Iterative Algorithm for Solving Two-player Zero-sum Extensive-form Games with Imperfect Information

Branislav Bošanský[1] and **Christopher Kiekintveld**[2] and **Viliam Lisý**[1] and **Michal Pěchouček**[1]

Abstract. We develop and evaluate a new exact algorithm for finding Nash equilibria of two-player zero-sum extensive-form games with imperfect information. Our approach is based on the sequence-form representation of the game, and uses an algorithmic framework of double-oracle methods that have been used successfully in other classes of games. The algorithm uses an iterative decomposition, solving restricted games and exploiting fast best-response algorithms to add additional sequences to the game over time. We demonstrate our algorithm on a class of adversarial graph search games motivated by real world border patrolling scenarios. The results indicate that our framework is a promising way to scale up solutions for extensive-form games, reducing both memory and computation time requirements.

1 Introduction

The field of computational game theory has made significant progress in recent years in developing more efficient algorithms for solving large, complex games. This is important because many real-world situations can naturally be modeled using a game-theoretic framework, but until recently many of these potential applications have been limited because they are too complex for existing solution methods. Some recent algorithmic advances are already used in impressive applications, including poker agents capable of defeating human experts [10] and methods for scheduling homeland security resources, such as Federal Air Marshals [12]. However, many classes of games are still computationally challenging, and further progress is needed to enable new applications.

Our primary motivation in this paper is a class of adversarial patrolling problems motivated by border security problems faced by the United States Customs and Border Patrol (CBP). Existing formulations of patrolling problems in the literature have focused on situations where there is no change in information during the game for either player (e.g, in [13]) — as soon as the evading player is detected or reaches the goal, the game ends. However, information structures are often significantly more complicated. For example, border patrol agents frequently patrol areas for signs of recent passage, or use remote sensing devices to provide (imperfect) remote detection, and track down illegal activity based on these signs.

The types of interactions we observe in the CBP patrolling problem can be formally modeled as extensive-form games with imperfect information. This class of games also includes classic games with private information, such as Poker and Kriegspiel. Since finding an exact solution is typically a computationally hard problem, existing algorithms for solving large extensive-form imperfect-information games typically use approximation. Examples include gradient methods with known error bounds [3], algorithms that exploit learning and regret minimization to converge to an approximative solution over time [7], and variants of Monte-Carlo tree search modified for imperfect information games [2] (these are not guaranteed to converge to an equilibrium).

In this paper we develop a novel algorithm for solving two-player zero-sum extensive-form games based on a double-oracle framework. Our method differs from the current state-of-the-art techniques in two key aspects: (1) it computes an exact Nash equilibrium (not an approximation), and (2) it iteratively expands the game by increasing the set of allowed strategies for players. Our approach is inspired by the oracle methods that have proven successful for solving large normal-form games [9, 4, 5], combined with the sequence form that allows a compact representation of strategies in extensive-form games [6, 14]. The main idea is to restrict the game to a limited number of possible sequences for each player, and iteratively expand the 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. We begin by presenting background and related work, and then describe our algorithm in detail before presenting a set of experimental results on adversarial search games motivated by border patrolling examples.

2 Background and Related Work

Adversarial situations with sequential moves and uncertainty can be modeled as extensive-form games (EFG) with imperfect information. We focus on two-player, zero-sum variants of EFGs that can be defined by a tuple $(N, A, H, Z, \chi, \rho, \tau, I, u)$ [11]. 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. A represents the set of actions, H denotes the set of all nonterminal choice nodes, and Z is a set of all terminal nodes of the game tree. The function $\chi : H \mapsto 2^A$ maps each nonterminal node to the subset of the actions can be selected in the node. The function $\rho : H \mapsto N$ assigns each nonterminal node to a player, and $\tau : H \times A \mapsto H \cup Z$ is a successor function that determines which node is reached after the players selects an action a in a nonterminal node h. The utility function $u_i : Z \mapsto \mathbb{R}$ assigns a utility value to each terminal node for player i, and the zero-sum assumption gives us $u_1(z) = -u_2(z)$.

Imperfect information is modeled through the use of information sets. The information sets I_i for player i form a partition of $\{h \in H : \rho(h) = i\}$ by defining equivalence classes such that $\chi(h) = \chi(h')$ and $\rho(h) = \rho(h')$ whenever there exists a j for which $h \in I_{i,j}$ and

[1] Agent Technology Center, Dept. of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague
[2] Computer Science Department, University of Texas at El Paso

$h' \in I_{i,j}$. We assume perfect recall so all nodes in some information set I_i have the same history of actions for player i (i.e., players cannot misremember their own actions).

Games with perfect recall can be represented using the compact *sequence form* [6, 14]. A sequence σ_i is an ordered list of actions of player i that occur on a path from the root to some node in the game tree; the set of all sequences for player i is denoted by Σ_i and the set of all sequences is $\Sigma = \bigtimes_i \Sigma_i$. The notation $\text{Ext}_i(\sigma_i)$ refers to the set of sequences that extend σ_i by exactly one action for player i. We use $\text{seq}_i(I)$ to denote the set of sequences of player i, which lead to information set I. We overload notation and also use $\text{seq}_i(h)$ to denote the sequence leading to node h. The notation $\mathcal{I}_i(\sigma_i)$ denotes an information set in which the last action of sequence σ_i was taken. The function $g_i : \Sigma \mapsto \mathbb{R}$ extends the utility function to all nodes by setting $g_i(\sigma) = u_i(z)$ if the execution of sequences of all players $\sigma \in \Sigma$ ends in a terminal node z, and $g_i(\sigma) = 0$ otherwise.

2.1 Sequence Form LP Method

Solving a game typically implies finding a profile of strategies that meet the criteria for a solution concept, such as Nash equilibrium in which each player plays a best response to the strategies of the other players. Formally, let Π_i be a set of *pure strategies* for player i, and Δ_i be a set of mixed strategies that are probability distributions over the pure strategies. A best response by player i to player $-i$ is a mixed strategy $\delta_i^* \in \Delta_i$ such that $u_i(\delta_i^*, \delta_{-i}) \geq u_i(\delta_i, \delta_{-i})$ for all strategies $\delta_i \in \Delta_i$. A strategy profile $\delta = \bigtimes_i \delta_i$ is in a *Nash equilibrium* if for all $i : \delta_i$ is a best response to δ_{-i}.

It is known that a Nash equilibrium of a two-player, zero-sum game in the normal form can be found efficiently using linear programming (LP). One way to solve an extensive-form game is to represent a pure strategy for player i as a combination of actions to take in each information set, and to transform it into a normal-form game. This, however, results in a game of an exponential size in the size of the game tree. Games with perfect recall can use a more compact representation of strategies called *behavioral strategies*, in which a mixed strategy is represented by a set of separate probability distributions over possible actions in each information set for a given player. Using the sequence form, we can represent the strategies of a player i as *realization plans* ($r_i : \Sigma_i \mapsto \mathbb{R}$) that are equivalent to behavioral strategies. Realization plans specify the probability of executing sequence σ_i, conditioned on the opponent choosing compatible actions that reach the information sets and the validity of taking actions specified in σ_i. Computing a Nash equilibrium using sequence form can be formulated as an LP [11] that is linear in the size of the game tree:

$$\min\ v_0$$

$$\text{s.t. } v_{\mathcal{I}_i(\sigma_i)} - \sum_{I' \in \mathcal{I}_i(\text{Ext}_i(\sigma_i))} v_{I'} \geq \geq \sum_{\sigma_{-i} \in \Sigma_{-i}} g_i(\sigma_i, \sigma_{-i}) r_{-i}(\sigma_{-i}) \qquad \forall \sigma_i \in \Sigma_i \tag{1}$$

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

$$\sum_{\sigma'_{-i} \in \text{Ext}_{-i}(I)} r_{-i}(\sigma'_{-i}) = r_{-i}(\text{seq}_{-i}(I)) \qquad \forall I \in I_{-i} \tag{3}$$

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

There are two types of variables in the program – variables v_{I_i} that represent the expected utility of the player i, and the variables r_{-i} that represent the strategy of the opponent in the form of realization plan. The 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).

Unfortunately, for many problems the size of the game tree quickly becomes prohibitive even for this LP, since the number of nodes in the tree grows exponentially in the length of the sequences. One approach used to solve large-scale optimization problems is to use decomposition to explore the solution space iteratively, without ever enumerating the full problem. These techniques are known in operations research as column generation, branch-and-cut, or branch-and-price methods [1]; in game theory they were adopted as the *oracle algorithms* [9]. We introduce a new algorithm based on this type of decomposition that operates on games in the sequence form after overviewing some related approaches.

2.2 Oracle Algorithms for NFG and Convex Games

The main idea used in oracle methods (i.e., column/constraint generation) is to solve the problem iteratively. First, a restricted and easier version of the full problem is formulated as an optimization problem (e.g., as a linear program, called the core problem or *coreLP*). Based on the solution of the coreLP, a second problem is solved to find the optimal way to relax the current restrictions imposed, in order to solve the original problem.

Oracle methods have been developed for normal-form games [9] as well as for convex games [8]. For two-player normal-form games (NFGs), the restricted coreLP problem corresponds to a game where players are restricted to using only a subset of the full strategy space. After solving this restricted game, new strategies are added to the restricted game by calculating *best responses* for each player to the current solution of the restricted game. This process converges to a Nash equilibrium [9]. Intuitively, when there is no best response to the restricted game solution that is not already included in the restricted game, the solution must be an equilibrium. The best-response algorithms are called *oracles*; restricting the strategy space only for one player is called *single-oracle* (SO), and *double-oracle* (DO) algorithms restrict the strategy space for both players.

A similar principle is used for convex games, in which the strategy space is a convex set [8]. Convex games can be used to represent a variety of different types of games, including extensive-form games. Although both of these approaches can, in principle, be used for computing solution of an extensive-form game, they do not effectively use the specific tree structure of EFGs. In the first case we would need a transformation to an exponentially-large NFG. For the case of convex games, the oracle approach requires enumeration of the complete set of sequences for both players, and the algorithm searches for the solution as a combination of a fixed number of realization plans.

3 A Double-Oracle Algorithm for Sequence Form

We apply the framework of oracle methods to general two-player zero-sum extensive-form games with imperfect information. We introduce a DO algorithm that operates directly on the sequence form of the game, and can find solutions without enumerating the full set of sequences. The main idea of our method is to restrict the strategy space of players by allowing them to play according to a limited subset of all sequences. We solve this restricted game using the sequence form LP as described in Section 2.1, and then add new sequences using a best-response oracle. However, the simplest form of this idea –

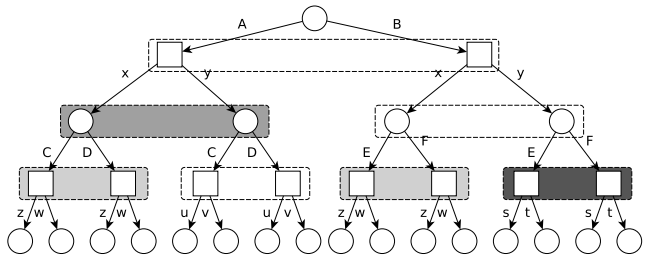


Figure 1: An extensive-form game between two players, circle and box. The same-colored rounded boxes on the same ply denote the same information sets.

adding a single best-response sequence in each iteration – does not work due to the problem of compatibility between sequences.

3.1 Sequence Compatibility

Consider the EFG shown in Figure 1. We arbitrarily select the initial sequences *AC* for the circle player and *xz* for the box player. Played together, these sequences lead to a leaf node. Now, suppose the best response sequence for the box player to *AC* is *yu*; hence, it is added to the restricted game and the execution of *AC* and *yu* also leads to a leaf node. Now, suppose that in the next iteration the sequence *BE* for the circle player is added as a best response. The execution of *BE* and *xz* terminates in a leaf node, but the execution of *BE* and *yu* does not because the action *u* is not legal after playing actions *B,y,E*. In the sequence form LP the value function $g(BE, yu)$ assigns the value of this node to 0, which can lead to an incorrect result. Suppose the values for the box player in the leaf nodes reached after moves *s* and *t* in the far right information set are all large negative numbers. In this case, the box player will never add sequences *ys* or *yt* as a best response sequence. However, solving the current restricted sequence-form LP can result in an incorrect solution because the value of the combination *BE* and *yu* is overestimated for the box player and underestimated for the circle player.

To solve this problem we define the concept of compatibility between sequences by introducing the function $\omega : \Sigma \mapsto H \cup Z$ which maps each possible combination of sequences to the node in the game tree that is reached by the execution of the two sequences of actions, and stopping when the next action choice is not valid, or a leaf is reached. We say that two sequences σ_i and σ_{-i} are *compatible* if $\omega(\sigma_i, \sigma_{-i})$ results in a terminal node from Z, and *incompatible* if it results in an internal choice node from H.

The solution we adopt to the problem of incompatible sequences is to add additional sequences to the restricted problem to ensure that the sequence form LP return a valid solution. To do this we consider the full-length sequences of the game (i.e., sequences that have no valid extensions, which we denote by $\Phi_i = \{\sigma_i : \sigma_i \in \Sigma_i \wedge \text{Ext}_i(\sigma_i) = \emptyset\}$). For every pair of full-length sequences, if the two sequences are incompatible, we seek for a full-length sequence within the set of all sequences of the player whose action was invalid, that extends the execution and that is compatible with the opponent sequence. If such a full-length sequence exists, it is added to the restricted game. The effect of adding these sequences is to ensure that each information set that is reachable in the restricted game is assigned a value based on a possible continuation of the path to a leaf node. The algorithm for checking compatibility and generating the new sequences is formalized in Figure 2.

Require: Φ'_1, Φ'_2 are the current sets of full-length sequences
1: **repeat**
2: changed $\leftarrow$ **false**
3: **for all** $\sigma_1 \in \Phi'_1, \sigma_2 \in \Phi'_2$ s.t. we have not checked σ_1, σ_2 for compatibility yet **do**
4: **if** $\omega(\sigma_1, \sigma_2) = h \in H$ **then**
5: $j \leftarrow \rho(h)$
6: **if** $\exists \sigma'_j \in \Phi_j$: $\text{seq}_j(h)$ is prefix $\sigma'_j \wedge \omega(\sigma_{-j}, \sigma'_j) \in Z$ **then**
7: **if** $\sigma'_j \notin \Phi'_j$ **then**
8: $\Phi'_j \leftarrow \Phi'_j \cup \{\sigma'_j\}$
9: changed $\leftarrow$ **true**
10: **until** changed
11: **return** Φ'

Figure 2: Compatibility Algorithm

Require: $\Phi'_1, \Sigma'_1 \leftarrow \emptyset$; $\Phi'_2, \Sigma'_2 \leftarrow \emptyset$
1: initialize Φ'_i with arbitrary full-length sequence σ_i
2: **repeat**
3: changed $\leftarrow$ **false**
4: $\forall i \in N$: $\Sigma'_i \leftarrow$ generateAllPrefixes(Φ'_i)
5: $(r'_1, r'_2) \leftarrow$ CoreLP(Σ'_1, Σ'_2)
6: **for** $i \in N$ **do**
7: $r_i^\pi \leftarrow \text{BR}_i(r'_{-i})$
8: **for** $\forall \sigma_i : r_i^\pi(\sigma_i) = 1 \ \wedge \ \sigma_i \notin \Phi'_i$ **do**
9: $\Phi'_i \leftarrow \Phi'_i \cup \{\sigma_i\}$
10: changed $\leftarrow$ **true**
11: ensure compatibility of Φ'
12: **until** changed
13: **return** (r'_1, r'_2)

Figure 3: Double Oracle Algorithm

3.2 Sequence Form Double-Oracle Algorithm

We can now present the main double-oracle algorithm for sequence form, which is depicted in Figure 3. The algorithm begins by initializing the sets Φ'_i with arbitrary (compatible) full-length sequences for each player; this can be done by selecting an arbitrary action in each information set until a leaf node is reached. Any time a new full-length sequence is added, all of the prefix sequences are added as well. The restricted game is solved using the coreLP for the sequence form (see Section 2.1), which generates a candidate solution consisting of a realization plan for each player over the sequences in the restricted game (line 5). Then the algorithm calculates a best response (BR) for each player to the realization plan of the opponent (we discuss BR algorithms in the next section). The BR algorithms return (one or more) sequences that are represented as a partial realization plan r_i^π for player i, and that are added to the restricted game (lines 8-10). Additional sequences are added using the compatibility algorithm described in Section 3.1. The algorithm terminates when no new sequences are added based on the best responses.

Theorem 1 *The sequence-form double-oracle algorithm terminates and computes a Nash equilibrium.*

Proof sketch The algorithm terminates, since the sets of sequences is finite and in each iteration we add at least one sequence.

The convergence of the double-oracle method relies on two things. First, the best-response algorithms used are complete in that they will always find a best response in the full strategy space if one exists. Second, the coreLP must calculate an optimal strategy for each player within the restricted game.

Our compatibility algorithm ensures that the sequences in the restricted game form a valid sub-game of the original game. If the BR algorithm does not add any new sequences, the expected values for all information sets (v variables in the sequence form LP) in the coreLP are equal to the their expected values in the full sequence LP, since the value cannot be underestimated for either player. This holds, because otherwise: (1) it would either mean that for some information set there exists another continuation sequence that yields better utility value, contradicting the assumption that the BR algorithm does not add any new sequence; or (2) it would mean that there is some succeeding information set that is not considered in the current coreLP, which is a contradiction with the compatibility algorithm that adds sequences that lead to all information sets conditioned on the current realization plan of the opponent. Thus, if we use the compatibility algorithm and correct BR algorithms, the final solution will be an equilibrium, since both players are playing a best-response strategy. □

4 Best-response Algorithms

A key component of the double-oracle methods is to design an oracle for finding additional best-response strategies to add to the restricted game. We begin by describing a full tree-search best-response algorithm for general extensive-form games, and then introduce methods for speeding up the calculations.

The pseudocode for a recursive tree-search best-response algorithm is shown in Figure 4. Based on the node currently considered by the algorithm, we distinguish two main cases. If the given node is terminal, the algorithm returns the utility value for the player i we are finding a best-response for (lines 1–5). The utility value is weighted by the probability of the opponent's realization plan if the sequence leading to this leaf node is in the restricted game ($\text{seq}_{-i}(I) \in \Sigma'_{-i}$), otherwise the raw value is returned.

If the current node h is an internal node it belongs to an information set I assigned to one of the two players. If the player is the searching player ($i = \rho(h)$) then we need to select the action with maximum expected payoff which will form part of the best-response sequence. We distinguish two cases: (1) there is at least one sequence leading to a node in I that has a non-zero probability in the opponent's realization plan, or (2) there is no non-zero opponent realization plan that leads to the information set. In the first case (lines 7–14) we can form a probability distribution over the nodes in I by calculating the realization probability for all possible nodes $h' \in I$, conditional on the searching player taking actions to reach this information set. For each possible action we make a recursive call to assess the value of the resulting node, and weight this value by the realization probability $r'_{-i}(\text{seq}_{-i}(h'))$. The action with maximum expected value is selected, and the value of this action is returned.

In the second case (lines 14–16) there is no probability information about the likelihood of the states $h' \in I$. In this case we choose the most optimistic value for the search player so that a best response cannot be missed. That is, for of each of the possible realizations of nodes in the information set, we select the one with the maximum value for the searching player for all possible continuation strategies.

Now, consider information sets belonging to the opponent $-i$, which are handled in lines 18–30. The same two cases apply. In the first case, there is a single sequence in the realization plan that leads to the current information set with non-zero probability (due to the perfect recall assumption). If the realization plan has a continuation of this sequence with non-zero probabilities the behavior in this information set is well-defined, and we can calculate the value of the

Require: $i \in N$ – player computing best response
$h \in H \cup Z$ – current node in the game tree
I information set for which $h \in I$
$V_h \leftarrow 0, V_a \leftarrow 0$
1: **if** $h \in Z$ **then**
2: **if** $\text{seq}_{-i}(I) \in \Sigma'_{-i} \wedge r'_{-i}(\text{seq}_{-i}(I)) > 0$ **then**
3: **return** $u_i(h) \cdot r'_{-i}(\text{seq}_{-i}(I))$
4: **else**
5: **return** $u_i(h)$
6: **if** $\rho(h) = i$ **then**
7: **if** $\exists \sigma'_{-i} \in \text{seq}_{-i}(I) : \sigma'_{-i} \in \Sigma'_{-i} \wedge r'_{-i}(\sigma'_{-i}) > 0$ **then**
8: **for all** $h' \in I$ **do**
9: **for all** $a \in \chi(h')$ **do**
10: $\sigma_{-i} \leftarrow \text{seq}_{-i}(h')$
11: $V_a \leftarrow V_a + r'_{-i}(\sigma_{-i}) \cdot BR_i(\tau(h', a))$
12: $a_{max} \leftarrow \arg\max_a(V_a)$
13: $V_h \leftarrow BR_i(\tau(h, a_{max})))$
14: **else**
15: **for all** $a \in \chi(h)$ **do**
16: $V_h \leftarrow \max(V_h, BR_i(\tau(h, a)))$
17: **else**
18: **if** $\text{seq}_{-i}(I) \in \Sigma'_{-i} \wedge r'_{-i}(\text{seq}_{-i}(I)) > 0$ **then**
19: **for all** $a \in \chi(h)$ **do**
20: $\sigma^a_{-i} \leftarrow \text{seq}_{-i}(I) \oplus a$
21: $V_a \leftarrow BR_i(\tau(h, a))$
22: **if** $\exists a \in \chi(h) : r'_{-i}(\sigma^a_{-i}) > 0$ **then**
23: **for all** $a \in \chi(h) : r'_{-i}(\sigma^a_{-i}) > 0$ **do**
24: $V_h \leftarrow V_h + V_a$
25: **else**
26: **for all** $a \in \chi(h)$ **do**
27: $V_h \leftarrow \max(V_a) \cdot r'_{-i}(\text{seq}_{-i}(I))$
28: **else**
29: **for all** $a \in \chi(h)$ **do**
30: $V_h \leftarrow \max(V_h, BR_i(\tau(h, a)))$
31: **return** backup(V_h)

Figure 4: Full Tree-search Best-response Algorithm

node by weighting the values of the succeeding nodes (calculated using recursion, lines 19–25).

It can also occur that all possible continuations have zero probability. This can occur because we search through the complete set of sequences for the searching player i in the BR algorithm which may lead the opponent to an information set with an undefined continuation plan when some of the sequences are not included in the restricted game used to generate the realization plan. This case is handled in lines 25–28 and it is handled as before by maximizing the value for the searching player. Finally, there is a case in which the sequence leading to I is not in the current realization plan of the opponent. Again, this is handled by selecting the maximum possible value for the current node h for the searching player (lines 28–30).

4.1 Improved Best-response Algorithm

The BR algorithm presented above is a straightforward depth-first search, and can be improved significantly with a number of techniques, including pruning and caching of partial results. Besides generic methods, domain-specific information can be used to speed up calculations even more dramatically. Since the BR method is invoked many times during a double-oracle algorithm, these improvements can have a substantial impact on the overall efficiency.

The first improvement we incorporate is a general pruning method for games with a bounded range of utility values, which is common. Suppose the searching player is evaluating successors of some node

$h \in H$ and is trying to find the maximum value (as in Figure 4, lines 14–16, and lines 28–30). Clearly, we can prune any remaining branches as soon as one branch is explored that gives the maximum utility of the game. We can also use domain-dependent *move ordering* heuristics to complement this pruning strategy by testing promising moves first (we describe such heuristics for our games later).

A second pruning strategy is possible if the node h belongs to the opponent and there is a non-zero probability of reaching this node according to the realization plan (in Figure 4, lines 19–25). We can aggregate the total probability of the sequences that extend this node based on the realization plan. As soon as the total probability of the extensions reaches the probability of reaching the initial node h (i.e. we reach the value $r'_{-i}(\text{seq}_{-i}(I))$), we can prune the remaining branches since they have zero probability and cannot modify the value of the node V_h.

5 Experiments

The performance of double-oracle methods depends on several factors: the speed of the coreLP, the speed of the best-response algorithms, and the number of strategies (sequences) that need to be added before terminating. Existing oracle methods for normal-form games show performance improvements [9, 4, 5, 13], but the results can vary substantially based on the properties of the game. Here we present experimental results for our algorithm on a realistic class of adversarial search games motivated by border patrolling scenarios.

5.1 Experimental Setting

As described in the introduction, one of the key tactics used by border patrolling agents is to look for recent signs of passage (e.g., foot or vehicle tracks) and use this information to capture illegal entrants. In addition, border patrol agents are also able to coordinate strategies among multiple agents. We developed a simplified patrolling scenario that captures these qualitative features in a game of incomplete information. There are two players, the *patroller* (or defender) and the *evader* (or attacker). The game is played on a graph, with the evader attempting to cross safely from a starting node to a destination node, and the defender patrolling the intermediate nodes to try to capture the evader. Two example graphs are shown in Figure 5. The evader starts in E and tries to get to D. The defender has two units that move in the shaded areas P1 and P2.

During each turn, both players move their units simultaneously from the current node to an adjacent node or stay in the same location. Players do not know the location of the other player's units, until the defender occupies the same node as the evader or the evader reaches the destination. In the first case the defender wins, and in the second case the attacker wins. If a pre-determined number of turns is made without either case occurring, the game is a draw. An additional feature of the game is that the evader leaves tracks in visited nodes that can be discovered if the defender visits the node later. In some game instances, we also include an option for the attacker to move slowly and avoid leaving tracks; this type of move requires two turns (the evader removes the tracks in a node in one turn).

These games are computationally challenging for several reasons. They have long sequences of moves for both players, and a high branching factor (particularly for the defender, which has two units to move on each turn). Furthermore, there is no simple structure to the information sets; the defender's observations depend on the actions of the evader. By modifying the structure of the graph we can also explore games with different characteristics, since the number

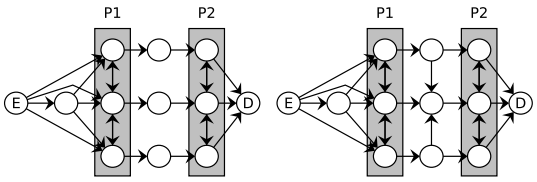


Figure 5: Two variants of the graph used in the experiments.

of compatible sequences and number of information sets can change dramatically with even small changes to the graph.

All the experiments were run on Intel i7 CPU running at 2.8GHz, each of the compared algorithms could use 10 GB of memory, and we used IBM CPLEX 12 for solving the LP.

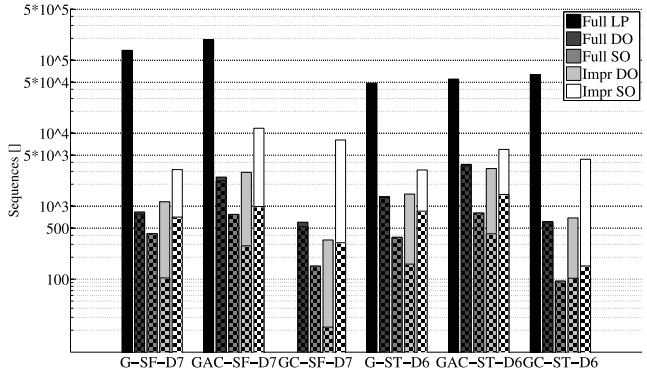
5.2 Experimental Results

We experiment with three graphs; two are shown in Figure 5: graph *G* on the left and *GAC* on the right. A final graph, *GC*, is similar to *GAC*, but with bidirectional edges connecting the nodes in the middle column. We vary the maximum number of turns in the game and denote it as depth. Since there are three units to move for each turn, the number of plies in the game tree is equal to $3 \times depth$. We also vary whether or not the evader has the option to move slowly and avoid leaving tracks. Our initial experiments compare three solutions methods: (1) FULL_LP generating and solving the full sequence form LP, (2) FULL_SO a single-oracle algorithm which uses all sequences for the evader and generates defender sequences, and (3) FULL_DO a double-oracle algorithm generating sequences for both players.

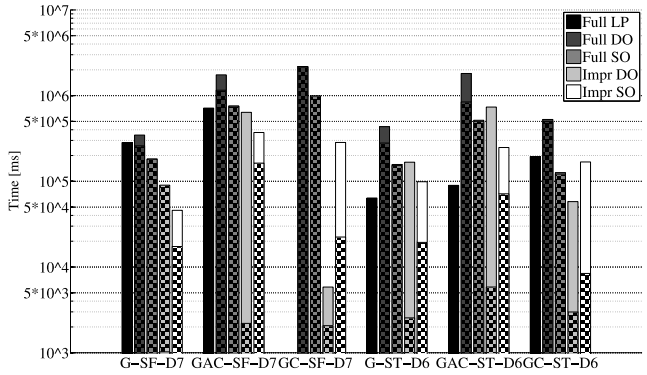
The first result we note is that all three algorithms found the same solution, experimentally confirming the correctness of our double-oracle algorithm. In addition, we found that both the SO and DO versions of the algorithm typically found solutions after adding a small fraction of the total number of sequences in the game (Figure 6a). For the defender, the maximum fraction of sequences used was 22%, and the effect was even stronger for larger games with typically less than 5% of the sequences used. The oracle algorithms also have lower memory requirements for larger games. They were able to compute an exact solution even in cases where FULL_LP exhausted all available memory (for example, in configuration *GC* graph and depth 7).

Comparing the performance of the SO and DO approaches, we find that the SO method often evaluates fewer sequences and uses less time than the DO method. This is likely because of the large imbalance in the number of sequences for the attacker and defender (due in large part to the defender having two units to control). For example, graph *G* with a depth of 7 has 137075 sequences for the defender compared to only 264 for the attacker. Another interesting feature of the data is that the graph *GAC* was hardest for the oracle methods to solve, while for the FULL_LP, the *GC* graph with the highest number of sequences is the most difficult one. The reason lies in the difficulty of computing compatible sequences in *GAC*; we can see that the compatibility algorithm uses a large fraction of time for this type of graph.

Overall, the running time of DO and SO is often slower than FULL_LP algorithm on smaller examples, though on larger example the DO and SO show improved performance in some cases. We also tested the DO and SO with the improved pruning and move ordering techniques described in Section 4.1. Move ordering uses domain knowledge: the evader first evaluates actions moving in the direction of the destination, and prefers slow movement if allowed. The defender evaluates moves towards the current location of the evader.



(a)



(b)

Figure 6: Selected results on the number of defender sequences (6a) and computation time (6b). Different settings are identified by three characteristics: (1) the graph type (G, GC, or GAC), (2) whether slow moves are allowed (ST), or not (SF), and (3) the number after D indicates the maximum number of turns in the game. The parts of the bars with pattern correspond to the number of sequences (or computational time) added by (or spent in) the BR algorithm.

These improved algorithms are termed IMPR_DO and IMPR_SO.

The data show that even these relatively simple improvements to the best-response algorithm result in a large improvement in the runtime for both the SO and DO algorithms. For most of the larger instances IMPR_DO and IMPR_SO outperform the FULL_LP algorithm, in some cases dramatically. The breakdown of the time spent in different parts of the algorithm give some additional insights. For example, in the *GC* graph with slow moves and depth 6, the full BR method took over 500 seconds and the improved version only 3. However, the results show that using improved versions of BR algorithms increases the time spent by the compatibility algorithm (BRs add less sequences due to pruning), and improving this part of the algorithm would likely lead to even better performance.

6 Conclusions and Future Work

In this paper we present a novel algorithmic framework for computing exact Nash equilibria for two-player zero-sum extensive form games with imperfect information. Our approach combines the iterative methodology of double-oracle algorithms with the compact sequence-form representation to provide a promising new way for scaling to larger, more realistic games. We demonstrate our new algorithm on a class of adversarial search games motivated by real challenges in patrolling large open areas, such as international borders. However, our algorithms are general and can be used for any problem that can be modeled as a two-player zero-sum extensive-form game with imperfect information.

Our experimental results show that the overall approach is promising, especially for larger problem instances where the DO method was able to solve the game using a small fraction of the full set of sequences. Using improved best-response methods our DO algorithm was significantly faster than solving the full sequence form game, and used less memory (a key limitation of solving large LPs in practice). This result is typical of oracle-based methods; to see the full benefits of the approach it is necessary to develop very fast oracles, sometimes using domain-specific knowledge. The framework is flexible enough that it can be used with a variety of different BR methods, and can even incorporate approximate BR methods. The compatibility algorithm is also a limiting factor in our current implementation, since it has not yet been optimized. We plan to explore additional improvements in both BR and compatibility in future work.

ACKNOWLEDGEMENTS

We would like to thank the three anonymous reviewers for helping in improving the presentation of this paper. This research was supported by the Czech Science Foundation (grant no. P202/12/2054), and by the United States Department of Homeland Security through the National Center for Border Security and Immigration (NCBSI).

REFERENCES

[1] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, 'Branch-and-price: Column generation for solving huge integer programs', *Operations Research*, (1998).
[2] P. Ciancarini and G. P. Favini, 'Monte Carlo tree search in Kriegspiel', *Artificial Intelligence*, (2010).
[3] A. Gilpin, J. Pena, and T Sandholm, 'First-Order Algorithm with O(ln(1/epsilon)) Convergence for epsilon-Equilibrium in Two-Person Zero-Sum Games.', *Mathematical Programming*, (2011).
[4] E. Halvorson, V. Conitzer, and R. Parr, 'Multi-step Multi-sensor Hider-Seeker Games', in *Proc. of IJCAI*, (2009).
[5] 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 AAMAS*, (2011).
[6] D. Koller, N. Megiddo, and B. von Stengel, 'Efficient computation of equilibria for extensive two-person games', *Games and Economic Behavior*, (1996).
[7] M. Lanctot, K. Waugh, M. Zinkevich, and M. Bowling, 'Monte carlo sampling for regret minimization in extensive games', in *Proc. of NIPS*, (2009).
[8] H. B. McMahan and G. J. Gordon, 'A fast bundle-based anytime algorithm for poker and other convex games', *Journal of Machine Learning Research*, (2007).
[9] H. B. McMahan, G. J. Gordon, and A. Blum, 'Planning in the presence of cost functions controlled by an adversary', in *ICML*, (2003).
[10] N. A. Risk and D. Szafron, 'Using Counterfactual Regret Minimization to Create Competitive Multiplayer Poker Agents', in *Proc of AAMAS*, (2010).
[11] Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithm, Game-Theoretic, and Logical Foundations*, Cambridge University Press, 2009.
[12] J. Tsai, S. Rathi, C. Kiekintveld, F. Ordóñez, and M. Tambe, 'IRIS - A Tool for Strategic Security Allocation in Transportation Networks Categories and Subject Descriptors', in *Proc. of AAMAS*, (2009).
[13] O. Vanek, B. Bosansky, M. Jakob, V. Lisy, and M. Pechoucek, 'Extending security games to defenders with constrained mobility', in *Proc. of AAAI Spring Symposium GTSSH*, (2012).
[14] B. von Stengel, 'Efficient computation of behavior strategies', *Games and Economic Behavior*, (1996).