Double-oracle Algorithm for Computing an Exact Nash Equilibrium in Zero-sum Extensive-form Games

Branislav Bošanský[1], Christopher Kiekintveld[2], Viliam Lisý[1], Jiří Čermák[1], Michal Pěchouček[1]
[1]Agent Technology Center, Dept. of Computer Science, FEE, Czech Technical University in Prague
{bosansky, lisy, cermak, pechoucek}@agents.fel.cvut.cz
[2]Computer Science Department, University of Texas at El Paso
cdkiekintveld@utep.edu

ABSTRACT

We investigate an iterative algorithm for computing an exact Nash equilibrium in two-player zero-sum extensive-form games with imperfect information. The approach uses the sequence-form representation of extensive-form games and the double-oracle algorithmic framework. The main idea is to restrict the game by allowing the players to play only some of the sequences of available actions, then iteratively solve this restricted game, and exploit fast best-response algorithms to add additional sequences to the restricted game for the next iteration. In this paper we (1) extend the sequence-form double-oracle method to be applicable on non-deterministic extensive-form games, (2) present more efficient methods for maintaining valid restricted game and computing best-response sequences, and finally we (3) provide theoretical guarantees of the convergence of the algorithm to a Nash equilibrium. We experimentally evaluate our algorithm on two types of games: a search game on a graph and simplified variants of Poker. The results show significant running-time improvements compared to the previous variant of the double-oracle algorithm, and demonstrate the ability to find an exact solution of much larger games compared to solving full linear program for the complete game.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: [Multiagent systems]

Keywords

game theory, extensive-form games, exact Nash equilibrium, algorithms

1. INTRODUCTION

Computational game theory is one of the fundamental methods for analyzing multi-agent systems using a formal mathematical framework. Improved algorithms for solving different classes of games have led to an increasing number of successful applications of game theory in areas ranging from auctions and trading agents [14], security [11], to Poker [15], and many others. We focus on making fundamental algorithmic advances for solving large instances of an important general class of games: two player, zero-sum extensive-form games (EFGs) with imperfect information. This class of games captures sequential interactions between two adversarial players in situations where they have uncertainty about the state of the world or the other player's actions. Many well known games are instances of this class, including Poker, Kreigspiel (blind chess), and various security and pursuit evasion games.

Solving these games is a very difficult computational task for problems of realistic size; hence, approximation methods are commonly used to solve them in practice, including: regret minimization techniques (e.g., CFR) [16], later improved with sampling methods [6]; Nesterov's Excessive Gap Technique (EGT) [2]; or variants of Monte-Carlo tree search algorithms applied on imperfect-information games (e.g., see [9]). The first two types of algorithms guarantee convergence to approximate ϵ-Nash equilibrium, while the third method has no theoretical guarantees for imperfect-information games, but it can produce good strategies in practice [9].

We focus on finding exact solutions, though our method could be used to develop approximate algorithms as well. The leading exact algorithm uses the *sequence-form* representation [5, 12] and linear programming optimization techniques to find a solution. Solving the linear program (LP) requires significantly more memory and time than the approximate methods. However, research on decomposition methods for solving large-scale optimization problems provides a framework for developing iterative approaches that do not need to enumerate the full problem. These techniques are known in the game theory as *oracle algorithms* [8] and they have been successfully used to solve large normal-form games [3].

In this paper we use the same iterative principle to develop a new exact algorithm for two-player, zero-sum EFGs with imperfect information based on the sequence-form double-oracle algorithm recently introduced by Bosansky et al. [1]. The main idea of the algorithm is to create a restricted game, in which the players have a limited number of allowed sequences, and then iteratively expand the restricted game by adding best-response sequences to the solution of the current restricted game. In the worst case, this approach may need to enumerate all possible sequences, but in typical cases a solution can be found by exploring a small fraction of the strategy space. There are two other approaches that use similar oracle-based decompositions for EFGs, but which operate on the complete strategy space of the game. The first was introduced by McMahan et al. in [7] for the more general class of convex games; this method searches a space of linear combinations of multiple mixed strategies on the full game tree. A similar approximative method using strategies defined on the complete game tree was used in [15] to create a competitive Poker bot. We build on ideas of the sequence-form double-oracle, since it exploits the tree structure of an EFG by using the compact sequence form so that the strategies do not need to be represented over the complete game tree. While this is a strong advantage, working with the sequence form also introduces

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013),* Ito, Jonker, Gini, and Shehory (eds.), May 6–10, 2013, Saint Paul, Minnesota, USA.

complexities in maintaining a valid restricted game and ensuring convergence to a Nash equilibrium.

The contributions of this paper include several major improvements of the sequence-form double-oracle method: (1) we extend the double-oracle framework to model more general games of imperfect information with moves by Nature, (2) we dramatically improve the performance of the algorithm by introducing a novel technique for maintaining a valid restricted game without adding unnecessary sequences to the restricted game, (3) we improve best-response calculations for sequences using a set of domain-independent pruning techniques, and (4) we provide formal theoretical analysis to guarantee that the algorithm converges to a Nash equilibrium. Finally, we present an experimental evaluation of our algorithm on two very different classes of games: a search game based on a pursuit evasion scenario and a game based on simplified variants of Poker. For search games, which have small support in the equilibrium, our algorithm is much faster than both computing the LP for the full game, and the previous sequence-form double-oracle algorithm. On the poker variants, which generally have larger support, the algorithm does not show speedups in all cases compared to the full LP, but it still has much lower memory requirements so it is possible to solve larger games using our approach.

2. TECHNICAL BACKGROUND

Adversarial situations with sequential moves can be modeled as extensive-form games (EFGs) visually represented as game trees. EFGs are sufficiently general to model stochastic changes in the environment, private information of players, and limited observability of the actions of the opponent. We study zero-sum two-player games, where a special *Nature* player is used to model stochastic events. An EFG is formally defined as follows (the outline of the main symbols is depicted in Table 1): N is a set of two players $N = \{1, 2\}$, we use i to refer to one of the two players (either 1 or 2), and $-i$ to refer to the opponent of i. H denotes a finite set of the *nodes* in the game tree. Each node corresponds to a unique *history* of actions taken by all players and Nature from the root of the game; hence, we use the terms history and node interchangeably. We denote $Z \subseteq H$ as the set of all *terminal nodes* of the game. For each terminal node $z \in Z$ we define a *utility function* for each player i ($u_i : Z \to \mathbb{R}$). Our games are zero sum, so $u_i(z) = -u_{-i}(z)$ holds for all $z \in Z$. $A(h)$ represents the set of actions available in node $h \in H$, we denote $ha = h' \in H$ to be a node h' reached from node h by performing an action $a \in A(h)$, and we say that h is a *prefix* of h'.

The function $P : H \to N \cup \{c\}$ assigns each node to a player who takes the action in the node; c denotes the Nature player that selects in the node an action based on a known fixed probability distribution. We represent the uncertainty that each player i has using *information sets* I_i which form a partition over the nodes assigned to player i ($P(h) = i$). That is, every node in the game tree belongs to exactly one information set, and the player does not know which of the nodes in the information set he is currently in when playing the game. All nodes h in a single information set $I_i^k \in I_i$ have the same set of possible actions $A(h)$. We assume *perfect recall*, which means that all nodes in any information set I_i^k have the same history of actions for player i (i.e., players cannot misremember their own actions). We use function $C : H \to [0, 1]$ to denote the probability of reaching node h if both players do their best to reach it, which is calculated as a product of probabilities of all actions of the Nature player in history h.

2.1 Sequence Form LP Method

Solving a game for a Nash equilibrium involves computing a profile of strategies (a strategy for each player) in which each player plays the best response to the strategies of the other players. Formally, let Π_i be the set of *pure strategies* for player i, i.e., a selection of exactly one action for each information set. A mixed strategy is a probability distribution over the set of all pure strategies of a player and we denote by Δ_i the set of all mixed strategies of player i. For any pair of strategies $\delta \in \Delta$ we denote $u_i(\delta) = u_i(\delta_i, \delta_{-i})$ the expected outcome of the game for player i. A best response of player i to the opponent's strategy δ_{-i} is a strategy δ_i^{BR} such that $u_i(\delta_i^{BR}, \delta_{-i}) \geq u_i(\delta_i', \delta_{-i})$ for all strategies $\delta_i' \in \Delta_i$. A strategy profile $\delta = (\delta_1, \delta_2)$ is in a *Nash equilibrium* if for each player i it holds that δ_i is a best response to δ_{-i}.

Extensive-form games with perfect recall can be represented using the compact *sequence form* [5, 12]. A *sequence* σ_i is an ordered list of actions of player i in a history h. The set of all possible sequences in a game for player i is denoted by Σ_i and the set of sequences for all players is $\Sigma = \Sigma_1 \times \Sigma_2$. A sequence $\sigma_i \in \Sigma_i$ can be extended by a single action a taken by a player i, denoted as $\sigma_i a = \sigma_i'$. In games with perfect recall, an action a taken to extend sequence σ_i uniquely identifies an information set I_i^k of player i. We use the function $\mathcal{I}_i(\sigma_i')$ to denote the information set in which the last action (i.e., a in this case) was executed. Similarly, all nodes in an information set I_i^k share the same sequence of actions of player i and we use $\mathsf{seq}_i(I_i^k)$ to denote this sequence. We overload the notation and also use $\mathsf{seq}_i(h)$ to denote the sequence of actions of player i leading to node h, and $\mathsf{seq}_i(H') \subseteq \Sigma_i$, where $\mathsf{seq}_i(H') = \bigcup_{h' \in H'} \mathsf{seq}_i(h')$ for some $H' \subseteq H$. Finally, we define function $g_i : H \to \mathbb{R}$ that extends the utility function to all nodes by setting $g_i(h) = u_i(h) \cdot C(h)$ if $h \in Z$ and $g_i(h) = 0$ if h is not a terminal node ($h \notin Z$).

We use function $\omega : \Sigma \to \mathcal{P}(H)$ to identify the set of nodes that can be reached using a fixed combination of sequences of players. More precisely, nodes in $\omega(\sigma_i, \sigma_{-i})$ can be reached by sequential execution of actions in the given order, stopping when either the next action is not valid in the reached node, or a leaf is reached. In non-deterministic games the execution of a single combination of sequences for two players can reach different nodes depending on the actions of Nature. Thus, function ω returns the set of *all* nodes in H that can be reached by execution of sequences for two players.

Using the sequence form we can find a Nash equilibrium of a two-player zero-sum extensive-form game using a linear program (LP) (e.g., see [10, p. 135]). This is enabled by an equivalent compact representation of mixed strategies of players in a form of *realization plans*. A realization plan of a sequence σ_i is a probability that player i will play this sequence of actions under the assumption that the opponent will choose compatible actions that reach the information sets in which actions specified in the sequence σ_i are applicable. We denote realization plans of player i by function $r_i : \Sigma_i \to \mathbb{R}$, and they can be computed using a LP:

$$\min v_{\mathcal{I}_i(\emptyset)}$$

$$v_{\mathcal{I}_i(\sigma_i)} - \sum_{I_i^k \in I_i : \mathsf{seq}_i(I_i^k) = \sigma_i} v_{I_i^k} \geq \sum_{\sigma_{-i} \in \Sigma_{-i}} \sum_{h \in \omega(\sigma_i, \sigma_{-i})} g_i(h) \cdot r_{-i}(\sigma_{-i}) \qquad \forall \sigma_i \in \Sigma_i \tag{1}$$

$$r_{-i}(\emptyset) = 1 \tag{2}$$

$$\sum_{\forall a \in A(I_{-i}^k)} r_{-i}(\sigma_{-i} a) = r_{-i}(\sigma_{-i}) \qquad \forall \sigma_{-i} = \mathsf{seq}_{-i}(I_{-i}^k),\ I_{-i}^k \in I_{-i} \tag{3}$$

$$r_{-i}(\sigma_{-i}) \geq 0 \qquad \forall \sigma_{-i} \in \Sigma_{-i} \tag{4}$$

A separate LP is constructed for computing the strategy of each player. It uses variables $v_{I_i^k}$ that represent the expected utility of the player i in an information set I_i^k, and variables r_{-i} represent the strategy of the opponent in the form of realization plan. The

H	game-tree nodes / histories
$Z \subseteq H$	leafs / terminal states
$\sigma'_i \in \Sigma'_i$	sequences of player i in the restricted game
$\phi'_i \in \Phi'_i$	full-length sequences of player i added by the best-response sequence algorithm
$\omega : \Sigma \to \mathcal{P}(H)$	all nodes reached by the maximal sequential execution of actions in a pair of sequences
$r_i : \Sigma_i \to \mathbb{R}$	realization plan of player i for a sequence
$C : H \to \mathbb{R}$	probability of reaching a node w.r.t. Nature play
$g_i : H \to \mathbb{R}$	extension of the utility function to all nodes; $g_i(h) = u_i(h) \cdot C(h)$ if $h \in Z$, $g_i(h) = 0$ otherwise
π_i	implicit default pure strategy for player i
seq_i	sequence(s) of actions of player i leading to a node / a set of nodes / an information set
$\mathcal{I}_i : \Sigma_i \to I_i$	an information set, in which the last action of sequence was executed

Table 1: Outline of the main used symbols.

first equation (1) ensures the maximization of the expected utility of player i for each information set, while the opponent is trying to minimize the utility by selecting the optimal realization plan, which is constrained by equations (2–4).

3. DOUBLE-ORACLE METHOD

In this section we describe in detail the sequence-form double-oracle algorithm for EFGs and our main contributions. The basic structure of the algorithm consists of iterating through three main steps until convergence: (1) create a restricted game by limiting the set of sequences that each player is allowed to play; (2) compute a pair of Nash equilibrium strategies in this restricted game; (3) for each player, compute a best response strategy against the equilibrium strategy of the opponent, which may be *any* sequence in the complete game. The best response sequences found in step 3 are added to the restricted game and allowed in the next iteration. The algorithm terminates if the value of the best-response sequences against the equilibrium strategies does not improve the value of the equilibrium strategies in the restricted game.

3.1 Definitions

We now introduce the concept of *full-length sequences* that lead to a terminal state of the game, and which we denote $\Phi \subseteq \Sigma$. The *restricted game* is defined by a limited set of allowed sequences for each player, which are used to construct the sequence-form linear program (LP). We denote the set of restricted sequences as $\Sigma' \subseteq \Sigma$ and we overload the notation and also use Σ' to refer to the restricted game corresponding to this set of sequences. The following section explains how this subset of sequences can be constructed based on the limited subset of full-length sequences $\Phi' \in \Phi$ using the best response methods.

Our algorithm also needs to use partial strategies from the restricted game in the context of the complete game. We extend the strategies from the restricted game to be defined in the complete game using the concept of a *default pure strategy* for player i, denoted $\pi_i \in \Pi_i$. This strategy is an arbitrary, fixed, pure strategy that can be implicitly defined for all information sets of player i (e.g., take the first action from each $A(h)$). The default strategy is used only where the strategy from the restricted game is not defined in the complete game.

3.2 Upper-bound Validity Algorithm

In [1] the authors show that a naïve implementation of the sequence-form double-oracle method may result in an incorrect solution. If we simply add best-response sequences to the restricted game, the game can get malformed because of incompatibilities among the sequences, resulting in incorrect solutions. We now describe the situations that lead to these inconsistencies in more detail, and introduce a novel solution to this problem.

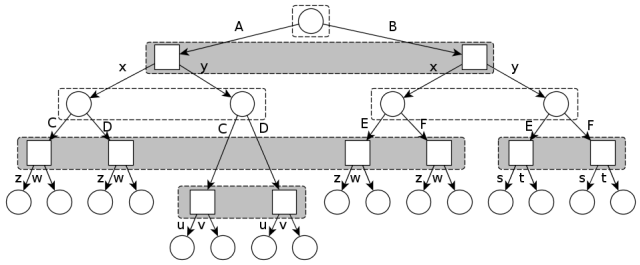

Figure 1: An extensive-form game between two players, circle and box. The dashed boxes indicate the information sets.

3.2.1 Inconsistencies in the Restricted Game

Consider the game depicted in Figure 1 and assume the restricted game is defined by sequences $\Sigma'_{\bigcirc} = \{\emptyset, A, AC\}$ for the circle player and $\Sigma'_{\square} = \{\emptyset, y, yu\}$ for the box player. Now assume, that the circle player adds sequences B, BE based on its best response calculation and we need to construct a new restricted game from sequences $(\Sigma'_{\bigcirc} \cup \{B, BE\}, \Sigma'_{\square})$. Executing sequences BE and yu together ends in an inner node $h = ByE$ in which the box player does not have any action available in the restricted game.

Having nodes such as h with no valid continuation in the restricted game causes inconsistencies because the LP does not know how to assign a value to these interior nodes. In general the problem appears whenever the best-response algorithm adds a sequence σ_i to a restricted game that already contains some sequence σ_{-i} for the opponent, such that the execution of these two sequences ends in an inner node h assigned to the player $-i = P(h)$, from which there is no continuation in the restricted game. That is, there is no sequence of player $-i$ in the restricted game that reaches h and contains a valid action to take in h. A continuation for h does exist in the complete game, otherwise it would be a terminal node and would have a utility function assigned to it. The problem with having interior nodes with no continuations in the restricted game LP is that the LP assigns a value of 0 to these nodes. This can result in the LP finding an "optimal" solution that ends in this node, and player $-i$ may not choose to add a new sequence to extend beyond node h if all possible continuations have very low values (less than 0), leading to incorrect termination of the double-oracle algorithm.

The method introduced in [1] solved this problem by adding additional sequences to the restricted game that provide valid continuations for all interior nodes. However, this method has large computational inefficiencies, and was the major bottleneck in the algorithm. We introduce a new approach that treats these interior nodes as *temporary leaf nodes* in the restricted game, and assigns them temporary utility values that will guarantee convergence of the algorithm to equilibrium. For convergence, these temporary values must be lower bounds on the utility for the player $-i$ who is moving in h (and has no valid continuation), and therefore an upper bound for player i. This accomplishes two things. First, player i will play in the restricted game so as to end in node h if there is any chance that there is a better outcome based on a continuation of this node. Second, player $-i$ will add a best response sequence continuing from this node if it occurs with non-zero probability in the solution to the restricted game, and there exists a better strategy for $-i$ that can improve over the pessimistic value assumed for this node in the current restricted game solution.

To find a temporary value that meets these criteria, we compute the utility of the best response of player i to the default strategy of the opponent. This is an upper bound on the value of node h for player i because the utility of the best response to the arbitrary default strategy π_{-i} will always be at least as good as if $-i$ was playing an intelligent strategy instead of the default. If we consider again the example from Figure 1, the node *ByE* will represent a temporary leaf in the restricted game and the temporary value will be equal to an upper bound estimation from the perspective of circle player.

3.2.2 Managing the Set of Temporal Leaves

Calculating the values of the temporary leaf nodes and updating them as new sequences are added to the restricted game requires some bookkeeping in the algorithm. First, we formalize the process of constructing the sequence-form LP for the restricted game (*CoreLP*). The set Φ' represents the set of full-length sequences added to the restricted game by best-response algorithms, and the set Σ' represents a set of sequences that are used to create the CoreLP. Intuitively, the sequences in Σ' always form a maximal compatible part of the game given by a set of full-length sequences Φ', so

$$\Sigma' \subseteq \{\mathsf{getAllPrefixes}(\phi) \ : \ \phi \in \Phi'\} \tag{5}$$

is maximal, such that:

$$\forall \sigma_i \in \Sigma'_i \ \exists \ \sigma_{-i} \in \Sigma'_{-i} \ s.t. \ \sigma_i \in \mathsf{seq}_i(\omega(\sigma_i, \sigma_{-i})). \tag{6}$$

This means that for each sequence σ_i in Σ' there exists a compatible sequence of the opponent σ_{-i} that allows an execution of the sequence σ_i in full.

Given the definition of Σ', the restricted game can now have temporary leaves instead of inner nodes in the complete game, and we define the set $\mathcal{L} \subseteq H$ to represent these temporary leaves. For each temporary leaf h, assigned to player $-i$, we define a temporary utility equal to the value of the best response of player i to the default strategy of the opponent. The sequences of both players leading to node h are extended to full-length sequences using either the default strategy (for player $-i$) or the best-response strategy (for player i). We denote these extended sequences as $\Phi^{BR(h)} \subset \Phi$. Now, the temporary utilities are incorporated into the CoreLP using a modified function g', defined as follows:

$$g'_i(h) = \sum_{\phi^{BR(h)} \in \Phi^{BR(h)}} \sum_{h' \in \omega(\phi^{BR(h)})} g(h') \tag{7}$$

if $h \in \mathcal{L}$, and we set $g'_i(h) = g_i(h)$ otherwise.

In the implementation, we maintain some additional information for each temporary leaf h, including the set of full-length sequences of player i that were added by the best-response sequence algorithm, and which end in h, with a corresponding sequence from the default strategy of player $-i$. In general there may be multiple such sequences for each h, which form the set $\Phi_i^{BR(h)}$. This information is useful for updating the temporary leafs as new sequences are added. If a new sequence σ_{-i}^{new} has an action that is applicable in node h, then h will no longer be a temporary leaf and it will be removed from the set $\mathcal{L}$. However, new temporary leaves could be created somewhere in the sub-tree rooted in node h. In this case, we can reuse the sequences from $\Phi_i^{BR(h)}$ to quickly estimate the temporary utilities of these new temporary leaves.

3.3 Best-response Sequence Algorithms

The goal of the best-response sequence (*BRS*) algorithm is to generate new sequences that will be added to the restricted game in the next iteration, or to prove that no more sequences need to be added. Throughout this section we use the term *searching player* to represent the player for whom the algorithm computes the best-response sequence. We denote this player as i. First, we describe the basic steps of the best-response sequence algorithm and then we focus on domain-independent pruning techniques.

3.3.1 Game-tree Search Algorithm

BRS algorithm returns a pure strategy that is a best response to the input strategy of the opponent completed by his default strategy where it is not defined. The algorithm returns both the set of sequences forming this strategy, as well as the expected value of this strategy against the completed realization plan of the opponent.

The algorithm is based on a depth-first search that traverses through the complete game tree, in which the behavior of the opponent $-i$ is fixed either to the strategy given by the realization plan r_{-i} from the *CoreLP* solution in information sets already included in the restricted game, or the default strategy π_{-i}. The behavior of the depth-first search algorithm in each currently evaluated node h depends primarily on the player, to which the node is assigned (searching player i, opponent $-i$, or Nature).

If node h is assigned to Nature (i.e., it is a chance node), the method recursively evaluates the succeeding nodes, computes the expected value of node h as a sum of the values of the successors weighted by the fixed probability distribution associated with node h, and propagates this expected value to the predecessor. If node h is assigned to the opponent, the algorithm acts similar, but the probability distribution is either given by the realization plan of the opponent r_{-i}, or by the default strategy π_{-i}.

Finally, if the algorithm evaluates node h associated with the searching player i, the algorithm selects the value of the best action played in the information set I_i^k, where this node h belongs. The probability of being in a specific node $h' \in I_i^k$ in this information set is given by the probability of the Nature play $C(h')$, and again either by the realization plan of the opponent $r_{-i}(\mathsf{seq}_{-i}(h'))$, or by the default strategy π_{-i}. Therefore, by applying each action $a \in A(h)$ in every node $h' \in I_i^k$ the algorithm finds the best action with the maximal expected utility.

3.3.2 Pruning Techniques

Since the best-response sequence (*BRS*) algorithm operates on (generally large) game tree of the complete game, its performance can be substantially improved by ensuring that irrelevant branches are not evaluated. To do this, we introduce several domain-independent pruning techniques. First, the method of selecting a fixed action according to the default strategy described in the previous subsection for nodes that belong to the opponent $-i$ not included in the restricted game results in pruning that significantly reduces the searched space.

Second, we can estimate the value for the searching player i in nodes assigned to the opponent that are included in the set of temporary leaves of the restricted game. The algorithm has already calculated and upper bound value for these temporary leaves according to the sequences of the searching player and the default strategy of the opponent. Therefore, the searching player can re-use this information directly without a need to search the state space beyond this node.

Finally, we introduce pruning for the nodes that are assigned to the searching player (i.e., $P(h) = i$, $h \in I_i^k$) and that are a part of the restricted game. The pruning relies on fact that (1) it evaluates the nodes from the information set I_i^k in an ordered fashion that is given by the probability of these nodes, and (2) we can prune the evaluation of some branches if the selection of the best action with maximal expected utility is certain. The pseudocode of this part of

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

1: $H' \leftarrow \{h';\ h' \in I_i^k\}$
2: sort H' descending according to value $r_{-i}(\mathsf{seq}_{-i}(h)) \cdot C(h)$
3: $w \leftarrow \sum_{h' \in H'} r_{-i}(\mathsf{seq}_{-i}(h')) \cdot C(h')$
4: maxVal $\leftarrow -\infty$
5: **for** $h' \in H'$ **do**
6: **if** (maxVal - secMaxVal) $> w \cdot$ (MaxUtility−MinUtility) **then**
7: $v^{h'}_{maxAction} \leftarrow v^{h'}_{maxAction}$+BRS$_i(h' maxAction)$
8: **else**
9: **for** $a \in A(h')$ **do**
10: $v^{h'}_{a} \leftarrow v^{h'}_{a}$+BRS$_i(h'a)$
11: **end for**
12: maxAction $\leftarrow \arg\max_{a \in A(h')} v_a$
13: maxVal $\leftarrow v^{h'}_{maxAction}$
14: secMaxVal $\leftarrow \max_{a \in \{A(h') \setminus maxAction\}} v_a$
15: **end if**
16: $w \leftarrow w - r_{-i}(\mathsf{seq}_{-i}(h')) \cdot C(h')$
17: **end for**
18: **return** $v^{h}_{maxAction}$

Figure 2: Pruning in the nodes of the searching player.

pruning is depicted in Figure 2. It uses $BRS_i(h)$ to denote the recursive call of the best-response sequence algorithm in node h. First, we sort all the nodes in this information set based on the probability of occurrence (set H'), which is determined by the realization plan of the opponent as well as moves by Nature (lines 1-2). Then, the algorithm evaluates recursively the nodes in the set H' (line 10) and for all actions it calculates the expected utility value (the recursive call returns the utility value already weighted by the probabilities). The pruning occurs if the choice for the best action cannot be changed — more formally, if the maximal expected value cannot be overcome in the remaining nodes given the remaining probabilities for these nodes (denoted as w) and bounds for minimal and maximal utility values in the game (line 6). If pruning is enabled, for the remaining nodes in H' the algorithm evaluates only the values by executing action *maxAction* (line 7), and these values are returned when some of these nodes is reached by the BRS algorithm.

4. THEORETICAL ANALYSIS

In this section we prove that our sequence-form double-oracle algorithms will always converge to a Nash equilibrium of the complete game. First, we define formally the strategy computed by the best-response sequence (*BRS*) algorithm, then we prove lemmas about the basic characteristics of the *BRS* strategies, and finally prove the main convergence result. In the proof we use the concept of an extension of a realization plan from a restricted game to the full game using a default strategy outside of the restricted game. Formally, if r'_i is a mixed strategy of player i in the form of realization plan in a restricted game Σ', then we define $\bar{r}_i'$ to be a strategy identical to r'_i on sequences in Σ' and identical to default strategy π_i on the remaining sequences from Σ that are not in Σ' .

LEMMA 4.1. *Let r'_{-i} be a realization plan of player $-i$ on some restricted game Σ'. $BRS(r'_{-i})$ then returns a strategy q_i in the complete game, such that q_i is a pure best response strategy to $\bar{r}_{-i}'$. The value returned by the algorithm is the value of executing the pair of strategies $u_i(q_i, \bar{r}_{-i}')$.*

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

For brevity we further use $v(BRS(r'_{-i}))$ to denote the value returned by the *BRS* algorithm that is equal to $u_i(q_i, \bar{r}_{-i}')$.

LEMMA 4.2. *Let r'_{-i} be a realization plan of player $-i$ on some restricted game Σ' and let v_i^* be the value of the complete game Σ for player i, then*

$$v(BRS(r'_{-i})) \geq v_i^*.$$

PROOF. The value of the game (v_i^*) is the value of the best response for player i in the restricted game against r'_{-i}. From Lemma 4.1, we know that $v(BRS(r'_{-i}))$ is the value of the best response to strategy $\bar{r}_{-i}'$ for player i. Since the best response in restricted game Σ' to r'_{-i} is also a possible response to the extended strategy $\bar{r}_{-i}'$, the value of the best response to $\bar{r}_{-i}'$ must be at least v_i^*, and could be greater since the responding player has strictly more strategies to choose from in the unrestricted game. □

LEMMA 4.3. *Let r'_{-i} be a realization plan of player $-i$ returned by the CoreLP for some restricted game Σ' and let v_i^{LP} be the value of the restricted game returned by the CoreLP, then*

$$v(BRS(r'_{-i})) \geq v_i^{LP}.$$

PROOF. r'_{-i} is a part of the Nash equilibrium strategy in a zero-sum game that guarantees value v_i^{LP} in Σ'. If the best response computation in the complete game selects only the actions from Σ', it creates the best response in game Σ' as well obtaining value v_i^{LP}. If the best response selects an action that is not included in Σ', there are two cases.

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

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

LEMMA 4.4. *Under the assumptions of the previous lemma, if $v(BRS(r'_{-i})) > v_i^{LP}$ then it returns sequences that extend game Σ' in the next iteration.*

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

THEOREM 4.5. *The double-oracle algorithm for extensive form games described in the previous section stops if and only if*

$$v(BRS(r'_{-i})) = -v(BRS(r'_i)) = v_i^{LP} = v_i^*, \tag{8}$$

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

Proof. First we show that the algorithm continues until all equalities (8) hold. If $v(BRS(r'_{-i})) \neq -v(BRS(r'_i))$ then from Lemma 4.2 and Lemma 4.4 we know that for some player i it holds $BRS(r'_{-i}) > v_i^{LP}$, thus the restricted game in the following iteration is larger by at least one action and the algorithm continues. In the worst case, the restricted game equals the complete game $\Sigma' = \Sigma$, and it cannot be extended any further. In this case the *BRS* cannot find a better response then v_i^* and the algorithm stops due to Lemma 4.4.

If the condition in the theorem holds, it is clear we have found a NE in the complete game, because from Lemma 4.1 we know that $q_{-i} = BRS(r'_i)$ is the best response to $\bar{r}_i{}'$ in the complete game. However, if the value of the best response to a strategy in a zero-sum game is the value of the game, then the strategy is optimal and it is a part of a Nash equilibrium of the game. □

5. EXPERIMENTS

We experimentally evaluate the performance of the described sequence-form double-oracle algorithm with novel methods for maintaining a valid restricted game and computing best-response sequences. Since both novel methods are complementary, we experimentally analyze the impact of each of them separately. We run the first set of experiments on the search games used for experimental evaluation in [1], the second set of experiments is on variants of a simplified Poker games inspired by Leduc Hold'em [13].

The test games have differing characteristics in terms of stochasticity (the first one is deterministic, while the second includes chance nodes) and private information of the players (the players cannot observe the actions of the opponent in the first game, however, the private information is completely determined by the moves of Nature in the second game). Also, these games differ greatly in the size of the support of Nash equilibria (i.e., the number of sequences actually used with non-zero probability in a Nash equilibrium). While there are only few sequences in the support of the equilibrium in the first game, the size of the support is considerably higher in poker games. Such differing characteristics give us an opportunity to present a broader evaluation of the performance of our algorithm and determine both strengths and current limitations.

Note that all algorithms were implemented using a generic framework for modeling arbitrary extensive-form games, and neither algorithm is using any domain-specific knowledge. Both the validity and best-response algorithms we present could be further enhanced with domain-dependent knowledge, which could significantly improve the runtime. However, our focus here is on generic methods. In all of the experiments we used a single thread on an Intel i7 CPU running at 2.8 GHz. Each of the algorithms was given at maximum of 10 GB of memory for Java heap space, and we used IBM CPLEX 12.2 for solving the linear programs.

5.1 Experiment Settings

Search Games.

The search game contains two players: the *patroller* (or the defender) and the *evader* (or the attacker). The game is played on a directed graph (see Figure 3), where the evader aims to cross safely from a starting node (E) to a destination node (D), and the defender moves in the intermediate nodes (the shaded areas) trying to capture the evader (i.e., to be at the same node as the evader). The defender controls two units and during each turn both players move their units simultaneously from the current node to an adjacent node, or stay in the same location. The only exception is the evader, who cannot stay in the two leftmost nodes. If a pre-determined number of turns is made without either player winning, the game is a draw.

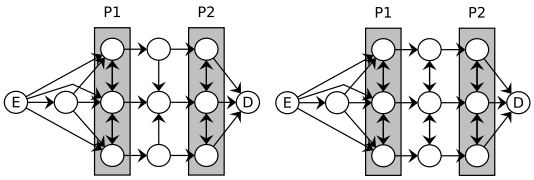

Figure 3: Two variants of the graph used in the experiments; we refer to them as G1 (left) and G2 (right).

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

Figure 3 shows two examples of the graphs used in the experiments. The patrolling units can move only in the shaded areas (P1,P2). Even though the graph is small, the concurrent movement of all units implies a large branching factor (up to ≈ 50 for one turn) and thus large game trees (up to $\approx 10^{11}$ nodes). In the experiments we altered three different graphs (G1 and G2 from Figure 3 and G3 that has no edges in the middle column), maximal number of turns of the game (from 3 to 7), and the option for the attacker to remove the tracks (*SA* if the slow moves were allowed, *SD* otherwise).

Poker Games.

Secondly, we use variants of a simplified two-player poker game inspired by Leduc Hold'em. Each player starts with the same amount of chips, and both players are required to put some number of chips in the pot (called *ante*). In the next step, the Nature player deals a single card to each player (the opponent is unaware of the card) and the betting round begins. A player can either *fold* (the opponent wins the pot), *check* (let the opponent start the round), *bet* (add some amount of chips as first in the round), *call* (add the amount of chips equal to the last bet of the opponent into the pot), or *raise* (match and increase the bet of the opponent). If no further raise is made by any of the players, the betting round ends, the Nature player deals one card on the table, and the second betting round with the same rules begins. After the second betting round ends, the outcome of the game is determined — a player wins if: (1) her private card matches the table card and the opponent's card does not match, or (2) none of the players' cards matches the table card and her private card is higher than the private card of the opponent. If no player wins, the game is a draw and the pot is split.

In the experiments we altered the number of types of the cards (ranging from 3 to 5; there are 3 types of cards in Leduc), number of cards of each type (ranging from 2 to 3; set to 2 in Leduc), maximum length of sequence of raises in a betting round (ranging from 1 to 4; set to 1 in Leduc), and the number of different sizes of bets (i.e., amount of chips added to the pot) for *bet*/*raise* actions (ranging from 1 to 4; set to 1 in Leduc).

5.2 Results

Search Games.

In the search game we measured the impact of each of the methods proposed in this paper. Therefore we compared 4 different instances of the double-oracle algorithm, where we used different version of the validity algorithms and the best-response sequence algorithms. The names of the algorithms are composed as follows: DO prefix denotes a double oracle algorithm; UBva denotes the upper-bound validity algorithm presented in this paper; SAva denotes the sequence-adding validity algorithm presented in [1];

Algorithm	Iterations	$\|\Sigma'\|$	VA [s]	BR [s]	LP [s]
FULLLP	-	-	-	-	278
DO-SAVA-ORGBRS	77	2064	1251	19	22
DO-SAVA-NEWBRS	77	1662	1028	14	22
DO-UBVA-ORGBRS	110	765	1	29	3
DO-UBVA-NEWBRS	123	839	1	28	5

Table 2: Time breakdown for scenario G1-SD of the search game; columns VA, BR, LP refer to the cumulative time spent in the validity algorithm, the best-response sequence algorithm, and the method for generating and solving the CoreLP.

NEWBRS denotes the best-response sequence algorithm presented in this paper; and ORGBRS denotes original best-response sequence algorithm presented in [1]. FULLLP denotes constructing and solving the full sequence-form linear program for the complete game.

In all scenarios all variants of double-oracle algorithm were able to find the exact solution by adding only a small fraction of the sequences to the restricted game. For the largest scenarios the size of the restricted game was $\approx 0.1\% - 0.4\%$ of the complete game. These results show that the double-oracle algorithms are particularly useful for games with small support of Nash equilibria, where they are able to quickly find the sequences actually needed for the solution without any domain-dependent knowledge.

The results for the selected scenarios of the search game with number of turns fixed to 7 are depicted in Figure 4 (missing values for FULLLP indicate that the algorithm run out of memory). The overview of the complete running time of the algorithms shows that the novel UBVA algorithm dramatically speeds-up the overall running time (note the logarithmic scale), and the double-oracle algorithms with UBVA significantly outperform other algorithms even on smaller instances of the search game. The impact of the new validity algorithm is demonstrated on the time breakdown for selected scenario G1-SD (see Table 2). The new validity algorithm increases the number of iterations somewhat, however, time needed for keeping the restricted game valid is dramatically lower. In addition, since we do not add additional sequences beyond the best responses, the restricted game is smaller so generating and solving the linear program is faster as well.

The novel NEWBRS algorithm also typically improves the runtime (it significantly speeds up calculating the best-response). However, it slows down the overall performance of the algorithm for some cases. The reason lies in the increased number of iterations in comparison to the original ORGBRS – for these instances of graphs the original optimistic *BRS* algorithm quickly found the correct sequences that form support of the Nash equilibrium.

Poker Games.

For variants of simplified Poker games we compared only the best algorithm from the previous set of experiments, the double-oracle method with the upper-bound validity algorithm and the new best-response algorithms that utilize default strategies (DO-UBVA-NEWBRS; from now on we refer to this algorithm only as DO), along with the FULLLP algorithm.

Contrary to the previous game, the variants of poker have relatively large support, so the running-time performance of the DO algorithm was typically somewhat worse. For the instances of experimental settings where we primarily increased the number of cards and card types the size of the final restricted game was $\approx 44\% - 65\%$ of the size of the complete game (up to 4481 sequences for each player in this case), and it took the double-oracle algorithm roughly $2 - 10$ times the amount of time needed by the FULLLP to find the exact solution (for example, ≈ 6 seconds for FULLLP compared to

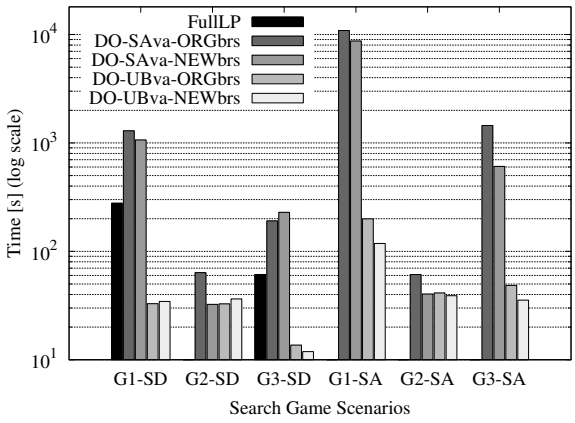

Figure 4: Running times for FULLLP and variants of DO algorithm on different scenarios of the search game (G1-G3 denotes graph;SA(SD) that slow moves were (dis)allowed). Missing values for FULLLP indicate that the algorithm ran out of memory.

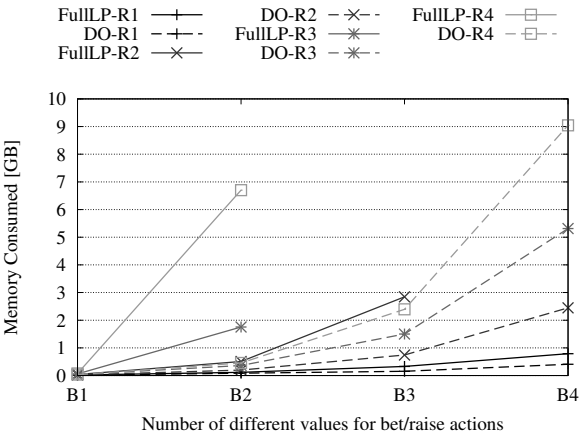

Figure 5: Memory consumption for instances of Poker games (R denotes number of re-raise actions, B number of possible betting amounts): solid lines represent FULLLP, dashed lines DO algorithm. Missing values for FULLLP indicate that the algorithm ran out of memory.

the ≈ 60 for DO on the instance with 5 types of cards, 3 cards of each type, and 4 different betting values in the second round).

The situation changes when we increase the depth of the game tree by allowing players to re-raise the opponent's bet, and increase the branching factor for player's actions by increasing number of different betting values for the actions bet and raise in both rounds. The size of the restricted game dropped to $\approx 2\% - 5\%$ for the largest instances solvable by both algorithms and FULLLP was quickly unable to construct and solve the linear program due to the memory restrictions. Figure 5 shows the scaling of the memory consumption for FULLLP and DO algorithm – we fixed number of types of cards to 3, number cards of each type to 2. We can see that with increasing depth of the game tree (increasing number of re-raise actions, denoted as R1-R4) and increasing branching factor at nodes assigned to players (differing number for betting amounts, denoted as B1-B4) the memory consumed by both algorithms grows exponentially. However, DO requires orders of magnitude less memory and we were able to find exact solution to much larger instances of Poker games compared to the FULLLP.

Case B2	$\lvert\Sigma'_1\rvert$	$\lvert\Sigma'_2\rvert$	Support P1	Support P2
DO-R1	1521 (67%)	1576 (69%)	492	479
DO-R2	3863 (34%)	4448 (39%)	871	914
DO-R3	6819 (13%)	7118 (14%)	717	1040
DO-R4	8140 (3%)	8579 (4%)	735	1188

Table 3: Sizes of the restricted sets of sequences (% of the complete game) and the number of support sequences with increasing depth.

Table 3 shows the exact sizes of the sets of sequences in the restricted game Σ' for a fixed number of betting amounts B2 and increasing depth R1-R4. We can see that although the size of the restricted game increases, it corresponds to smaller fractions of the complete game. Similarly, the size of the support in the restricted game increases, but we can see that there is an increasing number of sequences in Σ' that are not actually used in the support. By reducing the number of sequences added by best-response algorithms, we can reduce the overall number of iterations of DO algorithm, and thus further improve the performance.

6. DISCUSSION AND CONCLUSION

In this paper we present a sequence-form double-oracle algorithm for computing exact solutions for two-player zero-sum extensive-form games with imperfect information. We extend the double-oracle framework to non-deterministic games, provide new algorithms for key parts of the double-oracle framework, and present theoretical proof that our double-oracle algorithm converges to a Nash equilibrium of the game. We experimentally evaluated our algorithm on two classes of games and show significant speed-up in running time for games with small support Nash equilibria, and significant reduction in memory consumption even for games with large support in comparison to state-of-the-art exact algorithms.

The benefits of the approach presented in this paper are twofold. Firstly, it computes exact Nash equilibria for extensive-form games prior to constructing the complete game by identifying the sequences of promising actions that the players should play without any domain knowledge. Secondly, our approach decomposes the problem of computing a Nash equilibrium into separate sub-problems. Therefore, it can be seen as a framework, in which the domain-independent validity and/or best-response sequence algorithms can be replaced with domain-specific and thus much faster implementations (e.g., exploiting new efficient best-response algorithms for poker games [4] would bring significant speed-up in these games). The presented formal analysis identifies the key properties that these domain-specific implementations need to satisfy to guarantee the convergence to the correct solution of the game.

The algorithm presented in this paper stimulates a large volume of possible future work. Besides applications to specific domains, several theoretical questions need to be investigated. First of all, the performance of the double-oracle algorithm currently depends on the number of iterations. We intend to exploit theoretic characteristics of the structure of support of Nash equilibria in extensive-form games and improve the methodology of adding sequences in order to lower the number of iterations. Secondly, we plan to investigate possibility of adapting this framework for general-sum games, in which pruning techniques and upper-bound calculations are much more complicated. Finally, a modification of this algorithm for different solution concepts in extensive-form games (such as refinements of Nash equilibria, or Stackelberg equilibria) is a natural continuation of this work.

Acknowledgements

This research was supported by the Czech Science Foundation (grant no. P202/12/2054) and by the Czech Ministry of Education, Youth and Sports (grant no. LH11051).

7. REFERENCES

[1] B. Bosansky, C. Kiekintveld, V. Lisy, and M. Pechoucek. Iterative Algorithm for Solving Two-player Zero-sum Extensive-form Games with Imperfect Information. In *Proc. of European Conf. on Artificial Intelligence (ECAI)*, 2012.

[2] S. Hoda, A. Gilpin, J. Peña, and T. Sandholm. Smoothing techniques for computing nash equilibria of sequential games. *Math. Oper. Res.*, 35(2):494–512, May 2010.

[3] M. Jain, D. Korzhyk, O. Vanek, V. Conitzer, M. Tambe, and M. Pechoucek. Double Oracle Algorithm for Zero-Sum Security Games on Graph. In *Proc. of the Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.

[4] M. Johanson, M. Bowling, K. Waugh, and M. Zinkevich. Accelerating best response calculation in large extensive games. In *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2011.

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

[6] M. Lanctot, K. Waugh, M. Zinkevich, and M. Bowling. Monte carlo sampling for regret minimization in extensive games. In *Advances in Neural Information Processing Syst. (NIPS)*, pages 1078–1086, 2009.

[7] H. B. McMahan and G. J. Gordon. A fast bundle-based anytime algorithm for poker and other convex games. *Journal of Machine Learning Research - Proceedings Track*, 2:323–330, 2007.

[8] H. B. McMahan, G. J. Gordon, and A. Blum. Planning in the presence of cost functions controlled by an adversary. In *Int. Conf. on Machine Learning*, pages 536–543, 2003.

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

[10] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. 2009.

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

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

[13] K. Waugh, N. Bard, and M. Bowling. Strategy grafting in extensive games. In *Advances in Neural Information Processing Systems 22 (NIPS)*, pages 2026–2034, 2009.

[14] M. P. Wellman. *Trading Agents*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Pub., 2011.

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

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