Combining Incremental Strategy Generation and Branch and Bound Search for Computing Maxmin Strategies in Imperfect Recall Games

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

ABSTRACT

Extensive-form games with imperfect recall are an important model of dynamic games where the players are allowed to forget previously known information. Often, imperfect recall games are the result of an abstraction algorithm that simplifies a large game with perfect recall. Unfortunately, solving an imperfect recall game has fundamental problems since a Nash equilibrium does not have to exist. Alternatively, we can seek maxmin strategies that guarantee an expected outcome. The only existing algorithm computing maxmin strategies in two-player imperfect recall games without absentmindedness, however, requires approximating a bilinear mathematical program that is proportional to the size of the whole game and thus has a limited scalability. We propose a novel algorithm for computing maxmin strategies in this class of games that combines this approximate algorithm with an incremental strategy-generation technique designed previously for extensive-form games with perfect recall. Experimental evaluation shows that the novel algorithm builds only a fraction of the game tree and improves the scalability by several orders of magnitude. Finally, we demonstrate that our algorithm can solve an abstracted variant of a large game faster compared to the algorithms operating on the unabstracted perfect-recall variant.

Keywords

Game Theory, Maxmin strategies, Imperfect recall

INTRODUCTION

Dynamic games with a finite number of moves can be modeled as extensive-form games (EFGs) that are general enough to model scenarios with stochastic events and imperfect information. EFGs can model games such as Poker as well as many real-world scenarios where players have sequential strategies and are able to react to information about the opponent. EFGs are visualized as game trees, where nodes correspond to states of the game and edges to actions performed by players. Imperfect information is modeled by grouping indistinguishable states into information sets.

Recent advancements in scalability of algorithms for solving EFGs has been primarily driven by the research around the Annual Computer Poker Competition[1] and has led to solving heads-up limit texas hold'em poker [3]. Most of the algorithms for solving EFGs assume that players remember all the information gained during the course of the game [20, 22, 6] – a property denoted as a *perfect recall.* The size of a strategy (a randomized selection of an action to play in each information set) grows exponentially with the number of moves in the game due to the perfect memory. One approach for solving large perfect recall EFGs is thus to create a small abstracted game where certain information sets are merged together, solve this abstracted game, and translate the strategy into the original game (e.g., see [4, 13, 14]). However, to achieve sufficient space reductions the assumption of perfect recall might need to be violated in the abstracted game resulting in *imperfect recall.*

There are fundamental difficulties when solving imperfect recall games. The best known game-theoretic solution concept, a Nash equilibrium (NE), does not have to exist even in zero-sum games [21]. As a consequence, standard algorithms (e.g., a Counterfactual Regret Minimization (CFR) [22]) can converge to exploitable strategies (see Example 1). Existing approaches avoid this issue by creating very specific abstracted games so that perfect recall algorithms are still applicable: e.g., in *(skew) well-formed games* [16, 14] and *normal-form games with sequential strategies* [17]. The restrictions posed by these classes are rather strict, however, and can prevent us from creating sufficiently small abstracted games and thus fully exploit the possibilities of abstractions and compact representation of dynamic games.

An alternative to finding NE is to compute a strategy that guarantees the best worst-case expected outcome for a player – *a maxmin strategy.* However, computing a maxmin strategy is NP-hard in imperfect recall games and it may require irrational numbers even when the input uses only rational numbers [9]. The first algorithm approximating maxmin strategies in two-player imperfect recall games without absentmindedness, where the minimizing player has a restricted type of imperfect recall denoted as A-loss recall, uses a mixed-integer linear program (MILP) and a branch-and-bound search over linear relaxations of this MILP (denoted as BnB) [1]. The main disadvantage of BnB is that it requires to repeatedly solve a linear program proportional to the size of the game, resulting in a limited scalability.

We propose a novel algorithm for finding maxmin strategies in imperfect recall games, that extends the BnB algo-

Appears in: *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*

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

rithm by employing incremental strategy generation. While such techniques exist for perfect recall games [2], transferring the ideas to imperfect recall games presents a number of challenges that we address in this paper. We define the restricted EFG that is a subset of the original EFG and describe how the restricted game is solved via the BnB search. Finally, we ensure the correct expansion of the restricted EFG so that our algorithm preserves guarantees for approximating maxmin strategies. The experimental evaluation shows that our algorithm improves the scalability of BnB by at least an order of magnitude. Moreover, we show that we can use our algorithm to solve an abstracted imperfect-recall variant of Phantom Tic-Tac-Toe faster compared to solving the original perfect-recall variant of the game.

EXTENSIVE-FORM GAMES WITH IMPERFECT RECALL

A two-player extensive-form game (EFG, see Figure 1) is a tuple $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, u, \mathcal{C}, \mathcal{I})$. $\mathcal{N} = \{1, 2\}$ is a set of players, by i we refer to one of the players, and by $-i$ to his opponent. $\mathcal{A}$ denotes the set of all actions. $\mathcal{H}$ 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. An ordered list of all actions of player i from root to node h is referred to as a *sequence*, $\sigma_i = \mathsf{seq}_i(h)$, Σ_i is a set of all sequences of player i. For each $z \in \mathcal{Z}$ we define a *utility function* $u_1 : \mathcal{Z} \to \mathbb{R}$. The player 1 maximizes u_1, while player 2 minimizes it. 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 due to chance. Imperfect observation of player i is modeled via *information sets* $\mathcal{I}_i$ that form a partition over $h \in \mathcal{H}$ where i takes action. Player i cannot distinguish between nodes in any $I_i \in \mathcal{I}_i$. $\mathcal{A}(I_i)$ denotes actions available in each $h \in I_i$. The action a uniquely identifies the information set where it is available. We use $\mathsf{seq}_i(I_i)$ as a set of all sequences of player i leading to I_i. Finally, we use $\mathsf{inf}_i(\sigma_i)$ to be a set of all information sets to which sequence σ_i leads.

A *behavioral strategy* $\beta_i \in \mathcal{B}_i$ is a probability distribution over actions in each information set $I \in \mathcal{I}_i$. We use $u_1(\beta) = u_1(\beta_i, \beta_{-i})$ for the expected outcome of the game when players follow β. A *best response* of player 1 against β_2 is a strategy $\beta_1^{BR} \in BR_1(\beta_2)$, where $u_1(\beta_1^{BR}, \beta_2) \geq u_1(\beta_1', \beta_2)$ for all $\beta_1' \in \mathcal{B}_1$ ($BR_1(\beta_2)$ denotes a set of all best responses to β_2). Best response of player 2 is defined analogously. $\beta_i(I, a)$ is the probability of playing a in I.

A *maxmin strategy* β_1^* is defined as $\beta_1^* = \arg\max_{\beta_1 \in \mathcal{B}_1} \min_{\beta_2 \in \mathcal{B}_2} u_1(\beta_1, \beta_2)$. When a Nash equilibrium in behavioral strategies exists in a two-player zero-sum imperfect recall game then β_1^* is a Nash equilibrium strategy for player 1.

Perfect, Imperfect, and A-loss Recall.

In *perfect recall* games all players remember the history of their own actions and all information gained during the course of the game. As a consequence, all nodes in any information set I_i have the same sequence for player i. If the assumption of perfect recall does not hold in an EFG, we talk about games with *imperfect recall*. In imperfect recall games, mixed and behavioral strategies are not comparable.

In games where one information set can be reached more than once during one playthrough (game with *absentmindedness*), the best response of a player might need randomization. We restrict to games with no absentmindedness where it is sufficient to consider pure strategy best responses (see, e.g., [19]). Finding a best response in perfect recall games can be done by selecting the best action to play in each information set. This type of response, termed *time consistent strategy* [8], does not have to be an ex-ante best response in general imperfect recall games (see [19] for an example). A class of imperfect recall games where it is sufficient to consider only time consistent strategies when computing best responses was termed as *A-loss* recall games [7, 8].

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

Condition (1) says that if player i has perfect recall then she also has A-loss recall. Condition (2) requires that each loss of memory of A-loss recall player can be traced back to a loss of memory of the player's own actions. The equivalence between time consistent strategies and ex-ante best responses simplifies the computation of the best responses of player 2 in case she has A-loss recall. It is sufficient to consider best responses that correspond to the best response in a coarsest perfect-recall refinement of the imperfect recall game for a player with A-loss recall. By a *coarsest perfect recall refinement* of an imperfect recall game G we define a perfect recall game G' where we split the imperfect recall information sets to biggest subsets satisfying the perfect recall assumption. Finally, we assume that there is a mapping between actions from the coarsest perfect recall refinement $\mathcal{A}'$ and actions in the original game $\mathcal{A}$ so that we can identify to which actions from $\mathcal{A}'$ an original action $a \in \mathcal{A}$ maps. We assume this mapping to be implicit since it is clear from the context.

Lemma 1. *Let G be an imperfect recall game where player 2 has A-loss recall and β_1 a strategy of player 1. Let G' be the coarsest perfect recall refinement of G for player 2. Let β_2' be a pure best response in G' to β_1 and let β_2 be a realization equivalent behavioral strategy in G, then β_2 is a pure best response to β_1 in G.*

Proof. Being able to connect any loss of memory to forgetting his own action allows the player 2 to reconstruct all the information lost due to his imperfect recall by conditioning his behavior on his previous choices (see [1] for more detailed proof). □

The NP-hardness proof of computing maxmin strategies due to Koller [9] still applies since the reduction provided by Koller is a special case of the setting assumed in our paper.

Finally let us discuss the CFR in imperfect recall games. The no-regret learning cannot work in general in imperfect recall games, since the loss function $l^t(b_i) = u_i(b_i^t, b_{-i}^t) - u_i(b_i, b_{-i}^t)$ used in computation of external regret (see, e.g., [22]) can be non-convex over the probability simplex of behavioral strategies (the loss function must be convex for a no-regret learning to have convergence guarantees [5]).

Example 1: Assume we are in the step T of a no-regret learning algorithm solving the game from Figure 1, and we evaluate the loss of some strategy β_1 in step $t < T$. Let $\beta_1^t(a) = \beta_1^t(g) = 0.5$ and $\beta_2^t(d) = \beta_2^t(e) = 1$. Let $\beta_1(a) =$

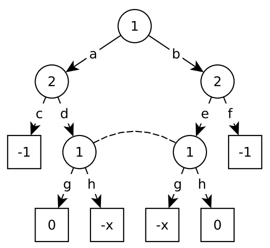

Figure 1: An imperfect recall game where CFR can reach a non-optimal strategy.

$\beta_1(g) = 1$, $\beta_1'(b) = \beta_1'(h) = 1$, and $\beta_1''(a) = \beta_1''(g) = 0.5$. The losses of these strategies are $l^t(\beta_1) = -x$, $l^t(\beta_1') = -x$, $l^t(\beta_1'') = 0$. Since β_1'' is a convex combination of β_1 and β_1' with uniform weights, it follows that the loss function is non-convex, hence the convergence guarantees used in CFR due to Gordon [5] no longer apply. This is not the case in perfect recall games since the behavior of i after any $a, a' \in A(I_i)$ is independent $\forall I \in \mathcal{I}_i$. Furthermore, the guarantee of convergence of CFR to (ϵ-)optimal strategies in (skew) well-formed games [14] is based on bounding the non-convexity of the loss function. By increasing $x > 2$ in the game from Figure 1, the CFR can find a strategy for player 1 with exploitability arbitrarily worse than the maxmin value -1, since mixing between actions a and b can yield the expected value strictly worse than the expected value reached by deterministic samples containing a and b if player 2 plays d and e with positive probability. The game has A-loss recall and has 2 NE, playing (a, g) or (b, h) deterministically for player 1 and (c, f) for player 2 (no mix between these two NE strategies for player 1 is a NE). We demonstrate the exploitability of strategies the CFR converges to in the experiments section.

MAXMIN BNB ALGORITHM

We base our method on the branch-and-bound algorithm (denoted BnB) from [1]. BnB algorithm is based on approximating the following bilinear program. We assume WLOG that player 1 is the maximizing player.

$$\max_{x,r,v} v(root, \emptyset) \tag{1a}$$

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

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

$$\sum_{a \in \mathcal{A}(I)} r(\sigma a) = r(\sigma) \qquad \forall \sigma \in \Sigma_1, \forall I \in \mathsf{inf}_1(\sigma_1) \tag{1d}$$

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

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

$$r(\sigma) \cdot x(a) = r(\sigma a) \qquad \forall I \in \mathcal{I}_1^{IR}, \forall a \in \mathcal{A}(I), \forall \sigma \in \mathsf{seq}_1(I) \tag{1g}$$

$$\sum_{\sigma_1 \in \Sigma_1} g(\sigma_1, \sigma_2 a) r_1(\sigma_1) + \sum_{I' \in \mathsf{inf}_2(\sigma_2 a)} v(I', \sigma_2 a) \geq v(I, \sigma_2)$$

$$\forall I \in \mathcal{I}_2, \forall \sigma_2 \in \mathsf{seq}_2(I), \forall a \in \mathcal{A}(I) \tag{1h}$$

Constraints (1a)–(1h) represent a bilinear reformulation of the sequence-form LP due to [20] applied to the information set structure of an imperfect recall game G where player 2 has A-loss recall. The objective of player 1 is to find a strategy that maximizes the expected utility against the best responding opponent in G. The strategy is represented by variables r that assign the probability to a sequence: $r(\sigma_1)$ is the probability that $\sigma_1 \in \Sigma_1$ will be played assuming that information sets, in which actions of the sequence σ_1 are applicable, are reached due to player 2. Probabilities r must satisfy so-called network flow Constraints (1b)–(1d). Finally, a strategy of player 1 is constrained by the best-responding opponent that selects an action minimizing the expected value in each $I \in \mathcal{I}_2$ and for each $\sigma_2 \in \mathsf{seq}_2(I)$ that was used to reach I (Constraint (1h)). These constraints ensure that the opponent plays the best response in the coarsest perfect recall refinement of G and thus also in G by Lemma 1. The expected utility for each action is a sum of the expected utility values from immediately reachable information sets I' and from immediately reachable leafs. For the later we use generalized utility function $g : \Sigma_1 \times \Sigma_2 \rightarrow \mathbb{R}$ defined as $g(\sigma_1, \sigma_2) = \sum_{z \in \mathcal{Z} | \mathsf{seq}_1(z) = \sigma_1 \wedge \mathsf{seq}_2(z) = \sigma_2} u_1(z)\mathcal{C}(z)$. In imperfect recall games multiple σ_i can lead to some imperfect recall information set $I_i \in \mathcal{I}_i^{IR} \subseteq \mathcal{I}_i$; hence, realization plans over sequences do not have to induce the same behavioral strategy for I_i. Therefore, for each $I_i \in \mathcal{I}_i^{IR}$ we define behavioral strategy $x(a)$ for each $a \in \mathcal{A}(I_i)$ (Constraints (1e)–(1f)). To ensure that the realization probabilities induce the same behavioral strategy in I_i, we add bilinear constraint $r(\sigma_i a) = x(a) \cdot r(\sigma_i)$ (Constraint (1g)).

Approximating Bilinear Terms

We use Multiparametric Disaggregation Technique (MDT) [12] for approximation of the bilinear Constraint (1g). The main idea of the approximation is to use a digit-wise discretization of one of the variables from a bilinear term. Let $a = bc$ be a bilinear term. MDT discretizes variable b and introduces new binary variables $w_{k,l}$ that indicate whether the digit on ℓ-th position is k.

$$\sum_{k=0}^{9} w_{k,\ell} = 1 \qquad \ell \in \mathbb{Z} \tag{2a}$$

$$w_{k,\ell} \in \{0, 1\} \tag{2b}$$

$$\sum_{\ell \in \mathbb{Z}} \sum_{k=0}^{9} 10^{\ell} \cdot k \cdot w_{k,\ell} = b \tag{2c}$$

$$c^L \cdot w_{k,\ell} \leq \hat{c}_{k,\ell} \leq c^U \cdot w_{k,\ell} \qquad \forall \ell \in \mathbb{Z}, \forall k \in \{0..9\} \tag{2d}$$

$$\sum_{k=0}^{9} \hat{c}_{k,\ell} = c \qquad \forall \ell \in \mathbb{Z} \tag{2e}$$

$$\sum_{\ell \in \mathbb{Z}} \sum_{k=0}^{9} 10^{\ell} \cdot k \cdot \hat{c}_{k,\ell} = a \tag{2f}$$

Constraint (2a) ensures that for each position ℓ there is exactly one digit chosen. All digits must sum to b (Constraint (2c)). Next, we introduce variables $\hat{c}_{k,\ell}$ that are equal to c for such k and ℓ where $w_{k,l} = 1$, and $\hat{c}_{k,\ell} = 0$ otherwise. c^L and c^U are bounds on the value of variable c. The value of a is given by Constraint (2f). This is an exact formulation that requires infinite sums and an infinite number of constraints. However, by restricting the set of all possible positions ℓ to a finite set $\{P_L, \ldots, P_U\}$ we get a lower bound approximation. Following the approach in [12] we can extend the lower bound formulation to compute an upper bound where Δb is assigned to every discretized variable b allowing it to take the value between two discretization points created due to the minimal value of ℓ (Constraints (3a)–(3b)).

$$\text{Constraints } (2a), (2d), (2e)$$

$$\sum_{\ell \in \{P_L,\ldots,P_U\}} \sum_{k=0}^{9} 10^{\ell} \cdot k \cdot w_{k,\ell} + \Delta b = b \tag{3a}$$

$$0 \le \Delta b \le 10^{P_L} \tag{3b}$$

$$\sum_{\ell \in \{P_L,\ldots,P_U\}} \sum_{k=0}^{9} 10^{\ell} \cdot k \cdot \hat{c}_{k,\ell} + \Delta a = a \tag{3c}$$

$$c^L \cdot \Delta b \le \Delta a \le c^U \cdot \Delta b \tag{3d}$$

$$\left(c - c^U\right) \cdot 10^{P_L} + c^U \cdot \Delta b \le \Delta a \tag{3e}$$

$$\left(c - c^L\right) \cdot 10^{P_L} + c^L \cdot \Delta b \ge \Delta a \tag{3f}$$

Similarly, we allow the product variable a to be increased with variable $\Delta a = \Delta b \cdot c$. To approximate the product of the delta variables, we use the McCormick envelope defined by Constraints (3d)–(3f).

Upper Bound MILP Approximation

By applying MDT to Constraint (1g) we represent every variable $x(a)$ using a finite number of digits. Binary variables $w^{I_1,a}_{k,\ell}$ correspond to $w_{k,\ell}$ variables from the example shown in previous subsection and are used for the digit-wise discretization of $x(a)$. $\hat{r}(\sigma_1)^a_{k,\ell}$ correspond to $\hat{c}_{k,\ell}$ variables used to discretize the bilinear term $r(\sigma_1 a)$. In order to allow variable $x(a)$ to attain an arbitrary value from $[0, 1]$ interval using a finite number of digits of precision, we add a real variable $0 \le \Delta x(a) \le 10^{-P}$ that can span the gap between two adjacent discretization points. Constraints (4d) and (4e) describe this loosening. Variables $\Delta x(a)$ also have to be propagated to bilinear terms $r(\sigma_1) \cdot x(a)$ involving $x(a)$. We cannot represent the product $\Delta r(\sigma_1 a) = r(\sigma_1) \cdot \Delta x(a)$ exactly and therefore we give bounds based on the McCormick envelope (Constraints (4i)–(4j)).

$$\max_{x,r,v} v(root, \emptyset) \tag{4a}$$

$$\text{s.t.} \quad \text{Constraints (1b) - (1f) , (1h)}$$

$$w^{I,a}_{k,\ell} \in \{0,1\} \qquad \forall I \in \mathcal{I}^{IR}_1, \forall a \in \mathcal{A}(I), \forall k \in \{0..9\}, \forall \ell \in \{-P..0\} \tag{4b}$$

$$\sum_{k=0}^{9} w^{I,a}_{k,\ell} = 1 \qquad \forall I \in \mathcal{I}^{IR}_1, \forall a \in \mathcal{A}(I), \forall \ell \in \{-P..0\} \tag{4c}$$

$$\sum_{\ell=-P}^{0} \sum_{k=0}^{9} 10^{\ell} \cdot k \cdot w^{I,a}_{k,\ell} + \Delta x(a) = x(a) \qquad \forall I \in \mathcal{I}^{IR}_1, \forall a \in \mathcal{A}(I) \tag{4d}$$

$$0 \le \Delta x(a) \le 10^{-P} \qquad \forall I \in \mathcal{I}^{IR}_1, \forall a \in \mathcal{A}(I) \tag{4e}$$

$$0 \le \hat{r}(\sigma)^a_{k,\ell} \le w^{I,a}_{k,\ell} \qquad \forall I \in \mathcal{I}^{IR}_1, \forall a \in \mathcal{A}(I), \forall \sigma \in \mathsf{seq}_1(I), \forall \ell \in \{-P..0\} \tag{4f}$$

$$\sum_{k=0}^{9} \hat{r}(\sigma)^a_{k,\ell} = r(\sigma) \qquad \forall I \in \mathcal{I}^{IR}_1, \forall \sigma \in \mathsf{seq}_1(I), \forall \ell \in \{-P..0\} \tag{4g}$$

$$\sum_{\ell=-P}^{0} \sum_{k=0}^{9} 10^{\ell} \cdot k \cdot \hat{r}(\sigma)^a_{k,\ell} + \Delta r(\sigma a) = r(\sigma a) \qquad \forall I \in \mathcal{I}^{IR}_1, \forall a \in \mathcal{A}(I), \forall \sigma \in \mathsf{seq}_1(I) \tag{4h}$$

$$(r(\sigma) - 1) \cdot 10^{-P} + \Delta x(a) \le \Delta r(\sigma a) \le 10^{-P} \cdot r(\sigma) \qquad \forall I \in \mathcal{I}^{IR}_1, \forall a \in \mathcal{A}(I), \forall \sigma \in \mathsf{seq}_1(I) \tag{4i}$$

$$0 \le \Delta r(\sigma a) \le \Delta x(a) \qquad \forall I \in \mathcal{I}^{IR}_1, \forall \sigma \in \mathsf{seq}_1(I), \forall a \in \mathcal{A}(I) \tag{4j}$$

Due to this loose representation of $\Delta r(\sigma_1 a)$, the reformulation of bilinear terms is no longer exact and this MILP therefore yields an upper bound of the bilinear sequence form program (1). Note that the MILP has both the number of variables and the number of constraints bounded by $O(|\mathcal{I}| \cdot |\Sigma| \cdot P)$, where $|\Sigma|$ is the number of sequences of both players. The number of binary variables is equal to $10 \cdot |\mathcal{I}^{IR}_1| \cdot \mathcal{A}^{max}_1 \cdot P$, where $\mathcal{A}^{max}_1 = \max_{I \in \mathcal{I}_1} |\mathcal{A}_1(I)|$.

The BNB Algorithm

The BnB algorithm works on the linear relaxation of the Upper Bound MILP and searches the BnB tree in the best first search manner. In every node n of the BnB tree, the algorithm solves the relaxed LP corresponding to node n, heuristically selects the information set I and action a contributing to the current approximation error the most, and creates successors of n by restricting the probability $\beta_1(I, a)$ that a is played in I. The algorithm adds new constraints to the LP depending on the value of $\beta_1(I, a)$ by constraining (and/or introducing new) variables $w^{I_1,a}_{k,l}$ and creating successors of the node in the BnB tree. Note that $w^{I_1,a}_{k,l}$ variables correspond to binary variables in the MILP formulation. This way, the algorithm simultaneously searches for the optimal approximation of bilinear terms as well as the assignment to binary variables. The algorithm terminates when ϵ-optimal strategy is found.

Algorithm 1: BnB algorithm

```
input      : Initial LP relaxation $LP_0$ of Upper Bound MILP
output     : $\epsilon$-optimal strategy for a maximizing player
parameters : Bound on maximum error $\epsilon$, bound $P_{max}$ for
             bilinear term precision approximation
1  fringe ← $(LP_0, -\infty, \infty)$
2  opt ← $(LP_0, -\infty, \infty)$
3  while fringe $\neq \emptyset$ do
4      $(LP, lb, ub) \leftarrow \arg\max_{n \in \text{fringe}} n.\text{ub}$
5      fringe ← fringe $\setminus (LP, lb, ub)$
6      if opt.lb $\ge ub$ then
7          return ReconstructStrategy(opt)
8      if opt.lb $< lb$ then
9          opt ← $(LP, lb, ub)$
10     if $ub - lb \le \epsilon$ then
11         return ReconstructStrategy(opt)
12     $(I_1, a)$ ← SelectAction($LP$)
13     AddSuccessors($LP$, $I_1$, $a$, $P_{max}$)
14 return ReconstructStrategy(opt)
```

More formally, the BnB algorithm (depicted in Algorithm 1) maintains the fringe of nodes. Each node corresponds to an LP with each bilinear term approximated to some level of precision (i.e., some number of decimal points). Among relaxed binary variables, all but the ones corresponding to the last level of precision are fixed to some value. The algorithm always solves the node with the highest upper bound (line 4). It keeps track of the current best solution with the highest lower bound, representing the highest guaranteed value for maximizing player (line 9). In each node, the algorithm checks the current bounds. If the bounds differ by more than the desired approximation ϵ, the algorithm generates new nodes by selecting bilinear term corresponding to some action and increases its precision (line 12), adds new variables and constraints into the LP that further restrict the maxmin strategy, and adds them to the fringe (line

13). The algorithm calculates an upper bound by solving the relaxed LP and a lower bound by constructing an imperfect recall strategy from the current LP and computing a best response against it. For more detailed description of the heuristics that can be used for ReconstructStrategy and SelectAction see [1].

In the experimental evaluation BnB often outperforms the IBM CPLEX MILP solver. There are two reasons for this: (1) BnB algorithm can compute a valid lower-bound candidate in each node of the search tree (while this is typically possible only in leaves in standard MILP search), (2) BnB algorithm can incrementally improve the precision of approximation of bilinear terms (thus improving the expected outcome of a maxmin strategy) and at the same time fix the values of binary variables.

The main disadvantage of BnB is that the LP is linear in the size of the game and thus the algorithm can refine bilinear terms in parts of the game that may not be relevant for the final solution. To overcome this disadvantage, an incremental strategy-generation technique can be employed.

DOUBLE ORACLE FOR PERFECT RECALL EFGS

The double oracle algorithm solving perfect recall EFGs (DOEFG, [2]) is the 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 the players and then incrementally expand this restricted game by allowing new actions. The restricted game is solved as a standard zero-sum extensive-form game using the sequence-form linear program [10, 20]. Afterward, best response algorithms search the original unrestricted game to find new sequences to add to the restricted game for each player. The algorithm terminates when the best response calculated on the unrestricted game provides no improvement to the solution of the restricted game for either of the players.

DOEFG uses two ideas in order to guarantee a linear number of iterations in the size of the game tree: (1) the algorithm assumes that players play some pure default strategy outside the restricted game (e.g., playing the first action in each information set given some orderings), (2) temporary utility values are assigned to leafs in the restricted game that correspond to an inner node in the original unrestricted game (so-called temporary leaf), which form an upper bound on the expected utility.

DOUBLE ORACLE BNB FOR IMPERFECT RECALL EFGS

In this section, we introduce our main algorithm, denoted as DOBnB, combining ideas of BnB and DOEFG. Adapting the ideas of DOEFG for games with imperfect recall poses several challenges that we need to solve. First, to solve the restricted game means to compute maxmin strategy for player 1. However, solving the restricted game does not provide us with a valid upper bound needed in the BnB. Second, solving the restricted game requires calling BnB search that iteratively refines the approximation of bilinear terms instead of solving a single (or a pair of) LPs in DOEFG for perfect recall games. Our algorithm thus makes an integration of two iterative methods and decides when to expand the restricted game and when to refine the approximation of bilinear terms already in the restricted game.

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

Algorithm 2: DOBnB algorithm

```
input      : Initial LP relaxation $LP_0$ of Upper Bound
             MILP, Initial restricted game $G$
output     : $\epsilon$-optimal strategy for the maximizing player
parameters: Bound on maximum error $\epsilon$, bound $P_{max}$ for
             bilinear term precision approximation
1  fringe ← $(LP_0, -\infty, \infty)$
2  opt ← $(LP_0, -\infty, \infty)$
3  while fringe ≠ ∅ do
4      $(LP, lb, ub) \leftarrow \arg\max_{n \in \text{fringe}} n.\text{ub}$
5      fringe ← fringe \ $(LP, lb, ub)$
6      if opt.lb ≥ $ub$ then
7          return ReconstructStrategy(opt)
8      if opt.lb < $lb$ then
9          opt ← $(LP, lb, ub)$
10     if $ub - lb \leq \epsilon$ then
11         return ReconstructStrategy(opt)
12     if FromSmallerG($n$, $G$) then
13         $(LP, lb, ub)$ ← Resolve($(LP, lb, ub)$, $G$)
14         Add($(LP, lb, ub)$, $G$)
15     else if CanBeExpanded($G$, $LP$) then
16         $G$ ← Expand($G$, $LP$)
17         $(LP, lb, ub)$ ← Resolve($(LP, lb, ub)$, $G$)
18         Add($(LP, lb, ub)$, $G$)
19     else
20         $(I_1, a)$ ← SelectAction($n$)
21         AddSuccessors($n$, $I_1$, $a$, $P_{max}$, $G$)
22 function Add($(LP, lb, ub), G$)
23     while PendingToAdd($G$, $LP$) do
24         $G$ ← AddPending($G$, $LP$)
25         $(LP, lb, ub)$ ← Resolve($(LP, lb, ub)$, $G$)
26     $fringe \leftarrow (LP, lb, ub)$
```

In Algorithm 2 we present the extension of the DOBnB algorithm. The algorithm starts with the empty restricted game G. Lines 1 to 11 are the same as in the BnB algorithm. There are two differences: (1) all the nodes use the current restricted game G, and (2) before we add any node to the fringe, we need to make sure that all the potential deviations of the maximizing player are in G using function Add (lines 22 to 26, see Updating the Restricted Game for details).

In every iteration of the DOBnB, we first check whether the bounds of the current node were computed in some smaller restricted game than the current G (line 12). If yes we recompute the bounds on the current restricted game (line 13) to make sure that the bounds are as precise as possible and return the node to the fringe (line 14). Else, if bounds come from the same game as the current restricted game, the algorithm checks whether G can be expanded (line 15, see Updating the Restricted Game). If yes, we expand G, resolve and return the node to the fringe (lines 16 to 18). Otherwise, if G cannot be expanded, the algorithm continues in the same way as BnB. It generates new nodes by selecting bilinear terms corresponding to some action from the current restricted game G (line 20), increases their precision and adds new variables and constraints into the LP that further restrict the maxmin strategy. Next, it adds the resulting nodes to the fringe (line 21) in the same way as in BnB (note that we add the nodes to the fringe using the Add function (lines 22 to 26)).

The Restricted Game

This section formally defines the restricted game $G' = (\mathcal{N}, \mathcal{H}', \mathcal{Z}', \mathcal{A}', p, u^{UB}, C, \mathcal{I}')$ as a subset of the original unrestricted game $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, p, u, C, \mathcal{I})$.

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

$$\Sigma'_i = \{\sigma_i \in \Phi'_i | \exists \sigma_{-i} \in \Phi'_{-i}\ \exists h \in \mathcal{H} \\ \forall j \in \mathcal{N} : seq_j(h) = \sigma_j\}, \forall i \in \mathcal{N}. \quad (5)$$

The sets $\mathcal{H}'$, $\mathcal{A}'$ and $\mathcal{I}'$ are the subsets of $\mathcal{H}$, $\mathcal{A}$ and $\mathcal{I}$ reachable when playing sequences from Σ'. The set of leaves in G' is a union of leaf nodes of G present in G' and inner nodes from G that do not have a valid continuation in Σ'

$$\mathcal{Z}' = (\mathcal{Z} \cap \mathcal{H}') \cup \{h \in \mathcal{H}' \setminus \mathcal{Z} : A'(h) = \emptyset\}. \quad (6)$$

We refer to the members of the set $\mathcal{Z}' \setminus \mathcal{Z}$ as temporary leaves and define a modified utility value u_1^{UB} such that the maxmin value of the restricted game is higher than the maxmin value of the original game. Formally,

$$u_1^{UB}(z) = \max_{\beta_2 \in \mathcal{B}_2^{LP} \cup \beta_2^{BR}} u_1^z(BR^z(\beta_2), \beta_2), \forall z \in \mathcal{Z}' \setminus \mathcal{Z}, \quad (7)$$

where β_2^{BR} is a best response of player 2 in G to be added in this iteration of the algorithm; $BR_1^z(\beta_2)$ is a set of the best responses of player 1[2], when starting in z against β_2; $u_1^z(\beta_1, \beta_2)$ is the expected value, when playing according to β_1 and β_2 and starting in z. Finally, $\mathcal{B}_2^{LP}$ is the set of all possible best responses of player 2 taken from the current LP by finding actions corresponding to active Constraint (1h). Notice that the u_1^{UB} might differ in every iteration of the algorithm. This provides guarantees that u_1^{UB} is an upper bound against all possible reactions of the minimizing player. Additionally, $\forall z \in \mathcal{Z} \cap \mathcal{Z}' u_1^{UB}(z) = u_1(z)$.

Note that if not stated otherwise, when we operate with a strategy from the restricted game in the whole unrestricted game, we automatically assume that it is extended by a default strategy as in DOEFG.

Updating the Restricted Game.

In this section we dicuss the oracles used in the DOBnB and the way their results are used to expand the restricted game (lines 16 and 24 in Algorithm 2).

Player 1 oracle. By solving the restricted game we compute a non-exploitable strategy for player 1. Therefore, we can use a best response algorithm as an oracle for player 2. In every iteration we compute $\beta_2^{BR} \in BR_2(\beta_1)$ in G, where β_1 is the strategy of player 1 computed by DOBnB in the current node. We extend Φ_2 by all the valid continuations of $\sigma_2 \in \Phi_2$ by actions in β_2^{BR} and update Σ' accordingly.

Player 2 oracle. The restricted game does not produce a non-exploitable strategy of player 2. This poses the most significant challenge in devising the oracle for the maximizing player since we cannot use a best response algorithm for adding sequences for player 1. Instead, we use a set of pending states

$$\mathcal{H}_p = \{h \in \mathcal{H} \setminus \mathcal{H}' | \exists h' \in \mathcal{H}'_1 \exists a \in A(h') : h'a = h\}, \quad (8)$$

as a set of possible extensions of the restricted game by taking actions in states of the maximizing player 1. We take a subset $\mathcal{H}'_p \subseteq \mathcal{H}_p$ such that all $h \in \mathcal{H}'_p$ are reachable by some $\beta'_2 \in \mathcal{B}_2^{LP} \cup \beta_2^{BR}$, where $\beta_2^{BR} \in BR_2(\beta_1)$ is the best response suggested by the minimizing player oracle for the current restricted game. By $\mathcal{H}^*_p$ we denote a subset of $\mathcal{H}'_p$, where for all $h \in \mathcal{H}^*_p$ holds that $u_1^{UB}(h) \geq u_1^{LB}(h')$ for $u_1^{LB}(h') = \min_{\beta'_2 \in \mathcal{B}_2^{LP}} u_1^{h'}(\beta_1, \beta'_2)$, where h' is the parent of h. When expanding, we add to the restricted game all the sequences leading to all $h \in \mathcal{H}^*_p$.

The function PendingToAdd returns *true* if $\mathcal{H}^*_p$ is nonempty, *false* otherwise. AddPending adds all the sequences suggested by the oracle for the maximizing player to the restricted game. CanBeExpanded checks whether the oracle of any player suggests any sequence to be added to the restricted game. Finally Expand adds all the sequences suggested by both oracles.

Theoretical Properties

Lemma 2. *$u_1^{UB}(z)$ forms an upper bound on the expected value player 1 can guarantee in all the $z \in \mathcal{Z}'$ in the original game against all the $\mathcal{B}^{LP'}$ obtained when solving the LP corresponding to the restricted game after the next expansion.*

Proof. $u_1^{UB}(z)$ in all $z \in \mathcal{Z}'$ considers all the best responses from the current LP $\mathcal{B}^{LP}$ and the β_2^{BR} obtained from the minimizing player oracle, hence we are sure that $u_1^{UB}(z) \geq \max_{\beta'_2 \in \mathcal{B}^{LP'}} u_1^z(BR^z(\beta'_2), \beta'_2)$, where $\mathcal{B}^{LP'}$ are all the possible best responses occuring in the LP solved after the restricted game expansion. This holds since the best responses can only be replaced by the β_2^{BR} or removed. □

Lemma 3. *All the nodes in the fringe in Algorithm 2 have a valid lower and upper bound on the solution with corresponding precision restrictions in the original game.*

Proof. The lower bound is valid, since it is computed as $u_1(\beta_1, \beta_2)$, where β_1 is the current solution of the corresponding LP applied to the current restricted game G', extended by the default strategy and $\beta_2 \in BR_2(\beta_1)$ in the whole game. If β_1 is not optimal given the current restrictions, this value is smaller than optimum, if β_1 is optimal, it is equal to the optimum with given precision restrictions.

To show that the upper bound is valid, first, notice that we make sure that we add a node to the fringe in Algorithm 2 only when we are sure that player 1 cannot increase his value by deviating outside of the restricted game. This is done in the function Add by adding $h \in \mathcal{H}^*_p$ and resolving the LP, until $\mathcal{H}^*_p$ is empty. Finally, since for all the $z \in \mathcal{Z}'$ holds that $u_1^{UB}(z)$ is the upper bound on the expected value the maximizing player can get in z (Lemma 2), we are sure that the upper bound obtained in this setting is higher or equal to the upper bound obtained in the whole game with the same precision restrictions. □

[2]Since the maximizing player does not have to have A-loss recall, we compute the best response in the coarsest perfect recall refinement of the solved unrestricted game for player 1. This allows us to efficiently obtain an upper bound on the correct value ($u_1^{UB}(h)$ is, therefore still an upper bound on the value obtainable in h).

Theorem 1. *If the* BnB *algorithm is guaranteed to return ϵ-optimal solution for some precision parameters, the* DOBnB *returns ϵ-optimal solution for the same precision parameters.*

Proof. When the upper and lower bound are at most ϵ distant, we are sure that we have found an ϵ-optimal solution (Lemma 3). We are guaranteed to reach such node in the space of precision restrictions since we never prune it away (again from Lemma 3). When we reach the precision restrictions guaranteeing an ϵ-optimal solution in the full game, we might have an upper bound which is higher due to the fact that we reach temporary leafs with overestimated upper bounds in the restricted game. In this situation, however, we are guaranteed to continue expanding the game and therefore increasing the precision of the upper bound, until the upper bound reaches the ϵ distance from the lower bound (condition on line 15 in Algorithm 2). This must happen after a finite number of steps since the algorithm only expands the restricted game and reaching ϵ distance is guaranteed by the correctness of BnB [1] when the restricted game is equal to the original game. $\square$

EXPERIMENTS

In this section, we provide an experimental evaluation of the DOBnB against the BnB algorithm and the baseline MILP (Base) implementation which iteratively solves the MILP resulting from the bilinear program approximation and iteratively increases the approximation precision. The main experiments are conducted on a set of Random games, however, we also report results on an imperfect recall search game and an imperfect recall variant of Tic-Tac-Toe. All algorithms were implemented in Java, each algorithm uses a single thread, 8 GB memory limit and we use IBM ILOG CPLEX 12.6.2 to solve all LPs/MILPs.

Random Games. Since there is no standardized collection of benchmark EFGs, we use randomly generated games in order to obtain statistically significant results. We randomly generate a perfect recall game with varying branching factor and fixed depth of 6. To control the information set structure, we use observations assigned to every action – for player i, nodes h with the same observations generated by all actions in history belong to the same information set. In order to obtain imperfect recall games with a non-trivial information set structure, we run a random abstraction algorithm which merges information sets according to parameter p ($p = 0$ means no merges, $p = 1$ means that all information sets with the same action count, which do not cause absent-mindedness are merged). We generate a set of experimental instances by varying the branching factor and the parameter p. Such games are rather difficult to solve since (1) information sets can span multiple levels of the game tree (i.e., the nodes in an information set often have histories with differing sizes) and (2) actions can easily lead to leafs with very differing utility values. The abstraction always has A-loss recall for the minimizing player.

Search Game. Our second domain is an instance of search (or pursuit-evasion) games, which are commonly used for evaluating incremental algorithms [18]. The game is played on a directed graph between attacker and defender. The attacker tries to cross from a starting node to his destination. The defender operates two units, each moving only in a restricted part of the graph, trying to intercept the evader by capturing him in a node. The players move simultaneously. The only information available to the defender is the position of both of his units without remembering the history of moves leading there. The evader knows only the sequence of his actions in the past. It is a zero-sum game, where the attacker obtains utility 1 for reaching his destination and defender obtains utility 1 for intercepting the attacker. If a given number of moves is depleted without either of the events happening, the game is considered a draw and both obtain utility 0. We assume the defender to be the maximizing player.

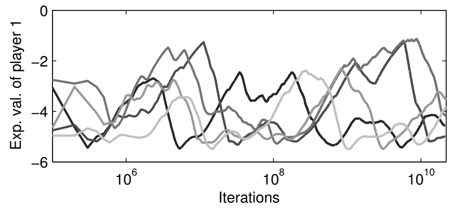

Figure 2: The exploitability of the average strategy computed by the CFR with outcome sampling (y-axis) with increasing number of iterations (logarithmic x-axis) for 5 different seeds.

Phantom Tic-Tac-Toe. The last domain is a blind variant of the Tic-Tac-Toe (e.g., used in [2]). The game is played on a 3×3 board, with standard rules except that the players observe only a partial state of the board and do not remember the history of actions. Players do not observe the position of opponent's stones. If a player tries to place his stone on a position that is occupied by opponent's stone the player learns this information and plays again. This game has A-loss recall for both players since the players forget only information about their own moves in the past.

Results

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

CFR. We first empirically demonstrate the performance of the outcome-sampling version of CFR [15] on the example game from Figure 1. Figure 2 depicts the expected utility of the average strategy computed by the CFR against its best response (i.e., the exploitability of the average strategy; logarithmic x-axis shows the number of iterations, the y-axis shows the exploitability for player 1, every line represents one run for a given seed). The algorithm does not converge to any fixed strategy, moreover, the exploitability differs significantly from the maxmin value of -1 for player 1. Therefore, we focus only on the comparison of the algorithms that guarantee the convergence to the maxmin strategies in the experiments on larger games. Note that vanilla CFR (see, e.g. [15] page 22) does not work either, since for example when initialized to uniform strategy, player 1 will never change this strategy since the expected values after his actions are always equal.

Random Games. In Figure 3 we present the runtime results in seconds obtained on random games. Every graph depicts the cumulative relative number of instances (y-axis) solved under a given time limit (logarithmic x-axis). The columns contain results for random games with varying p, the first row for branching factor 3, second for branching factor 4. The runtime of the algorithms was limited to 2 hours on every instance, the relative number of instances terminated after this limit is reported in the bars labeled *cutoff*.

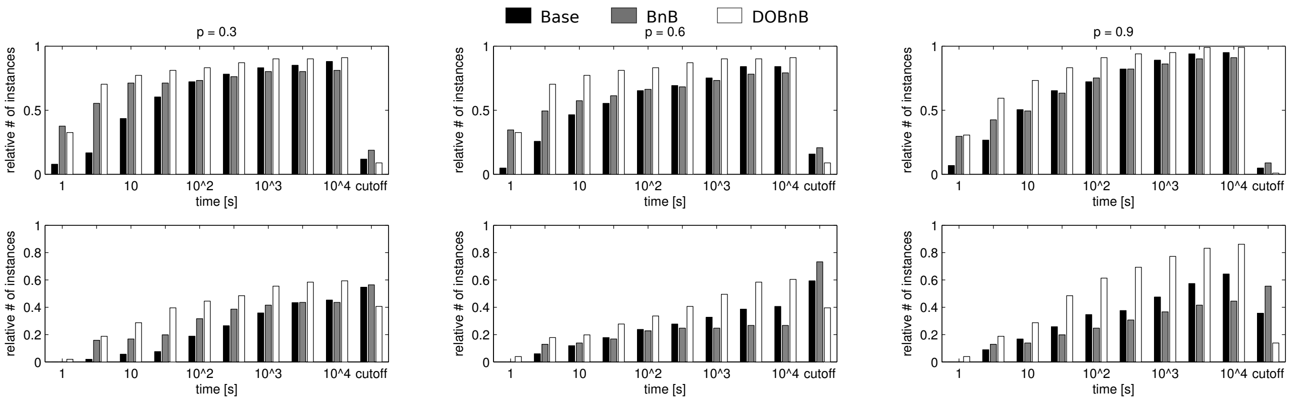

Figure 3: Results showing the relative cumulative number of instances (y-axis) solved under a given time limit (x-axis) and the relative amount of instances terminated due to the exceeded runtime in bars labeled *cutoff*. Rows contain results for branching factor 3 and 4, columns show results for $p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

Table 1: Average relative amount of sequences for maximizing and minimizing player respectively, added to the restricted game by DOBnB.

b.f. \ p	0.3	0.6	0.9
3	0.468, 0.231	0.598, 0.247	0.689, 0.248
4	0.585, 0.165	0.731, 0.163	0.780, 0.192

The results show that the DOBnB outperforms the other two algorithms across all the settings and achieves on average at least an order of magnitude better performance than the second best BnB algorithm, e.g., in case of branching factor 4 and $p = 0.3$ the average runtime on the instances solved by all the algorithms was $425s \pm 218s$, $337s \pm 181s$ and $24s \pm 12s$ for Base, BnB and DOBnB, respectively. This is due to the fact that the DOBnB limits adjustments to approximation precision to the relevant parts of the game tree present in the restricted game while keeping the underlying LP smaller. Additionally, we can see a significant decrease in the number of instances not solved in a given 2 hour limit, compared to Base and BnB. Note that the random games form an unfavorable scenario for all the presented algorithms since the construction of the abstraction is completely random, which makes conflicting behavior in merged information sets common. As we can see, however, even in these scenarios the DOBnB is capable of solving the majority of instances with branching factor 4 which have $\sim$ 3000 nodes in under 2 hours.

In Table 1 we present the average relative amount of sequences for each player needed by DOBnB to solve the random games for each setting. The relative amount of sequences needed by the minimizing player is consistently smaller because the restricted game is build to compute maximizing player's robust strategy, while the minimizing player only plays best responses during the computation. Even though the size of the restricted game remains similar across all values of p, we observe an increase in the relative size, since the number of sequences decreases as p increases.

Search game. In case of Search game, the DOBnB was able to solve a game with 6 moves allowed for each player (with 863126 states, 949 sequences for the attacker and 19291 sequences for the defender) using 19.5% of sequences for the defender and 9.7% sequences for the attacker in 5 minutes, while the rest of the algorithms did not finish in 48 hours.

Phantom Tic-Tac-Toe. The DOBnB was capable of solving the Phantom Tic-Tac-Toe in under 3 hours while building only 0.6% and 0.05% of sequences for the first and second player respectively (it has $\sim 10^9$ states, $\sim 1.3 \cdot 10^6$ and $\sim 4.4 \cdot 10^6$ sequences for the first and second player). The rest of the algorithms needs to build the whole game tree, which is not feasible for this game. This result shows that DOBnB is capable of outperforming the current state-of-the-art algorithms assuming perfect recall since the most successful of these algorithms is capable of solving the Phantom Tic-Tac-Toe in 4.88 hours [2].

CONCLUSION

We describe the first scalable algorithm for approximating maxmin strategies in imperfect recall games. Our approach is a novel combination of two iterative algorithms: an incremental strategy generation and a branch-and-bound search. The experimental evaluation shows that our algorithm can solve difficult randomly generated games and, more importantly, our algorithm can solve an abstracted variant of a large game faster than the algorithms operating on the unabstracted perfect-recall variant.

Our algorithm allows new directions of research on imperfect recall abstractions in EFGs and thus can be very valuable in understanding compact representations of sequential games. The algorithm can be modified to find the best imperfect recall abstractions in EFGs. Similarly, it can be adapted to operate on existing compact representations (e.g., Multi-Agent Influence Diagrams [11]) to further improve the scalability and allow real-world applications. Finally, it provides the baseline for evaluation of the quality of strategies produced by CFR in abstracted imperfect recall games that we plan to evaluate in the future work.

Acknowledgments

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

REFERENCES

[1] B. Bosansky, J. Cermak, K. Horak, and M. Pechoucek. Computing Maxmin Strategies in Extensive-Form Zero-Sum Games with Imperfect Recall. *ICAART*, 2017.

[2] B. Bosansky, C. Kiekintveld, V. Lisy, and M. Pechoucek. An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information. *Journal of Artificial Intelligence Research*, 51:829–866, 2014.

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

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

[5] G. J. Gordon. No-regret algorithms for online convex programs. In *NIPS*, pages 489–496, 2006.

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

[7] M. Kaneko and J. J. Kline. Behavior Strategies, Mixed Strategies and Perfect Recall. *International Journal of Game Theory*, 24:127–145, 1995.

[8] J. J. Kline. Minimum Memory for Equivalence between Ex Ante Optimality and Time-Consistency. *Games and Economic Behavior*, 38:278–305, 2002.

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

[10] D. Koller, N. Megiddo, and B. von Stengel. Efficient Computation of Equilibria for Extensive Two-Person Games. *Games and Economic Behavior*, 14(2):247–259, 1996.

[11] D. Koller and B. Milch. Multi-agent influence diagrams for representing and solving games. *Games and Economic Behavior*, 45(1):181–221, 2003.

[12] S. Kolodziej, P. M. Castro, and I. E. Grossmann. Global optimization of bilinear programs with a multiparametric disaggregation technique. *Journal of Global Optimization*, 57(4):1039–1063, 2013.

[13] C. Kroer and T. Sandholm. Extensive-Form Game Abstraction with Bounds. In *Proceedings of the fifteenth ACM conference on Economics and computation*, pages 621–638. ACM, 2014.

[14] C. Kroer and T. Sandholm. Imperfect-recall abstractions with bounds in games. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 459–476. ACM, 2016.

[15] M. Lanctot. *Monte Carlo sampling and regret minimization for equilibrium computation and decision-making in large extensive form games.* University of Alberta, 2013.

[16] M. Lanctot, R. Gibson, N. Burch, M. Zinkevich, and M. Bowling. No-Regret Learning in Extensive-Form Games with Imperfect Recall. In *Proceedings of the 29th International Conference on Machine Learning (ICML 2012)*, pages 1–21, 2012.

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

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

[19] J. Čermák and B. Bošanský. Towards Solving Imperfect Recall Games. In *Proceedings of AAAI Computer Poker Workshop*, 2017.

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

[21] P. C. Wichardt. Existence of nash equilibria in finite extensive form games with imperfect recall: A counterexample. *Games and Economic Behavior*, 63(1):366–369, 2008.

[22] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret Minimization in Games with Incomplete Information. *Advances in Neural Information Processing Systems (NIPS)*, 20:1729–1736, 2008.