arXiv:1803.05392v1 [cs.GT] 14 Mar 2018

Constructing Imperfect Recall Abstractions to Solve Large Extensive-Form Games

Jiří Čermák, Viliam Lisý*, Branislav Bošanský

Artificial Intelligence Center, Department of Computer Science, Czech Technical University in Prague

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 the player (and hence also the size of his strategy) to grow exponentially with the number of actions taken in the past. On the other hand, relaxing the perfect recall requirement (resulting in so-called imperfect recall abstractions) can significantly increase the computational complexity of solving the resulting abstracted game. Hence, there is only a limited amount of work that focuses on using imperfect recall abstractions. These approaches either use computationally complex algorithms to solve the abstracted game or restrict to trivial subclasses of imperfect recall abstractions that are easy to solve. 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 FPIRA and CFR+IRA which are able to start with an arbitrary imperfect recall abstraction of the solved

*Corresponding author

Email addresses: cermak@agents.fel.cvut.cz (Jiří Čermák), lisy@agents.fel.cvut.cz (Viliam Lisý), bosansky@agents.fel.cvut.cz (Branislav Bošanský)

two-player zero-sum perfect recall extensive-form game. 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. We experimentally demonstrate that even when the algorithms start with trivial coarse imperfect recall abstraction, they are capable of approximating Nash equilibrium of large extensive-form games using abstraction with as little as 0.9% of information sets of the original game. Moreover, the results suggest that the relative memory requirements of the algorithms will further decrease as the size of the solved game increases.

Keywords: Extensive-Form Games, Information Abstraction, Imperfect Recall, Nash Equilibrium, Fictitious Play, Counterfactual Regret Minimization

---

1. Introduction

Dynamic games with a finite number of moves can be modeled as extensive-form games (EFGs) that are capable of describing scenarios with stochastic events and imperfect information. EFGs can model recreational games, such as poker [1], as well as real-world situations in physical security [2], auctions [3], or medicine [4]. EFGs are represented 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 points of the players.

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 [6], 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., [7]) without the need of large computational resources necessary when using online algorithms. We thus focus on offline algorithms.

Most of the existing offline algorithms [8, 9, 10] 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 [11], even storing strategy for a game of this size requires prohibitive memory. Therefore, a popular approach is to use *information abstractions* [12] 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 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 algorithms (e.g., see [13, 14, 15, 16]) create perfect recall abstractions. However, the requirement of perfect recall in the abstraction severely limits possible memory savings, since it still forces players to remember all the moves they made in the past; hence 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 [17, 18]. 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* [19, 20] and *normal-form games with sequential strategies* [21, 2]. 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. In normal-form games with sequential strategies, players cannot observe actions of the opponent at all. 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. Existing methods for using imperfect recall abstractions without severe limitations cannot provide any guarantees of the quality of computed strategies [22], or assume that the abstraction is given and require computationally complex approximation of a bilinear program to solve it [23].

1.1. Our Contribution

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)[1]. FPIRA is based on the Fictitious Play (FP, [25]). 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 CFR+ for Imperfect Recall Abstractions (CFR+IRA). CFR+IRA replaces the FP by CFR+ algorithm [26] 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 provide a bound on the average external regret of CFR+IRA and hence show 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, [27]) since they create a smaller version of the original game and repeatedly refine it until the desired approximation of the Nash equilibrium of

[1] A part of this work appeared in [24]. Here, we provide an improved version of the FPIRA which can use a significantly smaller initial abstraction. Additionally, we significantly extend the experimental evaluation of the algorithm.

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.

1.2. Related Work on Abstractions

Here, we provide an overview of the related work concerning the use of abstractions to solve large EFGs. 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 [28, 29]. Algorithms for creating information abstractions followed [30, 12] 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 [13]. 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 [14]. Additionally, both lossless and lossy imperfect recall abstractions were provided in [19, 20]. The authors show that running the CFR algorithm on this class of imperfect recall abstractions leads to a bounded regret in the full game. In this work, we also focus on imperfect recall abstractions (see the next section for the comparison of our approach to the state-of-the-art).

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 [31, 32]. The algorithm for simultaneous action abstraction finding and game solving was introduced in [15]. This approach is similar to the algorithms presented in this work since it starts with a coarse abstraction and iteratively refines it until guaranteed convergence to the Nash equilibrium of the solved game (see the next section for the comparison of our approach to the state-of-the-art).

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., [33]). Furthermore, the algorithm Libratus [16] using both mentioned abstractions led to a superhuman performance in heads-up no-limit Texas Hold'em.

In other domains, abstractions with bounded error were introduced in Patrolling Security Games [34]. Additionally, the first general framework for lossy game abstraction with bounds was provided for stochastic games [35].

1.3. Comparison of our Contribution to the Related Work

There are several advantages of our algorithms compared to the above mentioned related work.

(1) Both algorithms are domain independent, while the majority of the work mentioned above 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 abstractions 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

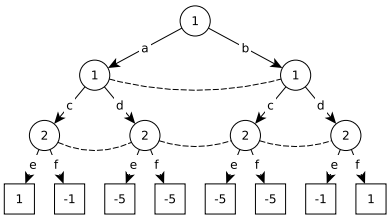


Figure 1: An imperfect recall game. Circle nodes represent the states of the game, labels in the circles show which player acts in that node (player 1, player 2 or chance player N), dashed lines represent indistinguishable states and box nodes are the terminal states with utility value for player 1 (the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

introduced in [15]. 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 we empirically demonstrate that both algorithms perform well on a set of domains with a diverse structure.

2. Extensive-Form Games

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

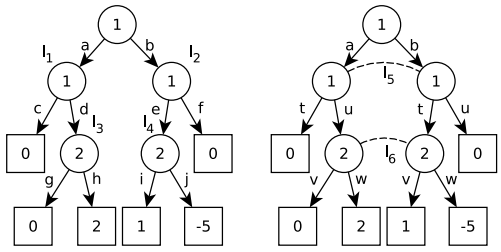


Figure 2: (a) Extensive-form game G. (b) Imperfect recall infromation abstraction of G.

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 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. A game has *perfect recall* iff $\forall i \in \mathcal{N}$ $\forall I \in \mathcal{I}_i$, for all $h, h' \in I$ 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*.

2.1. Information Abstraction

By *information abstraction* of a game $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, u, \mathcal{I}, \mathcal{A})$ we denote a game $G^x = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, u, \mathcal{I}^x, \mathcal{A}^x)$ (when iteratively building the abstraction, we willl use x to refer to the abstraction in a specific iteration). G^x differs from G in the structure of information sets and hence also in the action labeling. Each $I \in \mathcal{I}^x$ groups one or more information sets in $\mathcal{I}$. $\mathcal{A}^x$ changes $\mathcal{A}$ so that each action a again uniquely identifies the information set in $\mathcal{I}^x$ where it is available. Furthermore, $\forall I \in \mathcal{I}^x \forall h, h' \in I$ $\mathcal{A}^x(h) = \mathcal{A}^x(h')$.

To formally describe the information abstraction, 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 1. *In Figure 2(a) we show an extensive-form game G and in Figure 2(b) its imperfect recall abstraction G^x. In this case,*

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

2.2. Strategies in Imperfect Recall Games

There are several representations of strategies in EFGs. A *pure strategy* s_i for player i is a mapping assigning $\forall I \in \mathcal{I}_i$ an element of $\mathcal{A}(I)$. $\mathcal{S}_i$ 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 $\mathcal{A}(I)$ for each $I \in \mathcal{I}_i$. $\mathcal{B}_i$ 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.

Definition 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.

Behavioral strategies and mixed strategies have the same expressive power in perfect recall games, but it can differ in imperfect recall games [36] (see [23] for more detailed discussion).

Moreover, the size of these representations differs significantly. Mixed strategies of player i state probability distribution over $\mathcal{S}_i$, where $|\mathcal{S}_i| \in \mathcal{O}(2^{|\mathcal{Z}|})$, behavioral strategies create probability distribution over the set of actions (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.

Finally, we define the Nash equilibrium, ε-Nash equilibrium and the exploitability of a strategy.

Definition 2. *We say that strategy profile $b = \{b_i^*, b_{-i}^*\}$ is a* Nash equilibrium *(NE) in behavioral strategies iff $\forall i \in \mathcal{N}\ \forall b_i' \in \mathcal{B}_i : u_i(b_i^*, b_{-i}^*) \geq u_i(b_i', b_{-i}^*)$.*

Informally, a strategy profile is a NE if and only if no player wants to deviate to a different strategy.

Definition 3. *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 a ε-NE if and only if no player can gain more than ε by deviating to a different strategy.

Definition 4. *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.

3. Iterative Algorithms for Solving EFGs

In this section we describe the Fictitious Play (FP, [25]), we follow with the description of ideas behind external regret, Counterfactual Regret Minimization (CFR, [9]) and its variant CFR+ [26, 37].

3.1. Fictitious Play

Fictitious play (FP) is an iterative algorithm originaly defined for normal-form games [25]. It keeps track of average mixed strategies $\bar{m}_i^T, \bar{m}_{-i}^T$ of both players. 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 [38]. Furthermore, there is a long-standing conjecture [39, 40] 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 [41].

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

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

defines a behavioral strategy b_i'' *realization equivalent to the mixed strategy* $m_i'' = \lambda_1 m_i + \lambda_2 m_i'$.

3.2. External Regret

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

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

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_i^t in every t. An algorithm is a no-regret algorithm for player i, if the average external regret approaches zero; i.e.,

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

3.3. Counterfactual Regret Minimization

Let $b_{I \to a}^t$ 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 $\pi_i^b(h) = \prod_{a \in seq_i(h)} b(a)$ being the contribution of player i and $\pi_{-i,c}^b(h) = \mathcal{C}(h) \prod_{a \in seq_{-i}(h)} b(a)$ 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 information set I when players play according to the strategy profile b be

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

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

$$R_i^T(I, a) = \sum_{t=1}^{T} \left[v_i(b_{I \to a}^t, I) - v_i(b^t, I) \right]. \tag{2}$$

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

$$b_i^T(I, a) = \begin{cases} \frac{\left(R_i^{T-1}(I,a)\right)^+}{\sum_{a' \in \mathcal{A}(I)} \left(R_i^{T-1}(I,a')\right)^+}, \text{ if } \sum_{a' \in \mathcal{A}(I)} \left(R_i^{T-1}(I,a')\right)^+ > 0 \\ \frac{1}{\mathcal{A}(I)}, \text{ otherwise.} \end{cases} \tag{3}$$

The immediate counterfactual regret is defined as

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

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

$$\frac{R^T_i}{T} \leq \sum_{I \in \mathcal{I}_i} \left(R^T_{i,imm}(I) \right)^+, \tag{4}$$

i.e., the external regret is bounded by the sum of positive parts of immediate counterfactual regrets [9].

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) in I during iterations {1, ..., T}, then

$$R^T_{i,imm}(I) \leq \frac{\Delta_I \sqrt{|\mathcal{A}(I)|}}{\sqrt{T}}. \tag{5}$$

Thus, from eq. (4)

$$\frac{R^T_i}{T} \leq \frac{\Delta |\mathcal{I}_i| \sqrt{|A_{max}|}}{\sqrt{T}}, \tag{6}$$

where $|A_{max}| = \max_{I \in \mathcal{I}_i} |\mathcal{A}(I)|$ and $\Delta = 2u_{max}$ [9].

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

$$\bar{b}^T_i(I, a) = \frac{\sum_{t=1}^T \pi_i^{b^t_i}(I) b^t_i(I, a)}{\sum_{t=1}^T \pi_i^{b^t_i}(I)}, \quad \forall I \in \mathcal{I}_i, \forall a \in \mathcal{A}(I). \tag{7}$$

If $\frac{R^T_i}{T} < \varepsilon$ for all $i \in \mathcal{N}$ in a two-player zero-sum EFG, the strategy profile $\bar{b}^T = (\bar{b}^T_1, \bar{b}^T_2)$ forms a 2ε-Nash equilibrium [9].

3.4. CFR+

The CFR+ [26, 37] replaces the regret-matching shown in equation (3) with *regret-matching*$^+$. To do that, the algorithm maintains alternative regret values $Q^T_i(I, a)$ in each iteration T and for each player i defined as

$$Q^0_i(I, a) = 0 \tag{8}$$

$$Q^T_i(I, a) = \left(Q^{T-1}_i(I, a) + R^T_i(I, a) - R^{T-1}_i(I, a) \right)^+. \tag{9}$$

The update of $Q^T_i(I, a)$ is identical to the update of $R^T_i(I, a)$ in eq. (2) 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

$$b^T_i(I, a) = \begin{cases} \frac{Q^{T-1}_i(I,a)}{\sum_{a' \in \mathcal{A}(I)} Q^{T-1}_i(I,a')}, \text{ if } \sum_{a' \in \mathcal{A}(I)} Q^{T-1}_i(I, a') > 0 \\ \frac{1}{\mathcal{A}(I)}, \text{ otherwise,} \end{cases} \tag{10}$$

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 [37], 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 $T > d$

$$\bar{b}_i^T(I,a) = \frac{2\sum_{t=d}^{T} t \cdot \pi_i^{b_i^t}(I) \cdot b_i^t(I,a)}{((T-d)^2 + T - d)\sum_{t=d}^{T} \pi_i^{b_i^t}(I)}, \quad \forall I \in \mathcal{I}_i, \forall a \in \mathcal{A}(I). \tag{11}$$

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. (6) 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 [37]. Additionally, the empirical convergence of CFR+ is significantly better compared to CFR [37].

4. Algorithms for Constructing and Solving Imperfect Recall Abstractions

In this section, we present the main algorithmic results of this paper. 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 4.2 we present the FP based approach denoted as Fictitious Play for Imperfect Recall Abstractions (FPIRA). In Section 4.3 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.

4.1. Abstraction

As discussed before, the algorithms presented in this section can start from an arbitrary initial imperfect recall abstraction. In Section 4.1.1, 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 $G^t = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, u, \mathcal{I}^t, \mathcal{A}^t)$.

4.1.1. 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.).

4.2. The FPIRA Algorithm

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 4.1.1. 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 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.

In Algorithm 1 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 4.1.1. 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 4.2.1). 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 4.2.2, Case

Algorithm 1: 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 $\quad t \leftarrow t + 1$

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 4.2.2 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 4.2.2, 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.

4.2.1. Best Response Computation

In Algorithm 2 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

Algorithm 2: 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
```

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 [27], Section 4.2.

4.2.2. 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 2 for more details).

Algorithm 3: Abstraction update to accomodate best response.

```
1  function RefineForBR(G^t, b_i^t, σ̄_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                     σ̄_i^t(I') ← σ̄_i^t(I)
21                 σ̄_i^t ← σ̄_i^t \ σ̄_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
```

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 3 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).

Algorithm 4: 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
```

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 $-i$.

More formally, the algorithm first constructs the average behavioral strategy $\hat{b}_i^t$ in G^t (line 8 in Algorithm 1). This is done according to Lemma 1 from $\bar{b}_i^{t-1}$ with weight $\frac{t_i-1}{t_i}$ and b_i^t with weight $\frac{1}{t_i}$, where t_i is the number of updates performed by i so far, plus 1 for the initial strategy (b_i^t is used with mappings Φ_t and Ξ_t). Next, FPIRA constructs $\widetilde{b}_i^t$ (line 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 1). 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 4 in the 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 4.2.3, 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

$$u'_{-i}(z) = \mathcal{C}(z) \left[\pi_i^{\widetilde{b}_i^t}(z) - \pi_i^{\hat{b}_i^t}(z) \right] u_{-i}(z), \quad \forall z \in \mathcal{Z} \tag{12}$$

$$u''_{-i}(z) = -\mathcal{C}(z) \left[\pi_i^{\widetilde{b}_i^t}(z) - \pi_i^{\hat{b}_i^t}(z) \right] u_{-i}(z), \quad \forall z \in \mathcal{Z}. \tag{13}$$

Δ_i^t can be computed in $\mathcal{O}(|\mathcal{Z}|)$ by ComputeDelta depicted in Algorithm 5. The computation consists of two calls of function ComputePartialDelta similar to the computation of the best response described in Section 4.2.1. The only difference is the use of functions shown in eqs. (12) and (13) 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 2. *Let us demonstrate several iterations of* FPIRA *algorithm. Consider the EFG from Figure 3(a) as G and the imperfect recall game from Figure*

Algorithm 5: Δ_i^t computation.

function `ComputeDelta`(G, $\hat{b}_i^t$, $\widetilde{b}_i^t$)
  Let u'_{-i} be the function from eq. (12)
  Let u''_{-i} be the function from eq. (13)
  return max(`ComputePartialDelta`($G.root$, u'_{-i}, $\hat{b}_i^t$, $\widetilde{b}_i^t$), `ComputePartialDelta`($G.root$, u''_{-i}, $\hat{b}_i^t$, $\widetilde{b}_i^t$))
function `ComputePartialDelta`(h, u'_{-i}, $\hat{b}_i^t$, $\widetilde{b}_i^t$)
  if $\pi_i^{\hat{b}_i^t}(h) = 0 \wedge \pi_i^{\widetilde{b}_i^t}(h) = 0$ **then**
    return 0
  if `IsGameEnd`(h) **then**
    return $u'_{-i}(h)$
  $v \leftarrow$ `GetValueFromCache`(h)
  if $v \neq null$ **then**
    return v
  if `GetPlayerToMove`(h) $= -i$ **then**
    $I \leftarrow$ `GetInformationSetFor`($h, \mathcal{I}$)
    $a_{max} \leftarrow \arg\max_{a \in \mathcal{A}(I)} \sum_{h' \in I}$ `ComputePartialDelta`($h' \cdot a$, u'_{-i}, $\hat{b}_i^t$, $\widetilde{b}_i^t$)
    For each $h' \in I$ store $v_{h'} =$ `BR`($h' \cdot a_{max}$, G, $\bar{b}_{-i}$, b_i^{BR}) to cache
    $v \leftarrow$ `ComputePartialDelta`($h \cdot a_{max}$, u'_{-i}, $\hat{b}_i^t$, $\widetilde{b}_i^t$)
  else
    $v \leftarrow \sum_{a \in \mathcal{A}(h)}$`ComputePartialDelta`($h \cdot a$, u'_{-i}, $\hat{b}_i^t$, $\widetilde{b}_i^t$)
  return v

3 (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 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, $\widetilde{b}_1^3(a) =$*

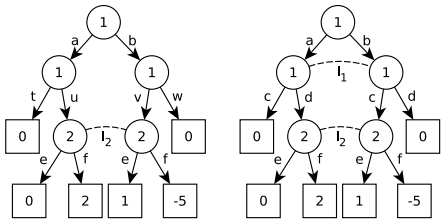


Figure 3: (a) G for demonstration of FPIRA iterations (b) G^1 for demonstration of FPIRA.

$\frac{1}{3}$, $\widetilde{b}_1^3(u) = 1, \widetilde{b}_1^3(w) = 0.5$ *(both according to Lemma 1 with* $\lambda_1 = \frac{2}{3}, \lambda_2 = \frac{1}{3}$*). In this case,* $\Delta_1^3 = \frac{1}{3}$ *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 4.2.2, Case 2, resulting in* G*. Therefore,* $G^4 = G$*,* $\bar{b}_1^3 = \widetilde{b}_1^3$ *and* Ξ_4 *is set to identity.*

4.2.3. Theoretical Properties

Here, we show that the convergence guarantees of FP in two-player zero-sum perfect recall game G [41] directly apply to FPIRA solving G.

Theorem 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

$$\forall b_{-i} \in \mathcal{B}_{-i}^p : u_{-i}(\bar{b}_i^t, b_{-i}) = u_{-i}(\bar{b'}_i^t, b_{-i}), \tag{14}$$

$$\forall b_i \in \mathcal{B}_i^p : u_i(b_i, \bar{b}_{-i}^t) = u_i(b_i, \bar{b'}_{-i}^t), \tag{15}$$

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

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

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'^t_i be the best response chosen in t in FP. From (15) and the use of the same tie breaking rule we know that $b_i^t = b'^t_i$. From Lemma 1 we know that

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

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 (14) and from the equality $b_i^t = b'^t_i$ 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

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

□

4.2.4. 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 4.1.1, 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 4.2.2, 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 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 4.2.2, 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.

4.2.5. 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 4.2.4.

When computing the best response (see Section 4.2.1), 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 2) and due to $\overline{b}_{-i}^t$ (lines 6 and 7 in Algorithm 2). 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. 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 [42]) 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.

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).

4.3. 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 4.1.1. 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 6: 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

```
1  G^1 ← InitAbstraction(G)
2  r ← InitRegrets(I^1)
3  b̄_1 ← UniformStrategy(I), b̄_2 ← UniformStrategy(I)
4  t ← 1, t_next ← 1, t_last ← 0, j ← 0
5  r_b ← ∅
6  while u_1(BR(G, b̄_2), b̄_2) − u_1(b̄_1, BR(G, b̄_1)) > ε do
7      Î_h ← SampleInformationSets(Ĩ_i^t, k_h)
8      r_h ← InitRegrets(Î_h)
9      if t = t_next then
10         Î_b ← SampleInformationSets(Ĩ^t, k_b)
11         r_b ← InitRegrets(Î_b)
12         t_last ← t
13         t_next ← t + 2^j
14         j ← j + 1
15     ComputeRegrets(I^t, G^t.root, GetPlayer(t), 1, 1, r, r_h, r_b, Î_h, Î_b)
16     RemoveNegativeRegrets(r)
17     if t > delay then
18         UpdateAverageStrategy(r, t)
19     G^{t+1} ← UpdateAbstractionForHeuristic(G^t, Î_h, r_h, r, b̄_1, b̄_2, t)
20     delete r_h
21     if t ≠ t_last then
22         G^{t+1} ← UpdateAbstractionForBound(G^{t+1}, Î_b, r, r_b, b̄_1, b̄_2, t, t_last)
23     t ← t + 1
```

In Algorithm 6 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 4.1.1). 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 and Algorithm 7). 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 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. (11).

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 4.3.1 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 4.3.2 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.

4.3.1. 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 6). 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 6 according to Algorithm 8. 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

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

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 8 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

$$\frac{1}{\sum_{I' \in \widetilde{\mathcal{I}}^t} |\Phi_t^{-1}(I')|}. \tag{16}$$

Algorithm 7: Regret update

```
1  function ComputeRegrets(I, h, i, π1, π2, r, rh, rb, 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,
                   rh, rb, Î_h, Î_b)
7          return Σ_{a∈A(h)} b_c(a)· ComputeRegrets(I, h·a, i, b_c(a)·π1, π2, r, rh,
                 rb, 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, rh, rb, Î_h, Î_b)
16         else
17             v[a] ← ComputeRegrets(I, h·a, i, π1, b^t(a)·π2, r, rh, rb, 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_I^h ← GetRegretsFor(I, rh)
24             for a ∈ A(h) do
25                 r_I^h[a] ← r_I^h[a] + π_{-i}·(v[a] − v_{b^t})
26         if I ∈ Î_b then
27             r_I^b ← GetRegretsFor(I, rb)
28             for a ∈ A(h) do
29                 r_I^b[a] ← r_I^b[a] + π_{-i}·(v[a] − v_{b^t})
30     return v_{b^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 7) 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 9 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

Algorithm 8: Sampling of unabstracted sets for unabstracted regret storage.

```
function SampleInformationSets(Ĩ, k)
    Î ← ∅
    while |Î| < k do
        if AllSetsSampled(Ĩ, Î) then
            break
        I ← GetRandomAbstractedSet(Ĩ)
        I'' ← Φ_t^{-1}(I)
        if |I''| + |Î| ≤ k then
            Î ← Î ∪ I''
        else
            Î ← Î ∪ random subset of I'' with size k − |Î|
    return Î
```

Algorithm 9: Abstraction update for regret bound

```
function UpdateAbstractionForBound(G^t, Î_b, r, r_b, b̄_1, b̄_2, t, t_last)
    for I ∈ Ĩ^t do
        I'' ← Φ_t^{-1}(I)
        for I'' ∈ Intersection(I'', Î_b) do
            r^b_{I''} ← GetRegretsFor(I'', r_b)
            if (max_{a∈A(I'')} r^b_{I''}[a]) / (t − t_last) > L^{t_last}_{I''}(t) then
                I_new ← CreateNewIS(I'' \ I'')
                G^t.I ← G^t.I \ I
                G^t.I ← G^t.I ∪ {CreateNewIS(I''), I_new}
                r_I ← GetRegretsFor(I, r)
                r ← r \ r_I
                r_{I_new} ← InitRegret(I_new)
                r ← r ∪ r_{I_new}
                r_{I''} ← InitRegret(I'')
                r ← r ∪ r_{I''}
                Î_b ← Î_b \ I''
                r_b ← r_b \ r^b_{I''}
                if GetPlayer(t) = 1 then
                    b̄_1 ← b̄_1 \ GetStrategyFor(I, b̄_1)
                else
                    b̄_2 ← b̄_2 \ GetStrategyFor(I, b̄_2)
    return G^t
```

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

If $R^{T'}_{T_j,imm}(I) > L^{T_j}_I(T')$ for some $T' \in \{T_j+1, ..., T_{j+1}\}$ (line 6 in Algorithm 9), 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 9). If I is disconnected, the average strategy in I in all $T > T'$ is computed $\forall a \in \mathcal{A}(I)$ as

$$\bar{b}_i^T(I,a) = \begin{cases} \frac{1}{|\mathcal{A}(I)|}, & \text{if } T = T'+1, \\ \frac{2\sum_{t'=T'+1}^{T} t' \cdot \pi_i^{b_i^{t'}}(I) \cdot b_i^{t'}(I,a)}{\left((T-T'-1)^2+T-T'-1\right)\sum_{t'=T'+1}^{T} \pi_i^{b_i^{t'}}(I)}, & \text{otherwise.} \end{cases} \tag{17}$$

The average strategy update in eq. (17) corresponds to the average strategy update descibed in Section 3.4, i.e., it does a weighted average of the b_i^t strategies over iterations $\{T'+1, ..., T\}$ with weight corresponding to the iteration.

4.3.2. Heuristic Update

Algorithm 10: Abstraction update for heuristic

```
1  function UpdateAbstractionForHeuristic(G^t, Î_h, r_h, r, b̄_1, b̄_2, t)
2      for I ∈ Ĩ_i^t do
3          I'' ← Φ_t^{-1}(I)
4          if Intersection(I'', Î_h) ≠ ∅ then
5              for I'' ∈ Intersection(I'', Î_h) do
6                  r^h_{I''} ← GetRegretsFor(I'', r_h)
7                  γ_{I''} ← set of indices of {a ∈ A(I'') | r^h_{I''}[a] ∈
                       [max_{a'∈A(I'')} (r^h_{I''}[a'] − 1/(5√t), max_{a'∈A(I'')} r^h_{I''}[a'])]}
8              Split I'' to subsets I''_1, ..., I''_k
9              Each I''_j ∈ {I''_1, ..., I''_k} contains all the I'' ∈ I'' with the same γ_{I''}
10             I''_max ← arg max_{I''_j ∈ {I''_1,...,I''_k}} |I''_j|
11             I''_f ← I''_max ∪ I'' \ Î_h
12             G^t.I ← G^t.I \ I
13             r ← r \ GetRegretsFor(I, r)
14             for I''_j ∈ ({I''_1, ..., I''_k} ∪ I''_f) \ I''_max do
15                 I_new ← CreateNewIS(I''_j)
16                 G^t.I ← G^t.I ∪ I_new
17                 r_{I_new} ← InitRegret(I_new)
18                 r ← r ∪ r_{I_new}
19             if GetPlayer(t) = 1 then
20                 b̄_1 ← b̄_1 \ GetStrategyFor(I, b̄_1)
21             else
22                 b̄_2 ← b̄_2 \ GetStrategyFor(I, b̄_2)
23     return G^t
```

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 6 according to Algorithm 8. In this case, the sampling in Algorithm 8 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 8 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

$$\frac{1}{\sum_{I' \in \widetilde{\mathcal{I}}_i} |\Phi_t^{-1}(I')|}. \tag{18}$$

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 7).

The actual abstraction update is done in a following way (Algorithm 10). 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 $\mathcal{I}_n^t$ created from $\left(\{\mathcal{I}''_1, ..., \mathcal{I}''_k\} \cup \mathcal{I}''_f\right) \setminus \mathcal{I}''_{max}$. The regrets in each $I_n \in \mathcal{I}_n^t$ 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 $I_n \in \mathcal{I}_n^t$ 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

$$\bar{b}_i^T(I, a) = \begin{cases} \frac{1}{|\mathcal{A}(I)|}, & \text{if } T = t+1, \\ \frac{2\sum_{t'=t+1}^{T} t' \cdot \pi_i^{b_i^{t'}}(I) \cdot b_i^{t'}(I,a)}{\left((T-t-1)^2+T-t-1\right)\sum_{t'=t+1}^{T} \pi_i^{b_i^{t'}}(I)}, & \text{otherwise.} \end{cases} \tag{19}$$

The average strategy update in eq. (19) corresponds to the average strategy update described in Section 3.4, i.e., it does a weighted average of the b_i^t strategies over iterations $\{t+1, ..., T\}$ with weight corresponding to the iteration.

4.3.3. 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 algo-

rithm uses only the regret bound abstraction update described in Section 4.3.1. 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 4.3.2. 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 $T_k < T$. $\boldsymbol{\tau}^T$ is the sequence of iteration counts corresponding to $\mathcal{T}^T$. Let $\boldsymbol{\tau}^T_{\bar{R}_i}$ be a sequence containing all the iteration counts $\boldsymbol{\tau}_j \in \boldsymbol{\tau}^T$, where for the corresponding T_j, T_{j+1} holds that

$$L_I^{T_j}(T_{j+1}) < \frac{\bar{R}_i}{|\mathcal{I}_i|}, \quad \forall I \in \mathcal{I}_i, \tag{20}$$

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

$$p_r(T, \bar{R}_i) = \begin{cases} 0, \text{if } |\boldsymbol{\tau}^T_{\bar{R}_i}| < |\mathcal{I}|, \\ 1 - \left(1 - \left(\frac{k_b}{|\mathcal{I}|}\right)^{|\mathcal{I}|}\right)^{\binom{|\boldsymbol{\tau}^T_{\bar{R}_i}|}{|\mathcal{I}|}} \text{, otherwise.} \end{cases} \tag{21}$$

Lemma 2. *$p_r(T, \bar{R}_i)$ 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 $\boldsymbol{\tau}^T_{\bar{R}_i}$ 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 $\boldsymbol{\tau}^T_{\bar{R}_i}$.

We know that

$$\bar{R}_i^T \leq \sum_{I \in \mathcal{I}_i} \left(R^T_{i,imm}(I)\right)^+ \leq |\mathcal{I}_i| \max_{I \in \mathcal{I}_i} \left(R^T_{i,imm}(I)\right)^+, \quad \forall T. \tag{22}$$

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. (22), there must exist $I \in \mathcal{I}_i$ such that for each iterations T', T'', $T'' > T'$

$$\left(R^{T''}_{T',imm}(I)\right)^+ > \frac{\bar{R}_i}{|\mathcal{I}_i|}. \tag{23}$$

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 $R^{T'}_{T_j,imm}(I) > L^{T_j}_I(T')$. Therefore, from eq. (23) 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 $L^{T_j}_I(T_{j+1}) < \frac{\bar{R}_i}{|\mathcal{I}_i|}$. Hence, from eq. (20) 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. (21). The first case of the piecewise function in eq. (21) 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 3. *The average external regret of the CFR+IRA is bounded in the following way*

$$\frac{R^T_i}{T} \leq B_i(T) = \frac{1}{T}\left(\Delta|\mathcal{I}_i|\sqrt{A_{max}}\sqrt{T} + \Delta|\mathcal{I}_i|T_{lim}\right), \tag{24}$$

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:

$$\max_{a \in \mathcal{A}(I)} R^T_i(I, a) \leq \Delta T_{lim} + \Delta\sqrt{|\mathcal{A}(I)|}\sqrt{T}. \tag{25}$$

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. (25) and (4) 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 2). □

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

Theorem 2. *Let*

$$\alpha(\delta) = \sqrt[3]{2\sqrt[|\mathcal{I}_i|]{|\mathcal{I}|!\log_{1-\left(\frac{k_b}{|\mathcal{I}|}\right)^{|\mathcal{I}|}}(\delta)+|\mathcal{I}|+3}A_{max}}. \tag{26}$$

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

$$\frac{R_i^T}{T} \leq \Delta|\mathcal{I}_i|\frac{\sqrt{A_{max}}}{\sqrt{T}} + \Delta|\mathcal{I}_i|\frac{\alpha(\delta)}{\sqrt[3]{T}} + \Delta|\mathcal{I}_i|\frac{1}{T} \in \mathcal{O}\left(\frac{1}{\sqrt[3]{T}}\right). \tag{27}$$

with probability $1-\delta$.

Proof. The bound in eq. (27) is created by substituting

$$T_{lim} = \left\lceil \sqrt[3]{2\sqrt[|\mathcal{I}_i|]{|\mathcal{I}|!\log_{1-\left(\frac{k_b}{|\mathcal{I}|}\right)^{|\mathcal{I}|}}(\delta)+|\mathcal{I}|+3}A_{max}} \cdot T^{\frac{2}{3}} \right\rceil, \tag{28}$$

to the bound from Lemma 3. Hence, we need to show that choosing this T_{lim} guarantees that

$$p_r(T_{lim}, B_i(T)) \geq 1-\delta. \tag{29}$$

The proof is conducted in the following way. First, in Lemma 4 we show the size of $\tau_{B_i(T)}^{T_{lim}}$ sufficient to guarantee that inequality (29) holds. In Lemma 5 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 6 we derive the T_{lim} that implies sufficient number of elements in $\tau^{T_{lim}}_{B_i(T)}$.

Lemma 4.

$$|\tau^{T_{lim}}_{B_i(T)}| > \sqrt[|\mathcal{I}|]{|\mathcal{I}|! \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta)} + |\mathcal{I}| \tag{30}$$

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

Proof.

$$|\tau^{T_{lim}}_{B_i(T)}| > \sqrt[|\mathcal{I}|]{|\mathcal{I}|! \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta)} + |\mathcal{I}| \tag{31}$$

$$\implies \left(|\tau^{T_{lim}}_{B_i(T)}| - |\mathcal{I}|\right)^{|\mathcal{I}|} > |\mathcal{I}|! \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta) \tag{32}$$

$$\implies \prod_{j=|\tau^{T_{lim}}_{B_i(T)}|-|\mathcal{I}|+1}^{|\tau^{T_{lim}}_{B_i(T)}|} j > |\mathcal{I}|! \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta) \tag{33}$$

$$\implies \frac{\prod_{j=|\tau^{T_{lim}}_{B_i(T)}|-|\mathcal{I}|+1}^{|\tau^{T_{lim}}_{B_i(T)}|} j}{|\mathcal{I}|!} > \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta) \tag{34}$$

$$\implies \frac{|\tau^{T_{lim}}_{B_i(T)}|!}{(|\tau^{T_{lim}}_{B_i(T)}| - |\mathcal{I}|)!|\mathcal{I}|!} > \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta) \tag{35}$$

$$\implies \binom{|\tau^{T_{lim}}_{B_i(T)}|}{|\mathcal{I}|} > \log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta) \tag{36}$$

$$\implies \left(1 - \left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}\right)^{\binom{|\tau^{T_{lim}}_{B_i(T)}|}{|\mathcal{I}|}} < \delta \tag{37}$$

$$\implies 1 - \left(1 - \left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}\right)^{\binom{|\tau^{T_{lim}}_{B_i(T)}|}{|\mathcal{I}|}} > 1 - \delta \tag{38}$$

□

Lemma 5.

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

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 2, 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

$$L^{T_j}_I(T_{j+1}) < \frac{B_i(T)}{|\mathcal{I}_i|}. \tag{39}$$

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

$$\max_{I \in \mathcal{I}_i} L^{T_j}_I(T_{j+1}) \leq \frac{\Delta \sqrt{A_{max}}}{\sqrt{T_{j+1} - T_j}}. \tag{40}$$

And so from eqs. (39) and (40), 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

$$T_{B_i(T)} = \left\lceil \left[\frac{T \sqrt{A_{max}}}{T_{lim} + \sqrt{A_{max}} \sqrt{T}} \right]^2 \right\rceil. \tag{41}$$

□

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 4 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 6. *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 $\boldsymbol{\tau}^{T_{lim}}$ *are higher or equal to* $T_{B_i(T)}$.

Proof.

$$T_{lim} = \left\lceil \sqrt[3]{2^{\sqrt[|\mathcal{I}|]{|\mathcal{I}|!\log_{1-\left(\frac{k_b}{|\mathcal{I}|}\right)^{|\mathcal{I}|}}(\delta)}+|\mathcal{I}|+3}} A_{max} \cdot T^{\frac{2}{3}} \right\rceil \tag{42}$$

$$\implies \quad T_{lim} \geq \sqrt[3]{2^{\sqrt[|\mathcal{I}|]{|\mathcal{I}|!\log_{1-\left(\frac{k_b}{|\mathcal{I}|}\right)^{|\mathcal{I}|}}(\delta)}+|\mathcal{I}|+3}} A_{max} \cdot T^{\frac{2}{3}} \tag{43}$$

$$\implies \quad T_{lim}^3 \geq 2 \cdot 2^{\sqrt[|\mathcal{I}|]{|\mathcal{I}|!\log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|+1}}(\delta)}+|\mathcal{I}|+2} A_{max} T^2 \tag{44}$$

$$\implies \quad (T_{lim}+1)\left(T_{lim} + \sqrt{A_{max}}\sqrt{T}\right)^2 \geq \tag{45}$$

$$\geq 2^{\sqrt[|\mathcal{I}|]{|\mathcal{I}|!\log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta)}+|\mathcal{I}|+2} 2A_{max} T^2 \tag{46}$$

$$\implies \quad \frac{T_{lim}+1}{2^{\sqrt[|\mathcal{I}|]{|\mathcal{I}|!\log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta)}+|\mathcal{I}|+2}} \geq \frac{2A_{max}T^2}{\left(T_{lim} + \sqrt{A_{max}}\sqrt{T}\right)^2} \tag{47}$$

$$\implies \quad \frac{T_{lim}+1}{2^{\sqrt[|\mathcal{I}|]{|\mathcal{I}|!\log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta)}+|\mathcal{I}|+2}} \geq \left\lceil \frac{A_{max}T^2}{\left(T_{lim} + \sqrt{A_{max}}\sqrt{T}\right)^2} \right\rceil \tag{48}$$

$$\implies \quad 2^{\log_2(T_{lim}+1)-2-\sqrt[|\mathcal{I}|]{|\mathcal{I}|!\log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta)}-|\mathcal{I}|} \geq T_{B_i(T)} \tag{49}$$

$$\implies \quad 2^{\log_2(T_{lim}+1)-1-\left\lceil \sqrt[|\mathcal{I}|]{|\mathcal{I}|!\log_{1-\left(\frac{k_b}{\mathcal{I}}\right)^{|\mathcal{I}|}}(\delta)}+|\mathcal{I}| \right\rceil} \geq T_{B_i(T)} \tag{50}$$

Eq. (50) states that 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 $\boldsymbol{\tau}^{T_{lim}}$ are at least $T_{B_i(T)}$, since the elements in $\boldsymbol{\tau}^{T_{lim}}$ are increasing. □

Hence, using T_{lim} from Lemma 6 guarantees that $p_r(T_{lim}, B_i(T)) > 1-\delta$ from Lemma 4. When substituting this T_{lim} to the bound from Lemma 3, i.e., to

$$\frac{R_i^T}{T} \leq \frac{1}{T}\left(\Delta|\mathcal{I}_i|\sqrt{A_{max}}\sqrt{T} + \Delta|\mathcal{I}_i|T_{lim}\right), \tag{51}$$

Table 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

we get

$$\frac{R_i^T}{T} \leq \frac{1}{T}\left(\Delta|\mathcal{I}_i|\sqrt{A_{max}}\sqrt{T} + \Delta|\mathcal{I}_i|\left\lceil\alpha(\delta)T^{\frac{2}{3}}\right\rceil\right) \tag{52}$$

$$\leq \Delta|\mathcal{I}_i|\frac{\sqrt{A_{max}}}{\sqrt{T}} + \Delta|\mathcal{I}_i|\frac{\alpha(\delta)}{\sqrt[3]{T}} + \Delta|\mathcal{I}_i|\frac{1}{T} \tag{53}$$

$$\in \mathcal{O}\left(\frac{1}{\sqrt[3]{T}}\right). \tag{54}$$

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

$$T \geq (\alpha(\delta)+1)^3 \tag{55}$$

$$\implies \quad \sqrt[3]{T} \geq (\alpha(\delta)+1) \tag{56}$$

$$\implies \quad T \geq (\alpha(\delta)+1)T^{\frac{2}{3}} \tag{57}$$

$$\implies \quad T \geq \left\lceil\alpha(\delta)T^{\frac{2}{3}}\right\rceil \tag{58}$$

$$\implies \quad T \geq T_{lim}. \tag{59}$$

□

Combining Regret Bound and Heuristic Abstraction Update.

When using the heuristic abstraction update in combination with the regret bound update, Theorem 2 still holds, since in the worst case the heuristic update does not 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 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 1(left) occur, we get

$$r^h_{I'}(c) = -b^t_i(I, d) \tag{60}$$
$$r^h_{I'}(d) = 1 - b^t_i(I, d) \tag{61}$$
$$r^h_{I''}(c) = -10b^t_i(I, d) \tag{62}$$
$$r^h_{I''}(d) = 10 - 10b^t_i(I, d). \tag{63}$$

When computing the regret r^h for heuristic update in iteration t' where the expected utilities from Table 1 (right) occur, we get

$$r^h_{I'}(c) = 10 - 10b^{t'}_i(I, c) \tag{64}$$
$$r^h_{I'}(d) = -10b^{t'}_i(I, c) \tag{65}$$
$$r^h_{I''}(c) = 1 - b^{t'}_i(I, c) \tag{66}$$
$$r^h_{I''}(d) = -b^{t'}_i(I, c). \tag{67}$$

The actions corresponding to d are always prefered in both I' and I'' when the utilties are given by Table 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 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''.

4.3.4. 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 4.2.4.

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 4.2.4 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 4.2.4 Case 1.

4.3.5. 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 4.3.4. 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 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 7) 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 we provide experimental evaluation of the runtime of CFR+IRA.

5. Experiments

In this section we present the experimental evaluation of FPIRA and CFR+IRA. First, we briefly describe the Double Oracle algorithm (DOEFG) [27] which will be used as a baseline in the experiments and 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 experimantal evaluation (these parameters control the memory used by CFR+IRA to update the abstraction, see Sections 4.3.1 and 4.3.2 for more details). We follow with the comparison of the memory requirements of FPIRA, CFR+IRA and the 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 $k_h = y$.

5.1. Experimental Settings

The experiments were performed using domain independent implementation of all algorithms in Java[2]. DOEFG uses IBM CPLEX 12.6. to solve the

[2]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/

underlying linear programs. Both FPIRA and CFR+IRA used the initial abstraction built as described in Section 4.1.1. 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 4.3.1), were set to

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

5.2. Double Oracle Algorithm

The Double oracle algorithm for solving perfect recall EFGs (DOEFG, [43]) 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 in 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.

We chose the DOEFG as a baseline for comparison since it is the state-of-the-art domain-independent algorithm for solving large EFGs with prohibitive memory requirements. Furthermore, the conceptual idea of DOEFG is similar to FPIRA and CFR+IRA since DOEFG 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, exploit a completely different type of sparseness than DOEFG.

5.3. 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

automatedabstractions/memeff.

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.

5.3.1. Poker

As a first domain, we use a two-player poker, which is commonly used as a benchmark in imperfect-information game solving [1]. 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 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. II-Goofspiel

II-Goofspiel is a modification of the Goofspiel game [44] which is commonly used as a benchmark domain (see, e.g., [45, 46]). 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.

5.3.3. Graph Pursuit

Graph pursuit is a game played between the defender and the attacker on the graph depicted in Figure 4. The attacker starts in the node labeled S and tries to reach the node labeled G. The defender controls two units which start

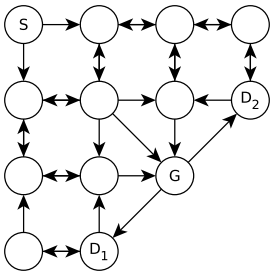


Figure 4: Graph used in the Graph pursuit domain.

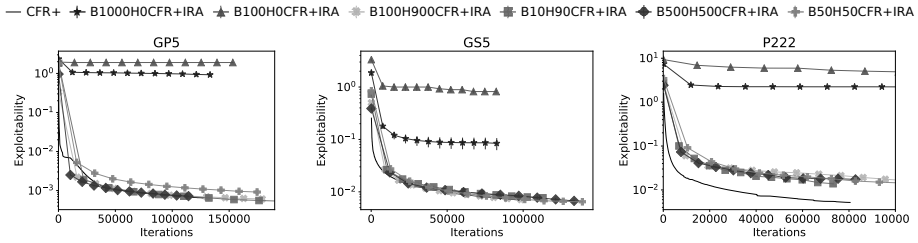


Figure 5: The plots showing the sum of exploitabilities of the resulting strategies for player 1 and 2 (log y-axis) as a function of iterations (x-axis) for GP5, GS5, P222.

in nodes D_1 and D_2. The players move simultaneously and are forced to move their units each round. Both the 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.

5.4. 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 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

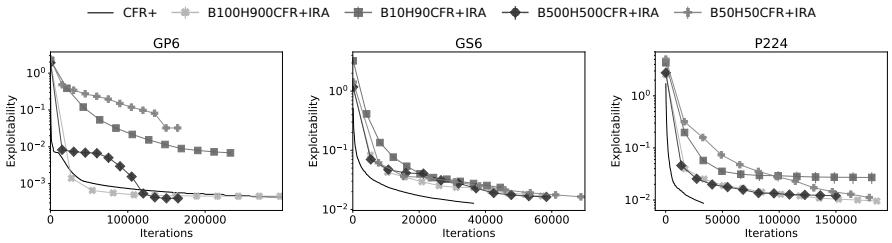


Figure 6: The plots showing the sum of exploitabilities of the resulting strategies of player 1 and 2 (log y-axis) as a function of iterations (x-axis) for GP6, GS6, P224.

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 $k_h > 0$.

In Figure 6 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.

5.5. 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

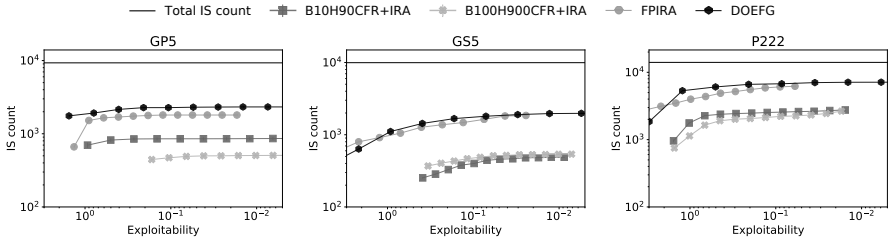


Figure 7: The plots depicting the number of information sets (log y-axis) used by algorithms to compute strategies with the exploitability depicted on the log x-axis for GP5, GS5, P222.

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.

5.5.1. GP5, GS5 and P222

In Figure 7 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, 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 8 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,

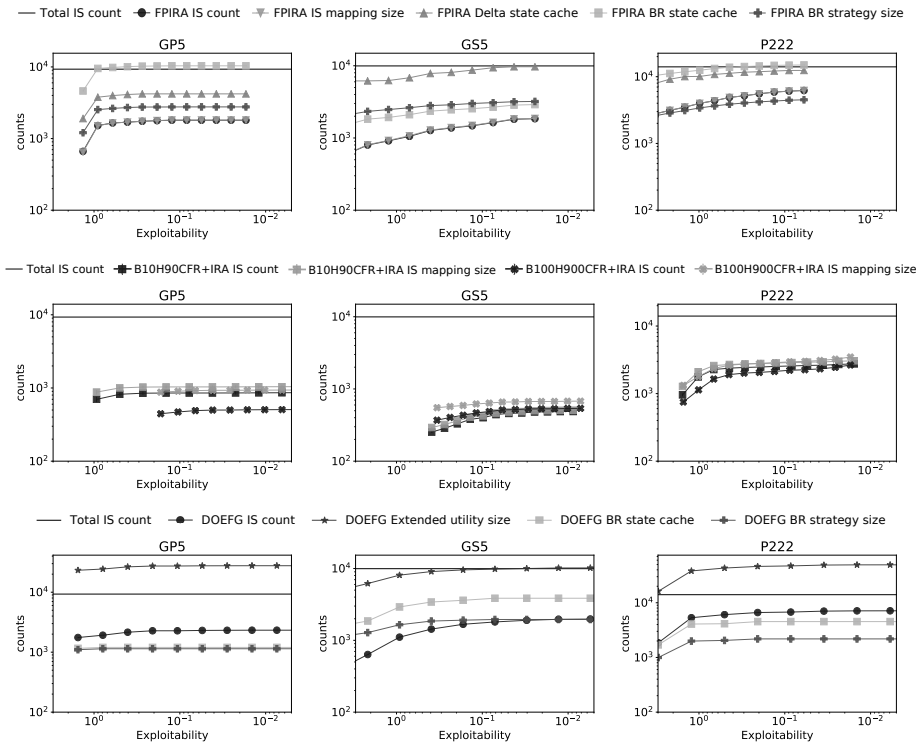


Figure 8: Plots showing size of data stored during the run of FPIRA in the first row, CFR+IRA in the second row and DOEFG in the third row (log y-axis) as a function of the sum of exploitabilities of the resulting strategies of player 1 and 2 (log x-axis) for GP5, GS5 and P222.

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 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

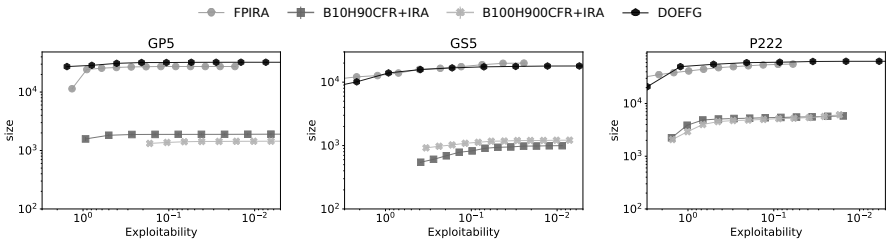


Figure 9: The number of 32 bit words the algorithms store (log y-axis) as a function of the exploitability of the resulting strategies (log x-axis) for GP5, GS5 and P222.

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 [27], 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 [27] for more details). Finally, we report the number of information sets in the 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 9 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 8. 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.

5.5.2. GP6, GS6 and P224

In Figure 10 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

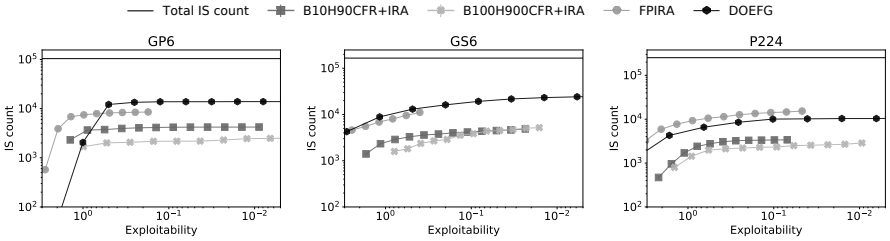


Figure 10: The plots depicting the number of information sets (log y-axis) used by algorithms to compute strategies with the sum of their exploitabilities depicted on the log x-axis for GP6, GS6 and P224.

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 5.6 for runtime analysis). Additionally, in case of GP6, GS6 and P224, $k_h + k_b = 100$ corresponds 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 11 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 12 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.

5.5.3. 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 13 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.

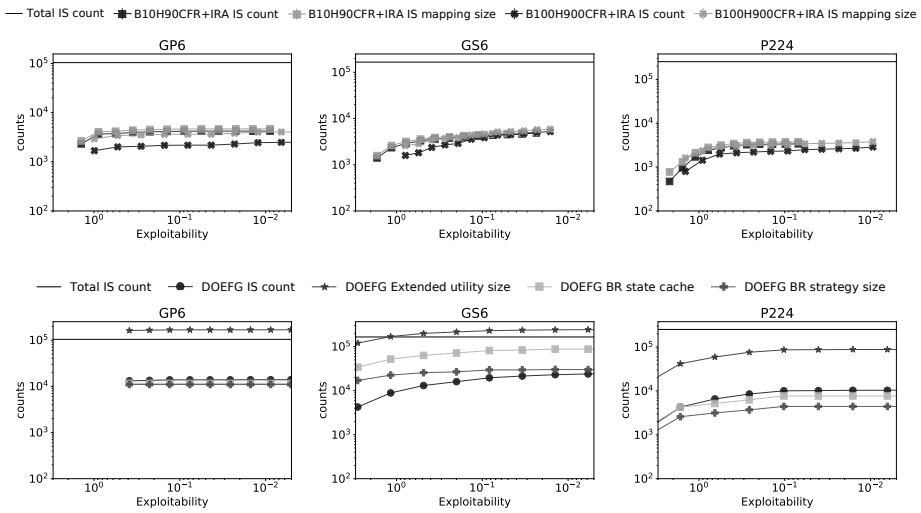


Figure 11: Plots showing size of data stored during the run of CFR+IRA in the first row and DOEFG in the second row (log y-axis) as a function of the sum of exploitabilities of the resulting strategies of player 1 and 2 (log x-axis) for GP6, GS6 and P224.

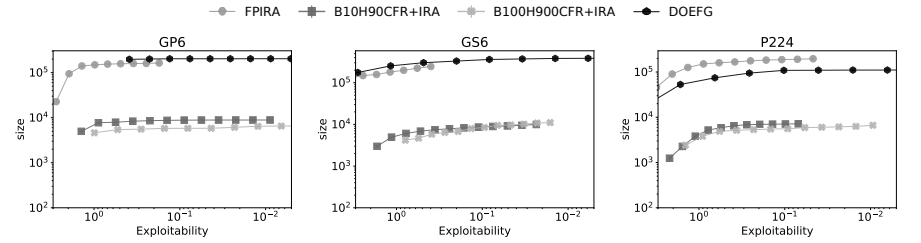


Figure 12: The number of 32 bit words the algorithms store (log y-axis) as a function of the exploitability of the resulting strategies (log x-axis) for GP6, GS6 and P224.

5.6. Runtime

In this section, we provide a comparison of the runtime of the algorithms. The plots in Figure 14 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 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 re-

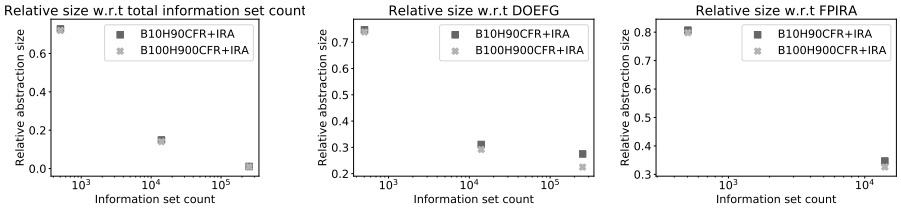


Figure 13: Plots showing the relative size of the abstractions used by CFR+IRA (y-axis) compared to the total information set count, the size of the restricted game used by DOEFG and the size of the abstraction used by FPIRA to compute resulting strategies with the sum of exploitabilities 0.05 as a function of the information set count of different poker instances (log x-axis).

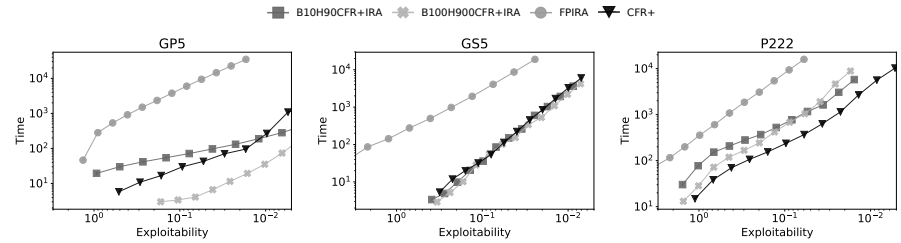


Figure 14: The plots showing runtime of FPIRA and CFR+IRA in seconds (log y-axis) required to reach the given sum of exploitabilities of resulting strategies of player 1 and 2 (log x-axis) for GP5, GS5, and P222 respectively.

quirements. 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 15 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 for the abstraction update in all 3 domains. Hence it takes longer to refine the abstraction to allow strategies with a smaller exploitability.

5.7. 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

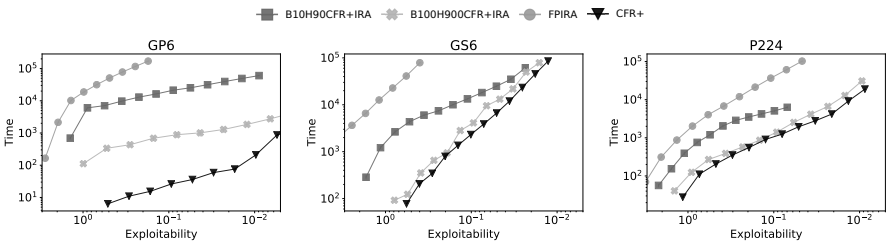


Figure 15: The plots showing runtime of FPIRA and CFR+IRA in seconds (log y-axis) required to reach the given sum of exploitabilities of resulting strategies of player 1 and 2 (log x-axis) for GP5, GS5 and P222 respectivelly.

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

6. Conclusion

The imperfect recall abstraction methodology can significantly reduce the memory required to solve large extensive-form games. However, solving the resulting imperfect recall abstracted game is a hard problem. Hence, there is only a limited amount of work that focuses on using imperfect recall abstractions. Previous works use either very restrictive subclasses of imperfect recall abstractions [19, 20, 21], heuristic approaches [22], or use computationally complex algorithms to solve the imperfect recall abstracted game [23].

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 FPIRA and CFR+IRA which are able to start with an arbitrary imperfect recall abstraction of the solved two-player zero-sum perfect recall extensive-form game. 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. As a consequence, the strategy from the abstracted game can be used directly in the original game without the need to use translation

techniques, and the quality of such strategy is not affected by the choice of the initial abstraction.

The experimental evaluation shows that CFR+IRA requires at least an order of magnitude less memory than FPIRA and the Double Oracle algorithm (DOEFG, [27]) to solve given domains. Even when using trivial automatically build initial imperfect recall abstraction, CFR+IRA is capable of closely approximating Nash equilibrium of large extensive-form games using abstraction with as little as 0.9% of information sets of the original game. Furthermore, the results suggest that the relative size of the abstraction used by CFR+IRA will further decrease as the size of the solved game increases.

6.1. Future Work

There are several directions for future work.

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. A more sophisticated mapping, which would take into account the similarity of the actions, could greatly reduce the number of refinements of the initial abstraction CFR+IRA and FPIRA need to perform to find the desired approximation of the Nash equilibrium of the original game.

CFR based approaches are known to perform extremely well in poker domains [6, 37, 16]. 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. Hence, 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.

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 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 such abstractions would have two benefits. (1) CFR+IRA could suggest further improvements of the abstractions in places where they are too coarse. (2) CFR+IRA would use these abstractions to provide a better approximation of the Nash equilibrium of the original domain than the current solvers, as it would improve the abstraction if necessary. This, combined with the domain-specific implementation could lead to significant improvement of the quality of the strategies computed in the existing abstractions, as there are typically no guarantees that the abstraction allows computation of sufficient approximation of the Nash equilibrium in the poker domain.

Acknowledgments

This research 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. SGS16/235/OHK3/3T/13. Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures".

References

References

[1] J. Rubin, I. Watson, Computer poker: A review, Artificial intelligence 175 (5-6) (2011) 958–987.

[2] V. Lisý, T. Davis, M. Bowling, Counterfactual Regret Minimization in Sequential Security Games, in: Proceedings of AAAI Conference on Artificial Intelligence, 2016.

[3] G. Christodoulou, A. Kovács, M. Schapira, Bayesian combinatorial auctions, Automata, Languages and Programming (2008) 820–832.

[4] T. Sandholm, Steering evolution strategically: computational game theory and opponent exploitation for treatment planning, drug design, and synthetic biology, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI Press, 2015, pp. 4057–4061.

[5] B. Bošanský, V. Lisý, M. Lanctot, J. Čermák, M. H. Winands, Algorithms for Computing Strategies in Two-Player Simultaneous Move Games, Artificial Intelligence 237 (2016) 1–40.

[6] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, M. Bowling, Deepstack: Expert-level artificial intelligence in heads-up no-limit poker, Science.

[7] F. Fang, T. H. Nguyen, R. Pickles, W. Y. Lam, G. R. Clements, B. An, A. Singh, B. C. Schwedock, M. Tambe, A. Lemieux, Paws—a deployed game-theoretic application to combat poaching, AI Magazine.

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

[9] M. Zinkevich, M. Johanson, M. H. Bowling, C. Piccione, Regret minimization in games with incomplete information., in: Advances in Neural Information Processing Systems, 2007, pp. 1729–1736.

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

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

[12] A. Gilpin, T. Sandholm, T. B. Sørensen, Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of texas hold'em poker, in: Proceedings of the National Conference on Artificial Intelligence, Vol. 22, 2007, p. 50.

[13] A. Gilpin, T. Sandholm, Lossless abstraction of imperfect information games, Journal of the ACM (JACM) 54 (5) (2007) 25.

[14] C. Kroer, T. Sandholm, Extensive-Form Game Abstraction with Bounds, in: Proceedings of the fifteenth ACM conference on Economics and computation, ACM, 2014, pp. 621–638.

[15] N. Brown, T. W. Sandholm, Simultaneous abstraction and equilibrium finding in games, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.

[16] N. Brown, T. Sandholm, Superhuman AI for heads-up no-limit poker: Libratus beats top professionals, Science (2017) eaao1733.

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

[18] K. A. Hansen, P. B. Miltersen, T. B. Sørensen, Finding Equilibria in Games of no Chance, in: Computing and Combinatorics, Springer, 2007, pp. 274–284.

[19] M. Lanctot, N. Burch, M. Zinkevich, M. Bowling, R. G. Gibson, No-regret learning in extensive-form games with imperfect recall, in: Proceedings of the 29th International Conference on Machine Learning (ICML-12), 2012, pp. 65–72.

[20] C. Kroer, T. Sandholm, Imperfect-Recall Abstractions with Bounds in Games, in: Proceedings of the seventeenth ACM conference on Economics and computation, ACM, 2016, pp. 459–476.

[21] B. Bošanský, A. X. Jiang, M. Tambe, C. Kiekintveld, Combining compact representation and incremental generation in large games with sequential strategies., in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015, pp. 812–818.

[22] K. Waugh, M. Zinkevich, M. Johanson, M. Kan, D. Schnizlein, M. H. Bowling, A Practical Use of Imperfect Recall, in: SARA, Citeseer, 2009.

[23] 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 93 (2018) 290–326.

[24] J. Cermak, B. Bosansky, V. Lisy, 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.

[25] G. W. Brown, Some notes on computation of games solutions, Tech. rep., DTIC Document (1949).

[26] O. Tammelin, Cfr+, CoRR, abs/1407.5042.

[27] B. Bosansky, C. Kiekintveld, V. Lisy, M. Pechoucek, An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information, Journal of Artificial Intelligence Research (2014) 829–866.

[28] J. Shi, M. L. Littman, Abstraction methods for game theoretic poker, Computers and Games 2063 (2000) 333–345.

[29] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, D. Szafron, Approximating game-theoretic optimal strategies for full-scale poker, in: IJCAI, 2003, pp. 661–668.

[30] A. Gilpin, T. 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, Vol. 21, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006, p. 1007.

[31] J. A. Hawkin, R. Holte, D. Szafron, Automated action abstraction of imperfect information extensive-form games., in: AAAI, 2011.

[32] J. A. Hawkin, R. Holte, D. Szafron, Using sliding windows to generate action abstractions in extensive-form games., in: AAAI, 2012.

[33] A. Gilpin, T. Sandholm, T. B. 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, International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 911–918.

[34] N. Basilico, N. Gatti, Automated abstractions for patrolling security games., in: AAAI, 2011.

[35] T. Sandholm, S. Singh, Lossy stochastic game abstraction with bounds, in: Proceedings of the 13th ACM Conference on Electronic Commerce, ACM, 2012, pp. 880–897.

[36] H. W. Kuhn, Extensive Games and the Problem of Information, Annals of Mathematics Studies.

[37] O. Tammelin, N. Burch, M. Johanson, M. Bowling, Solving heads-up limit texas hold'em., in: IJCAI, 2015, pp. 645–652.

[38] J. Robinson, An iterative method of solving a game, Annals of mathematics (1951) 296–301.

[39] S. Karlin, Mathematical methods and theory in games, programming, and economics, Vol. 2, Courier Corporation, 2003.

[40] C. Daskalakis, Q. Pan, A counter-example to karlin's strong conjecture for fictitious play, in: Annual Symposium on Foundations of Computer Science, IEEE, 2014, pp. 11–20.

[41] J. Heinrich, M. Lanctot, D. Silver, Fictitious self-play in extensive-form games., in: ICML, 2015, pp. 805–813.

[42] M. Johanson, K. Waugh, M. Bowling, M. Zinkevich, Accelerating best response calculation in large extensive games, in: IJCAI, Vol. 11, 2011, pp. 258–265.

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

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

[45] V. Lisý, M. Lanctot, M. 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, International Foundation for Autonomous Agents and Multiagent Systems, 2015, pp. 27–36.

[46] M. Lanctot, V. Lisý, M. H. Winands, Monte carlo tree search in simultaneous move games with applications to goofspiel, in: Workshop on Computer Games, Springer, 2013, pp. 28–43.