Combining Compact Representation and Incremental Generation in Large Games with Sequential Strategies

Branislav Bošanský[1,2], **Albert Xin Jiang**[3], **Milind Tambe**[4], **Christopher Kiekintveld**[5]
[1] Agent Technology Center, Faculty of Electrical Engineering, Czech Technical University in Prague
[2] Department of Computer Science, Aarhus University
[3] Department of Computer Science, Trinity University, San Antonio, Texas
[4] Computer Science Department, University of Southern California, Los Angeles
[5] Computer Science Department, University of Texas at El Paso

Abstract

Many search and security games played on a graph can be modeled as normal-form zero-sum games with strategies consisting of sequences of actions. The size of the strategy space provides a computational challenge when solving these games. This complexity is tackled either by using the compact representation of sequential strategies and linear programming, or by incremental strategy generation of iterative double-oracle methods. In this paper, we present novel hybrid of these two approaches: compact-strategy double-oracle (CS-DO) algorithm that combines the advantages of the compact representation with incremental strategy generation. We experimentally compare CS-DO with the standard approaches and analyze the impact of the size of the support on the performance of the algorithms. Results show that CS-DO dramatically improves the convergence rate in games with non-trivial support.

Introduction

Scalable algorithms have driven many recent applications of game theory in domains ranging from security (Tambe 2011) to Poker (Sandholm 2010). We present a new algorithm for strictly competitive games in which players have sequential strategies, but cannot directly observe the immediate effects of the actions of their opponent during the game. The strategy of each player in these games is modeled as a finite-horizon Markov Decision Process (MDP) and we term these games as *normal-form games with sequential strategies* (NFGSS). In typical examples, the strategies of the players correspond to a movement on an underlying graph in time, such as maritime transit (Vanek et al. 2010; 2012), search games (McMahan, Gordon, and Blum 2003), network interdiction (Washburn and Wood 1995), or securing road networks (Jain et al. 2011; Jain, Conitzer, and Tambe 2013). Other examples assume a similar structure of strategies, but do not focus on strictly competitive scenarios (e.g., in the fare evasion domain (Yin et al. 2012; Jiang et al. 2013)).

In spite of the sequential nature of these scenarios, the existing works do not use the standard sequential model of *extensive-form games* (EFGs; players are able to observe and react to some actions of the opponent) due to the computational difficulties. EFGs have a crucial property called *perfect recall* that holds when the players perfectly remember the history of play and gained information. Most of the exact (Koller, Megiddo, and von Stengel 1996; von Stengel 1996; Bosansky et al. 2013) and approximative algorithms (Zinkevich et al. 2008; Hoda et al. 2010) require perfect recall to guarantee finding the solution. Unfortunately, the games of perfect recall suffer from extremely large state space[1]. In EFGs without perfect recall, however, equilibria might not exist (see e.g. (Wichardt 2008)) and finding maximin strategies is known to be very hard (Koller and Megiddo 1992). Moreover, practical algorithms (Lanctot et al. 2012) require strong assumptions on the game structure that do not in general hold in NFGSS.

Previous works therefore tackled the algorithmic challenges in NFGSS along two main lines: (1) using a compact *network-flow* representation of strategies with linear programming (McMahan 2006; Jiang et al. 2013), or (2) iterative, strategy-generation framework of double-oracle algorithms (McMahan, Gordon, and Blum 2003; Halvorson, Conitzer, and Parr 2009; Jain et al. 2011; Vanek et al. 2012; Jain, Conitzer, and Tambe 2013). The central idea of the iterative approach is to restrict the size of the game by limiting the strategies that can be played, and then to iteratively relax the restrictions by adding new strategies into the restricted game based on best-response calculations in the complete game. As we show, however, both of the existing approaches have limited scalability.

In this paper we combine these two lines of research and introduce new compact-strategy double-oracle (CS-DO) algorithm for NFGSS. We first formally define NFGSS, describe the compact network-flow representation, and linear program for computing strategies in this representation. We follow by the description of the CS-DO algorithm. Our algorithm is inspired by the sequence-form double-oracle algorithm designed for solving EFGs (Bosansky et al. 2013) that also uses compact representation, but which cannot be used directly for NFGSS due to its requirement of perfect recall. CS-DO thus extends this idea to NFGSS and it is able to: (1) handle the imperfect recall in this class of games, (2)

[1] A single player without considering an opponent would have over 2×10^6 states when remembering the history in the smallest graph (3×6) in our experiments.

operate with a more general structure of the utility values that appear in NFGSS, and (3) use a more fine-grained expansion of the restricted game. Afterwards we turn to the experimental evaluation of CS-DO, where two previous lines of work act as the baseline approaches. The results show that CS-DO provides significant computation time improvements and allows us to scale to much larger instances of games. Moreover, we offer novel insights into the performance of the double-oracle algorithms by (1) directly comparing two substantially different approaches of expanding the restricted game, and (2) by demonstrating a strong correlation between the size of the support in the complete game and the relative performance of iterative algorithms.

Technical Background

This section describes the key components of normal-form games with sequential strategies (NFGSS). We focus on two-player zero-sum games, where $N = \{1, 2\}$ is a set of players; we use index i to denote one of the players, $-i$ is the opponent of player i. To solve a game we must find a strategy profile (i.e., a strategy for each of the players) that satisfies the conditions of a solution concept. Here we use the *Nash equilibrium* (NE) solution concept that describes behavior of agents under the assumption of rationality.

In normal-form games the *pure strategies* are associated with actions that the players can take in the game. A *mixed strategy* is a probability distribution over the set of all pure strategies of a player and we denote by Δ the set of all pairs of mixed strategies for both players. For any pair of strategies $\delta \in \Delta$ we denote utility $U(\delta) = U(\delta_1, \delta_2)$ to be the expected outcome for player 1 that tries to maximize this utility, while player 2 minimizes it. A best response of player 1 to the strategy of the opponent δ_2 is a strategy δ_1^{BR} for which $U(\delta_1^{BR}, \delta_2) \geq U(\delta_1', \delta_2)$ for all strategies $\delta_1' \in \Delta_1$. Best response is defined similarly for player 2. A strategy profile $\delta^* = (\delta_1^*, \delta_2^*)$ is a NE if for each player i it holds that the strategy δ_i^* is a best response to the strategy of the opponent. A game can have multiple NE, but all of them have the same expected utility for the players, called the *value of the game* and denoted by V^*.

Normal-Form Games with Sequential Strategies

In NFGSS, the strategy space of a player formally corresponds to a finite-horizon, acyclic Markov decision process (MDP). Each player follows a different MDP that represent her observable states (we use lower index to indicate the player). We denote S_i to be the set of all states in MDP, A_i to be the set of actions in the MDP of player i (called *marginal actions*). We also use A_i as a function to refer to a subset of actions applicable in particular state $s \in S_i$ as $A_i(s) \subseteq A_i$. We allow stochastic transitions in MDPs; $T : S_i \times A_i \times S_i \to \mathbb{R}$ is the function that defines the transition probabilities. In NFGSS, a pure strategy is a selection of an action to play in each state of the MDP, and a mixed strategy is a probability distribution over pure strategies.

We define the utility function in NFGSS by following the approach present in literature (e.g., see (McMahan 2006; McMahan and Gordon 2007; Jiang et al. 2013)) and assume that each combination of actions (a_i, a_{-i}) applicable in some states s_i and s_{-i} can have assigned a utility value (further termed as *marginal utilities*). We denote this utility value as $U\left((s_i, a_i), (s_{-i}, a_{-i})\right)$. The overall expected outcome of the game can be then linearly composed from the utility values assigned to combinations of marginal actions. Formally, for a mixed strategy profile $\delta = (\delta_i, \delta_{-i})$ we set:

$$U(\delta_1, \delta_2) = \sum_{S_1 \times A_1} \sum_{S_2 \times A_2} \delta_1(s_1, a_1)\delta_2(s_2, a_2) \cdot U((s_1, a_1), (s_2, a_2)) \tag{1}$$

where $\delta_i(s_i, a_i)$ represents the probability that state s_i is reached and then action a_i is played in this state when player i follows mixed strategy δ_i. This assumption is sometimes called a *separability condition* (Jiang et al. 2013).

Compact Representation of Strategies A separable utility function allows the mixed strategies to be compactly represented as a network flow (McMahan 2006; Yin et al. 2012; Jiang et al. 2013). We use $x : S_1 \times A_1 \to \mathbb{R}$ to represent the *marginal probability* of an action being played in a mixed strategy of player 1, X denotes the set of all possible network-flow strategies (we use y and Y for player 2). We also use $x(s_1)$ to refer to the probability that the state would be reached following this strategy. Formally, this probability is calculated as the sum of marginal probabilities incoming to state s_1 and it must be equal to the sum of the marginal probabilities assigned to the actions played in this state:

$$x(s_1) = \sum_{s_1' \in S_1} \sum_{a_1' \in A(s_1')} x(s_1', a_1') \cdot T(s_1', a_1', s_1) = \\ = \sum_{a_1 \in A(s_1)} x(s_1, a_1) \qquad \forall s_1 \in S_1 \tag{2}$$

$$0 \leq x(s_1, a_1) \leq 1 \qquad \forall (s_1, a_1) \in S_1 \times A(s_1) \tag{3}$$

We assume that each MDP has a starting root state, denoted s_1^r for player 1, with probability $x(s_1^r) = 1$.

Solving the Game We compute a pair of equilibrium strategies x^*, y^* using a linear program (LP) that is similar to sequence-form LP (Koller, Megiddo, and von Stengel 1996; von Stengel 1996). Let v_{s_2} be the expected utility value that can be achieved from state s_2 for player 2. In equilibrium strategies, both players have the same expected value in their root states (i.e., in $v_{s_1^r}$ and $v_{s_2^r}$) equal to the value of the game V^*. In NE, player 2 is playing the best response by selecting such action a_2 in state s_2 that the expected utility value of the state v_{s_2} is minimal. Therefore, the expected utility v_{s_2} is smaller than or equal to the expected value for each state-action combination (s_2, a_2) that consists of immediate expected utility of this action and the expected utility of the succeeding states s_2', weighted by the probability that this state would be reached $T(s_2, a_2, s_2')$:

$$\max_{x \in X} \; v_{s_2^r} \tag{4}$$

$$v_{s_2} \leq \sum_{s_1, a_1 \in S_1 \times A_1} x(s_1, a_1) \cdot U((s_1, a_1), (s_2, a_2)) + \\ \sum_{s_2' \in S_2} v_{s_2'} T(s_2, a_2, s_2') \quad \forall (s_2, a_2) \in S_2 \times A(s_2) \tag{5}$$

where x satisfies network-flow constraints (2)-(3). A similar program can be constructed for player 2 with a minimizing objective and reversing the inequality in constraint (5).

Compact-Strategy Double-Oracle Algorithm

This section describes the compact-strategy double-oracle (CS-DO) algorithm. We first give an overview of the algorithm, following by the detailed description of each of the main components.

Our algorithm builds on the double-oracle algorithm (McMahan, Gordon, and Blum 2003; Vanek et al. 2010; Jain et al. 2011; Jain, Conitzer, and Tambe 2013) that repeats three steps until convergence: (1) create a *restricted game* by limiting the set of pure strategies that each player is allowed to play, (2) compute a pair of Nash equilibrium strategies in this restricted game, and (3) compute a best response strategy for each player against the current equilibrium strategy of the opponent. The best response may be *any* pure strategy in the unrestricted game. This strategy is added to the restricted game and may be a part of the solution in the next iteration. The algorithm terminates if neither of the best responses to the equilibrium strategies improve over the value of the equilibrium strategies in the restricted game.

CS-DO does not use the pure strategies. Instead, the main idea is to (1) create the restricted game by restricting the players to play a specific implicit *default strategy* in each node of their respective MDPs; (2) solve this restricted game using the LP described in the previous section, and (3) gradually relax this restriction by allowing new marginal actions to be played for specific states of MDPs.

There are several notable differences between CS-DO and the most related algorithm (Bosansky et al. 2013) that makes the methods for creating, expanding, and maintaining valid restricted game more complex. First, CS-DO expands two separate MDPs instead of a single EFG game tree. Second, the utility value is defined for any pair of marginal actions in NFGSS instead of only for the terminal states in EFGs. Third, states in MDPs do not encode the complete history and the assumption of perfect recall does not hold for strategies in NFGSS. Finally, CS-DO allows a more fine-grained expansion of the restricted game, where the best-response algorithms can suggest only a single marginal action to add to the restricted game.

Restricted Game

We use an example MDP to illustrate the methods for the restricted game (see Figure 1). For brevity, we use a deterministic MDP, however, all formulas and ideas hold for the stochastic transitions as well.

The restricted game in CS-DO is determined by the sets of states $S_i^R \subseteq S_i$ and actions $A_i^R \subseteq A_i$ for each player (we omit player index when not needed). We say that state s is *included* in the restricted game iff $s \in S^R$ (states t_1–t_5 and one terminal state in the example); we say that an included state is *expanded* iff either there is some action playable in this state in the restricted game (i.e., $A^R(s) \neq \emptyset$; states t_1,t_2, and t_5 in the example), or this state is terminal in the original MDP (state after playing b_{10} in the example). Finally,

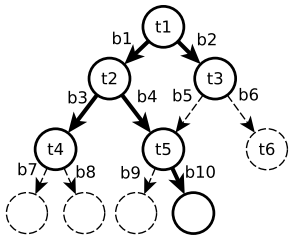

Figure 1: An example of a partially expanded deterministic MDP representing a player's strategy. Solid nodes and edges represent the states and actions included in the restricted game, the dashed ones are not included in the restricted game.

we say that action $a \in A^R$ is *fully expanded* iff all the succeeding states that can be immediately reached by playing this action (i.e., $s' \in S$ s.t. $T(s, a, s') > 0$) are expanded in the restricted game (actions b_1, b_4, and b_{10}). The remaining actions in the restricted game are not fully expanded (actions b_2 and b_3).

The CS-DO algorithm assumes a default strategy to be played in nodes that are not included, or not expanded in the restricted game (e.g, choosing the leftmost action in the example). Formally, the default strategy is a mixed strategy[2] $\delta_i^{\text{DEF}} \in \Delta_i$. As new states and actions are added into the restricted game, the player can choose a strategy that differs from the default strategy and that consists of actions added to the restricted game. The algorithm assumes that whenever action a is added into the restricted game, all immediate successor states are also added into the restricted game.

Finally, we use $\overline{x}$ (or $\overline{y}$ for player 2) to represent the *extended strategy* in the complete game, which is created by extending the strategy x from the restricted game with the default strategy. Depending on which states and actions are included in the restricted game, we put $\overline{x}(s, a)$ equal to:

- 0, if state s is expanded, but action a is not in $A_1^R(s)$
- $x(s, a)$, if state s is expanded and action a is in $A_1^R(s)$
- $\overline{x}(s) \cdot \delta_1^{\text{DEF}}((s, a)|s)$ otherwise, where the marginal probability is calculated as a product of probability that this state would be reached from its predecessors $\overline{x}(s)$, and the probability distribution over actions in this state according to the default strategy calculated as a conditional probability $\delta_1^{\text{DEF}}((s, a)|s)$.

Maintaining a Valid Restricted Game The algorithm needs to ensure that the strategy in the restricted game can always be extended into a well-formed strategy in the complete game – i.e., to keep the restricted game valid. However, naïvely adding new actions and states into the restricted game can make the extended strategies malformed, causing the double-oracle algorithm to converge to an incorrect result. This is due to the imperfect recall, since one state can be reached using different combinations of actions.

[2]This strategy can be any arbitrary but fixed mixed strategy. We use a pure default strategy that selects the first action according to an arbitrary but fixed ordering of $A(s)$.

An example situation is depicted in Figure 1 (choosing the leftmost action is the default strategy), where the extended strategy violates the network-flow constraints in state t_5. According to the LP of the restricted game, the marginal probability of action b_{10} equals to the marginal probability of action b_4. However, the overall incoming probability into the state t_5 is equal to $b_4 + b_2$ in the extended strategy due to playing the default strategy in state t_3.

The algorithm prevents these situations and whenever a new state s is added into the restricted game, the algorithm checks: (1) whether this state can be reached by following the default strategy from a different state s' that is already included, but not yet expanded in the restricted game, and (2) if this state s is not expanded in the restricted game yet, whether it is possible to reach some different state s'' that is already included in the restricted game by following the default strategy from s. If such a sequence of actions and state s' or s'' is detected, all states and actions corresponding to this sequence are also added into the restricted game.

Utility Values in Restricted Game The solution of a valid restricted game is found with network-flow LPs constructed from the restricted sets S^R and A^R for each player. However, the algorithm must also modify the utility function to address the default strategy. Consider again our example. State t_4 is included in the restricted game, but it is not expanded yet and the default strategy is used in this state. In case there is a large negative utility assigned to actions b_7 and b_8, the unmodified utility function overestimates the value after playing action b_3 and the solution of the restricted game would not be optimal.

The algorithm thus uses a modified utility function U^R for each action in the restricted game that is not fully expanded. This modified utility function propagates the utility values of all actions that will be played using the default strategy in not expanded states from now on. Formally we define the new utility function U^R to be equal to U for each combination of actions in states, if either (1) *both* of these actions *are fully expanded* in the restricted game, or (2) *neither* of these actions *is included* in the restricted game. Otherwise, we add to the utility of an action also the utility value corresponding to the continuation of the strategy according to the default strategy. From the perspective of player 1, for each not fully expanded action a_1 played in state s_1, we use:

$$U^R((s_1,a_1),(s_2,a_2)) = U((s_1,a_1),(s_2,a_2)) + \sum_{s'_1 \in S_1 \,:A^R(s'_1)=\emptyset} \; \sum_{a'_1 \in A(s'_1)} U^R((s'_1,a'_1),(s_2,a_2)) \cdot \delta_1^{\text{DEF}}((s'_1,a'_1)|(s_1,a_1)) \quad (6)$$

where $\delta_1^{\text{DEF}}((s'_1,a'_1)|(s_1,a_1))$ denotes the conditional probability of playing an action a'_1 in state s'_1 after playing action a_1 in s_1 (this probability is zero if state s'_1 is not reachable after playing a_1). The algorithm calculates utility values for every state and action of the opponent (s_2, a_2) that can have a non-zero extended strategy $\overline{y}$.

Note that we use U^R for any subsequent nodes s'_1 and actions a'_1 in the definition. This is due to the possibility of action of the opponent a_2 also being not fully expanded. Since we assume that both MDPs forming strategy spaces for the players are finite and acyclic, this modified utility function is well-defined. CS-DO algorithm uses a two-step dynamic program to calculate these values in order to avoid repeated calculations:

Step 1: First, the algorithm calculates utility values U^R for each action of player 1 that is included but not fully expanded in the restricted game, against all actions of the opponent (s_2, a_2) that are either (1) included in the restricted game (i.e., $s_2 \in S_2^R \wedge a_2 \in A_2^R$), or (2) they can have a non-zero probability in the extended strategy from the restricted game (i.e., $\exists s' \in S_2^R : \delta_2^{\text{DEF}}((s_2,a_2)|s'_2) > 0$). The algorithm calculates the values according to Formula 6 using only original utility function U on the right side of the equation.

Step 2: Next, the algorithm calculates utility values U^R for each action of player 2 that is included but not fully expanded in the restricted game according to Formula 6, and uses utility values U^R calculated during the first step when necessary on the right side of the equation.

Best Response: s - current state, $\overline{y}$ - extended strategy of the opponent

```
1: if s is terminal then
2:     return 0
3: for a ∈ A_1(s) do
4:     v_a ← Σ_{s_2∈S_2, a_2∈A_2} ȳ(s_2,a_2) U((s,a),(s_2,a_2))
5:     for s' ∈ S_1 s.t. T(s,a,s') > 0 do
6:         v_a ← v_a + BR_1(s', ȳ) · T(s,a,s')
7: maxAction ← arg max_{a∈A_1(s)} v_a
8: if maxAction is not in the default strategy or
       there is a change in some successor of s then
9:     backup s,maxAction
10: return v_{maxAction}
```

Figure 2: The best-response algorithm for player 1.

Best-Response Algorithm

The best-response algorithm (BR) determines which states and actions to add to the restricted game. For each player, BR returns the set of best-response actions as well as the expected utility value of the best response. Best response is calculated against the current optimal *extended strategy* of the opponent, so the default strategy of the opponent is taken into account. BR is a depth-first search through the unrestricted MDP selecting the best action to play in each state (BR for player 1 is in Figure 2). In each state, the algorithm calculates the expected utility value for each action applicable in this state (line 4) and then recursively calculates the expected utility for each of the successors (line 6). Finally, the algorithm selects the best action for this node (line 7) and keeps it as a best-response candidate (line 9). Best-response actions are then found by following the best-response candidates from the root state.

There is one notable exception to the described behavior. If the selected $maxAction$ for some state s corresponds to an action prescribed by the pure default strategy, this action is stored as a best-response candidate only if some action

a' was selected in one of the successors of s and this action a' is not prescribed by the pure default strategy (line 8). This condition ensures that the algorithm adds the default-strategy actions into the restricted game only if necessary.

Convergence Analysis

CS-DO algorithm converges to a Nash equilibrium of the complete game. The convergence follows from three steps: (1) note that a best-response algorithm returns a value that is strictly better for the player if and only if there exists an action not included in the restricted game that is a part of the best response; (2) if such an action does not exist for either of the players, the value of the best responses must be equal to the current value of the restricted game, as well as the value of the complete game V^*; (3) the algorithm is finite because in each iteration either at least one action is added to the restricted game, or the algorithm terminates.

Experiments

We experimentally compare the performance of the CS-DO algorithm with the standard pure-strategy double-oracle algorithm (PS-DO), and the algorithm solving the full linear program using compact strategies (FULLLP). We use variants of search games inspired by existing games. The first game is inspired by the maritime security problem (*Transit Game*; (Vanek et al. 2012)), the second game is inspired by the border security (*Border Protection Game*; (Bosansky et al. 2013)). In both of these games, the evader tries to cross an area represented as a graph, while the defender tries to discover the evader. The evader receives a reward of 1 for reaching the goal undetected, while the defender receives a reward of 1 for encountering the evader. The games are played for a fixed number of time steps and differ in the proportional sizes of the strategy space for the players as well as the size of the support for typical solutions. They are both parameterizable, allowing us to evaluate performance across different conditions.

Neither of the algorithms use any domain-specific knowledge. Experiments were run using a single thread on a standard PC. Each of the algorithms was given a maximum of 7 GB of memory for a process, and we used IBM CPLEX 12.5 to solve the linear programs.

Experiment Settings

Transit Game The game is played on a grid, undirected graph (see Figure 3, left subfigure). The evader tries to cross the graph from left to right (the selection of a node in the left-most column is the first move) and receives a small penalty (0.02) for each step. The defender controls a single patrolling unit that starts in the base (the black node). The patrolling unit has limited resources and receives a large penalty if it does not return to the base by the end of the game. The movement of the units may fail with probability 0.1 (the unit stays in the same node). We vary the size of the graph by increasing the number of the rows in the graph (*width*), the number of the columns in the graph (*length*), and the number of steps for both players.

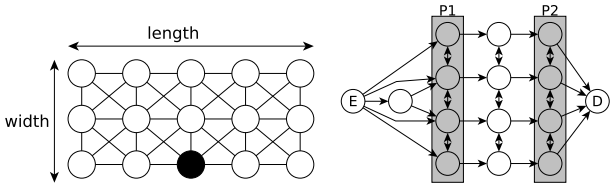

Figure 3: Examples of the graphs for the search games. The evader aims to cross from left to right.

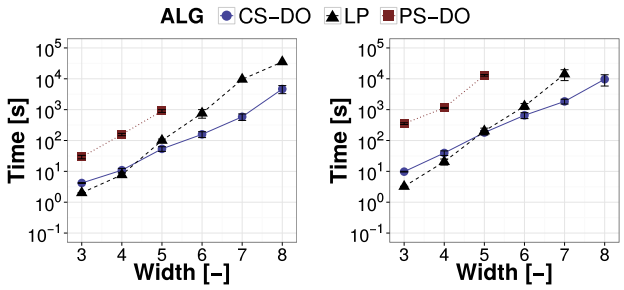

Figure 4: Comparison of the computation times with increasing size of the scenario in Transit Game. *Length* of the graph is fixed to $2\times$*width*, the number of steps in the game are set to *length*$+2$ (left graph) and *length*$+4$ (right graph).

Border Protection Game The game is played on a directed graph (see Figure 3, right subfigure), where the evader aims to cross safely from a starting node (E) to a destination node (D), and the defender controls two patrolling units operating in the selected nodes of the graph (the shaded areas P1 and P2). Both patrolling units of the defender move simultaneously and may observe the signs of the recent passage in the visited nodes. Nature determines whether the signs appear in a visited node (the probability is 0.1).

Results

The results are means from several runs (at least 10 for smaller instances, the error bars in figures visualize 95% confidence interval). In each run we used a different random ordering of actions in states of MDPs to eliminate the effect of some particular ordering that also determines the default strategy in CS-DO.

Transit Game
Results in the Transit Game demonstrate the benefits of our algorithm and show better scalability than both FULLLP and PS-DO. We scaled the size of the scenario for fixed ratio of the graph (*length*$= 2\times$*width*) and fixed number of steps, where *steps*=*length*$+2$, or *steps*=*length*$+4$. The results (see Figure 4) show that CS-DO outperforms PS-DO by several orders of magnitude even for the smallest scenarios. This is caused by the exponential number of possible pure strategies and relatively large support in the complete game (the size of the support is typically $10-25\%$ of all marginal actions for each player). These factors contribute to the large number of iterations and slow convergence of PS-DO.

CS-DO also outperforms FULLLP and the difference is increasing with the increasing size of the game. This is due to the fact that CS-DO is still able to create substantially smaller restricted games (only $\approx 20\%$ of all marginal actions for each player are part of the restricted games). On the other hand, FULLLP outperforms PS-DO due to the compact strategy representation and the fact that the depicted graph sizes fit into memory (the largest game in this comparison has $10,503$ marginal actions for the evader and $13,210$ for the defender).

The following table shows the cumulative times the algorithms spent while solving LP, calculating best responses (BR), and constructing the restricted game (RG), in a scenario with width 5, length 10, and 12 steps:

Algorithm	Iterations	Total [s]	LP [s]	BR [s]	RG [s]
FULLLP	-	100	90	-	-
PS-DO	585	907	190	72	576
CS-DO	146	53	18	31	1

Results show a large number of iterations of PS-DO causing the algorithm to be the slowest. The majority of the time is used to construct and solve the restricted game. Constructing the restricted game is more time consuming since computing the utility value for a pair of pure strategies is much more expensive than calculating utility for pair of marginal actions. CS-DO, on the other hand, converges rather quickly and the majority of time is used to calculate best-responses. This is because the size of LP for the restricted game is rather small (on average it adds 675 evader actions (25% of all marginal actions) and 793 defender actions (24%)).

Better scalability of CS-DO is apparent when we further increase the sizes of the graphs. While CS-DO successfully solves even large graphs (width set to $10 - 11$) in less than 3 hours, FULLLP fails to construct such games due to the memory constraints (the game has $26,610$ marginal actions of the evader and $33,850$ actions of the defender). Moreover, FULLLP is often unable to solve the largest instances that fit into memory in 60 hours.

Border Protection Game This game is better suited for the standard double-oracle algorithm, since it has unequal sizes of the strategy space (the defender controls two units that move simultaneously and can observe signs in nodes, while evader observes only its own position), and extremely small support solutions ($\approx 3\%$ of the marginal actions for each player). The left subfigure in Figure 5 shows the computation times. The results confirm that the performance of PS-DO dramatically improves. Both double-oracle algorithms perform comparably and they both significantly outperform FULLLP. The time breakdown for the game with 5 steps is shown in the following table:

Algorithm	Iterations	Total [s]	LP [s]	BR [s]	RG [s]
FULLLP	-	12,177	12,174	-	-
PS-DO	9	9.5	0.05	8.13	0.52
CS-DO	11	13	0.2	8.8	1.92

Both double-oracle algorithms are extremely efficient at finding all strategies needed for solving the game. CS-DO adds less than 7% of all marginal actions for the evader and less than 3% for the defender. The slightly worse performance of CS-DO is a result of additional overhead when expanding the restricted game, and the slightly larger number of iterations due to the more conservative expansion of the restricted game.

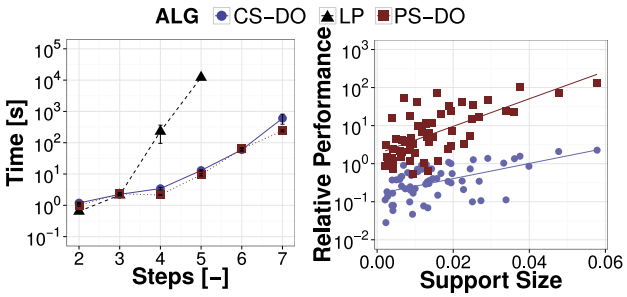

Figure 5: (Left) Comparison of computation times for Border Protection Game with graph from Figure 3 and increasing number of steps. (Right) Comparison of the relative performance of the DO algorithms depending on the size of the support.

Support Size Analysis

The previous results suggest that there is a correlation between the size of the support in the complete game and the performance of the DO algorithms. We analyze this factor in more detail by using the Transit Game to produce games with differing sizes of the support. We alter the utility values for the evader's reward from 1 to 20 and her penalty for a time step from 0.01 to 2. We ran the experiments on small graphs of width 4 and 5, where FULLLP performs well.

The right subfigure in Figure 5 shows the relative performance of the double-oracle algorithms with respect to FULLLP. The x-axis shows the size of the support for the solution relative to the complete game, calculated by multiplying the proportional support sizes for both players. The results show that there is a strong correlation between these two factors (correlation coefficients are 0.715 for CS-DO and 0.729 for PS-DO); the double-oracle algorithms typically perform worse with larger support. Moreover, the comparison shows that CS-DO is less sensitive to the changes in the size of the support (see the linear fit of data, note the logarithmic y-scale). These experimental results demonstrate that the size of the support can be a good indicator of the expected performance of the iterative approach in practice.

Conclusions

This paper presents a novel hybrid algorithm for solving normal-form games with sequential strategies that combines a compact representation with incremental strategy generation. Experimental results confirm that our novel algorithm outperforms existing approaches and scales to much larger scenarios. Moreover, this paper provides unique insights into the performance of double-oracle algorithms by comparing two substantially different methods of incremental strategy generation and correlating their relative performance to the size of the equilibrium support.

The presented algorithm and experimental results can stimulate future research. Primarily, our approach can be tailored to specific domains and deployed in many real-world scenarios. Secondly, the main idea of our algorithm, the concept of a default strategy to be played in local states of the players and incremental relaxation of this restriction, can transferred to more generic models such as decentralized MPDs, or stochastic games. Finally, the experimental results on support size suggest the directions for theoretical analysis of the convergence rate for double-oracle algorithms.

Acknowledgments

This research was supported by the Czech Science Foundation (grant no. P202/12/2054), by the Danish National Research Foundation and The National Science Foundation of China (under the grant 61361136003) for the Sino-Danish Center for the Theory of Interactive Computation, by the Center for Research in Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council, by the MURI grant W911NF-11-1-0332, and by the Army Research Office (ARO) under award number W911NF-13-1-0467.

References

Bosansky, B.; Kiekintveld, C.; Lisy, V.; Cermak, J.; and Pechoucek, M. 2013. Double-oracle Algorithm for Computing an Exact Nash Equilibrium in Zero-sum Extensive-form Games. In *Proceedings of 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 335–342.

Halvorson, E.; Conitzer, V.; and Parr, R. 2009. Multi-step Multi-sensor Hider-seeker Games. In *Proceedings of the Joint International Conference on Artificial Intelligence (IJCAI)*, 159–166.

Hoda, S.; Gilpin, A.; Peña, J.; and Sandholm, T. 2010. Smoothing techniques for computing nash equilibria of sequential games. *Mathematics of Operations Research* 35(2):494–512.

Jain, M.; Korzhyk, D.; Vanek, O.; Conitzer, V.; Tambe, M.; and Pechoucek, M. 2011. Double Oracle Algorithm for Zero-Sum Security Games on Graph. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 327–334.

Jain, M.; Conitzer, V.; and Tambe, M. 2013. Security Scheduling for Real-world Networks. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 215–222.

Jiang, A. X.; Yin, Z.; Zhang, C.; Tambe, M.; and Kraus, S. 2013. Game-theoretic Randomization for Security Patrolling with Dynamic Execution Uncertainty. In *Proceedings of 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 207–214.

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

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

Lanctot, M.; Gibson, R.; Burch, N.; Zinkevich, M.; and Bowling, M. 2012. No-Regret Learning in Extensive-Form Games with Imperfect Recall. In *ICML*, 1–21.

McMahan, H. B., and Gordon, G. J. 2007. A unification of extensive-form games and Markov decision processes. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, 86.

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

McMahan, H. B. 2006. *Robust Planning in Domains with Stochastic Outcomes, Adversaries, and Partial Observability*. Ph.D. Dissertation, Carnegie Mellon University.

Sandholm, T. 2010. The State of Solving Large Incomplete-Information Games, and Application to Poker . *AI Magazine* special issue on Algorithmic Game Theory:13–32.

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

Vanek, O.; Bosansky, B.; Jakob, M.; and Pechoucek, M. 2010. Transiting Areas Patrolled by a Mobile Adversary. In *Proceedings of IEEE Conference on Compuational Intelligence and Games*.

Vanek, O.; Bosansky, B.; Jakob, M.; Lisy, V.; and Pechoucek, M. 2012. Extending Security Games to Defenders with Constrained Mobility. In *Proceedings of AAAI Spring Symposium GTSSH*.

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

Washburn, A., and Wood, K. 1995. Two-person Zero-sum Games for Network Interdiction. *Operations Research* 43(2):243–251.

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

Yin, Z.; Jiang, A. X.; Johnson, M. P.; Tambe, M.; Kiekintveld, C.; Leyton-Brown, K.; Sandholm, T.; and Sullivan, J. P. 2012. TRUSTS: Scheduling Randomized Patrols for Fare Inspection in Transit Systems. In *Proceedings of 24th Conference on Innovative Applications of Artificial Intelligence (IAAI)*.

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