The Thirteenth International Symposium
on Combinatorial Search (SoCS 2020)

Planning Against Adversary in Zero-Sum Games: Heuristics for Selecting and Ordering Critical Actions

Lukáš Chrpa, Pavel Rytíř, Rostislav Horčík
Faculty of Electrical Engineering
Czech Technical University in Prague

Abstract

Effective and efficient reasoning in adversarial environments is important for many real-world applications ranging from cybersecurity to military operations. Deliberative reasoning techniques, such as Automated Planning, often restrict to static environments where only an agent can make changes by its actions. On the other hand, such techniques are effective and can generate non-trivial solutions. To explicitly reason in environments with an active adversary such as zero-sum games, the game-theoretic framework such as the Double Oracle algorithm can be leveraged.

In this paper, we leverage the notions of critical and adversary actions, where critical actions should be applied before the adversary ones. We propose heuristics that provide a guidance for planners about what (critical) actions and in which order have to be applied in a good plan. We empirically evaluate our approach in terms of quality of generated strategies (by leveraging Double Oracle) and CPU time required to generate such strategies.

Introduction

Automated Planning is an important tool for enabling deliberative reasoning of intelligent agents. However, many application domains consist of multiple actors – agents, independent on each other, that act in order to achieve their goals – that can either willingly or unknowingly interfere with each other. Hence, the planning approach has to be modified in order to handle multiple agents (Bowling, Jensen, and Veloso 2003; Brafman et al. 2009). To be more specific, in scenarios in which agents have conflicting goals such as in zero-sum games each agent has to consider a possible strategy of its opponent while generating its own strategy. Such scenarios include, for instance, competing for limited resources in games or competing for customers in on-demand transport services.

Actions of other agents can be represented as exogenous events (Dean and Wellman 1990). There is a range of techniques that tackle plan generation and execution under presence of exogenous events. For example, there are techniques based on Markov Decision Process (MDP) (Kolobov, Mausam, and Weld 2012), Monte-Carlo Tree Search (Patra et al. 2019), or reasoning about "dangerous states" (Chrpa, Gemrot, and Pilát 2017). The above techniques, however, do not consider scenarios where agents have conflicting goals and hence hinder each other their pursuit towards goals.

In order to explicitly reason about adversaries while generating plans, *game-theoretic* methods have to be leveraged. Importantly, agents may need to randomise over several plans, so the other agents have uncertainty about which plan is going to be executed making it difficult for them to exploit such a strategy (LaValle 2006). Existing techniques involving planning and game theory focus on *congestion games* where the task is to find an optimal robust multi-agent plan for non-cooperating agents (Jordán et al. 2018) or on *Stackelberg games* where the task is to find a pure plan of the leader that is robust against the adversary (Speicher et al. 2018). Such techniques are not able to find randomised strategies (or plans). While there are several successful applications of game-theoretic algorithms in practice, for example in domains of physical security (Tambe 2011) or protecting wildlife (Fang, Stone, and Tambe 2015), most of the methods used for scaling-up are domain-dependent and their transferability to other domains is limited.

One of the best known game-theoretic algorithms is the incremental strategy generation method called the *Double Oracle* (DO) algorithm (McMahan, Gordon, and Blum 2003). DO algorithm tackles one common problem of games – the exponential number of possibilities to choose from. The number of plans needed to achieve certain goals is usually exponential with respect to the number of agent's actions (while omitting plans with loops, i.e., during the plan execution none of the states is visited more than once). DO therefore restricts 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 *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 possibilities (see e.g (Bošanský et al. 2014; Lanctot et al. 2017)).

In terms of using domain-independent planning algorithms for computing the (best) response plans, finding an optimal plan that accounts for best response is often computationally harder than finding any plan (Helmert 2003).

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

In particular, many planners produce sub-optimal plans and improve them until the allocated time expires or the plans cannot be further improved (Vallati, Chrpa, and McCluskey 2018). Rytíř, Chrpa, and Bošanský (2019) studied how effective and efficient is to combine DO with domain-independent planning algorithms while considering varying time limits for providing the response plans and varying granularity of the underlying planning tasks. The results indicate that using domain-independent planning albeit sub-optimal struggles with scalability.

In this paper, we, in analogy to Rytíř, Chrpa, and Bošanský (2019), provide a framework for formulating planning tasks in zero-sum games. To tackle the scalability issue we develop (i) an algorithm that computes lower bound of the time in which each action, defined in the planning task, can be applied, (ii) a greedy heuristic algorithm for selecting and ordering critical actions in response plans, and (iii) a local search algorithms for improving the selection and ordering of critical actions. Whereas (i) prunes alternatives that are useless to pursuit with respect to a strategy of the adversary and thus maintaining optimality (if an optimal planner is used for response plan generation), (ii) and (iii) provide a structure of response plans and despite compromising optimality it makes it more efficient to generate response plans. We evaluate our approach on a resource collection problem with two competing 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 (as the resource taken by the opponent will no longer be available to collect) (Rytíř, Chrpa, and Bošanský 2019). The aim of the evaluation is to demonstrate how the proposed algorithms improve the "classical" approach involving using domain-independent planning within the DO algorithm (Rytíř, Chrpa, and Bošanský 2019).

Related Work

The idea of combining planning and game theory has appeared in previous works, although mostly with different goals. Often, the goal was to update the planning formalism in order to handle multiple agents and multiple goals the agents can pursue (Bowling, Jensen, and Veloso 2003; Brafman et al. 2009). A body of work concerning non-cooperative multi-agent planning exploits game theory for generating plans for each agent while minimizing conflicts with plans of other agents. Resolving such conflicts can be done by translating the task into an invertible planning problem (Galuszka and Swierniak 2010), or by selecting the best plan for each agent from a set of pre-computed plans using a two-game approach (Jordán and Onaindia 2015). Closer to our work, the conflicts can also be resolved by a *best-response* approach that iteratively improves plans of each agent (Jonsson and Rovatsos 2011). Such an approach has been used for planning Electric Autonomous Vehicles (Jordán et al. 2018). These works, however, focus on *congestion games*, for which a single plan can be optimally robust (a pure equilibrium is guaranteed to exist for this class of games). This, however, is not true for most of the non-cooperative games and adversarial scenarios. Speicher et al.. (2018) used game theoretic framework of Stackelberg games and seek a pure plan of the leader that is robust against actions of the adversary. Again, we seek a possibly randomized strategy which poses computation challenges that are not present when restricting to pure strategies.

There are several existing methods that use the double-oracle incremental strategy generation method. The original paper by McMahan, Gordon, and Blum (2003) was used in the setting where one player sought an optimal way to get through an area unobserved while the other player placed the surveillance cameras. In that work and many other follow-up works (e.g., see (Jain, Conitzer, and Tambe 2013; Bošanský et al. 2014)), the standard assumption is that the best response algorithm is capable of computing the optimal plan (or at least a best response with a bounded error) given the strategy of the opponent. On the other hand, the recent work combined reinforcement learning with double oracle algorithm (Lanctot et al. 2017; Wang et al. 2019) on domains where computing (approximate) best response is not possible.

Recently, an effort to combine domain-independent planning and DO has been made (Rytíř, Chrpa, and Bošanský 2019). In particular, the (best) response in DO is encoded as a planning task in which "critical actions" are associated with cost such that the cost reflects probability of applying these actions before those of the adversary. Whereas the results are promising, such as approach is prone to low scalability. In contrast, to this work, we propose heuristics in order to tackle the scalability issue.

Technical Background

This section introduces the terminology we use in this paper.

Automated Planning

We assume a restricted form of Automated Planning, with durative actions while having a static, deterministic and fully observable environment. Solution plans amount to partially ordered sequences of actions with known time of application. We consider durative actions defined as in PDDL 2.1 (Fox and Long 2003).

Let V be a set of ***variables*** where each variable $v \in V$ is associated with its domain $D(v)$. An ***assignment*** of a variable $v \in V$ is a pair (v, val), where its value $val \in D(v)$. Hereinafter, an assignment of a variable is also denoted as a ***fact***. A (partial) ***variable assignment*** p over V is a set of assignments of individual variables from V, where $vars(p)$ is a set of all variables in p and $p[v]$ represents a value of v in p. To accommodate the notion of time, we denote that an assignment f or a (partial) variable assignment p holds in time t as $f(t)$ or $p(t)$ respectively.

An ***action*** is a tuple $a = (dur(a), pre^{\vdash}(a), pre^{\vdash\dashv}(a), pre^{\dashv}(a), eff^{\vdash}(a), eff^{\dashv}(a))$, where $dur(a)$ represents duration of a's application and the other elements are sets of partial variable assignments. In particular, $pre^{\vdash}$ represents action precondition before its application, $pre^{\dashv}$ represents action precondition before finishing its application, $pre^{\vdash\dashv}$ represents action precondition for the whole time interval of its application, $eff^{\vdash}(a)$ represents action effects taking place after starting its application

and $\mathit{eff}^{\dashv}(a)$ represents action effects taking place after finishing its application. We say that an action a is ***applicable*** in time t if and only if $\mathit{pre}^{\vdash}(a)(t)$, $\mathit{pre}^{\dashv}(a)(t + \mathit{dur}(a))$ and $\forall t' \in [t, t + \mathit{dur}(a)] : \mathit{pre}^{\vdash\dashv}(a)(t')$. The ***result*** of applying a in time t (if possible) is that $\mathit{eff}^{\vdash}(a)$ becomes true in t and $\mathit{eff}^{\dashv}(a)$ becomes true in $t + \mathit{dur}(a)$. It should be noted that an assignment of a variable can change in time t only when an action effect modifying the variable takes place in time t. Note that we denote $\mathit{pre}(a) = \mathit{pre}^{\vdash}(a) \cup \mathit{pre}^{\vdash\dashv}(a) \cup \mathit{pre}^{\dashv}(a)$ and $\mathit{eff}(a) = \mathit{eff}^{\vdash}(a) \cup \mathit{eff}^{\dashv}(a)$ unless otherwise stated.

A ***planning task*** is a quadruple $\mathcal{P} = (V, A, I, G)$, where V is a set of variables, A a set of actions, I a complete variable assignment representing the ***initial state*** and G a partial variable assignment representing the ***goal***.

A ***plan*** $\pi = \{(a_1, t_1), \ldots, (a_n, t_n)\}$ (for a planning task $\mathcal{P}$) is a set of couples (action,time) such that $I(0)$ (i.e., the initial variable assignment is true in time 0), for each $1 \leq i \leq n$ it is the case that $a_i \in A$ is applicable in t_i, no actions are conflicting (i.e., no action deletes a precondition of another action at the same time or no two or more actions modify the same variable at the same time), and $G(\max_{i=1}^{n}(t_i + \mathit{dur}(a_i)))$ holds (i.e., a goal is achieved after all actions are applied).

Typically, plans are optimised for makespan, i.e., duration of their execution. For our purpose, it is more important to apply some actions within given dead-lines and hence we define a cost function that assigns each action and timestamp a non-negative cost, i.e., $\mathit{cost} : A \times T \to \mathbb{R}_0^+$. We say that a plan $\pi = \{(a_1, t_1), \ldots, (a_n, t_n)\}$ (for $\mathcal{P}$) is ***cost-optimal*** if for every plan $\pi' = \{(a'_1, t'_1), \ldots, (a'_m, t'_m)\}$ (for $\mathcal{P}$) it is the case that $\sum_{i=1}^{n} \mathit{cost}(a_i, t_i) \leq \sum_{j=1}^{m} \mathit{cost}(a'_j, t'_j)$.

Another variant of planning task definition considers, rather than a single (hard) goal, a set of ***soft goals*** (each goal is a set of variable assignments) such that failing to achieve a goal is penalised. Formally, for a planning task $\mathcal{P} = (V, A, I, G)$, $G = \{G_1, \ldots, G_n\}$, where each G_i is associated with a cost M_i ($1 \leq i \leq n$) such that for a plan π it is the case that $\mathit{cost}(\pi) = \sum_{i \in \{i \mid G_i \text{ not achieved}\}} M_i$.

Normal-Form Games

The baseline representation for modelling strategic interaction is *normal-form games* (NFGs), see e.g. (Shoham and Leyton-Brown 2009). A normal-form game Γ is a tuple (N, S, u), where N is the finite set of players, $S = S_1 \times \cdots \times S_N$ for finite sets of pure strategies $S_1, \ldots, S_N$ of players $1, \ldots, N$ and $u = (u_1, \ldots, u_N)$ is a N-tuple of utility functions that assign a real-valued utility of player i 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 for a player i is a probability distribution σ_i over the set of player's pure strategies S_i. An N-tuple of mixed strategies $\sigma = (\sigma_1, \ldots, \sigma_N)$ is called mixed-strategy profile. We extend the definition of utility functions so that a given mixed-strategy profile σ the value $u_i(\sigma)$ is the expected utility 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 a mixed strategy of one player σ_i is the best response to the strategy of the opponent σ_{-i} (denoted as $\sigma_i = br(\sigma_{-i})$) when $u_i(\sigma_i, \sigma_{-i}) \geq u_i(\sigma'_i, \sigma_{-i})$ for all mixed strategies σ'_i over S_i. We say that a mixed-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 (Shoham and Leyton-Brown 2009)). The expected utility of player 1 in a 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 sets of possible pure strategies available to players are 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 linear program. Next, each player computes a best pure response from all its strategies 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).

Planning in Zero-sum Games

In zero-sum games, any gain or loss of the agent is a loss or gain for the competitor, respectively. Whereas the agent's plans maximise reward, or minimise cost, in consequence, reward of the competitor's plans is minimised, or their cost is maximised. In plain words, it is of adversary's best interest to hinder as many agent's goals as possible.

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.

Rytíř, Chrpa, and Bošanský (2019) introduced the notions of *critical* and *adversary* actions. In a nutshell, critical actions require a fact that adversary actions can delete. We denote such a fact as a *critical* fact.

Definition 1. *Let $\mathcal{P} = (V, A, I, G)$ be an agent's planning task and A' be a set of competitor's actions. We say that (v, val), where $v \in V$ and $val \in D(v)$, is a* **critical fact** *if and only if $(v, val) \in I$, $\exists a \in A : (v, val) \in \mathit{pre}(a)$, $\forall a \in A : (v, val) \notin \mathit{eff}(a)$, and $\exists a' \in A' : (v, val') \in \mathit{eff}(a') \wedge val \neq val'$.*

Definition 2. *Let $\mathcal{P} = (V, A, I, G)$ be an agent's planning task, A' be a set of competitor's actions and (v, val) be a critical fact. We say that $A_c \subseteq A$, where for each $a_c \in A_c$ it is the case that $(v, val) \in \mathit{pre}(a_c)$, is a set of* **critical actions** *over (v, val). We also define a set of* **adversary actions** *$A'_a \subseteq A'$ over (v, val), where for each $a'_a \in A'_a$ it is the case that $(v, val') \in \mathit{eff}(a'_a)$ and $val \neq val'$.*

Knowing competitor's plan gives us an information about when adversary actions are applied. Therefore, in order to successfully apply critical actions the agent has to plan them before those of the competitor. In other words, adversary actions set deadlines for agent's critical actions. For example, the agent has to collect resources before the competitor, so competitor's collect actions set deadlines for agent's collect actions.

Definition 3. *Let A_c, A'_a and (v, val) be as in Definition 2. Let $\pi' = \{(a'_1, t'_1), \ldots, (a'_m, t'_m)\}$ be a plan of the competitor. We define a function $at(f, a, x)$, where f is a fact, a is an action and $x \in \{pre, eff\}$, such that $at(f, a, x) = 0$ iff $f \in x^{\vdash}(a) \cup x^{\vdash\!\!\vdash}(a)$, or $at(f, a, x) = dur(a)$ iff $f \in x^{\dashv}(a)$ and $f \notin x^{\vdash}(a) \cup x^{\vdash\!\!\vdash}(a)$. Then, for each $a_c \in A_c$ and $(v, val) \in pre(a_c)$, we can determine a* **deadline** *with respect to π', denoted as $dl(a_c, \pi')$ as $\min\{t - at((v, val), a_c, pre) + at((a, val'), a', eff) \mid (a', t) \in \pi', a' \in A'_a, (v, val') \in eff(a'), val \neq val'\}$.*

For solving zero-sum games, we can leverage the well known Double Oracle algorithm that iteratively improves strategies of the competing agents. In our case, strategies are in form of sets of plans where each of the plans can be applied with a given probability, i.e., $\{(\pi_1, p_1), \ldots, (\pi_n, p_n)\}$ such that $\sum_{i=1}^{n} p_i = 1$. Improving agent's strategy concerns generating (best) response to the competitor's current strategy, that is, generating a plan that increases the utility of agent's strategy.

We consider a set of (heterogeneous) ***units***[1] such that at least one unit has to be associated with each action. In plain words, it means that each action has to be executed by one or multiple units. Also, no unit can execute more than one action at the same time.

In analogy to Rytíř, Chrpa, and Bošanský (2019), we associate each (soft) goal with a critical atom and we assume that each critical action has exactly one critical fact in its precondition. Hence, for a set of critical actions over the same critical fact, it is the case that one critical action from the set has to be applied in order to achieve the goal. We divide critical actions into ***clusters*** such that for each critical action it is the case that it belongs to exactly one cluster and where each cluster is associated with a distinct critical fact.

Competitor's strategy provides multiple deadlines for agent's critical actions which occur with a given probability. In consequence while considering the previous assumption, the probability of reaching a given soft goal equals probability of successful application of a corresponding critical action. We can formulate a planning task such that critical actions are associated with costs reflecting their probability to be applied before competitor's adversary actions. Hence the agent's plans are optimised for maximising their expected utility, in other words, the likeliness of reaching the goals, with respect to the given competitor's strategy.

Definition 4. *Let $\Pi' = \{(\pi'_1, p'_1), \ldots, (\pi'_n, p'_n)\}$ be competitor's strategy, $\mathcal{P} = (V, A, I, G)$ be agent's planning task such that $G = \{G_1, \ldots, G_k\}$ is a set of soft goals associated with costs for failure of their achievement $M_1, \ldots, M_k$, and $A_c \subseteq A$ be agent's critical actions distributed in clusters $C_1, \ldots, C_k$ corresponding to the goals. We define a* **cost function** *$c_{\mathcal{P},\Pi'} : A_c \times T \to \mathcal{R}_0^+$ that assigns non-negative cost for critical actions in given timestamps (from T). For each $1 \leq j \leq k$ $a \in C_j$ and $t \in T$, $c_{\mathcal{P},\Pi'}(a, t) = \sum\{p'_i M_j \mid dl(a, \pi'_i) < t\} + 0.5 \sum\{p'_i M_j \mid dl(a, \pi'_i) = t\}$.*

For the agent's planning task $\mathcal{P}$ and the competitor's strategy Π', we define an agent's **response planning task** *$\mathcal{P}_{\Pi'}$ such that critical actions A_c are associated with the $c_{\mathcal{P},\Pi'}$ cost function. Other (non-critical) actions are of zero cost.*

[1]Note that in our terminology an agent can control multiple units

Algorithm 1 Estimating the earliest application time of all actions

Require: Planning Task $\mathcal{P} = (V, A, I, G)$
Ensure: The earliest application time for all actions

function EARLIESTACTIONTIME($\mathcal{P}$)
  $F \leftarrow I$
  $\forall f \in I : time(f) \leftarrow 0$
  $\forall f \notin I : time(f) \leftarrow \infty$
  $O \leftarrow \emptyset$
  return EarliestTime($F, O, A, time$)
end function

function EARLIESTTIME($F, O, A, time$)
  repeat
    $A' \leftarrow \{a \mid a \in A, a \notin O, pre^{\vdash}(a) \cup pre^{\vdash\!\!\vdash}(a) \subseteq F\}$
    $\forall a \in A' : time(a) \leftarrow \max\{time(f) \mid f \in F, f \in pre^{\vdash}(a) \cup pre^{\vdash\!\!\vdash}(a)\}$
    $A'' \leftarrow \{a \mid a \in A', \forall a' \in A' : time(a) \leq time(a')\}$
    $O \leftarrow O \cup A''$
    for all $f \in \bigcup_{a \in A''} eff(a)$ **do**
      $time(f) \leftarrow \min(time(f), \min\{time(a) + at(f, a, eff) \mid a \in A'', f \in eff(a)\})$
      $F \leftarrow F \cup \{f\}$
    end for
  until $A'' = \emptyset$
  $\forall a \notin O : time(a) \leftarrow \infty$
  return $time$
end function

Note that we consider "ties", i.e., situations where critical actions are planned exactly on their deadlines, such that the agent has a half chance to succeed (and similarly the competitor has a half chance of succeeding). Also, it can be observed that an optimal response plan accounts for the best agent's response on the competitor's strategy.

Estimating the Earliest Action Application Time

Delete relaxation has a tradition in classical planning (Hoffmann and Nebel 2001) as well as in temporal planning (Coles et al. 2008). Inspired by the h^{max} heuristics (Bonet and Geffner 2001), we propose an algorithm that estimates lower bound of time in which each action defined in the planning task can be applied. For this purpose, we

define a function $time : A \cup F \rightarrow T$ assigning an action (from A) or a fact (from F) a timestamp (from T) of its earliest application or occurrence. The proposed algorithm is summarised in Algorithm 1. Initially, facts (variable assignments) present in the initial state are set to be true at time 0 and the other facts at time ∞. We keep track of facts (variable assignments) and actions occurring at some point in the process in sets F and O respectively. The algorithm iterates (Lines 9–18) until the set of actions O remains unchanged (after the iteration). Actions that have not yet been considered (are not in O), whose precondition is present in F, are selected (the set A'). For each action from A' its earliest application time is determined as the latest time in which its "at start" or "over all' precondition is met (Line 11). Then, from A' we select actions with minimum application time (the set A'') and add them into O. For each effect f of these actions (from A'') we update its earliest occurrence time as minimum of the current time of f and minimum time in which f is achieved by any action from A'' having f in its effects (Line 15). The fact f is then added to F (if it is not yet there). Lastly, for actions not present in O we set their earliest application time to ∞ as they are unreachable from the initial state (their precondition is never met).

Proposition 5. *Let $\mathcal{P} = (V, A, I, G)$ be a planning task. Let $time(a)$ be determined for each $a \in A$ by running Algorithm 1. Then, for each plan of $\mathcal{P}$ in form $\{(a_1, t_1), \ldots, (a_n, t_n)\}$ it is the case that for each couple (a_i, t_i) $time(a_i) \leq t_i$ holds.*

Proof (Sketch). *Algorithm 1 iteratively selects actions not applied before that can be applied in the smallest time while considering the time of first occurrence of each fact. Time of the first occurrence of facts are determined as minimum of the times of the selected action effects and the current time of their first occurrence. Hence, as action effects cannot take place before application of the action, if the first occurrence of a fact is modified, then it cannot be earlier than the action application time. Hence it is the case that for each action $time(a)$ determines a lower bound of its application time in every valid plan.*

Note that Algorithm 1 does not consider "at end" preconditions in the main loop (the EarliestTime function). The reason is that another action (or a sequence of actions) applied after an action a started its application can influence the time in which the "at end" precondition of a (i.e., $pre^{\dashv}(a)$) is satisfied. Consequently, Algorithm 1 would have to be more complicated (even in terms of computational complexity) to reflect the issue and guarantee the lower bound of action application time. Omitting "at end" preconditions does not violate lower bounds of action application time (although it might underestimate the bound more). Actions whose "at end" preconditions are unreachable could be omitted. For the sake of clarity, we do not consider this option in Algorithm 1.

The consequence of Proposition 5 is that critical actions whose earliest time of application (determined by Algorithm 1) is greater than their latest deadline (with respect to competitor's strategy) do not have to be considered for generating robust plans.

Algorithm 2 Estimating application time of all actions while considering fixed application time of a subset of actions

Require: Planning Task $\mathcal{P} = (V, A, I, G)$, actions $A' \subseteq A$ with their application timestamps ($time$)
Ensure: Estimated application time for all actions

1: **function** EstActionTimeWActions($\mathcal{P}, A', time$)
2: $F \leftarrow \emptyset$
3: **for all** $v \in \bigcup_{a' \in A'} vars(pre(a') \cup eff(a'))$ **do**
4: $val \leftarrow \arg\max_x \{time(a') + at((v,x), a', pre) \mid a' \in A', (v,x) \in pre(a')\} \cup \{time(a') + at((v,x), a', eff) \mid a' \in A', (v,x) \in eff(a'))\}$
5: $time((v, val)) \leftarrow \max\{time(a') + at((v,x), a', pre) \mid a' \in A', (v,x) \in pre(a')\} \cup \{time(a') + at((v,x), a', eff) \mid a' \in A', (v,x) \in eff(a'))\}$
6: $F \leftarrow F \cup \{(v, val)\}$
7: **end for**
8: **for all** $f \in I$ s.t. $vars(f) \cap vars(F) = \emptyset$ **do**
9: $time(f) \leftarrow 0$
10: $F \leftarrow F \cup \{f\}$
11: **end for**
12: $\forall f \notin F : time(f) \leftarrow \infty$
13: $O \leftarrow A'$
14: **return** EarliestTime($F, O, A, time$)
15: **end function**

Corollary 6. *Let $\mathcal{P} = (V, A, I, G)$ be an agent's response planning task, $A_c \subseteq A$ be agent's critical actions and $\Pi' = \{(\pi'_1, p'_1), \ldots, (\pi'_n, p'_n)\}$ be a strategy of the competitor. Let $A' = A \setminus \{a \mid a \in A_c, time(a) > \max_{1 \leq i \leq n} dl(a, \pi'_i)\}$ be a set of actions and $\mathcal{P}' = (V, A', I, G)$ be a response planning task. Then, π is an optimal plan for $\mathcal{P}$ with respect to Π' if and only if π is an optimal plan for $\mathcal{P}'$ with respect to Π'.*

Proof (Sketch). *It immediately implies from Proposition 5 as critical actions planned after the latest possible deadline do not have any chance to be successfully applied.*

Heuristic Estimation of Critical Actions Ordering

To improve scalability of the approach, we propose a heuristics that estimates promising selection and ordering of critical actions. The structure of response planning tasks concerns dividing critical actions into clusters and assuming that no unit can apply more than one action at the same time. Hence we focus on selecting the most promising critical action from each cluster and order these actions according to the units associated with them.

For this purpose, we adapt Algorithm 1 to consider actions that are known to be applied in a certain time as summarised in Algorithm 2. Hence instead of starting in the initial state with initial variable assignments, the considered actions determine the latest time and value of the variable assignment. In particular, variables that appear in preconditions or effects of the considered actions are processed such that their latest values with corresponding timestamps are

Algorithm 3 Selecting and ordering of critical actions

Require: Planning Task $\mathcal{P} = (V, A, I, G)$, a set of critical actions $A_c \in A$ with the cost function $c_{\mathcal{P},\Pi'}$ divided into clusters $C = C_1, \ldots, C_n$
Ensure: Selecting and ordering of critical actions

```
1: time ←EarliestActionTime(P)
2: C ←EliminateMaxCostClusters(C)
3: sa ← ∅
4: while C ≠ ∅ do
5:     c' ← ∞
6:     for all a ∈ ⋃_{i=1}^{n} C_i do
7:         time' ←EstActionTimeWActions(P,sa ∪ {a}, time)
8:         c ←MinCost(C, time')
9:         if c < c' then
10:            c' ← c
11:            a' ← a
12:        end if
13:    end for
14:    sa ← sa ∪ {a'}
15:    C ←RemoveCluster(C, a')
16:    time ←EstActionTimeWActions(P,sa,time)
17:    C ←EliminateMaxCostClusters(C)
18: end while
19: DetermineOrdering(sa)
20: return sa
```

set as initial (Lines 3–7). Variables that do not appear in the considered actions are set to their initial values with the timestamp 0 (Lines 8–10). Timestamps of other (not yet considered) variable assignments are set to infinity (Line 12). Then, the *EarliestTime* function from Algorithm 1 is called to estimate application time of remaining actions.

Algorithm 3 greedily selects the most promising critical actions and (partially) orders them with respect to given units. For the purpose of selecting the most promising critical action in each step, we need to estimate the cost of the response plan from the action application time estimation (given by Algorithm 2):

$$\text{MinCost}(C, time) = \sum_{C_i \in C} \min\{c_{\mathcal{P},\Pi'}(a, (time(a)) \mid a \in C_i\}$$

Clusters whose cost is determined as maximal, i.e., for a cluster C_i it is the case that $\min\{c_{\mathcal{P},\Pi'}(a, time(a)) \mid a \in C_i\}$ equals to M_i (the cost for failing to achieve a goal G_i), do not have to be considered as no critical action from that cluster can be applied before any adversary action from competitor's set of plans. The *EliminateMaxCostClusters* function eliminates such clusters.

Initially, Algorithm 3 estimates the earliest action application time by Algorithm 1 and eliminates clusters that yield maximum cost. Then, until all the clusters are considered (or eliminated) the algorithm iteratively selects a critical action (from any remaining cluster) that yields minimum cost (Lines 6–13). The minimum cost is determined by estimating action application time by Algorithm 2 while considering the already selected critical actions and the "to be selected" critical action and applying the *MinCost* function (defined as above). The selected critical action is added into the set of already selected critical actions (Line 14) and the cluster to which the selected critical action belongs is removed (since it is unnecessary to apply more than one critical action from the same cluster) (Line 15). Then, similarly to the initial situation, action application time is estimated (by Algorithm 2) and the maximum cost clusters are eliminated (Lines 16 and 17). After the loop terminates, the selected critical actions are partially ordered such that ordering constraints are provided between critical actions associated with the same unit and the order reflects the action timestamps (the $time$ function).

Algorithm 3 hence provides a selection of critical actions and their partial ordering. A planner therefore has to consider only selected critical actions while pruning the other critical actions. On top of that, the (selected) critical actions become applicable after the predecessing critical actions are applied. Roughly speaking, the planner has to only "fill the gaps" between partially ordered critical actions in order to generate response plans. It should be noted that as Algorithm 3 is greedy, hence response plans might not be optimal even if an optimal planner is used. On the other hand, such an approach might increase scalability as response plans might be generated more easily.

Algorithm 4 Simulated annealing algorithm

Require: Ordering of critical actions for all units O, initial temperature t, constant K and temperature decrease ϵ.
Ensure: Optimised ordering of critical actions O.

```
1: while t > 0 do
2:     O' ← SampleNextOrdering(O)
3:     Δ ← OrderingCost(O') − OrderingCost(O)
4:     if Δ < 0 then
5:         O ← O'
6:     else
       With probability exp(−Δ/(t ∗ K)) do
7:         O ← O'
8:     end if
9:     t ← (t − ε)
10: end while
11: return O
```

Improving Critical Actions Ordering by Local Search

We try to improve the initial ordering found by the greedy algorithm (Algorithm 3) by using simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983) as the local search algorithm (depicted in Algorithm 4).

The SampleNextOrdering function takes an ordering O and with probability 0.5 returns ordering O' where two randomly chosen critical actions of the same unit are swapped, and with probability 0.5 returns ordering O' where a randomly chosen critical action is replaced by another critical action from the same cluster.

The OrderingCost function calculates the total cost of or-

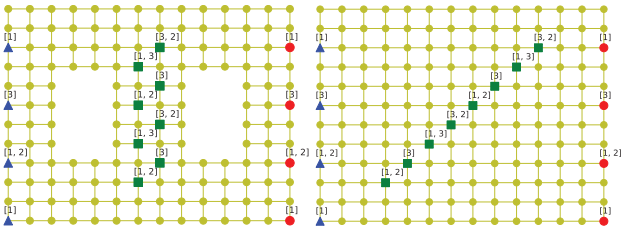

Figure 1: A game with 4 UAVs and 6 resources. UAVs of Player 1 are depicted by blue triangles. UAVs of Player 2 are depicted by red circles. Resources are depicted by green squares. The numbers in the brackets are sensors that a UAV has or are required for a resource to be collected.

dering O, given the cost function $c_{\mathcal{P},\Pi'}$. It is done by adapting Algorithm 3 such that the for loop is removed and the action a' (to be added into sa) is determined from the given ordering O. After all actions from O are considered then the *MinCost* function is applied to estimate the cost of O.

Incorporating Planning into the Double Oracle Algorithm

As stated before, Double Oracle considers a restricted game with a set of pure strategies for each player and where each player iteratively generates (best) response to the opponent strategy until neither player can improve its strategy. In our case, pure strategies of agents are set of plans such that each plan can be applied with a given probability. For finding the (best) response to competitor's strategy, an agent formulates a response planning task as in Definition 4. If the response planning task is solved optimally, i.e., the response plan has minimum cost, then the agent obtained the best response.

However, finding an optimal plan might be too expensive. Also, if the heuristics (as in Algorithm 3 and 4) is used, then suboptimal response plans are generated. If the response plan despite being suboptimal improves agent's strategy, the response plan is considered and the Double Oracle algorithm continues. If none of the agents can improve its strategy, then the Double Oracle algorithm terminates.

Experiments

We evaluated our algorithms on a two-player zero sum game called Resource Hunting, recently specified by Rytíř, Chrpa, and Bošanský (2019). The map of the game is modelled as a graph in which the vertices represent locations and edges connect neighbouring locations. The (soft) goals for each player are to collect resources that are placed in some locations on the map. Each player controls a group of unmanned aerial vehicles (UAVs). Each UAV has at most two different sensors. Each resource requires one or two sensors for being collected. There are two types of actions the player can take: the *move* action, which moves an UAV from one location to another such that the locations are connected by an edge, and the *collect* action, where one or more UAVs collect a resource present in the same location as the UAVs and where the UAVs posses the required sensors. The examples of two types of scenarios, the "middle" and "diagonal" ones, we use for experiments are depicted in Figure 1

We modelled the domain in PDDL, which is a well known language for describing planning tasks (Fox and Long 2003). We abstracted the graph by considering only locations of interest (with an UAV, or a resource), where length of edges between these locations correspond to minimum path length in the original graph. To reason with timelines we can introduce specific "timeline" objects. 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 task. Reasoning with timestamps can be embedded into the model by introducing "arithmetic" and "relation" predicates that represent essential operations (e.g., adding, comparing). Enforcing ordering constraints for critical actions is done by introducing special facts representing the order of each critical actions. Critical actions are enhanced in a way that in their precondition a required order fact is present and in their effects the order fact is incremented.

The parameters for Algorithm 4 were set as $t = 100$, $K = 2.1$ and $\epsilon = 0.002$. The values of the parameters were determined experimentally for the given domain.

As an optimal planner, for scenarios with three UAVs, we used Fast Downward (Helmert 2006) with the potential heuristic (Pommerening et al. 2015) optimised by the diversification method proposed by Seipp, Pommerening, and Helmert (2015).

For scenarios with more than three UAVs, we used the well known LAMA planner (Richter and Westphal 2010). The time limit for generating a single response plan was set to 1800 seconds while considering *anytime* mode. That is, plans are being generated until either a response plan improving player's strategy is found, or the time limit expires. Such a setting has shown to be the most promising (Rytíř, Chrpa, and Bošanský 2019). We ran the experiments on Linux with CPU Intel Xeon E5-2620 v4 2.10 GHz with 32GB RAM.

Results

At first, we have compared the classical approach (Rytíř, Chrpa, and Bošanský 2019), the pruning heuristics (leveraging Corollary 6), the selecting and ordering heuristics (Alg. 3) and the local search approach (Alg. 4). The comparison leveraging the optimal planner has been done in scenarios with 3 UAVs and 6 resources as the classical approach as well as the pruning heuristics do not scale beyond that size. Table 1 shows the results of the comparison. The *approximation error* is equal to the difference of between the value of best response to Player 1 strategy and best response to Player 2 strategy computed by the DO algorithm analogously to Rytíř, Chrpa, and Bošanský (2019). In the diagonal case, the pruning heuristics led to the highest number of iterations, i.e., the number of generated response plans until it converges, while the ordering heuristics approach led to the highest error. Whereas the latter is expectable, the former is caused by the fact that the response plans might omit collecting some resources (all the related critical actions are pruned out) and hence not setting the deadline for the other

	"Middle" Scenario					"Diagonal" Scenario				
Algorithm	Error	P1 Value	P2 Value	time (s)	Iters	Error	P1 Value	P2 Value	time (s)	Iters
Classical	0.0	3.1	2.9	3430	34	0.0	3.11	2.89	2590	23
PruningHeur	0.0	3.1	2.9	3077	34	0.0	3.11	2.89	3178	31
orderingHeur	1.72	2.82	3.18	1043	15	1.13	3	3	2126	13
localSearchHeur	1.12	3.21	2.79	2834	24	0.6	3.2	2.8	2929	17

Table 1: Comparison of the approaches for Scenarios with 3 UAVs and 6 resources. P1 and P2 stand for Player 1 and Player 2 respectively.

		PruningHeur				orderingHeur				localSearchHeur			
UAVs	Res	P1 Val	P2 Val	Time(s)	Iters	P1 Val	P2 Val	Time(s)	Iters	P1 Val	P2 Val	Time(s)	Iters
3	6	3.08	2.92	1238	33	2.72	3.28	3838	21	2.96	3.04	4931	21
4	8	4.31	3.69	3810	14	3.43	4.57	10250	27	4.26	3.74	7958	34
5	10	1.85	8.15	2189	10	4.02	5.98	10248	30	4.48	5.52	16127	24
6	12	6.43	5.57	4209	9	5.5	6.5	13507	35	6	6	14546	16
8	16	10.08	5.92	10522	17	3	13	6616	6	9.5	6.5	18113	14

Table 2: Scalability results for the "Middle" Scenarios. Comparing pruning heuristics, order heuristics and local search heuristics. "Res" denotes the number of resources. P1 and P2 stand for Player 1 and Player 2 respectively.

		PruningHeur				orderingHeur				localSearchHeur			
UAVs	Res	P1 Val	P2 Val	Time(s)	Iters	P1 Val	P2 Val	Time(s)	Iters	P1 Val	P2 Val	Time(s)	Iters
3	6	3.14	2.86	1631	20	2.62	3.38	7459	21	2.96	3.04	1292	20
4	8	3.67	4.33	6235	8	5.61	2.39	3932	17	3.82	4.18	7114	24
5	10	2	8	5739	9	10	0	4562	11	5.67	4.33	12984	23
6	12	4.29	7.71	2618	9	7.8	4.2	4483	17	6.83	5.17	10122	11
8	16	8	8	9229	17	11.05	4.95	8777	16	11.5	4.5	17997	12

Table 3: Scalability results for the "Diagonal" Scenarios. Comparing pruning heuristics, order heuristics and local search heuristics. "Res" denotes the number of resources. P1 and P2 stand for Player 1 and Player 2 respectively.

player. For the other player it means that it can collect the resource at any time. However, since randomised strategies are generated, the deadlines for resources (for both players) are eventually set.

All the heuristic approaches with the LAMA planner scale, in contrast to the classical approach, up to the scenarios with 8 UAVs and 16 resources (see Tables 2 and 3). For scenarios with 4 and more UAVs, we were not able to compute optimal strategies and hence we are not able to determine the error. On the other hand, the scenarios are similar (albeit larger) to those with 3 UAVs and hence, as a rule of the thumb, we believe that the optimal strategies are those in which values of both players are close to each other. From that perspective, the local search heuristics provides reasonable results and in the most cases the strategies are of better quality (closer to an equilibrium) than for the other approaches. Another aspect that contributes to worse quality of generated strategies is the use of a suboptimal planner. This is especially the case in the "diagonal" scenario with 5 UAVs for the ordering heuristics (Table 3), where Player 2 was not able to generate plans that collected at least some resources before the opponent. The reason is that in the generated plans the UAVs do not move directly towards the resources and hence miss the deadlines.

In summary, the results show the tradeoff between the optimal approaches – classical and pruning heuristics with an optimal planner – and the sub optimal heuristic approaches with a satisficing planner. The latter sacrifices quality of the solutions for getting higher scalability. It also should be noted that generating optimal strategies is feasible only for small scenarios (up to 3 UAVs in our case).

Conclusion

Planning in zero-sum games concerns of finding plans that maximise the reward (or minimise the cost) for accomplished (or failed) goals in the presence of an opponent sharing the same goals such that only one of the actors can accomplish a given goal. Automated Planning can be incorporated into the Double Oracle algorithm such that plans are optimised for applying critical actions before adversary can invalidate their preconditions (Rytíř, Chrpa, and Bošanský 2019). In this paper, we presented three approaches, one complete that prunes alternatives that do not lead to optimal solutions, and two incomplete that suggest what critical actions and in which order they should be applied. Whereas the former approach maintains optimality of generated strategies, the latter addresses (to some extent) the scalability issue of the complete approaches despite reducing the quality of generated strategies.

In future, we plan to adapt our heuristic approaches to decompose the problem into smaller ones in order to further improve scalability of the approach.

Acknowledgements This research was funded by AFOSR award FA9550-18-1-0097, by the Czech Science Foundation (project no. 18-07252S) and by the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".

References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artif. Intell.* 129(1-2):5–33.

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

Bowling, M.; Jensen, R.; and Veloso, M. 2003. A formalization of equilibria for multiagent planning. In *IJCAI*, 1460–1462.

Brafman, R. I.; Domshlak, C.; Engel, Y.; and Tennenholtz, M. 2009. Planning games. In *Twenty-First International Joint Conference on Artificial Intelligence*.

Chrpa, L.; Gemrot, J.; and Pilát, M. 2017. Towards a safer planning and execution concept. In *29th IEEE International Conference on Tools with Artificial Intelligence, ICTAI*, 972–976.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Planning with problems requiring temporal coordination. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, 892–897.

Dean, T., and Wellman, M. 1990. *Planning and Control*. Morgan Kaufmann Publishers.

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

Fox, M., and Long, D. 2003. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.* 20:61–124.

Galuszka, A., and Swierniak, A. 2010. Planning in multiagent environment using strips representation and non-cooperative equilibrium strategy. *Journal of Intelligent and Robotic Systems* 58(3-4):239–251.

Helmert, M. 2003. Complexity results for standard benchmark domains in planning. *Artif. Intell.* 143(2):219–262.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res.* 14:253–302.

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

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

Jordán, J., and Onaindia, E. 2015. Game-theoretic approach for non-cooperative planning. In *the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 1357–1363.

Jordán, J.; Torreño, A.; de Weerdt, M.; and Onaindia, E. 2018. A better-response strategy for self-interested planning agents. *Appl. Intell.* 48(4):1020–1040.

Kirkpatrick, S.; Gelatt, C. D.; and Vecchi, M. P. 1983. Optimization by simulated annealing. *Science* 220(4598):671–680.

Kolobov, A.; Mausam; and Weld, D. S. 2012. LRTDP versus UCT for online probabilistic planning. In *AAAI*.

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

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge University Press.

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

Patra, S.; Ghallab, M.; Nau, D. S.; and Traverso, P. 2019. Acting and planning using operational models. In *The 33rd AAAI Conference on Artificial Intelligence*, 7691–7698.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From non-negative to general operator cost partitioning. In *AAAI*, 3335–3341.

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

Rytíř, P.; Chrpa, L.; and Bošanský, B. 2019. Using classical planning in adversarial problems. In *the IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, 1327–1332.

Seipp, J.; Pommerening, F.; and Helmert, M. 2015. New optimization functions for potential heuristics. In *ICAPS*, 193–201.

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

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

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

Vallati, M.; Chrpa, L.; and McCluskey, T. L. 2018. 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* 33:e3.

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