Competing for Resources: Estimating Adversary Strategy for Effective Plan Generation

Lukáš Chrpa, Pavel Rytíř, Rostislav Horčík, Stefan Edelkamp

Faculty of Electrical Engineering, Czech Technical University in Prague
{lukas.chrpa, pavel.rytir, rostislav.horcik, stefan.edelkamp}@fel.cvut.cz

Abstract

Effective decision making while competing for limited resources in adversarial environments is important for many real-world applications (e.g. two Taxi companies competing for customers). Decision-making techniques such as Automated planning have to take into account possible actions of adversary (or competing) agents. That said, the agent should know what the competitor will likely do and then generate its plan accordingly.

In this paper we propose a novel approach for estimating strategies of the adversary (or the competitor), sampling its actions that might hinder agent's goals by interfering with the agent's actions. The estimated competitor strategies are used in plan generation such that agent's actions have to be applied prior to the ones of the competitor, whose estimated times dictate the deadlines. We empirically evaluate our approach leveraging sampling of competitor's actions by comparing it to the naive approach optimising the make-span (not taking the competing agent into account at all) and to Nash Equilibrium (mixed) strategies.

Introduction

Planning in static environments accounts for generating plans that are optimised, for instance, for their length, makespan or action cost. However, in environments, where an adversarial (or competing) agent is present, such a naive approach is rarely effective.

The concept of planning in adversarial environment is not new (Applegate, Elsaesser, and Sanborn 1990). Succinct symbolic representations of state sets helped generating optimistic and strong cyclic adversarial plans (Jensen, Veloso, and Bowling 2001; Kissmann and Edelkamp 2009), a setting conceptually related to FOND planning (Cimatti et al. 2003). Such a setting, however, has to explore most if not all alternatives (in analogy to traditional game-tree methods such as minimax). Monte-Carlo Tree Search (MCTS) and Online Evolutionary Planning have been applied in adversarial environments such as the Hero Academy game (Justesen et al. 2018), or Starcraft (Justesen and Risi 2017). Deep Reinforcement Learning (DRL) has shown impressive results in Starcraft (Vinyals et al. 2019) and other (adversarial) domains such as the games of Chess or Go (Silver et al. 2018). MCTS and DRL approaches work "online": they select the most promising action (or move) in the current state of the environment and they continue to do so until the terminal state is reached.

From the planning side, Speicher et al. (2018) used the game-theoretic framework of *Stackelberg games* for generating robust plans against actions of the adversary. In a similar spirit, *Plan Interdiction Games* have been proposed to describe the problem of attackers and defenders, where the former plans to intrude a computer network, while the latter tries to prohibit attackers' actions (Letchford and Vorobeychik 2013; Vorobeychik and Pritchard 2020). A recent work about "Counterplanning" goes in a similar direction as one agent tries to invalidate landmarks required by the opposite agent (Pozanco et al. 2018). Planning-based techniques work offline, i.e., they generate plans upfront, which are then executed (as they are).

In this paper, we define a class of *Resource Competition* problems in which two agents compete for limited resources. Such problems involve, for example, competing for limited resources in strategy games, or on-demand transport companies competing for passengers requiring transporting from one place to another. We also assume that each agent has to generate its plan upfront and the plan cannot be amended after the agent starts executing it because, for instance, there is a lack of reliable communication between the units (e.g. UAVs) the agent controls. Although Resource Competition problems can be addressed by MCTS (Lelis 2020) or DRL (Silver et al. 2018), these techniques do not seem to be feasible for our assumption.

To generate mixed strategies (composed from plans) in Nash Equilibrium, we can leverage the Double Oracle algorithm (McMahan, Gordon, and Blum 2003), which can take tens of iterations until it converges (and none of the player can improve its strategy) even for smaller tasks (Rytíř, Chrpa, and Bošanský 2019). It involves cost-optimally solving n planning tasks in each iteration (n is the number of players), which is computationally expensive. This paper tackles the issue by proposing a heuristic method for estimating a mixed strategy of the other (adversarial) agent. Leveraging a heuristic for estimating earliest action application time, developed by Chrpa, Rytíř, and Horčík (2020), we propose a "sampling" method which provides potential application times of actions of the adversary that interfere with agent's actions. The sampling method, hence, provides *deadlines* for those agent's actions (later called *critical actions*) the agent has to take into consideration while it generates its plan. To evaluate potential of the "sampling" method, we hence resort to cost-optimal planning as it removes biases made by sub-optimal techniques. In particular, we compare plans generated by the sampling method with *naive* plans, which optimise for make-span while ignoring the presence of adversary, and mixed strategies in Nash Equilibrium generated by Double Oracle (Rytíř, Chrpa, and Bošanský 2019).

Preliminaries

This section introduces the terminology we use in this paper.

Automated Planning

We assume a restricted form of Temporal Planning in a static, deterministic and fully observable environment. Solution plans are sets of pairs (action, time of its application). We consider durative actions as defined in PDDL 2.1 (Fox and Long 2003) and discretized timelines (in contrast to PDDL 2.1).

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 the value of v in p. To accommodate the notion of time, we denote that a fact f or a (partial) variable assignment p holds in time t as $f(t)$ or $p(t)$ respectively. In an ***action*** $a = (dur(a), pre^{\vdash}(a), pre^{\vdash\!\dashv}(a), pre^{\dashv}(a), \mathit{eff}^{\vdash}(a), \mathit{eff}^{\dashv}(a))$, $dur(a)$ is a non-negative integer representing duration of a's application and the other elements are sets of partial variable assignments. In particular, $pre^{\vdash}(a)$ represents action precondition before its application, $pre^{\dashv}(a)$ represents action precondition before finishing its application, $pre^{\vdash\!\dashv}(a)$ represents action precondition for the whole time interval of its application, $\mathit{eff}^{\vdash}(a)$ 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 $pre^{\vdash}(a)(t)$, $pre^{\dashv}(a)(t + dur(a))$ and $\forall t' \in (t, t + dur(a)) : pre^{\vdash\!\dashv}(a)(t')$. 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 $t + dur(a)$. 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 $pre(a) = pre^{\vdash}(a) \cup pre^{\vdash\!\dashv}(a) \cup pre^{\dashv}(a)$ and $\mathit{eff}(a) = \mathit{eff}^{\vdash}(a) \cup \mathit{eff}^{\dashv}(a)$ unless otherwise stated.

We say that actions a_i and a_j ***possibly interfere*** if $vars(pre(a_i) \cup \mathit{eff}(a_i)) \cap vars(pre(a_j) \cup \mathit{eff}(a_j)) \neq \emptyset$.

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 have conflicts (i.e., no two or more actions modify the same variable, or one action modifies a variable some other action requires in its precondition at the same time), and $G(\max_{i=1}^{n}(t_i + dur(a_i)))$ 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 deadlines and hence we define a cost function that assigns each action and timestamp a non-negative cost, i.e., $cost : A \times \mathbb{N}_0 \to \mathbb{R}_0^+$. 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 $\sum_{i=1}^{n} cost(a_i, t_i) \leq \sum_{j=1}^{m} cost(a'_j, t'_j)$.

Another variant of planning task definition considers, rather than a single (hard) goal, a set of ***soft goals*** (each goal is a partial variable assignment) such that failing to achieve a goal is penalised. Therefore, 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 its cost is $\sum_{i \in \{i \mid G_i \text{ not achieved}\}} M_i$.

Normal-form Games

A ***normal-form game*** Γ is a tuple (N, S, u), where N is the number of players, $S = S_1, \ldots, S_N$ represents finite sets of pure strategies of players $1, \ldots, N$ and $u = (u_1, \ldots, u_N)$ is an 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); $u_i : S_1 \times \cdots \times S_N \to \mathbb{R}$. We say that a normal-form game is a ***zero-sum game*** if $\sum_{i=1}^{N} u_i = 0$. From now, we focus only on 2-player games, i.e., $N = 2$.

A ***mixed strategy*** for a player i is a probability distribution σ_i over the set of player's pure strategies S_i. A pair of mixed strategies $\sigma = (\sigma_1, \sigma_2)$ is called a ***mixed-strategy profile***. We extend the definition of utility functions so that for a given mixed-strategy profile σ, the value $u_i(\sigma)$ is the expected utility of player i. 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.

One way for tackling normal-form games is to incrementally build the game using the *Double-Oracle* algorithm (McMahan, Gordon, and Blum 2003). The algorithm starts with a restricted game, where each player's mixed strategy is composed from a subset of pure strategies, then, iteratively each player computes the best response expanding the restricted game. The algorithm terminates when neither of the players can add a best response strategy that improves the expected outcome from the restricted game. The NE of the restricted game matches the one in the original game, since best response is computed over the unrestricted set of all strategies (McMahan, Gordon, and Blum 2003).

The algorithm returns an optimal strategy, but is not monotone (in the upper and lower bounds on the game value in each iteration), and might have to consider, in the worst case, all pure strategies during its computation.

Case Studies

Resource Hunting Domain We consider a two-player game introduced by Rytíř, Chrpa, and Bošanský (2019), called *Resource Hunting*, where each player controls its fleet of unmanned aerial vehicles (UAVs) that tries to collect as many resources as possible. Each UAV can *move* from one location to another. Each UAV can carry at most two sensors. For each resource to be collected, one or two (different) sensors are required. One or two UAVs can *collect* an available resource if the UAV(s) are at the same location as the resource and carry the required sensors.

Taxi Domain We consider an on-demand transport scenario in which there are two taxi companies competing for passengers who require to be transported from one location to another. When one company picks up a passenger, she/he can no longer be transported by the other company. The goal of each taxi company is to maximise its rewards by transporting passengers, at the expense of the competing taxi company. Each taxi company operates a fleet of cars. In the *standard* variant, each car can carry at most one passenger at time, while in the *infinity* variant, each car has unlimited capacity. The car can move between two connected location by the *drive* action, can *load* a passenger into itself if both are at the same location, and can *unload* the passenger if being in his/her destination location.

Resource Competition Planning Task

In multi-agent environments, each agent executes its own actions in order to achieve its own goals. In non-cooperative settings, however, actions of one agent might interfere with actions of other agent(s). In adversarial or competitive settings, such conflicts between actions of multiple agents are usually inevitable as "winning" the conflict might be essential for achieving a given (soft) goal.

To illustrate the problem, we can observe in our case studies that after one agent collects a given resource or picks up a passenger, the other agent can no longer collect the resource or pick up the passenger. The conflicting actions are hence those trying to collect the same resource or to pick up a passenger. "Winning" the conflict in this context means that one agent collects a resource or picks up a passenger before the other agent tries to do so.

In general, we can define a planning task for 2-player normal form games. Our definition is partially inspired by the MA-STRIPS formalism (Brafman and Domshlak 2008) used in Multi-agent planning. In contrast to MA-STRIPS, we consider soft goals for each agent and durative actions.

Definition 1. *Let $\mathcal{NP} = (V, A^1, A^2, I, G^1, G^2)$ be a* **2-Player Normal-form Game (2PNG) Planning Task**, *where V is a set of variables, A^1 and A^2 such that $A^1 \cap A^2 = \emptyset$ are sets of (durative) actions for the first and second agent (or player), respectively, I is an initial state and G^1 and G^2 are sets of soft goals for the first and the second agent (or player), respectively.*

To address the 2PNG planning task, each agent might generate its own plan by solving an underlying planning task, where each agent uses only its actions to achieve its own goals. Plans of both agents are executed simultaneously and we will assume that plans of both agents start their execution at the same time. We can reasonably assume that a plan of an individual agent is conflict-free on its own, i.e., actions do not invalidate preconditions of other actions or actions do not try to change the value of a variable at the same time. However, while executing plans of both agents simultaneously, conflicts can arise (the agents compete against each other).

Definition 2. *We say that actions a_i and a_j have* **back interference** *iff $vars((pre^{\vdash\dashv}(a_j) \cup pre^{\dashv}(a_j)) \cap \mathit{eff}(a_i)) \neq \emptyset$ or $vars((pre^{\vdash\dashv}(a_i) \cup pre^{\dashv}(a_i)) \cap \mathit{eff}(a_j)) \neq \emptyset$. We also say that actions a_i and a_j have* **front interference** *iff $vars(pre^{\vdash}(a_j) \cap \mathit{eff}^{\vdash}(a_i)) \neq \emptyset$ or $vars(pre^{\vdash}(a_i) \cap \mathit{eff}^{\vdash}(a_j)) \neq \emptyset$.*

Back interference might cause situations in which an action of one agent might invalidate precondition of another agent's action that is already running. Invalidation of a precondition of a running action might have different outcomes that have to be specified in the action model. For the sake of simplicity of the execution model, we assume, in this paper, that no actions $a_i \in A^1, a_j \in A^2$ have back interference (it is the case for the considered case studies).

If actions having a front interference are scheduled at the same time, then one action is randomly selected by the "coin toss" (i.e., with an equal chance) to be applied while the other becomes inapplicable. Inapplicable actions are skipped during plan execution. Conflicts between effects are also resolved by the "coin toss". After both plans are executed, the cost of the plan for each agent is determined as the sum of costs of soft goals the agent failed to achieve. The utility value for each player is computed by subtracting the cost from the sum of costs of all player's soft goals.

Facts, i.e., variable assignments, that are required by actions of one agent (or they are part of agent's goals) can be deleted by actions of the competing agent. We call such facts *conflicting*. Specific conflicting facts that are initially true but neither are part of any agent's goals nor can be reachieved by either agent are called *critical facts* (the notion is adapted from (Chrpa, Rytíř, and Horčík 2020)).

Definition 3. *Let $\mathcal{NP} = (V, A^1, A^2, I, G^1, G^2)$ be a 2PNG planning task. We say that (v, val), where $v \in V$ and $val \in D(v)$, is a* **conflicting fact** *iff there exists $a \in A^i : (v, val) \in pre(a)$ or (v, val) is a part of G^i and there exists $a' \in A^j : (v, val') \in \mathit{eff}(a') \wedge val \neq val'$ with $i \neq j$.*

We say that a conflicting fact (v, val) is a **critical fact** *iff $(v, val) \in I$, (v, val) is neither a part of G^1 nor G^2 and for each $a \in A^1 \cup A^2 : (v, val) \notin \mathit{eff}(a)$.*

Resource Competition planning tasks, as defined below, are a subclass of 2PNG planning tasks in which all conflicting facts are critical facts and no actions of different agents

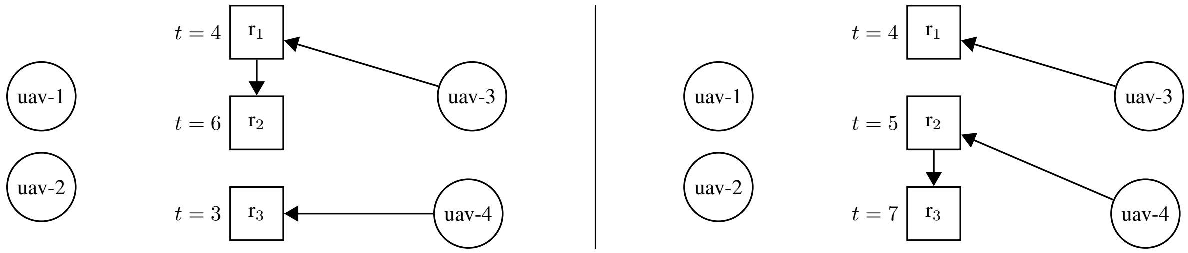

Figure 1: Figure depicts two pure strategies of an adversary in a Resource Hunting scenario, where agent's UAVs (uav-1 and uav-2) compete for resources r_1,r_2 and r_3 with adversary's UAVs (uav-3 and uav-4). In the left-hand-side strategy, the adversary collects r_1 and r_2 (in this order) by uav-3 in time 4 and 6, respectively, and r_3 by uav-4 in time 3. In the right-hand-side strategy, the adversary collects r_2 and r_3 (in this order) by uav-4 in time 5 and 7, respectively, and r_1 by uav-3 in time 4.

have back interference. Note that critical facts represent resource availability before they are collected by either agent.

Definition 4. *Let $\mathcal{RP} = (V, A^1, A^2, I, G^1, G^2)$ be a 2PNG planning task. We say that $\mathcal{RP}$ is a* **Resource Competition (RC) Planning Task** *iff each conflicting fact over V is a critical fact and for no pair of actions $a^1 \in A^1$ and $a^2 \in A^2$ it is the case that a^1 and a^2 have back interference.*

Critical and Adversary Actions

To achieve its (soft) goals the agent has to apply certain *critical* actions requiring specific *critical* facts. The adversary, on the other hand, can apply *adversary* actions deleting these critical facts and making them no longer achievable. For example, an available(r1)=true fact is required by agent's collect(uav-1,r1) action while adversary's collect(uav-3,r1) deletes the fact by setting available(r1)=false. Hereinafter, we present the terminology from the perspective of agent 1, so agent 2 is considered as an adversary or a competitor. We adopted the notions of critical and adversary actions from Chrpa, Rytíř, and Horčík (2020).

Definition 5. *Let $\mathcal{RP} = (V, A^1, A^2, I, G^1, G^2)$ be a RC planning task and (v, val) be a critical fact. We say that $A^c = \{a^c \mid a^c \in A^1, (v, val) \in pre^{\vdash}(a^c)\}$ is a set of* **critical actions** *over (v, val). We also define a set of* **adversary actions** *as $A^a = \{a^a \mid a^a \in A^2, (v, val') \in eff(a^a), val \neq val'\}$.*

Conceptually, the agent needs to apply its critical actions before the adversary applies its adversary actions, e.g., the agent has to collect a resource before the adversary does. Therefore, adversary actions set deadlines for agent's critical actions. For example, in Figure 1 we depict two plans of the adversary in which the adversary collects the r_2 resource in time 6 and 5, respectively. Hence, the deadlines for the agent to collect r_2 are 6 and 5, respectively. We define a function $at(f, a, eff)$ such that $at(f, a, eff) = 0$ iff $f \in eff^{\vdash}(a)$, or $at(f, a, eff) = dur(a)$ iff $f \in eff^{\dashv}(a)$ and $f \notin eff^{\vdash}(a)$.

Definition 6. *Let A^c, A^a and (v, val) be as in Definition 5. Let $\pi' = \{(a'_1, t'_1), \ldots, (a'_m, t'_m)\}$ be a plan of the adversary. Then, for each $a^c \in A^c$ and $(v, val) \in pre^{\vdash}(a^c)$, we can determine a* **local deadline** *with respect to π' and (v, val), denoted as $dl(a^c, \pi', (v, val))$ as $\min\{t + at((v, val'), a', eff) \mid (a', t) \in \pi', a' \in A^a, (v, val') \in eff(a'), val \neq val'\}$.*

A **(global) deadline** *for a critical action a^c with respect to π', denoted as $dl(a^c, \pi')$ is defined as $\min_{f' \text{ is a critical fact for } a^c} dl(a^c, \pi', f')$.*

Response Planning Task

Deadlines set by adversary actions determine whether corresponding critical actions can be successfully applied. Of course, not all critical actions are required to achieve the goal(s). We are therefore interested in critical actions that are *action landmarks* or part of *disjunctive action landmarks* (Hoffmann, Porteous, and Sebastia 2004). In particular, for achieving a goal an action landmark has to be present in every (valid) plan while for a disjunctive action landmark at least one its action has to be present in every plan. Considering planning tasks with sets of soft goals, we determine for each soft goal which critical actions are (possibly) necessary for achieving it. For example, *collect* actions over a resource r_1 form a disjunctive action landmark for r_1 and thus are relevant to the goal of collecting r_1.

Definition 7. *Let $\mathcal{RP} = (V, A^1, A^2, I, G^1, G^2)$ be a RC planning task with $G^1 = \{G_1, \ldots, G_n\}$ being a set of soft goals. Let $A^c \subseteq A^1$ be a set of all critical actions and $\mathcal{A}^l_i$ be the set of all disjunctive action landmarks for $\mathcal{P}_i = (V, A^1, I, G_i)$. We say that a set of critical actions $A^c_i = \bigcup_{A^l_i \in \mathcal{A}^l_i} A^l_i \cap A^c$ is* **relevant** *to G_i.*

It is reasonable to assume that the adversary will not follow a single plan to set (firm) deadlines for the agent's critical actions, but a randomised mixed strategy in form of a set of plans, where each plan has a given probability for being selected (and executed) (Rytíř, Chrpa, and Bošanský 2019). As the deadlines for agent's critical actions are determined only by the adversary actions, we do not have to consider whole plans of the adversary but rather sets of adversary actions with their application timestamps. Given a critical action and a timestamp, we can project for which pure strategies of the adversary the critical action will miss the deadline, or be exactly on the deadline, as formalised in the following definition.

Definition 8. *Let $\mathcal{RP} = (V, A^1, A^2, I, G^1, G^2)$ be a RC planning task and $A^d \subseteq A^2$ be a set of all adversary actions. We say that* **adversary strategy** *σ over $\mathcal{RP}$ is a set*

$\sigma = \{(ad_1, p_1), \ldots, (ad_n, p_n)\}$ *such that* $\sum_{i=1}^{n} p_i = 1$ *and for each* i *it is the case that* $ad_i = \{(a_1^i, t_1^i), \ldots, (a_k^i, t_k^i)\}$*, where* $a_j^i \in A^d$ *and* t_j^i *is the timestamp of* a_j^i*'s application* ($1 \leq j \leq k$).

Let $A^c \subseteq A^1$ *be the set of all agent's critical actions. We define an* **after deadline** *strategy projection over* σ *as* $d_\sigma^{>} : A^c \times \mathbb{N}_0 \rightarrow 2^\sigma$ *such that* $d_\sigma^{>}(a, t) = \{(ad_i, p_i) \mid (ad_i, p_i) \in \sigma, dl(a, ad_i) < t\}$*. Analogously, we define an* **on deadline** *strategy projection over* σ *as* $d_\sigma^{=} : A^c \times \mathbb{N}_0 \rightarrow 2^\sigma$ *such that* $d_\sigma^{=}(a, t) = \{(ad_i, p_i) \mid (ad_i, p_i) \in \sigma, dl(a, ad_i) = t\}$.

Adversary's strategies, in form of sets of adversary actions, provide multiple deadlines for agent's critical actions. Each competitor's strategy occurs with a given probability. In consequence while considering the assumption that to achieve a soft goal at most one critical action is necessary, the probability of reaching a given soft goal equals the 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 adversary actions. Note that in situations in which a critical action is applied at the same time as the corresponding adversary action, we evenly "split the cost". For example, let us assume that the left-hand-side plan in Figure 1 has the probability of being selected 0.6 while the right-hand-side plan 0.4 and the cost of failing to collect r_3 is 100. Then collect(uav-1,r_3) and collect(uav-2,r_3) will get the following cost depending on their application time. For $t < 3$, the cost will be 0 as both deadlines will be met. For $3 < t < 7$, one deadline will be missed and the cost will be 60 (as the agent will fail to collect r_3 with probability 0.6). For $t > 7$, both deadlines will be missed and hence the cost will be 100. Note that "on deadline" cases, i.e., with $t = 3$ and $t = 7$, the corresponding deadline will be missed on 50%, and thus the costs will be 30 and 80, respectively.

Plans are then optimised for minimising the total action cost, in other words, maximising agent's expected utility with respect to the given adversary's strategy. The next definition is adapted from Rytíř, Chrpa, and Bošanský (2019).

Definition 9. *Let* $\mathcal{RP} = (V, A^1, A^2, I, G^1, G^2)$ *be a RC planning task with* $G^1 = \{G_1, \ldots, G_k\}$ *being a set of soft goals and* σ *be an adversary strategy over* $\mathcal{RP}$*. Let* $A_i^c \subseteq A^1$ *be the set of critical actions relevant to* G_i *and* M_i *be the cost for failing to achieve* G_i ($1 \leq i \leq k$)*. We define a* **cost function** $c_{\mathcal{RP},\sigma} : A^1 \times \mathbb{N}_0 \rightarrow \mathcal{R}_0^+$*, where* $c_{\mathcal{RP},\sigma}(a, t) =$

$$\sum_{i \in \{j \mid a \in A_j^c\}} M_i \Big(\sum_{(ad_i, p_i) \in d_\sigma^{>}(a,t)} p_i + 1/2 \sum_{(ad_i, p_i) \in d_\sigma^{=}(a,t)} p_i \Big), \text{ if } a \in \bigcup_{i=1}^{n} A_i^c,$$

and $c_{\mathcal{RP},\sigma}(a, t) = 0$*, otherwise. This sets up an agent's* **response planning task** $\mathcal{P}_\sigma = (V, A^1, I, G^1)$ *such that actions are associated with the* $c_{\mathcal{RP},\sigma}$ *cost function.*

The cost function in the above definition ensures that the cost-optimal plan is the best response if at most one critical action is needed for achieving each soft goal (we then refer to a *best response planning task*). Note that for both our domains this condition is satisfied. In more general cases, the cost function might depend on other critical actions relevant to the same goal that are (already) present in a partial plan. Hence classical cost-optimal planning might not be applicable in such cases.

Algorithm 1: Estimating earliest action application and fact occurrence time

1: **function** EARLIESTTIME($F, O, A, time$)
2:   **while** true **do**
3:     $A' \leftarrow \{a \mid a \in A \setminus O, pre^{\vdash}(a) \cup pre^{\vdash\!\dashv}(a) \subseteq F\}$
4:     $\forall a \in A' : time(a) \leftarrow \max\{time(f) \mid f \in pre^{\vdash}(a) \cup pre^{\vdash\!\dashv}(a)\}$
5:     $a \leftarrow \arg\min_x \{time(x) \mid x \in A'\}$
6:     **if** a is undefined **then** break
7:     $O \leftarrow O \cup \{a\}$
8:     $\forall f \in \mathit{eff}(a) : time(f) \leftarrow \min(time(f), time(a) + at(f, a, \mathit{eff}))$
9:     $F \leftarrow F \cup \mathit{eff}(a)$
10:   **end while**
11:   $\forall a \notin O : time(a) \leftarrow \infty$
12:   **return** $F, time$
13: **end function**

Estimating Earliest Action Application Time

We adopt the algorithm for estimating lower bounds of action application and fact occurrence time proposed by Chrpa, Rytíř, and Horčík (2020). For this purpose, we define a function $time : A \cup F \rightarrow \mathbb{N}_0$ assigning a timestamp to either an action from A or to a fact from the set of all variable assignments F. The *EarliestTime* function in Algorithm 1 is inspired by the h^{max} heuristics in classical planning (Bonet and Geffner 2001), with sets F and O representing processed facts and actions, respectively. Each iteration involves selection of not yet processed actions (Line 3), determining their application time as maximum across the times of its "at start" and "over all" preconditions (Line 4), selecting the action (not yet processed) with the lowest application time (Line 5), and determining times of its effects as minimum of the current time and the time when the effect of the selected action takes place (Line 8). Note that "at end" preconditions are relaxed out as their presence might cause that actions whose application starts later influence possible application starting time of earlier actions.

Estimating Adversary Strategy

Formulating a (best) response planning task requires knowledge of an adversary (mixed) strategy. Computing adversary strategy by the Double Oracle algorithm is computationally expensive as even for small problems, tens planning tasks have to be (optimally) solved (Rytíř, Chrpa, and Bošanský 2019). We hence propose a heuristic method that estimates when the competitor can apply its adversary actions as such an information is important for setting the deadlines for agent's critical actions and thus formulating the (best) response problem. That said, we do not need to compute whole plans of the competitor.

Algorithm 2 summarises the routine for estimating an adversary strategy. Initially, for each (soft) goal G_i, "clusters"

Algorithm 2: Estimating adversary strategy

Require: RC planning task $\mathcal{RP} = (V, A^1, A^2, I, G^1, G^2)$ with $G^1 = \{G_1, \ldots, G_n\}$, number of samples k
Ensure: Estimated adversary strategy σ
1: **for** $i = 1$ to n **do**
2: Let $\mathcal{A}_i^c$ be relevant to G_i (as in Def. 7)
3: $C_i = \{a \mid a \in A^2, a$ is an adversary action over f, $a^c \in A_i^c, A_i^c \in \mathcal{A}_i^c$, a^c is a critical action over $f\}$
4: **end for**
5: $\sigma \leftarrow \emptyset$
6: **for** $i = 1$ to k **do**
7: $ad \leftarrow$ EstimateAdversaryTime($\mathcal{RP}, \{C_1, \ldots, C_n\}$)
8: **if** $(ad, p) \in \sigma$ **then**
9: $\sigma \leftarrow \sigma \setminus \{(ad, p)\} \cup \{(ad, p + \frac{1}{k})\}$
10: **else**
11: $\sigma \leftarrow \sigma \cup \{(ad, \frac{1}{k})\}$
12: **end if**
13: **end for**

14: **function** ESTIMATEADVERSARYTIME($\mathcal{RP}, C$)
15: $sa \leftarrow \emptyset, F \leftarrow \emptyset$
16: set $time$ as undefined
17: **for all** $f \in I$ **do**
18: $time(f) \leftarrow 0, F \leftarrow F \cup \{f\}$
19: **end for**
20: **while** $C \neq \emptyset$ **do**
21: $F, time' \leftarrow$EarliestTime($F, sa, A^2, time$)
22: $A' \leftarrow \{a \mid a \in \bigcup_{C_i \in C} C_i, time'(a) \neq \infty\}$
23: **if** $A' = \emptyset$ **then** break **end if**
24: $a' \leftarrow$Select(A')
25: $sa \leftarrow sa \cup \{a'\}, time(a') \leftarrow time'(a')$
26: $C \leftarrow C \setminus \{C' \mid a' \in C'\}$
27: $F, time \leftarrow$ UpdateFactTime($a', F, time'$)
28: **end while**
29: **return** $\{(a, time(a)) \mid a \in sa\}$
30: **end function**

of adversary actions (denoted as C_i) corresponding to sets of critical actions relevant for G_i are determined (Line 3). Then, it k times samples a "skeleton" of an adversary's plan in the form of a set of sampled adversary actions with their estimated application times (the *EstimateAdversaryTime* routine). EstimateAdversaryTime initially sets the set of selected adversary actions (sa) as empty and the initial facts with time of 0. Then, in each iteration of the while loop, Algorithm 1 is called to estimate application time of the actions (Line 21), and then an adversary action, which is reachable, is selected (Line 24) and the corresponding cluster is removed (Line 26). Lastly, we update the set of facts F and the $time$ function according to the selected action a' (Line 27). For facts in F whose variable is not part of a', the value of $time$ will be the same as the value of $time'$. For variables being part of a', the value that remains true after application of a' is considered in F and $time$ is set to when a' accessed the fact (in precondition of a') or modified the fact (in effects of a') for the last time. The facts representing the other variable values are removed from F and $time$ for them is set to ∞.

Probability of selecting a particular adversary action (the *Select* function) depends on their estimated application time such that those with lower application time are preferred. The following expression, inspired by the roulette wheel selection in genetic programming (Goldberg and Deb 1991), denotes how the probability is computed:

$$p_a = \frac{1 - \left(\frac{time(a)}{\sum_{a' \in A'} time(a')}\right)}{|A'| - 1}$$

As adversary actions are selected randomly (according to the probability), different runs of The EstimateAdversaryTime routine produce different outcomes. The k outcomes of The EstimateAdversaryTime routine are then "arranged" into an adversary strategy σ.

For example, the EstimateAdversaryTime routine can produce the left-hand-side pure strategy in Figure 1 by iteratively sampling collect(uav-4,r_3), collect(uav-3,r_1), collect(uav-3,r_2) (in this order). Note that after sampling, for example, collect(uav-3,r_1) the F set will contain the fact that uav-3 is in the location of r_1 at time 4 plus the duration of collect(uav-3,r_1).

Let us have two adversary actions a_i, a_j such that $vars(pre(a_i) \cup eff(a_i)) \cap vars(pre(a_j) \cup eff(a_j)) \neq \emptyset$. Also let us assume if either a_i or a_j is in A', then the other action can be applied after the one in A' finishes its application. Under this assumption we can derive that the EstimateAdversaryTime routine does not overestimate application time of selected adversary actions (in a given order). Such an observation is implied from the fact that the EarliestTime function (Alg. 1) does not overestimate action application time and fact occurrence time (Chrpa, Rytíř, and Horčík 2020).

The last step, after adversary strategy σ is obtained, is the formulation of the response planning task (according to Definition 9) and solving the task by generating a plan.

Experimental Evaluation

The aim of the experiments is to evaluate i) the quality of plans generated with the use of our sampling method vs the naive one (minimising plan makespan) and vs the mixed (planning) strategy generated with the Double Oracle approach, ii) the CPU time required for running the above methods, and iii) the exploitability of the naive and sampling method showing how much generated plans are subject to exploitation from the competitor.

We encoded both case study domains as temporal domains (in PDDL 2.1) for the naive approach, and in a classical subset of PDDL for the sampling and Double Oracle approach. To reason with (discrete) time in the classical models we introduced "timeline" objects analogously to Rytíř, Chrpa, and Bošanský (2019). To obtain state-variable representation we used Temporal Fast Downward (Eyerich, Mattmüller, and Röger 2009). For discovering disjunctive action landmarks, we used the back-chaining method (Hoffmann, Porteous, and Sebastia 2004). As an optimal classical planner, we used the Fast Downward planner (Helmert 2006) with the potential heuristic (Pommerening et al. 2015) optimised by the diversification method proposed by Seipp,

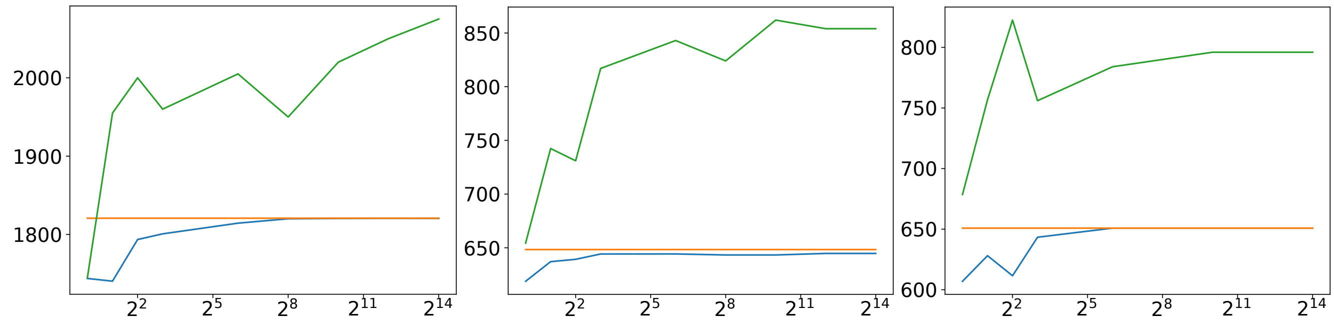

Figure 2: Comparing the cumulative utility value of plans (y-axis) generated by the optimal planner using our strategy sampling approach against the optimal mixed strategy generated by Double Oracle (blue line) and the naive plans (green line) with an increasing number of samples (x-axis). Orange line denotes the Nash Equilibrium utility value. The Resource Hunting domain is on the left, the standard Taxi domain in the middle and the infinity variant of the Taxi domain is on the right.

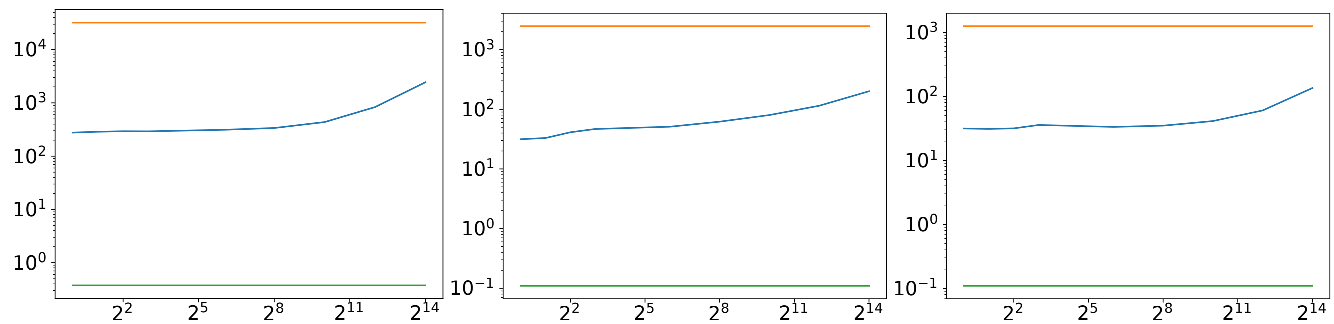

Figure 3: Comparing the cumulative runtime of plan generation (y-axis, in seconds) by our strategy sampling approach (blue line) against the runtimes of optimal mixed strategy generation by Double Oracle (orange line) and the naive plan generation (green line) with an increasing number of samples (x-axis). Note that runtime is in logarithmic scale. The Resource Hunting domain is on the left, the standard Taxi domain in the middle and the infinity variant of the Taxi domain is on the right.

Pommerening, and Helmert (2015). As an optimal temporal planner we used CPT4 (Vidal 2011). We ran the experiments on Linux with 2.10GHz Intel Xeon CPU E5-2620 v4 with 32GB RAM.

An interesting property of the Resource Hunting domain as well as the infinity variant of the Taxi domain is that Algorithm 2 provides a perfect heuristic estimation of the application time of the adversary's *collect* (or *load*) actions because in each iteration of Algorithm 2 the application time of the next *collect* (or *load*) action depends on the distance of the required UAVs (or car) that is never underestimated. In contrast to Resource Hunting, passengers have to be delivered to their required destinations, which might create larger discrepancies between makespan optimal and "adversary aware" plans. In the standard variant of the Taxi domain, Algorithm 2 provides perfect heuristic estimation only for cars loading their first passenger, then Algorithm 2 underestimates loading times of subsequent passengers, since the heuristic incorrectly assumes that delivering the current passenger (to empty the car) and going to load the next passenger can be done simultaneously.

For the experiments, we consider eight scenarios of the Resource Hunting domain ranging from 1 UAV per player to 3 UAVs per player and 2 to 6 resources. Further, we consider five problems for each variant of the Taxi domain, ranging from 1 to 2 taxi per player and 3 to 5 passengers.[1]

Figure 2 summarises the results of a comparison between the Sampling method, the Double Oracle method (setting the Nash Equilibrium value), and the naive one in terms of utility values of agent's plans against competitor's plans or (mixed) strategies. The sampling method was run 40 times for each setting and the presented results show average values. We observed that the utility values fluctuated for each run (much) more with smaller number of samples and were rather stable with the large number of samples. As the results show, the sampling method generates plans whose utility values usually converge to the equilibrium with a growing number of samples. Comparing to the naive method, the sampling method often provides a better plan (with growing number of samples), i.e., the utility value of the agent is greater than the equilibrium value (there are, however, a few exceptions – 2 problems in Resource Hunting and 1 problem in each variant of the Taxi domain).

Figure 3 depicts cumulative runtimes of the sampling method (with respect to the number of samples), the Double Oracle method for generating Nash Equilibrium mixed strategies and naive plan generation. Naive plans were gen-

[1]Code and benchmarks can be found at https://gitlab.com/FRASProject/aaai22-competing-for-resources

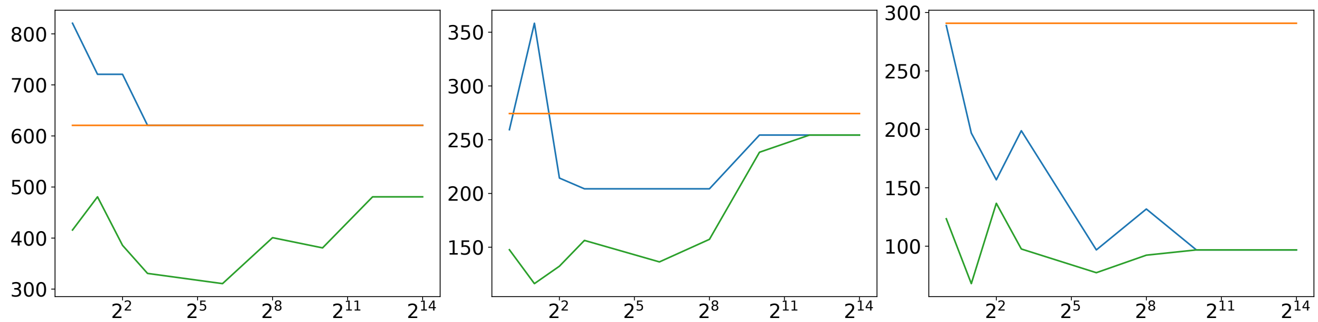

Figure 4: Comparing the cumulative exploitability value (y-axis) of naive plans (orange line), single plans generated by the optimal planner using our strategy sampling approach (blue line) and the uniform mixed strategies of 10 plans generated by different runs of the sampling method (green line) with an increasing number of samples (x-axis). The Resource Hunting domain is on the left, the standard Taxi domain in the middle and the infinity variant of the Taxi domain is on the right is on the right.

erated in less than 100ms for each of the considered problems and hence cumulative runtimes were lower than 1 second in each domain. Unsurprisingly, runtimes of the sampling method increased with the higher number of samples. For the largest number of samples (2^{16}) the runtime ranged from 7 to 473 seconds per problem while for 2^8 samples the runtime ranged from 6 to 75 seconds per problem. Even though the higher number of samples leads to higher runtime of the adversary strategy estimating algorithm (Algorithm 2), the main reason for runtime increase with the higher number of samples is the "complexity" of the response planning tasks. That said, more samples produce more deadlines for critical actions and hence generating cost-optimal plans for such response planning tasks is harder for the planner. Generating Nash Equilibrium mixed strategies (by Double Oracle) is slower by an order of magnitude than the sampling method, in average 9 – 53 times depending on the number of samples. Such a result comes with no surprise as Double Oracle has to optimally solve a number of response planning tasks (between 5 – 44).

Figure 4 considers the robustness of generated plans by measuring their "exploitability" (lower is better): for a given plan or a (mixed) strategy, we calculate the difference of the best response plan against it and the equilibrium value. Note that a strategy in Nash Equilibrium has 0 exploitability. For single plans, there is no difference between the sampling method generated plans (with more samples) and the naive plans in Resource Hunting, while in Taxi, single plans generated by the sampling method have lower exploitability. We have, in addition, considered mixed strategies consisting of 10 plans generated by 10 different runs of the sampling method (in the same setting). With not so small number of samples, where the variance of outcomes can be large, we find more diverse plans that —if combined into a strategy— are harder to exploit. The exploitability results demonstrate that single plans are easier to exploit (if they are known to the adversary) than mixed strategies combining more different plans (which matches the observation of LaValle (2006)). The agent, however, does not have to generate such a mixed strategy as due to randomness of the sampling method, the competitor might not be able to guess an actual plan even if it knows the method (and settings) the agent uses. This means that plans generated by the sampling method are more robust, since they are harder to guess.

It is possible to use satisficing planners with all the methods. Whereas the use of satisficing planners can improve scalability, the results are heavily dependent on how suboptimal the generated plans are. It often happens that different best response tasks over the same problem instance (i.e., differing only in critical action cost distribution) lead to plans of varying suboptimality. The use of optimal planners alleviates such a bias.

Conclusion

Planing in adversarial environments requires to predict the strategy of the competitor, so the agent can optimise its plan accordingly. We defined a subclass of adversarial problems - the Resource Competition problems in which two agents compete for (limited) resources. We have presented a method that, by sampling, estimates the ordering and the application time of actions, by which the adversary collects the resources. By applying the method several times, we derived a (randomised) mixed strategy of the adversary, which imposed deadlines for the agent's actions collecting the resources. Missing the deadlines is encoded by action cost and hence cost-optimal plan is the best response to the estimated adversary strategy.

The results show that the sampling method outperforms the naive one (i.e., plans that optimise for make-span) in terms of quality (although the plan generation time is higher for the sampling method). Also, plans generated by the sampling method are harder to guess and exploit due to randomness of adversary action selection; even if the competitor conveys full knowledge of the sampling method (including the parameters), it might come up with a different plan.

In future, we plan to extend the sampling method for more than 2 players. We also plan to use the sampling method in an online mode, for example, with MCTS techniques or by dynamically adapting plans on the fly.

Acknowledgements

We thank Jan Čuhel and Anastasiia Livochka for their help in the initial stages of the implementation.

This research was funded by AFOSR award FA9550-18-1-0097 and by the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".

References

Applegate, C.; Elsaesser, C.; and Sanborn, J. C. 1990. An architecture for adversarial planning. *IEEE Trans. Systems, Man, and Cybernetics*, 20(1): 186–194.

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

Brafman, R. I.; and Domshlak, C. 2008. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *The Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008*, 28–35.

Chrpa, L.; Rytíř, P.; and Horčík, R. 2020. Planning Against Adversary in Zero-Sum Games: Heuristics for Selecting and Ordering Critical Actions. In *The Thirteenth International Symposium on Combinatorial Search, SOCS 2020*, 20–28.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.*, 147(1-2): 35–84.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the Context-enhanced Additive Heuristic for Temporal and Numeric Planning. In *The 19th International Conference on Automated Planning and Scheduling, ICAPS 2009*.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.*, 20: 61–124.

Goldberg, D. E.; and Deb, K. 1991. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. volume 1 of *Foundations of Genetic Algorithms*, 69–93. Elsevier.

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

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. *J. Artif. Intell. Res.*, 22: 215–278.

Jensen, R.; Veloso, M.; and Bowling, M. 2001. OBDD-based optimistic and strong cyclic adversarial planning. In *ECP*, 265–276.

Justesen, N.; Mahlmann, T.; Risi, S.; and Togelius, J. 2018. Playing Multiaction Adversarial Games: Online Evolutionary Planning Versus Tree Search. *IEEE Trans. Games*, 10(3): 281–291.

Justesen, N.; and Risi, S. 2017. Continual online evolutionary planning for in-game build order adaptation in StarCraft. In *The Genetic and Evolutionary Computation Conference, GECCO 2017*, 187–194.

Kissmann, P.; and Edelkamp, S. 2009. Solving Fully-Observable Non-deterministic Planning Problems via Translation into a General Game. In *KI*, volume 5803, 1–8. Springer.

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge University Press. ISBN 9780511546877.

Lelis, L. H. S. 2020. Planning Algorithms for Zero-Sum Games with Exponential Action Spaces: A Unifying Perspective. In *The Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, 4892–4898.

Letchford, J.; and Vorobeychik, Y. 2013. Optimal interdiction of attack plans. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13*, 199–206. IFAAMAS.

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.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From Non-Negative to General Operator Cost Partitioning. In *Proc. AAAI'15*, 3335–3341.

Pozanco, A.; E-Martín, Y.; Fernández, S.; and Borrajo, D. 2018. Counterplanning using Goal Recognition and Landmarks. In *The Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018*, 4808–4814.

Rytíř, P.; Chrpa, L.; and Bošanský, B. 2019. Using Classical Planning in Adversarial Problems. In *Proceedings of the 31st IEEE 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 *Proc. ICAPS'15*, 193–201.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.

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*.

Vidal, V. 2011. CPT4: An Optimal Temporal Planner Lost in a Planning Competition without Optimal Temporal Track. In *Proceedings of the 7th International Planning Competition (IPC-2011)*, 25–28. Freiburg, Germany.

Vinyals, O.; Babuschkin, I.; Czarnecki, W.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D.; Powell, R.; Ewalds, T.; Georgiev, P.; Oh, J.; Horgan, D.; Kroiss, M.; Danihelka, I.; Huang, A.; Sifre, L.; Cai, T.; Agapiou, J.; Jaderberg, M.; and Silver, D. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575.

Vorobeychik, Y.; and Pritchard, M. 2020. Plan Interdiction Games. In *Adaptive Autonomous Secure Cyber Systems*, 159–182. Springer.