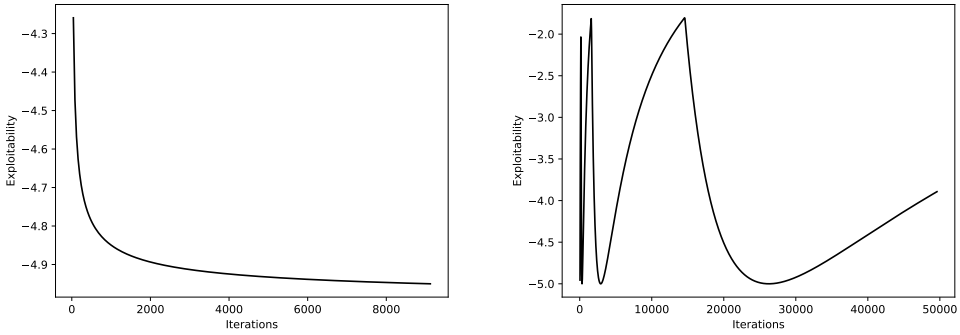

Figure A.1: The expected value of the average strategy of player 1 against the best response tof player 2 (y-axis) as a function of iterations (x-axis) for CFR (left) and FP (right).

In this Chapter we demonstrate the performance of FP and CFR applied to the imperfect recall game from Figure 3.5 where we set $x = 10$. The game has A-loss recall and has 2 NE, playing (a, g) or (b, h) deterministically for player 1 and (c, f) for player 2 (no mix between these two NE strategies for player 1 is a NE).

A.1 Experimental Evaluation of Strategies Computed by FP in Imperfect Recall Game

Here, we empirically demonstrate the performance of FP on the example game from Figure 3.5 when we set $x = 10$.. Figure A.1 (right) depicts the expected value of the average strategy of player 1 against the best response of player 2 computed by the FP (x-axis shows the number of iterations, the y-axis shows the expected value). The FP does not

converge to any fixed strategy. Moreover, the exploitability of the strategy in every iteration significantly differs from the maxmin value -1 for player 1 achievable, e.g., by playing a and g deterministically).

A.2 Experimental Evaluation of Strategies Computed by CFR in Imperfect Recall Game

Here, we empirically demonstrate the performance of CFR on the example game from Figure 3.5 when we set $x = 10$. Figure A.1 (left) depicts the expected value of the average strategy of player 1 computed by the CFR against the best response of player 2 (x-axis shows the number of iterations, the y-axis shows the expected value). The algorithm converges to the average strategy with exploitability -5, while the maxmin value of player 1 is -1. Notice that by increasing x, the exploitability of the resulting strategy will further grow, while the maxmin value remains -1 (achievable, e.g., by playing a and g deterministically).

Appendix B

Publications

B.1 Publications Related to the Thesis

B.1.1 Journal Publications (with IF)

- J. Čermák, B. Bošanský, K. Horák, V. Lisý, M. Pěchouček. Approximating maxmin strategies in imperfect recall games using A-loss recall property. International Journal of Approximate Reasoning (2018) (55%)

- J. Čermák, B. Bošanský, V. Lisý. Constructing imperfect recall abstractions to solve large extensive-form games. submitted to Artificial Intelligence Journal (2018) (70%)

B.1.2 Conference Publications

- J. Čermák, B. Bošanský, V. Lisý. An Algorithm for Constructing and Solving Imperfect Recall Abstractions of Large Extensive-Form Games. Proceedings of 26th International Joint Conference on Artificial Intelligence (IJCAI 2017) (50%)

- J. Čermák, B. Bošanský, M. Pěchouček. Combining incremental strategy generation and branch and bound search for computing maxmin strategies in imperfect recall games. Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2017) (55%)

- J. Čermák. Solving Imperfect Recall Games. Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2017) (100%)

- B. Bošanský, J. Čermák, K. Horák, M. Pěchouček. Computing Maxmin Strategies in Extensive-Form Zero-Sum Games with Imperfect Recall. Proceedings of International Conference on Agents and Artificial Intelligence (ICAART 2017) (30%)

- J. Čermák, B. Bošanský. Towards Solving Imperfect Recall Games. Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI 2017) (60%)

B.2 Other Publications

B.2.1 Journal Publications (with IF)

- B. Bošanský, V. Lisý, M. Lanctot, J. Čermák, M. H. M. Winands. Algorithms for computing strategies in two-player simultaneous move games. Artificial Intelligence 2016

B.2.2 Conference Publications

- J. Čermák, B. Bošanský, K. Durkota, V. Lisý, C. Kiekintveld. Using Correlated Strategies for Computing Stackelberg Equilibria in Extensive-form Games. Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI 2016)

- B. Bošanský, J. Čermák. Sequence-Form Algorithm for Computing Stackelberg Equilibria in Extensive-Form Games. Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)

- J. Čermák, B. Bošanský, N. Gatti. Effectiveness of Game-Theoretic Strategies in Extensive-Form General-Sum Games. Proceedings of the 15th Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)

- J. Čermák, B. Bošanský, V. Lisý. Practical Performance of Refinements of Nash Equilibria in Extensive-Form Zero-Sum Games. Proceedings of The 21st European Conference on Artificial Intelligence (ECAI 2014)

- B. Bošanský, C. Kiekintveld, V. Lisý, J. Čermák, M. Pěchouček. Double-Oracle Algorithm for Computing an Exact Nash Equilibrium in Zero-Sum Extensive-Form Games. Proceedings of the 12th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2013)

- B. Bošanský, V. Lisý, J. Čermák, R. Vítek, M. Pěchouček. Using Double-Oracle Method and Serialized Alpha-Beta Search for Pruning in Simultaneous Move Games. Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)

B.3 Responses

Following is a list of publications with at least 3 citations based on Google Scholar as of February 2018 (excluding self-citations).

- B. Bošanský, C. Kiekintveld, V. Lisý, J. Čermák, M. Pěchouček. Double-Oracle Algorithm for Computing an Exact Nash Equilibrium in Zero-Sum Extensive-Form Games. Proceedings of the 12th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2013) (7 citations)

- B. Bošanský, V. Lisý, J. Čermák, R. Vítek, M. Pěchouček. Using Double-Oracle Method and Serialized Alpha-Beta Search for Pruning in Simultaneous Move Games. Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI 2013) (4 citations)
- B. Bošanský, V. Lisý, M. Lanctot, J. Čermák, M. H. M. Winands. Algorithms for computing strategies in two-player simultaneous move games. Artificial Intelligence (2016) (3 citations)
- J. Čermák, B. Bošanský, V. Lisý. Practical Performance of Refinements of Nash Equilibria in Extensive-Form Zero-Sum Games. Proceedings of The 21st European Conference on Artificial Intelligence (ECAI 2014) (3 citations)