Using Classical Planning in Adversarial Problems

Pavel Rytíř[1], Lukáš Chrpa[1,2], Branislav Bošanský[1]
[1]Dept. of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague
[2]Faculty of Mathematics and Physics, Charles University
{rytirpav, chrpaluk, bosansky}@fel.cvut.cz

***Abstract*—Many problems from classical planning are applied in the environment with other, possibly adversarial agents. However, plans found by classical planning algorithms lack the robustness against the actions of other agents – the quality of computed plans can be significantly worse compared to the model. To explicitly reason about other (adversarial) agents, the game-theoretic framework can be used. The scalability of game-theoretic algorithms, however, is limited and often insufficient for real-world problems. In this paper, we combine classical domain-independent planning algorithms and game-theoretic strategy-generation algorithm where plans form strategies in the game. Our contribution is threefold. First, we provide the methodology for using classical planning in this game-theoretic framework. Second, we analyze the trade-off between the quality of the planning algorithm and the robustness of final randomized plans and the computation time. Finally, we analyze different variants of integration of classical planning algorithms into the game-theoretic framework and show that at the cost a minor loss in the robustness of final plans, we can significantly reduce the computation time.**

I. Introduction

Classical domain-independent planning is among the key areas of artificial intelligence mainly focusing on plan-generation methods [1]. However, many application domains for planning algorithms have multiple actors – e.g. planning defense measures, information collection in an adversarial environment or planning a robust mission where nature acts as the other actor. Ignoring the presence of other actors (or agents) can have severe consequences as the actual quality of the plan can be considerably worse than expected. Therefore, the planning formalism can be updated in order to handle multiple agents and multiple goals the agents pursue [2], [3].

However, in order to explicit reason about other agents and find a provably robust plan, *game-theoretic* methods have to be used. Game theory states that in order to act optimally in a game, agents may need to randomize over several plans or actions, so the other agents have uncertainty about which plan is going to be executed making it difficult for them to exploit such strategy. Existing planning algorithms focus on *congestion games* where the task is to find an optimal robust multi-agent plan for non-cooperating agents [4], [5], [6] or on *Stackelberg games* where the task is to find a pure plan of the leader that is robust against the adversary [7]. Such algorithms are not able to find such randomized plans. While there are several successful applications of game-theoretic algorithms in practice, for example in domains of physical security [8] or protecting wildlife [9], most of the methods used for scaling-up are domain-dependent and their transferability to other domains is limited.

One of the general game-theoretic algorithms is the incremental strategy generation method called the *double oracle* algorithm [10]. Double oracle (DO) algorithm tackles one common problem of games – in case agents have exponentially many possible plans to choose from, the size of the solved game is intractable. To this end, DO algorithms restrict the space of possible plans to choose from – the algorithm forms *a restricted problem* that is iteratively expanded by calculating and adding into the problem new plans as *best responses* to the current strategy of the other agent from the restricted problem. Although, in the worst case, all plans have to be added into the restricted problem, it rarely happens in practice and DO algorithms are often able to find an optimal strategy using only a fraction of all possible plans (see e.g [11], [12], [13]). While this methodology is general, practical applications use domain-specific algorithms for computing best response plans [14], [13], hence it typically requires a non-trivial work on designing and implementing best response algorithms for each domain.

This limitation can be overcome by using domain-independent planning algorithms for computing the best response plans. However, the standard assumption in DO is that the best response algorithm finds an optimal plan (or at least one with a bounded error) given the strategy of the opponent. This is, however, not true for general domain-independent planning algorithms. Similarly to other DO-based approaches with heuristic best response algorithms (based on reinforcement learning) [13], [15], [16], we parametrize the planning algorithms with computation time thus having heuristic best response algorithms of differing quality. Contrary to existing approaches, we are also able to compute an exact best response using a domain-specific A^* algorithm. Therefore, we are able to exactly identify possible weaknesses of using heuristic best response algorithms.

Our main contributions include (1) the methodology for using algorithms from classical domain-independent planning algorithms in the game-theoretic framework for computing robust randomized plans; (2) the experimental analysis of the trade-off between the quality of the plans computed as the best responses, the robustness of the final randomized strategy, and the required computation time; (3) comparison of several variants of integration of the classical domain-independent planning into the DO framework and proposition of a novel variant that for a minor loss in the final robustness achieves a significant computation speed-up. We demonstrate

our approach on a simple, yet computationally challenging resource collection problem[1] with two agents where each agent controls a group of UAVs and the goal is to collect information from a defined set of resources before the opponent. Once the information is collected from a resource, it can no longer be collected by the other agent.

II. Technical Background

This section introduces the terminology used in the paper.

A. Classical Planning

Classical planning, the simplest form of Automated Planning, assumes a static, deterministic and fully observable environment; a solution plan amounts to a sequence of actions.

A ***Planning Domain Model*** is a couple $\mathcal{D} = (L, A)$, where L is the set of propositional atoms used to describe the state of the environment, and A is the set of actions over L. A set of ***states*** S over L is defined as $S \subseteq 2^L$. In classical planning, we assume that an atom present in a state is true in that state while an atom not being present in a state is considered to be false in that state. An ***action*** is a quadruple $a = (\mathit{pre}(a), \mathit{del}(a), \mathit{add}(a), \mathit{cost}(a))$, where $\mathit{pre}(a), \mathit{del}(a)$ and $\mathit{add}(a)$ are sets of atoms from L representing a's precondition, delete, and add effects, respectively, and $\mathit{cost}(a)$ represents cost of a's execution. An action a is *applicable* (or executable) in a state s if and only if $\mathit{pre}(a) \subseteq s$.

If possible, application (or execution) of a in s, denoted as $\gamma(s, a)$, yields the successor state of the environment $(s \setminus \mathit{del}(a)) \cup \mathit{add}(a)$, otherwise $\gamma(s, a)$ is undefined. The notion of applicability can be extended to sequences of actions, i.e., $\gamma(s, \langle a_1, \ldots, a_n \rangle) = \gamma(\ldots \gamma(s, a_1) \ldots, a_n)$.

A ***Planning problem*** (or a problem instance) is a triple $\mathcal{P} = (\mathcal{D}, I, G)$, where $\mathcal{D}$ is a planning domain model, I is the initial state of the environment, and G is the goal in the form of a set of propositions. A ***plan*** $\pi = \langle a_1, \ldots, a_n \rangle$ (for a planning problem $\mathcal{P}$) is a sequence of actions (defined in $\mathcal{D}$) such that their consecutive application starting in the initial state results in a state satisfying the goal (i.e., a goal state), i.e., $G \subseteq \gamma(I, \pi)$. We say that a plan $\pi = \langle a_1, \ldots, a_n \rangle$ (for $\mathcal{P}$) is ***optimal*** if for every plan $\pi' = \langle a'_1, \ldots, a'_m \rangle$ (for $\mathcal{P}$) it is the case that $\sum_{i=1}^{n} \mathit{cost}(a_i) \leq \sum_{j=1}^{m} \mathit{cost}(a'_j)$.

To reason with (discrete) time we can introduce a "timeline" objects into the domain model representing specific timestamps. Although we have to know the upper bound (i.e., the latest timestamp) upfront as in PDDL all the objects have to be specified upfront, it can be estimated from the size of a given problem. Reasoning with timestamps can be embedded into the model by introducing "arithmetic" and "relation" predicates that represent essential operations (e.g., adding, comparing).

[1]Note that we use this problem as an example domain that is challenging enough for the classical domain-independent planning algorithms and we can still compute an optimal plan using a domain-specific algorithm. The goal is not, per se, to design an algorithm for this particular domain.

B. Normal-Form Games

The baseline representation for modeling strategic interaction is *normal-form games* (NFGs). A normal-form game Γ is a tuple (N, S, u), where N is the finite set of players, S_i is a finite set of pure strategies of player i, and u_i is a utility function that assigns a real value for each outcome of the game defined by a strategy profile – an N-tuple of pure strategies (one for each player); $u_i : S \to \mathbb{R}$. A mixed strategy is a probability distribution over pure strategies, $\sigma(s_1)$ represents the probability with which strategy s_1 is played by player 1 (for brevity, we use σ_i to denote some mixed strategy of player i). We restrict on the two player zero-sum setting where $|N| = 2$ and the sum of utility values of players equals to 0 ($u_1 = -u_2$). We say that strategy of one player s_i is the best response to the strategy of the opponent σ_{-i} (denoted as $s_i = br(\sigma_{-i})$) when $u_i(s_i, \sigma_{-i}) \geq u_i(s'_i, \sigma_{-i})$ for all $s'_i \in S_i$. We say that strategy profile σ is in Nash equilibrium (NE) if each player is playing best response to the strategy of the opponent. NE of a zero-sum game can be computed using the standard linear program (e.g., see eqs (4.1)–(4.4) in [17]). The expected utility of player 1 in an NE of the game is termed *value of the game*.

When the number of possible strategies is exponential, solving the linear program becomes computationally intractable. One way for tackling this issue is to incrementally build the game using the *double-oracle algorithm*. The algorithm starts with a restricted game $\Gamma' = (N, S', u)$, where the set of possible pure strategies available to players S' is restricted such that players can select only from a limited set of pure strategies (generally, $S' \subseteq S$). In each iteration of the algorithm, the restricted game Γ' is solved using the LP. Next, each player computes a best response from all strategies S to the strategy of the opponent from the restricted game Γ'. These best response strategies are added into S' and the restricted game is expanded. The algorithm terminates when neither of the players can add a best response strategy that improves the expected outcome from the restricted game. When the algorithm terminates, NE of the restricted game is the same as in the original game (since best response is computed over unrestricted set of all strategies).

III. Planning in Adversarial Domains

In adversarial environments, the quality of plans depends on possible actions of adversaries. As we focus on zero-sum game, for maximizing the reward (or minimizing the cost), the crucial aspect is to apply specific action before the adversary. To illustrate the problem, let us consider two agents who compete against each other in collecting resources. After one agent collects a given resource, the other agent can no longer collect it. Intuitively, a good plan for the agent is such that the agent collects resources before its competitor. To reflect this observation the cost of the "collect" action should be higher if it is planned to be executed (more likely) after the competitor's "collect" action.

Assuming that we have full knowledge of actions and intentions (goals) of the competitor (or adversary) we can

estimate its plans. Specifically, we are interested in actions of the competitor that hinder agent's goals. In a nutshell, agent's *critical* actions for achieving its goals (e.g. the "collect" action) have to be applied before the *adversary* actions of the competitor. Knowing the timestamps of the adversary actions (in the competitor's plan estimate), we can determine *deadlines* for agent's critical actions.

Definition 1. *Let A be a set of agent's actions and A' be a set of competitor's actions. Then, let $A^c \subseteq A$ be a set of* **critical actions** *and for each $a^c \in A^c$ we define a set of* **adversary actions** *$ad(a^c) \subseteq A'$. Let $\pi' = \langle a'_1[t'_1], \ldots, a'_m[t'_m] \rangle$ be a plan of the competitor, respectively (the notation $a'_i[t'_i]$ represents that an action a'_i is executed at timestamp t'_i). Then, for each $a^c \in A^c$ we can determine a* **deadline** *with respect to π' as $\min\{t \mid a[t] \in \pi', a \in ad(a^c)\}$.*

Estimated competitor's plans provide deadlines for agent's critical actions. We can formulate a planning problem such that the agent's plans are optimized for planning critical actions (e.g. the "collect" action) before the deadlines.

Definition 2. *For a single competitor's plan π' and a critical action $a^c \in A^c$ and a timestamp t we define a* **cost function** *$c(a[t], \pi')$ such that $c(a^c[t], \pi') = 0$, for t smaller than the deadline (with respect to π'), $c(a^c[t], \pi') = M$, for t greater than the deadline, and $c(a^c[t]) = M/2$, for t equal to the deadline, where M corresponds to the penalty agent receives for not executing critical action a^c before the competitor's adversary action. For an agent's planning problem $\mathcal{P}$ and competitor's plan π', we define an agent's* **response planning problem** *$\mathcal{P}_{\pi'}$ such that critical actions of $\mathcal{P}$ are associated with the above cost function.*

Optimal plan of the response planning problem (minimizing the total action cost) accounts for the best possible agent's response on the competitor's plan.

IV. Formulating the Game

We now describe how the classical planning algorithms can be integrated into the double oracle algorithm. We define a strictly competitive game $\Gamma_{\mathcal{P}}$ between an agent and its competitor. All possible plans Π_1 of an agent in the planning problem $\mathcal{P}$ form the set of pure strategies in the game. Finally, the utility function for a combination of plans is the sum of the marginal costs of actions in respective plans:

$$u(\pi_1, \pi_2) = \sum_{a^c[t] \in \pi_1} c(a^c[t], \pi_2) \quad \pi_1 \in \Pi_1, \pi_2 \in \Pi_2 \qquad (1)$$

Solving game $\Gamma_{\mathcal{P}}$ requires the double oracle algorithm, since it is not usually possible to enumerate all possible plans in the planning problem for an agent. There are two key steps that must be addressed: (1) how to formulate a planning problem in order for the classical planning algorithm can be used as the best response algorithm; (2) how to use the classical planning algorithm if we do not have guarantees that an optimal plan will be found.

A. Classical Planners as Best Response Algorithms

We need to formulate a response planning problem that corresponds to finding a best response for agent i against a strategy of the opponent from the restricted game σ'_{-i}. Since σ'_{-i} can be a randomized strategy, the opponent of agent i can randomly choose from several plans to execute. This affects the definition of costs in the planning problem of agent i.

As we deal with a randomized strategy, i.e., the opponent can choose a plan from a given set of plans with some probability, we have to generalize the cost function of critical actions from Definition 2 as follows. Let $\pi'_{-i,1}, \ldots, \pi'_{-i,n}$ be the plans of the opponent of agent i and each of them is played with probabilities $\sigma'_{-i}(\pi'_{-i,j})$ (for $j = 1, \ldots, n$ and $\sum_{j=1}^{n} \sigma'_{-i}(\pi'_{-i,j}) = 1$) according to the solution of the restricted game. Then, the cost of each critical action a^c in a given timestamp t is calculated as:

$$cost(a^c[t]) = \sum_{j=1}^{n} \sigma_{-i,j}(\pi'_{-i,j}) \cdot c(a^c[t], \pi'_{-i,j}) \qquad (2)$$

An example of the cost of a mixed strategy follows: Let π_1 and π_2 be two plans such that π_1 collects both resources in time 20 and plan π_2 collects both resources in time 30. Let σ be a mixed strategy of Player 1 that plays π_1 with probability 0.75 and π_2 with probability 0.25. The penalty M is set to 100. Then the cost for each critical action collecting a given resource will be:

- If the resource is collected till 20, the cost is 0
- If the resource is collected exactly at 20, the cost is $37.5 = 100 * 0.75 * 0.5$
- If the resource is collected between 20 and 30, the cost is $75 = 100 * 0.75$.
- If the resource is collected exactly at 30, the cost is $87.5 = 100 * (0.75 + (1.0 - 0.75) * 0.5)$
- If the resource is collected after 30, the cost is 100

B. Using Planners Without Optimality Guarantees

In large scenarios, finding an optimal plan is often not possible using domain-independent planners. Therefore, these planners act as a (heuristic) estimators of best responses in the double oracle algorithm. Classical planning algorithms are typically run with a given time limit and the best plan found within this time limit is returned (e.g., in the planning competition [1]). We adopt this setting and use planning algorithms with a fixed deadline on computation time.

However, in order for the double oracle algorithm to proceed to a next iteration, adding a best response is not necessary and it is sufficient to add a plan (pure strategy) with a better expected outcome for agent i. The new plan is added into the restricted game and the new plan is going to be used by agent i. Therefore, we also use setting where the planning algorithm has a strict deadline on computation time, but the planning algorithm is terminated whenever a plan that is strictly better than the outcome in the restricted game for a particular player is found. This setting is more common in larger game-theoretic scenarios [14] or when reinforcement learning algorithms are used as best responses [13], [16].

C. Termination of Double Oracle with Non-optimal Planners

Finally, we investigate alternatives of termination condition of the double oracle algorithm. Typically, double oracle terminates in case neither of the players can improve the optimal strategy from the restricted game (hereinafter denoted as *both* DO). However, it can happen that during the actual execution of the DO algorithm, new strategies of only one player are added over several iterations – for example, in iteration t, player 1 finds a marginally better plan that is added into the restricted game and player 2 does not have a better response. In iteration $t+1$, player 1 again finds a new marginally better plan, but player 2 does not add anything. This repeats until player 1 is able to find a best response. However, in each iteration, player 2 fails to generate its best response spending the allocated computation time.

To avoid this phenomenon, we propose a novel second variant of termination of double oracle (hereinafter denoted as *single* DO) – the algorithm stops in case at least of the players cannot improve the optimal strategy from the restricted game. This variant can reduce the quality of final strategies but our experimental results show that the loss is rather marginal and the savings in the computation time are substantial.

V. Experiments

We evaluated our algorithm on a two-player game where each player controls multiple unmanned aerial vehicles (UAVs). The goal of each player is to collect resources before the opponent. After a resource is collected, it can no longer be collected by the opponent. A resource can be collected only by a UAV with a suitable sensor. Some resources may require two sensors. In that case, a UAV equipped with the two sensors or two UAVs, each of them with the required sensor that can collect the resource.

The map of the domain is modeled as a graph $G = (V, E)$, where V is the set of locations and E is the set of edges between the locations. Each UAV is located in some location and can move to another location only if there is an edge between them. Each UAV has two available actions. It can move to another location or it can collect a resource, given that UAV is at the same location as some resource and it has a suitable sensor.

We evaluated the DO algorithm on multiple scenarios. For the presentation, we selected three case studies (see Figure 1, blue triangles denote an initial position of the agent's UAVs, red circles are the positions of opponent's UAVs, green squares represent the resources). The first scenario represents an easier case where each resource requires only one sensor to collect (numbers in the brackets next to the resources denote required types of sensors, the numbers next to the UAVs represent types of sensors they are carrying). In the second scenario, the situation is more complicated since 4 out of 6 resources require two types of sensors.

As a domain independent planner, we used LAMA [18]. Since LAMA does not guarantee optimality of found plans, we used a domain-specific A^* algorithm with admissible heuristics for computing provably optimal plans. We ran the experiments on a Linux machine with processor Intel Xeon E5-2620 v4 at 2.10 GHz with 32GB RAM.

As the final measure of quality of produced plans, we compute *approximation error* as the difference between the values of best responses computed to the strategies computed by the double oracle algorithm in the restricted game:

$$error(\sigma) = |u_i(\sigma_i, br(\sigma_i)) - u_i(br(\sigma_{-i}), \sigma_{-i})| \ .$$

For zero-sum games, the error should converge to zero when using best response algorithms with optimality guarantees. Note that we always used LAMA planning algorithm for generating best responses and A^* was used solely for calculating the error with respect to the ground truth.

A. Results

We first examined the quality of final strategies of the double oracle algorithm – the results are depicted in Figure 2. Then, we have calculated the error of the strategies computed as optimal in the restricted game from the last iteration of the double oracle algorithm. The error is calculated either using the LAMA planner with 10-hours deadline, or with our domain-specific A^* planner.

For the first scenario (left graph in Figure 2), the results show that LAMA planner with larger deadline is able to find optimal plans (note that the difference between the error using LAMA planner and A^* is close to 0). Otherwise, the error of final strategies decreases with increasing available computation time given to the planner. However, this improvement in the quality of strategies is not monotonous. Note that while the error of strategies for the deadline 10 seconds is 0.175, the error for the deadline 150 seconds the error is 0.182 (see the left graph in Figure 2) and similar can be see also in other scenarios (see Figure 2).

The explanation of this counter-intuitive phenomenon comes from the fact that, in DO, the players are iteratively computing better and better plans (i.e., the plans with more tight deadlines). Hence a planner might compute a plan setting the near-optimal deadlines for the response planning problem for which the planning algorithm fails to find a better plan for the opponent. Moreover, the set of plans might differ for different time limits.

The second aspect that we analyzed is whether it is better to use the domain-independent planner with a fixed deadline or to use the first better plan found. For the first scenario, the latter method was significantly better reaching optimal strategy (for variant $Any1800$) in only 3850 seconds. For more complex scenarios, the results of the first-better variant are not consistently better (i.e., under the curve depicted by the variants with fixed deadlines), but they are never worse. Most of the time, the better plan was found within a couple of seconds. Therefore, the experimental results suggest that planning algorithms should be used similarly to reinforcement learning algorithms in the first better plan setting.

Finally, our experimental results show that the lack of optimality guarantees can have a significant impact on the quality of final strategies. In more complex scenarios (Scenario

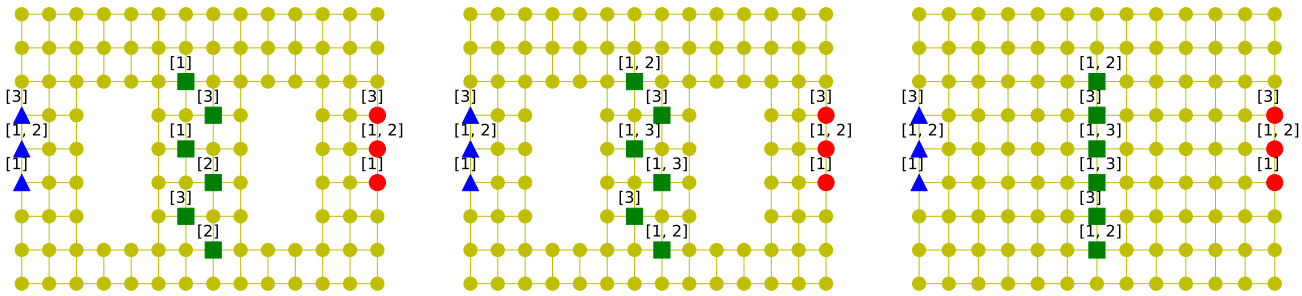

Fig. 1: Visualization of scenarios (numbered from the left, scenario 1, 2, and 3). Blue triangles denote UAVs of the agent, red circles denote UAVs of the opponent and green squares denote resources. Numbers in the brackets next to the resources denote required sensors, the numbers next to the UAVs represent sensors they are carrying.

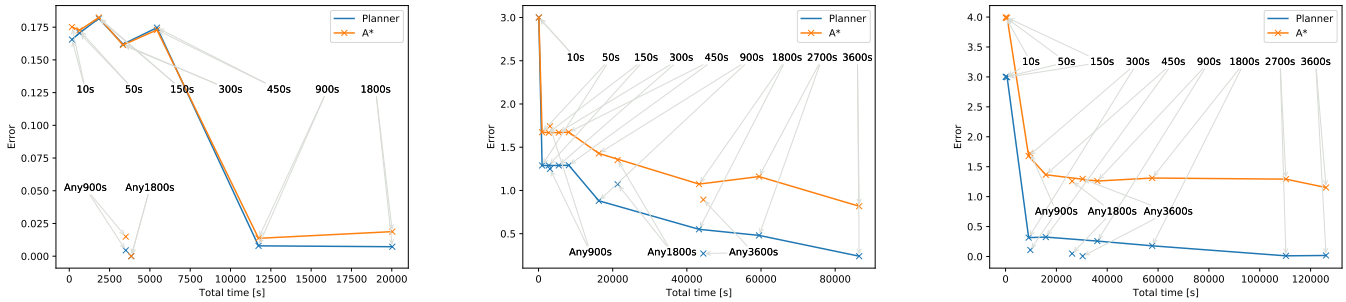

Fig. 2: Quality of final strategies of our algorithm when using planning algorithms with a fixed deadline (best plan found in the given time; depicted as lines) and maximal computation time (first better plan is used; depicted as points *Any X*) on different scenarios (1, 2, and 3; numbered from the left).

granularity	DO *both* time [s]	iter.	Error	DO *single* time[s]	iter.	Error
Scenario 1						
1.0	70569	38	1.18	8925	17	1.40
2.0	52608	49	2.12	8561	25	2.56
3.0	17550	28	3.77	4662	21	3.83
4.0	10353	35	4.24	2601	15	4.13
5.0	15954	32	4.88	6785	25	4.55
Scenario 2						
1.0	21542	53	1.43	11285	52	1.31
2.0	7381	20	2.88	1717	11	2.50
3.0	3245	10	3.00	2038	9	3.00
4.0	5457	25	5.00	4271	24	5.00
5.0	4320	28	5.63	3325	27	5.63
Scenario 3						
1.0	2557	14	0.00	1621	13	0.00
2.0	2143	13	2.75	756	10	2.50
3.0	1509	6	4.00	806	5	4.00
4.0	3851	6	4.50	1831	5	4.50
5.0	2364	2	5.00	1168	1	5.00

TABLE I: Statistics of double oracle computation for all three scenarios. Error is measured by the A^* planner.

2 and 3; see Figures 1 and 2), the difference between the error calculated by LAMA planner with 10 hours of computation time differs considerably from the error computed using the optimal A^* algorithm. Consider, for example, the variant $Any3600$ in Scenario 3, where DO terminates after 30300 seconds since neither player can improve the strategy in the restricted game. Similarly, the LAMA planner with 10-times more computation time also cannot find an improving plan (note that the error is 0 with LAMA used in the error computation). However, the actual error (computed using A^*) is 1.296. Also note, that error of the $Any3600$ variant is higher than the 3600 seconds one (the error is 1.15 in this case), however, it takes more than 126000 seconds for DO to converge.

B. Reducing problem complexity by increasing granularity

Now, we consider the $Any1800$ variant and focus on reducing the size of planning problems by decreasing their granularity, specifically by reducing the number of the timestamp objects used in problem formulation.

In the original planning problem, the number of timestamp objects is equal to the length of longest path in the problem that does not visit one vertex twice divided by the length of the shortest edge in the graph. In the reduced problem, the number of timestamps for each edge is divided by a coefficient β. Note that after the planner finds a plan, the plan is then interpolated into the original problem.

As the second aspect, we analyzed two variants of termination of DO. Results in Table I show the computation times of the two variants of DO (*both* and *single*) for all three scenarios.

The results show that, unsurprisingly, the computation time is considerably smaller for the single DO cases than for both DO one, while the error is only marginally higher for the single DO cases. Consider, for example, Scenario 1 (top in Table I), where without reducing the problem size (i.e., granularity is set to 1.0), the original DO both method solved the instance in over 70000 seconds, while our novel variant DO single took less than 9000 seconds to solve. At the same time, the error increased from 1.18 to only 1.4. That said, it is usually the case that a player whose opponent failed to generate best response does not improve its strategy that much (given the time the player spends for generating its best responses).

Granularity-wise, the results show that with increasing the coefficient β, i.e., with decreasing granularity, the error is higher. Such an observation is indeed expectable. Time-wise, the results show that the largest CPU time is spent in Scenarios 1 and 2 for cases where $\beta = 1$. However, counter-intuitively, the smallest spent CPU time varies from $\beta = 2$ to $\beta = 4$ per scenario and variant of DO.

To explain the above observation, we have to stress that although the size of the problem itself should have to have positive impact on planner's performance, low granularity, on the other hand, makes the problem representation very inaccurate and thus hard to optimize (as the optimization metric is also skewed by low granularity). Hence considering very low granularity (large β) might not save time for generating strategies that are, expectably, of low quality (have a large error).

VI. Conclusions

We demonstrate how domain-independent classical planning algorithms can be used for finding robust randomized plans, thus allowing such planners to be used in real-world environment that often involves intelligent adversaries that actively hinder the pursuit towards the goals. We formally specify a planning problem against a strategy of the opponent and embed domain-independent planning algorithms in a game-theoretic double oracle algorithm. We use LAMA planner for experimental evaluation and despite the intuition, giving more time to the planner does not necessarily lead to more robust plans. The complexity of the planning problem is increasing over iterations of the double oracle algorithm and thus it can be the case that at some point one of the agents is unable to appropriately react to the opponent's plans, even when using a planner with more computation time. Moreover, we have introduced a modification to the termination condition of the double oracle algorithm that significantly reduces computation time while only slightly decreasing quality of computed randomized plans in practice.

There are several directions for future work. In the presented work, only costs are affected by the plans of the opponent. Similarly, we assume that once an agent chooses a plan, it is executed in full. Both of these assumptions can be relaxed in order to extend the possibilities of application domains.

Acknowledgements

This research was funded by AFOSR award FA9550-18-1-0097 and by the Czech Science Foundation (project no. 17-17125Y).

References

[1] M. Vallati, L. Chrpa, and T. L. McCluskey, "What you always wanted to know about the deterministic part of the international planning competition (IPC) 2014 (but were too afraid to ask)," *Knowledge Eng. Review*, vol. 33, p. e3, 2018.

[2] M. Bowling, R. Jensen, and M. Veloso, "A formalization of equilibria for multiagent planning," in *IJCAI*, 2003, pp. 1460–1462.

[3] R. I. Brafman, C. Domshlak, Y. Engel, and M. Tennenholtz, "Planning games," in *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[4] A. Jonsson and M. Rovatsos, "Scaling up multiagent planning: A best-response approach," in *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS*, 2011.

[5] J. Jordán and E. Onaindia, "Game-theoretic approach for non-cooperative planning," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015, pp. 1357–1363.

[6] J. Jordán, A. Torreño, M. de Weerdt, and E. Onaindia, "A better-response strategy for self-interested planning agents," *Appl. Intell.*, vol. 48, no. 4, pp. 1020–1040, 2018.

[7] P. Speicher, M. Steinmetz, M. Backes, J. Hoffmann, and R. Künnemann, "Stackelberg planning: Towards effective leader-follower state space search," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

[9] F. Fang, P. Stone, and M. Tambe, "When Security Games Go Green: Designing Defender Strategies to Prevent Poaching and Illegal Fishing," in *In Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[10] H. B. McMahan, G. J. Gordon, and A. Blum, "Planning in the Presence of Cost Functions Controlled by an Adversary," in *ICML*, 2003, pp. 536–543.

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

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

[13] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Pérolat, D. Silver, and T. Graepel, "A unified game-theoretic approach to multiagent reinforcement learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4190–4203.

[14] M. Jain, V. Conitzer, and M. Tambe, "Security Scheduling for Real-world Networks," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2013, pp. 215–222.

[15] F. A. Oliehoek, R. Savani, J. Gallego-Posada, E. van der Pol, E. D. de Jong, and R. Gross, "GANGs: Generative Adversarial Network Games," *ArXiv e-prints*, 2017.

[16] Y. Wang, Z. R. Shi, L. Yu, Y. Wu, R. Singh, L. Joppa, and F. Fang, "Deep reinforcement learning for green security games with real-time information," in *AAAI Conference on Artificial Intelligence*, 2019.

[17] Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, Feb. 2009.

[18] S. Richter and M. Westphal, "The lama planner: Guiding cost-based anytime planning with landmarks," *Journal of Artificial Intelligence Research*, vol. 39, pp. 127–177, 2010.