Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)

On Generating Robust Plans and Linear Execution Strategies in Planning Against Nature

Lukáš Chrpa[1], **Erez Karpas**[2]

[1]Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Prague, Czechia
[2]Faculty of Data and Decision Sciences, Technion - Israel Institute of Technology, Haifa, Israel
chrpaluk@cvut.cz, karpase@technion.ac.il

Abstract

Planning against nature is a recent concept describing planning and acting in environments in which nature can non-deterministically trigger exogenous events, where the agent has to consider that the state of the environment might change without its consent. Therefore, the agent has to make sure that it eventually achieves its goal (if possible) despite the acts of nature. In this paper, we leverage the recent concept of *robust plans*, which assumes that nature might act as an adversary, to design a method for generating *linear execution strategies*, which assume that nature acts randomly but *fairly*. In particular, we consider events that have to eventually occur and facts that even if deleted by events will be eventually reachieved by (other) events (because nature acts fairly). To improve the efficiency of both robust plan and linear execution strategy generation methods, we provide an approach allowing us to adopt delete-relaxed heuristics that are used in classical planning.

Introduction

Autonomous agents controlled by automated planners (Ingrand and Ghallab 2017) operating in the real world, in scenarios such as planetary rovers (Ai-Chang et al. 2004) or autonomous underwater vehicles (Chrpa et al. 2015), face a challenge which classical planning does not account for – exogenous events. These exogenous events are triggered by an actor which we refer to as *nature*, to signify that it has no goal of its own, but rather acts randomly. A rational agent which wants to achieve its goal in the presence of such exogenous events must take these into account during both planning and execution. We refer to this class of problems as *planning against nature*. While planning against nature can be framed as fully-observable non-deterministic (FOND) planning (Cimatti et al. 2003), this is not really a tractable approach when we consider that nature might apply an arbitrarily finitely long sequence of exogenous events after every action of the agent. Thus, the number of non-deterministic alternatives (per action) might be exponential with respect to the size of the representation as it is bounded by the number of states.

For example, the *AUV domain* (Chrpa, Gemrot, and Pilát 2020), inspired by real-world AUV operations (Chrpa et al. 2015), simulates a scenario in which the AUV has to perform sampling of given objects of interest in the presence of ships (controlled by nature) passing by in corridors. If a ship enters the AUV's location, then the AUV is destroyed.

Early works on planning against nature (Chrpa, Gemrot, and Pilát 2020; Chrpa, Pilát, and Med 2021) aimed at alleviating non-deterministic branching by studying under which conditions a sequential plan is guaranteed to eventually achieve the goal, under the assumption that nature can only execute a set of independent events (events that do not delete preconditions or effects of each other (Blum and Furst 1997)) after every agent action. More recent work (Chrpa and Karpas 2024a,b) considered the less restrictive assumption, where nature can execute a finite sequence of events after every agent action, with a fairness assumption that nature will eventually execute every event that becomes available infinitely often – similarly to the fairness assumption behind strong cyclic FOND planning (Cimatti et al. 2003). More precisely, two types of plans were proposed: *robust plans* (Chrpa and Karpas 2024a), which are sequences of actions that work no matter what events nature executes, and *linear execution strategies* (Chrpa and Karpas 2024b), which are specialized policies in which actions are executed in a fixed order, but there is a condition associated with when to execute each action in the sequences. For the sake of brevity, we use the term *plan* to refer to both.

Importantly, this previous work focused on *verifying* whether a given plan achieves the goal, regardless of the nature. Plan generation was investigated only for robust plans but limited to a blind search which enumerates all plans in a breadth-first order that tends to scale poorly even for relatively small tasks (Chrpa and Karpas 2024a). In this paper, we present new techniques for *generating* plans which work against nature. Initially, we present heuristics for guiding the search for robust plans that are based on delete-relaxation, well established in classical planning (Bonet, Loerincs, and Geffner 1997; Bonet and Geffner 2001). After that, we extend the method for generating robust plans for generating linear execution strategies. Our empirical evaluation demonstrates that our plan generation methods provide better coverage than the verification approaches, and that using the heuristics considerably reduces the runtime (with respect to the existing blind approach), and that the runtime is in the same order of magnitude as for the verification approaches.

Related Work

Planning against nature extends classical planning with the addition of exogenous events. Exogenous events were considered in planning (Dean and Wellman 1990; Iocchi, Nardi, and Rosati 2000) in systems such as Circa (Musliner, Durfee, and Shin 1993). These systems usually have to reason with a whole (or almost whole) state space. Markov Decision Process (MDP)-based approaches can be leveraged to tackle events (Mausam and Kolobov 2012) and aim to generate a policy with the most promising action in each state. Monte-Carlo Tree Search (MCTS) approaches provide similar benefits; however, their success rate tends to drop for problems with dead-ends (Patra et al. 2021).

A lazy approach for addressing non-determinism is to relax events and generate plans by classical planners, and if, during acting, the agent encounters an unknown state or cannot apply the next action, it replans (Komenda, Novák, and Pechoucek 2014). The success of FF-replan (Yoon, Fern, and Givan 2007) in the International Planning Competition 2006 (where it was an unofficial winner of the probabilistic track) indicates this is often a viable approach. However, in domains with dead-ends this might not be effective, and might even be dangerous (Little and Thiebaux 2007).

To address the issue of encountering dead-ends while using the (classical) planning and replanning strategy Chrpa, Gemrot, and Pilát (2020) adapted the notion of *safe states* (Cserna et al. 2018) such that no sequence of events can transform a safe state to a dead-end state and proposed a technique that iteratively generates *robust plans*, guaranteed to always succeed regardless of events, connecting safe states until the goal is achieved. A subsequent work of Chrpa, Pilát, and Med (2021) introduced a technique for generating *eventually applicable plans* that, in our terminology, refer to linear execution strategies. In contrast to us, these works assumed that nature can only execute one set of independent events after every agent's action. Later work (Chrpa and Karpas 2024b,a) addressed a different model in which nature can execute a series of events after every agent action – this is the setting we address in this paper, although we focus on *generating* plans rather just on verifying given plans.

Conformant planning deals with the problem of generating linear plans in partially or unobservable environments (Cimatti and Roveri 2000; Bonet 2010). Conformant planning can be addressed, for instance, by extending classical planners (Hoffmann and Brafman 2006) or by compiling it to classical planning (Palacios and Geffner 2009). Although robust plans are similar in spirit to conformant plans, the settings are different – planning against nature assumes full observability, but the environment can be modified by an act of nature. Furthermore, linear execution strategies can be interpreted as a policy with loops that waits until the wait-for precondition of the next action is satisfied, similarly to waitfor conditions in social laws (Karpas, Shleyfman, and Tennenholtz 2017; Tuisov, Shleyfman, and Karpas 2024).

Fully Observable Non-deterministic (FOND) planning concerns similar tasks, in which the environment is fully observable while actions have several different outcomes and if one such action is applied a random outcome occurs (Cimatti et al. 2003). The task is to find a strong plan that is a solution of a FOND planning task. For instance, the well-known PRP planner (Muise, McIlraith, and Beck 2012) looks for strong plans by leveraging classical planning techniques and handling non-determinism by attempting to "close" states from which there does not yet exist a plan. FOND planning is known to be EXPTIME-complete (Littman, Goldsmith, and Mundhenk 1998).

Although both FOND planning and planning against nature deal with non-determinism, there is a fundamental difference in how non-determinism occurs. In FOND planning, non-determinism is triggered by (non-deterministic) actions of the agent while in planning against nature non-determinism is triggered by events that nature can apply. Thus the resulting state from an agent's action can be any of the reachable states by any sequence of events. In the AUV example, ships can move freely regardless of the movement of the AUV. After the AUV acts, each ship can then move to any of its reachable positions or stay, so the number of non-deterministic alternatives is the number of combinations of reachable positions of the ships. Hence, in each "turn" the number of outcomes of nature might be exponential with respect to the size of the representation of the planning task.

Preliminaries

This section introduces the terminology used in the paper.

Planning against Nature

Planning against nature can be understood as a special case of multi-agent planning (Brafman and Domshlak 2008) in which an intelligent agent that plans towards its goal acts against a random agent (or nature) that acts randomly without a specific purpose (or goal) (Chrpa and Karpas 2024b). To represent the environment, we use Finite Domain Representation (FDR) (Helmert 2009). This is captured in the following definition.

Definition 1. *A* **planning task against nature** *(or* **planning task***, for short) is a tuple $\mathcal{P} = (V, A, E, I, G)$, where V is a set of finite-domain variables, A is a set of actions of the agent, E is a set of actions of nature (or,* **events***), I is a complete variable assignment representing the* **initial state** *and G is a partial variable assignment representing the* **goal***.*

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 the set of all variables in p and $p[v]$ represents the value of v in p. A ***state*** is a complete variable assignment (over V). We say that a (partial) variable assignment q ***holds*** in a (partial) variable assignment p, denoted as $p \models q$, iff $vars(q) \subseteq vars(p)$ and for each $v \in vars(q)$ it is the case that $q[v] = p[v]$.

An ***action*** is a pair $a = (pre(a), eff(a))$, where $pre(a)$ is a partial variable assignment representing a's precondition and $eff(a)$ is a partial variable assignment representing a's effects. We say that an action a is ***applicable*** in state s if and

only if $s \models \mathit{pre}(a)$. The ***result*** of applying a in s, denoted as $\gamma(s, a)$, is a state s' such that for each variable $v \in V$, $s'[v] = \mathit{eff}(a)[v]$ if $v \in vars(\mathit{eff}(a))$ while $s'[v] = s[v]$ otherwise. If a is not applicable in s, $\gamma(s, a)$ is undefined. The notion of action application can be extended to sequences of actions, i.e., $\gamma(s, \langle a_1, \ldots, a_n \rangle) = \gamma(\ldots \gamma(s, a_1) \ldots, a_n)$. ***Events*** are defined analogously.

The next definition of *event reachability* will be useful later. Essentially, it defines δ_E which maps a set of states to the set of states nature can reach from any of them by applying any sequence of events.

Definition 2. *$\delta_E : 2^S \to 2^S$ is defined as $\delta_E(S') = \{s'' \mid s'' = \gamma(s', \langle e_1 \ldots, e_k \rangle), s' \in S',\ k \geq 0,\ e_1, \ldots, e_k \in E\}$.*

We define two different solution concepts for planning against nature: robust plans and linear execution strategies.

Robust Plans

The notion of *robust plans* has been defined by Chrpa, Gemrot, and Pilát (2020) and Chrpa and Karpas (2024a) as a sequence of actions that always achieves the goal regardless of nature. That is, even in the worst case scenario, nature cannot invalidate the precondition of any action, nor the goal. In this sense, a robust plan is similar to a conformant plan (Cimatti and Roveri 2000) or to a strong plan in FOND (Cimatti et al. 2003). We adopt the definition of *robust plans* from (Chrpa and Karpas 2024a).

Definition 3. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task. Let $\pi = \langle a_1, \ldots, a_n \rangle$ ($a_1, \ldots, a_n \in A$) be a sequence of actions. We define sets of states $S^0, S^1, \ldots, S^n$ as follows.*

- $S^0 = \delta_E(\{I\})$
- $S^i = \delta_E(\{\gamma(s, a_i) \mid s \in S^{i-1}\})$ *($1 \leq i \leq n$)*

We say that π is a **robust plan** *for $\mathcal{P}$ if and only if $\forall s \in S^{i-1} : s \models \mathit{pre}(a_i)$ ($1 \leq i \leq n$) and $\forall s \in S^n : s \models G$.*

The above definition indicates that it is problematic if nature can modify the values of variables that might be needed by later actions or the goal. We will call these variables *affected*. We adopt the formal definition of sets of *affected variables* that nature can modify within an action sequence from (Chrpa and Karpas 2024a) as well as the lemma stating that none of the variables of the precondition of the next action (or the goal at the end) can be affected.

Definition 4. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task and $\pi = \langle a_1, \ldots, a_n \rangle$ be a sequence of actions. Let $S^0, \ldots, S^n$ be sets of states as defined in Definition 3. We define $\langle V^0, \ldots, V^n \rangle$ as sets of* **affected variables** *such that $V^i = \{v \mid s_x, s_y \in S^i, s_x[v] \neq s_y[v]\}$ for $0 \leq i \leq n$.*

Lemma 1. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task, $\pi = \langle a_1, \ldots, a_n \rangle$ be a sequence of actions, and $\langle V^0, \ldots, V^n \rangle$ be the sets of affected variables. Then, π is a robust plan if and only if $V^{i-1} \cap vars(\mathit{pre}(a_i)) = \emptyset$ ($1 \leq i \leq n$) and $V^n \cap vars(G) = \emptyset$.*

Linear Execution Strategies

Robust plans are often impossible to find, as they enforce a very strict requirement. Thus, we consider another solution concept, called *linear execution strategies* (Chrpa and Karpas 2024b). In general, an execution strategy is a function that maps states to agent actions, similarly to a policy in an MDP (Mausam and Kolobov 2012). A policy solves a planning task if following the policy is guaranteed to eventually achieve the goal (possibly under some fairness assumptions, as we discuss later).

Unfortunately, general policies are hard to find and are also hard to explain. Therefore, we restrict our attention to *linear* execution strategies, which are specified by a sequence of actions which will always be executed in the specified order, similarly to a classical plan. However, unlike a classical plan, we associate with each action a condition which specifies *when* to execute this action. This condition allows us to capture situations in which the agent has to wait until nature executes a certain event, which may not be explicitly captured in the preconditions of the agent's actions.

To give an example, the AUV might need to wait until the ship has passed the location even though the "move-AUV" action does not explicitly require it. Therefore, Chrpa and Karpas (2024b) defined the notion of *linear execution strategies with waitfors*, in which each action in the sequence is associated with a waitfor precondition (Karpas, Shleyfman, and Tennenholtz 2017; Tuisov, Shleyfman, and Karpas 2024) which provides further constraints on when to execute this action.

We adapt the definition of a (valid) linear execution strategy with waitfor conditions (Chrpa and Karpas 2024b) such that it complies with our terminology and guarantees that the agent eventually achieves its goal if it executes actions whenever their waitfor conditions are met.

One important point is that for linear execution strategies, we assume nature is *fair*. This is a similar assumption to the one underlying the notion of strong cyclic solutions for FOND planning (Cimatti et al. 2003), or to strong fairness in Alternating-time Temporal Logic (Alur, Henzinger, and Kupferman 2002). That is, we assume that if nature can execute some event, there is some chance that it will execute that event. More formally, we assume that if some state s is visited infinitely often during the execution of a policy, and some event e is applicable at state s, then nature will eventually execute e. This fairness assumption is already included in the following definition (requiring that in each step the nature can always reachieve the precondition and the waitfor precondition of the next action, or the goal).

Definition 5. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task, and $\pi = \langle (w(a_1), a_1), \ldots, (w(a_n), a_n) \rangle$ be a sequence of actions ($a_1, \ldots, a_n \in A$), where each action a_i is associated with a waitfor precondition $w(a_i)$ in the form of a logical formula over the set of facts over V. We define sets of states $S^0, S^1, \ldots, S^n$ as follows.*

- $S^0 = \delta_E(\{I\})$
- $S^i = \delta_E(\{\gamma(s, a_i) \mid s \in S^{i-1}, s \models \mathit{pre}(a_i), s \models w(a_i)\})$ *($1 \leq i \leq n$)*

We say that π is a **linear execution strategy with waitfors** *for $\mathcal{P}$ if and only if for all $1 \leq i \leq n$ it holds that $\forall (s \in S^{i-1}) \exists (s' \in \delta_E(\{s\})) : s' \models \mathit{pre}(a_i) \wedge s' \models w(a_i)$, and it also holds that $\forall (s \in S^n) \exists (s' \in \delta_E(\{s\})) : s' \models G$.*

Algorithm 1: A blueprint algorithm for delete-relaxed generation of a robust plan

Require: Planning task $\mathcal{P} = (V, A, E, I, G)$
Ensure: π being a robust plan for $\mathcal{P}$.
1: $f \leftarrow I; \pi \leftarrow \langle\rangle$
2: **while** $f \not\models G$ or $vars(G) \cap \mathit{aff}(f) \neq \emptyset$ **do**
3: $f \leftarrow$ExpandRelaxed(E, f)
4: non-deterministically select $a \in A$ s.t. $f \models \mathit{pre}(a)$ and $vars(\mathit{pre}(a)) \cap \mathit{aff}(f) = \emptyset$
5: **if** no such a can be selected **then**
6: **return** fail
7: **end if**
8: π.append(a)
9: $f \leftarrow f \setminus \{(v, val) \mid v \in vars(\mathit{eff}(a))\} \cup \mathit{eff}(a)$
10: **end while**
11: **return** π

12: **function** EXPANDRELAXED((E, f))
13: $E_{sel} \leftarrow \emptyset$
14: **repeat**
15: $E_{last} \leftarrow E_{sel}$
16: $E_{sel} \leftarrow \{e \mid e \in E, f \models \mathit{pre}(e)\}$
17: $f \leftarrow f \cup \bigcup_{e \in E_{sel}} \mathit{eff}(e)$
18: **until** $E_{sel} = E_{last}$
19: **return** f
20: **end function**

Relaxed Robust Plan Generation

We now turn our attention to plan generation in planning against nature, starting with the simpler case of robust plan generation. Previous work (Chrpa and Karpas 2024a) proposed a blind search approach, which we present below, and later extend with heuristics to better guide the search. Algorithm 1 extends a traditional progressive state-space search classical plan generation routine by using delete-relaxation for over-approximating how nature's events can modify the state of the environment. The ExpandRelaxed function, starting from f, constructs a Relaxed Planning Graph (RPG) by using only nature's events (from E). The resulting set of facts contains all the facts that can be potentially achieved by events (e.g. (Hoffmann and Nebel 2001)). As a result of delete-relaxed reachability by events, we can identify variables that are potentially affected (in fact we overapproximate sets of affected variables) as follows:

$$\mathit{aff}(f) = \{v \mid (v, val), (v, val') \in f, val \neq val'\}$$

Importantly, the remaining variables are guaranteed not to be affected. Then, action applicability as well as whether the goal has been achieved is verified by checking that f entails the precondition of a selected action, or the goal, and that none of the variables in the precondition of the selected action, or the goal, is affected as required by Lemma 1 (Lines 4, and 2, respectively). After the selected action is applied, all conflicting values of the effects' variables are removed from f and the effects are added to f (Line 9).

Algorithm 1 is sound, i.e., the generated plan is robust, but incomplete as it over-approximates the effects of events and thus might prune out some robust plans (Chrpa and Karpas 2024a).

One of the theoretical results of Chrpa and Karpas (2024a) claims that the set of affected variables monotonically grows if the actions have all their effect variables "covered" by their preconditions, and that a variable might become unaffected only if an action that has that variable in its effects but not in its precondition is applied.

Lemma 2. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task, $\pi = \langle a_1, \ldots, a_n \rangle$ be a robust plan for $\mathcal{P}$, and $\langle V^0, \ldots, V^n \rangle$ be the sets of affected variables (as in Definition 4). The following claims hold for all ($1 \leq i \leq n$):*
(a) If $vars(\mathit{eff}(a_i)) \subseteq vars(\mathit{pre}(a_i))$, then $V^{i-1} \subseteq V^i$
(b) $(V^{i-1} \setminus V^i) \subseteq (vars(\mathit{eff}(a_i)) \setminus vars(\mathit{pre}(a_i)))$

The above lemma provides grounds for leveraging delete-relaxed heuristics as we know from classical planning (e.g. (Bonet and Geffner 2001; Hoffmann and Nebel 2001)) for generating robust plans. Note that it is a different use of delete relaxation than over-approximating effects of events as in Algorithm 1.

Definition 6. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task, S be a set of states, and f be a set of facts. We denote $h^*_{\mathcal{P}}(S)$ as the minimum number of actions forming a robust plan for $\mathcal{P}$ from S. We also denote $h^*_{\mathcal{P}_+}(f)$ as the minimum number of actions forming a robust plan using the delete-relaxed method (Algorithm 1) for $\mathcal{P}$ from f.*

From the soundness of Algorithm 1 (Chrpa and Karpas 2024a), we can immediately derive that $h^*_{\mathcal{P}}(S) \leq h^*_{\mathcal{P}_+}(f)$ if $f = \{(v, s[v]) \mid s \in S\}$.

Definition 7. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task and f be a set of facts. We define a **base** of a set of facts f such that all facts referring to affected variables are removed, i.e:*

$$\mathit{base}(f) = f \setminus \{(v, val) \mid v \in \mathit{aff}(f)\}$$

We also define $h^+_G(f)$ as the minimum number of delete-relaxed actions (i.e., actions whose application does not "remove" facts) needed to achieve the goal G from f.

Next, we formally show that $h^+_G(\mathit{base}(f))$ does not overestimate the value of $h^*_{\mathcal{P}_+}(f)$.

Theorem 1. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task. For each f, being a set of facts over V, it holds that $h^+_G(\mathit{base}(f)) \leq h^*_{\mathcal{P}_+}(f)$.*

Proof. Without loss of generality let us assume that $a \in A$ is applicable in a set of facts f, i.e., $f \models \mathit{pre}(a)$ and $vars(\mathit{pre}(a)) \cap \mathit{aff}(f) = \emptyset$. Let $f' = f \setminus \{(v, val) \mid v \in vars(\mathit{eff}(a))\} \cup \mathit{eff}(a)$.

We can derive that $\mathit{aff}(f) \setminus \mathit{aff}(f') = vars(\mathit{eff}(a)) \setminus vars(\mathit{pre}(a))$ because all facts related to variables from $V \setminus vars(\mathit{eff}(a))$ are present in f' if and only if they are present in f, and f contains exactly one fact related to variables from $vars(\mathit{pre}(a))$ (otherwise a would not be applicable). This observation follows the claim of Lemma 2 (albeit for over-appliapproximated sets of affected variables).

Now, we show that a sequence of actions $\langle a_1, \ldots, a_n \rangle$ forming a robust plan for $\mathcal{P}$ from f is a delete-relaxed plan achieving G from $\mathit{base}(f)$. Since $vars(\mathit{pre}(a_1)) \cap \mathit{aff}(f) = \emptyset$ (otherwise a_1 would not be applicable in f), it is the

case that $\mathit{base}(f) \models \mathit{pre}(a_1)$. In the i-th step, it is the case that for each $v \in vars(\mathit{pre}(a_i)) \cap \mathit{aff}(f)$ there exists a_k $(1 \leq k < i)$ such that $v \in vars(\mathit{eff}(a_k)) \setminus vars(\mathit{pre}(a_k))$ according to the above observation. Also, it holds that for each $(v, val) \in \mathit{pre}(a_i)$, $(v, val) \in f$ or there exists a_l $(1 \leq l < i)$ such that $(v, val) \in \mathit{eff}(a_l)$ as nature needs not to apply any event to make any agent's action applicable. From both observations, we can derive that if a_i is applicable, it is the case that $\mathit{base}(f) \bigcup_{j=1}^{i-1} \mathit{eff}(a_j) \models \mathit{pre}(a_i)$. The same applies for G (as it can be treated in the same way as we treat actions).

Hence, we can derive that $h^+_G(\mathit{base}(f)) \leq h^*_{\mathcal{P}^+}(f)$. □

The above theorem guarantees that any classical delete-relaxed heuristic computed in $\mathit{base}(f)$ is *safe* with respect to delete-relaxed robust plan generation. On top of that, if the classical delete-relaxed heuristic is *admissible* in classical planning (such as h_{max} (Bonet and Geffner 2001)), it is also admissible, when computed in $\mathit{base}(f)$, with respect to delete-relaxed robust plan generation (as in Algorithm 1).

Generating Linear Execution Strategies

It can be straightforwardly observed that the existence of a robust plan entails the existence of a linear execution strategy as the latter operates under a weaker assumption that nature acts *fairly*. In particular, a robust plan can be adjusted by (trivial) waitfor preconditions that are the same as the preconditions of the actions in the plan to become a linear execution strategy. It is formalised as follows.

Proposition 1. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task. If $\pi = \langle a_1, \ldots, a_n \rangle$ is a robust plan for $\mathcal{P}$, then $\theta = \langle (w(a_1), a_1), \ldots, (w(a_n), a_n) \rangle$, where $w(a_i) = \bigwedge \mathit{pre}(a_i)$ $(1 \leq i \leq n)$, is a linear execution strategy (with waitfors) for $\mathcal{P}$.*

However, when the strong fairness assumption is in place, focusing on robust plans might be too restrictive. In the next paragraphs, we will show how we can identify a subset of events that nature has to eventually apply, and how we can identify a subset of facts that even if deleted by nature, will eventually be reachieved by nature. These two properties will allow us to generate linear execution strategies for some classes of problems for which a robust plan does not exist.

To simplify reasoning about possible acts of nature, we adopt *Nature Domain Transition Graph (NDTG)* introduced by (Chrpa and Karpas 2024b), which is a special case of a *Domain Transition Graph* (Jonsson and Bäckström 1998), that represents how the values of variables can be changed by nature.

Definition 8. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task. For each $v \in V$, we define the* **Nature Domain Transition Graph (NDTG)** *as a directed graph $\mathcal{G}^v = (D(v), T^v)$, where $D(v)$ is a set of nodes and T^v a set of edges such that for all $x, y \in D(v)$ with $x \neq y$, $(x, y) \in T^v$ iff there exists $e \in E$ such that $\mathit{eff}(e)[v] = y$ and either $\mathit{pre}(e)[v] = x$ or $v \notin vars(\mathit{pre}(e))$. Also, we denote $x \rightarrow_v y$ if there is a path from x to y in $\mathcal{G}^v$, and $x \nrightarrow_v y$ if not.*

We can identify events that if applied in some state (they are applicable in) can never become applicable (i.e., their preconditions are no longer nature-reachable). In other words, such events make irreversible changes to the environment that prohibit nature from making them applicable again. We denote such events as *self disabling*.

Definition 9. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task, $e \in E$ be an event and s be a state such that $s \models \mathit{pre}(e)$. We say that e in* **self-disabling** *in s if and only if e is not event-reachable from $\gamma(s, e)$.*

By analysing NDTGs, we can identify a subset of events

Algorithm 2: Generating Linear Execution Strategy

Require: Planning task $\mathcal{P} = (V, A, E, I, G)$
Ensure: π being a linear execution strategy (with waitfors) for $\mathcal{P}$.

$E_- \leftarrow \{e \mid e \in E, \exists v \in vars(\mathit{eff}(e)) \cap vars(\mathit{pre}(e)) : \mathit{eff}(e)[v] \nrightarrow_v \mathit{pre}(e)[v]\}$
safe $\leftarrow \{(v, val) \mid (v, val)$ is safe$\}$
$f \leftarrow I$; $\theta \leftarrow \langle\rangle$
while not CHECK($G, f,$ *safe*) **do**
  randomly select $e \in E_-$ with $f \models \mathit{pre}(e)$ and CHECK(*pre*(e),ExpandRelaxed($E \setminus \{e\}, f$), *safe*) being true
  if such e exists **then**
    θ.append(e)
    $f \leftarrow f \setminus \{(v, val) \mid v \in vars(\mathit{eff}(e))\} \cup \mathit{eff}(e)$
    continue
  end if
  $f \leftarrow$ExpandRelaxed(E, f)
  non-deterministically select $a \in A$ with CHECK(*pre*$(a), f,$ *safe*) being true
  if no such a can be selected **then**
    return fail
  end if
  θ.append(a)
  $f \leftarrow f \setminus \{(v, val) \mid v \in vars(\mathit{eff}(a))\} \cup \mathit{eff}(a)$
end while
$s, s' \leftarrow I$
$\pi \leftarrow \langle\rangle$
while $\theta \neq \langle\rangle$ **do**
  $x \leftarrow \theta$.popfront
  if $x \in E$ **then**
    $s' \leftarrow \gamma(s', x)$
  else
    $w(x) \leftarrow \bigwedge \mathit{pre}(x) \wedge \bigwedge_{v \in \{v' \mid s[v'] \neq s'[v']\}} \left(\bigvee_{val \in \{val' \mid s'[v] \rightarrow_v val'\}} (v, val) \right)$
    $s \leftarrow s' \leftarrow \gamma(s', x)$
    π.append($(w(x), x)$)
  end if
end while
return π

function CHECK($(q, f,$ *safe*$)$)
  if $f \not\models q$ **then**
    return false
  end if
  if $vars(q) \cap \mathit{aff}(f) = \emptyset$ **then**
    return true
  end if
  if $\{v\} = vars(q) \cap \mathit{aff}(f)$ and $(v, q[v]) \in$ *safe* **then**
    return true
  end if
  return false
end function

that are self-disabling in all states they are applicable in. It is the case if for some variable there is no path in its NDTG from the effect value to the precondition value.

Lemma 3. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task and $e \in E$ be an event. If for some $v \in vars(\mathit{eff}(e))$ it is the case that $\mathit{eff}(e)[v] \nrightarrow_v \mathit{pre}(e)[v]$, then e is self-disabling for all states s such that $s \models \mathit{pre}(e)$.*

Proof. It immediately follows from the observation that if $\mathit{eff}(e)[v] \nrightarrow_v \mathit{pre}(e)[v]$, then there does not exist a sequence of events from E changing the value of v from $\mathit{eff}(e)[v]$ back to $\mathit{pre}(e)[v]$. □

In a different context, we can observe that some facts, even if they are deleted by some events, can always be eventually reachieved. In the context of linear execution strategies, these facts might be temporarily deleted and invalidate the precondition of the next action, yet assuming that nature is fair, these facts will eventually be reachieved again, and the agent can continue executing its actions. We call these facts *safe*.

Definition 10. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task and (v, val) be a fact. Let S be the set of states reachable from I (by both actions and events). We say that (v, val) is* **safe** *if and only if for each $s \in S$ with $s[v] = val$ and each event $e \in E$ applicable in s and deleting (v, val) (i.e., $\gamma(s, e)[v] \neq val$) it holds that there exists $s' \in S$ such that $s'[v] = val$ and s' is event-reachable from $\gamma(s, e)$.*

We can identify some safe facts such that if an event deletes the fact, it always enables another event that can reachieve the fact. The following lemma is a generalised version of Lemma 3 from (Chrpa and Karpas 2024b).

Lemma 4. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task and (v, val) be a fact. If for each $e \in E$ deleting (v, val), there exists $e' \in E$ such that $\mathit{eff}(e) \cup \{(v, \mathit{pre}(e)[v]) \mid v \in vars(\mathit{pre}(e)) \setminus vars(\mathit{eff}(e))\} \models \mathit{pre}(e')$, $\mathit{eff}(e')[v] = val$ and for each $e'' \in E$ deleting some precondition of e' it holds that either $\mathit{eff}(e'')[v] = val$ or $\mathit{pre}(e'')[v] = val$, then (v, val) is safe.*

It should be noted that two (or more) facts that are safe individually might not be safe together (assuming an extension of Definition 10 for sets of facts).

Algorithm 2 summarizes the procedure for generating linear execution strategies (with waitfors). Initially, we identify a subset of self-disabling events E_{-} by leveraging Lemma 3 (Line 1) and a subset of safe facts by leveraging Lemma 4 (Line 2). The CHECK routine tests the conditions of *satisfiability* of a partial variable assignment q in a set of facts f under the consideration of *safe* facts. It has to hold that $f \models q$. Then, either none of the variables from q is affected (in f), or exactly one variable from q is affected but its value (in q) is a safe fact.

Starting in the initial state (i.e., $f \leftarrow I$), we iterate until the goal G is satisfied in f, or if we cannot (nondeterministically) select an action. Firstly, we look for self-disabling events such that their preconditions are *satisfied* in ExpandRelaxed($E \setminus \{e\}, f$), meaning that at most one safe fact from the precondition of e can be invalidated by any event reachable from f other than e (Line 5). If such an event exists, it is applied in f and the main loop continues (Lines 6–10). Otherwise, we proceed to the action selection part (Lines 11–17). Note that such events are guaranteed to eventually occur due to the fairness assumption as nature cannot (ultimately) invalidate their preconditions. The idea is motivated by the notion of *confluence* that, roughly speaking, refers to the property of achieving the same state regardless of how randomly events are triggered (Elahi, Fadnis, and Rintanen 2024). The main difference here is that we consider a subset of events that might occur in a given state.

The action selection part of Algorithm 2 (Lines 11–17) is similar to the one from Algorithm 1 for generating robust plans. The only difference is in considering safe facts.

Waitfor preconditions are extracted from the found sequence of actions and events (θ). We monitor two states, one that is achieved after applying an action in a given step (s) and the other that also takes into account the effects of events (s'). Discrepancies between s and s' provide grounds for determining the waitfor precondition for the next action, besides the precondition of the next action. In particular, the difference of values of some variables in s and s' refers to the occurrence of (self-disabling) events that have to eventually occur before the next action. In other words, we have to wait until the values of variables in s', or any other values that might be reachable from these values (by checking paths in the corresponding NDTGs) are achieved. This is formally described in Line 26.

The following theorem proves the soundness of Algorithm 2. The algorithm is, however, incomplete (e.g. because of over-approximating effects of events).

Theorem 2. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task. If π is an output of Algorithm 2, then π is a linear execution strategy (with waitfors) for $\mathcal{P}$.*

Proof sketch. Initially, we can observe that soundness of the action selection part of Algorithm 2 (Lines 11–17) is implied from soundness of Algorithm 1 (Chrpa and Karpas 2024a), and from Lemma 4 and the strong fairness assumption for nature's event occurrence that together allow for considering at most one safe fact in one step.

The event triggering part of Algorithm 2 (Lines 5–10) selects events that, under the strong fairness assumption, are guaranteed to eventually occur. Also, the number of such events has to be finite (in each step), otherwise the algorithm might not terminate. The latter condition is implied from using the finite set of self-disabling events that, according to Lemma 3, can be triggered at most once in a given step. The eventual event occurrence condition is ensured by checking whether the other events can invalidate the precondition of the event in question. The principle is the same as in the action case.

The waitfor preconditions as extracted by Algorithm 2 reflect the occurrence of "forced" events between the actions. In a nutshell, before applying the next action, the agent has to wait until all the variables modified by events are set to the value resulting from the occurrence of the events, or any possibly subsequent value. □

	1	2	3	4	5	6
AUV						
Time (V)	0.20	0.14	✗	0.17	0.20	0.20
Length (V)	8	8	✗	18	30	28
Time (B)	0.04	0.04	16.97	16.49	870.24	856.24
Length (B)	8	8	20	18	30	26
Time (M)	0.05	0.05	15.60	15.71	-	-
Length (M)	8	8	20	18	-	-
Time (A)	0.04	0.04	0.08	0.08	0.14	0.12
Length (A)	8	8	20	18	34	26
AUV Extended						
Time (V)	✗	✗	0.17	✗	✗	✗
Length (V)	✗	✗	16	✗	✗	✗
Time (B)	0.05	0.09	0.63	0.71	16.53	14.53
Length (B)	14	17	14	22	30	34
Time (M)	0.05	0.11	0.41	0.69	20.60	17.34
Length (M)	14	17	14	22	30	34
Time (A)	0.04	0.06	0.08	0.11	0.19	0.17
Length (A)	14	21	22	28	48	52
ServiceRobot						
Time (V)	0.13	✗	✗	0.13	✗	✗
Length (V)	8	✗	✗	10	✗	✗
Time (B)	0.03	0.04	0.11	0.08	1.49	61.18
Length (B)	8	12	20	8	14	14
Time (M)	0.04	0.05	0.12	0.06	0.90	27.50
Length (M)	8	12	20	8	14	14
Time (A)	0.03	0.04	0.04	0.04	0.04	0.05
Length (A)	8	12	20	8	14	16

Table 1: Results for robust plan generation. Runtimes are in seconds. "Length" represents the number of actions in a robust plan. "✗" denotes instances in which the verification approach failed. "-" denotes instances in which robust plan generation ran out of time. Results of the verification denoted by (V).

	1	2	3	4	5	6
AUV (LES)						
Time (V)	0.10	0.11	0.13	0.14	0.18	NA
Length (V)	16	25	36	35	48	NA
Time (G)	0.05	0.08	0.12	0.17	0.24	NA
Length (G)	16	35	42	43	58	NA
HomeRobot						
Time (V)	0.09	0.12	0.23	0.60	1.57	NA
Length (V)	10	16	22	28	34	NA
Time (G)	0.03	0.04	0.06	0.09	0.14	NA
Length (G)	10	16	22	28	34	NA
AUV (RP)						
Time (V)	✗	✗	✗	✗	✗	✗
Length (V)	✗	✗	✗	✗	✗	✗
Time (G)	0.04	0.05	0.12	0.08	0.19	0.14
Length (G)	8	8	26	18	42	26
AUV - Extended						
Time (V)	✗	0.11	0.12	✗	✗	0.13
Length (V)	✗	15	16	✗	✗	30
Time (G)	0.04	0.06	0.09	0.11	0.15	0.15
Length (G)	8	17	20	28	40	42
AUV - Extended - back and forth						
Time (V)	✗	✗	0.11	✗	✗	✗
Length (V)	✗	✗	16	✗	✗	✗
Time (G)	0.05	0.07	0.09	0.12	0.20	0.17
Length (G)	14	23	20	28	46	40
ServiceRobot						
Time (V)	0.08	✗	✗	✗	✗	✗
Length (V)	8	✗	✗	✗	✗	✗
Time (G)	0.03	0.03	0.04	0.04	0.04	0.05
Length (G)	8	12	20	8	14	16

Table 2: Results for linear execution strategy generation (G). Runtimes are in seconds. "Length" represents the number of actions in a linear execution strategy. "✗" denotes instances in which the verification approach failed. Results for the verification denoted by (V).

Delete-relaxation Heuristics

Since we consider *safe* facts, we might not compute delete-relaxed heuristics for linear execution strategy generation from $base(f)$ as we could while generating robust plans. As can be seen in the CHECK function (see Algorithm 2), the use of safe facts can "bypass" some affected variables. To accommodate safe facts, we have to compute delete-relaxed heuristics from the following set of facts (add facts that are in f and are safe):

$$base(f) \cup (f \cap safe)$$

As we consider safe facts that are true in f for heuristics computation, we can observe that we do not prune any applicable actions (according to the CHECK function). Together with Theorem 2, we can claim that $h^+_G(base(f) \cup (f \cap safe))$ is safe and does not overestimate the minimum number of actions required to form a linear execution strategy.

Experimental Evaluation

The experimental evaluation evaluates i) how the use of delete-relaxed heuristics can improve robust plan generation, and ii) how the introduced method for linear execution strategies performs. The results demonstrate that both robust plan and linear execution strategy generation methods using Greedy Best First Search with the h_{add} heuristic have better coverage than the verification-based approaches while having the runtime in the same order of magnitude.

The verification methods (Chrpa and Karpas 2024b,a) rely on an input plan that they verify. These input plans are generated by LAMA (Richter and Westphal 2010) from underlying classical planning tasks for the robust plan verification (Chrpa and Karpas 2024a), or from classical planning tasks constructed by merging actions and events into (classical) actions and then taking out events from resulting plans (for the linear execution strategy verification (Chrpa and Karpas 2024b)). The time limit for each problem was 900 seconds, and the memory limit was 4GB. The experiments were run on AMD Ryzen 5 5500u 2.1GHz, 16GB RAM, Ubuntu 22.04.[1]

Robust Plan Generation

For the evaluation, we use three benchmark domains – AUV, Extended AUV, and Service Robot – with 6 problem instances each taken from (Chrpa and Karpas 2024a). In the

[1]Our source code and benchmark data are provided at https://github.com/lchrpa/Planning-against-nature-ICAPS-25

AUV domain, we have two AUVs that have to collect resources in a grid-like environment, while there are ships that can move within their corridors in a given direction (each ship has a single column in the grid). If a ship collides with an AUV, the AUV is destroyed. The Extended AUV domain allows AUVs to go into depth, so they can avoid a potential collision with ships. The problem instances consider a single AUV. The Service Robot domain, in a nutshell, describes a task in which two-handed robots have to move objects between different rooms. However, some objects are fragile and can be damaged if the robot carries the fragile object and some other object (in its other hand) or the robot encounters another robot in a (narrow) corridor at the same time. Details about problem instances for all domains can be found at (Chrpa and Karpas 2024a).

Table 1 shows results for robust plan generation. In particular, we consider the delete-relaxed verification approach (V) (Chrpa and Karpas 2024a), delete-relaxed generation approach using Breadth First Search with dead-end detection (B) (Chrpa and Karpas 2024a), delete-relaxed generation approach using A* with the h_{max} heuristic (M), and delete-relaxed generation approach using Greedy Best First Search with the h_{add} heuristic (A).

Unsurprisingly, the coverage of the verification approach is not so good, especially in the AUV Extended and Service Robot domains, because a plan that is initially generated without considering nature's events might not conform to the conditions for being a robust plan (which has also been observed by Chrpa and Karpas (2024a). Both BFS and h_{max} based robust plan generation methods can generate shortest plans (w.r.t Algorithm 1). However, these methods tend to scale poorly with the increasing size of problem instances. In both variants of the AUV domain, h_{max} performs rather worse than BFS because the number of expanded nodes is only about 10–15% less (than for BFS) and thus the overheads related to h_{max} computation outweigh the benefits of expanded nodes reduction. In the Service Robot, h_{max} performs better (on larger instances) because the expanded nodes reduction is more than 25%.

The use of h_{add} led to the best results in terms of runtime, which is in the same order of magnitude as the "generate and verify" approach. The scalability hence can be improved by using a rather simple heuristic approach that can be on the same level as if "candidate" plans are generated by advanced off-the-shelf planners such as LAMA (and verified afterwards). The downside of the h_{add} approach is the generation of suboptimal (delete-relaxed) robust plans, which is more apparent in the Extended AUV domain.

Linear Execution Strategy Generation

For the evaluation, we use two additional benchmark domains – AUV, Home Robot – with 5 problem instances each taken from (Chrpa and Karpas 2024b). The AUV domain is the same as in the robust plan case, however, the problem instances consider only a single AUV. To distinguish, we denote AUV (LES) the benchmarks taken from (Chrpa and Karpas 2024b), and AUV (RP) those taken from (Chrpa and Karpas 2024a). The HomeRobot domain involves a robot that needs to make up rooms. Rooms are connected by a narrow corridor. There are also humans that can move between rooms as well, yet there can be at most one entity (a robot or a human) in the corridor. We have also considered a variant of the Extended AUV domain, referred as "back and forth", in which the ships can move in both directions (the problem instances are the same as in the original Extended AUV domain). Note that both AUV and HomeRobot problem instances do not have a robust plan, since the ship can block or destroy the AUV that has to cross its corridor, and humans might block the narrow corridor forever and prevent the robot from moving between rooms.

The results of the comparison of our introduced Linear Execution Strategy generation method (G) using Greedy Best First Search with the h_{add} heuristic against the verification method introduced by (Chrpa and Karpas 2024b) (V) are shown in Table 2. Noteworthy, the use of h_{add} heuristic might result in generation of longer action sequences (as seen in the robust plan case), yet in the context of linear execution strategies the length (or the cost) of an action sequence might not be necessarily a determining factor of the quality as it might be affected by how long the agent might need to wait until the waitfor preconditions for its actions become satisfied. The question of quality and/or optimality of Linear Execution Strategies is still open.

The results show that our Linear Execution Strategy generation method considerably outperforms the verification one as the latter is only fully successful in the AUV (LES) and Home Robot domains. Our method, on the other hand, generates linear execution strategies (with waitfors) for all considered problem instances. Runtime-wise, our method scales in the same order of magnitude as the verification method that, again, demonstrates that using a rather simple heuristic approach can be on the same level as generating candidate plans by advanced off-the-shelf planners such as LAMA (and verifying them afterwards).

Conclusion

In this paper, we have investigated how to generate robust plans and linear execution strategies in planning against nature. We have amended the recent robust plan generation method (Chrpa and Karpas 2024a) by delete-relaxed heuristics that are widely used in classical planning. Then, we have extended the method to generate linear execution strategies assuming the *fair* nature (i.e., if nature can apply an event, there is a non-zero chance it does). We have empirically shown that both methods can outperform their verification counterparts (Chrpa and Karpas 2024b,a) in terms of coverage since the verification methods rely on an input plan that might not conform to the conditions for being a robust plan or a linear execution strategy. On top of that, our generation methods, when using Greedy Best First Search with the h_{add} heuristic, keep the runtime in the same order of magnitude as the verification methods.

In the future, we plan to investigate how to extract and leverage knowledge about agent/nature interaction in plan generation (e.g. not passing the ship corridor if the ship can return). Also, we plan to investigate how the concept of action reversibility (Morak et al. 2020) can be leveraged to identify groups of safe facts.

Acknowledgements

This research is supported by the Czech Science Foundation (project no. 24-13337L) and by the European Union under the project ROBOPROX (reg. no. CZ.02.01.01/00/22_008/0004590), and by the Israeli Ministry of Science and Technology.

References

Ai-Chang, M.; Bresina, J. L.; Charest, L.; Chase, A.; Hsu, J. C.; Jónsson, A. K.; Kanefsky, B.; Morris, P. H.; Rajan, K.; Yglesias, J.; Chafin, B. G.; Dias, W. C.; and Maldague, P. F. 2004. MAPGEN: Mixed-Initiative Planning and Scheduling for the Mars Exploration Rover Mission. *IEEE Intelligent Systems*, 19(1): 8–12.

Alur, R.; Henzinger, T. A.; and Kupferman, O. 2002. Alternating-time temporal logic. *J. ACM*, 49(5): 672–713.

Blum, A.; and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2): 281–300.

Bonet, B. 2010. Conformant plans and beyond: Principles and complexity. *Artif. Intell.*, 174(3-4): 245–269.

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

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A Robust and Fast Action Selection Mechanism for Planning. In Kuipers, B.; and Webber, B. L., eds., *AAAI*, 714–719. AAAI Press / The MIT Press.

Brafman, R. I.; and Domshlak, C. 2008. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *ICAPS*, 28–35. AAAI Press.

Chrpa, L.; Gemrot, J.; and Pilát, M. 2020. Planning and Acting with Non-Deterministic Events: Navigating between Safe States. In *AAAI*, 9802–9809. AAAI Press.

Chrpa, L.; and Karpas, E. 2024a. On Verifying and Generating Robust Plans for Planning Tasks with Exogenous Events. In *KR*, 273–283.

Chrpa, L.; and Karpas, E. 2024b. On Verifying Linear Execution Strategies in Planning Against Nature. In *ICAPS*, 86–94. AAAI Press.

Chrpa, L.; Pilát, M.; and Med, J. 2021. On Eventual Applicability of Plans in Dynamic Environments with Cyclic Phenomena. In *KR*, 184–193.

Chrpa, L.; Pinto, J.; Ribeiro, M. A.; Py, F.; de Sousa, J. B.; and Rajan, K. 2015. On mixed-initiative planning and control for Autonomous underwater vehicles. In *IROS*, 1685–1690.

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.

Cimatti, A.; and Roveri, M. 2000. Conformant Planning via Symbolic Model Checking. *J. Artif. Intell. Res.*, 13: 305–338.

Cserna, B.; Doyle, W. J.; Ramsdell, J. S.; and Ruml, W. 2018. Avoiding Dead Ends in Real-Time Heuristic Search. In *AAAI*.

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

Elahi, M.; Fadnis, S.; and Rintanen, J. 2024. Termination Properties of Transition Rules for Indirect Effects. In *ICAPS*, 178–186. AAAI Press.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artif. Intell.*, 173(5-6): 503–535.

Hoffmann, J.; and Brafman, R. I. 2006. Conformant planning via heuristic forward search: A new approach. *Artif. Intell.*, 170(6-7): 507–541.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research (JAIR)*, 14: 253–302.

Ingrand, F.; and Ghallab, M. 2017. Deliberation for autonomous robots: A survey. *Artif. Intell.*, 247: 10–44.

Iocchi, L.; Nardi, D.; and Rosati, R. 2000. Planning with sensing, concurrency, and exogenous events: logical framework and implementation. In *KR*, 678–689.

Jonsson, P.; and Bäckström, C. 1998. State-Variable Planning Under Structural Restrictions: Algorithms and Complexity. *Artif. Intell.*, 100(1-2): 125–176.

Karpas, E.; Shleyfman, A.; and Tennenholtz, M. 2017. Automated Verification of Social Law Robustness in STRIPS. In *ICAPS*, 163–171.

Komenda, A.; Novák, P.; and Pechoucek, M. 2014. Domain-independent multi-agent plan repair. *J. Network and Computer Applications*, 37: 76–88.

Little, I.; and Thiebaux, S. 2007. Probabilistic Planning vs Replanning. In *Proceedings of ICAPS Workshop on International Planning Competition: Past, Present and Future*.

Littman, M. L.; Goldsmith, J.; and Mundhenk, M. 1998. The Computational Complexity of Probabilistic Planning. *J. Artif. Intell. Res.*, 9: 1–36.

Mausam; and Kolobov, A. 2012. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Morak, M.; Chrpa, L.; Faber, W.; and Fišer, D. 2020. On the Reversibility of Actions in Planning. In *KR*, 652–661.

Muise, C. J.; McIlraith, S. A.; and Beck, J. C. 2012. Improved Non-Deterministic Planning by Exploiting State Relevance. In *ICAPS*.

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1993. CIRCA: a cooperative intelligent real-time control architecture. *IEEE Trans. Systems, Man, and Cybernetics*, 23(6): 1561–1574.

Palacios, H.; and Geffner, H. 2009. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *J. Artif. Intell. Res.*, 35: 623–675.

Patra, S.; Mason, J.; Ghallab, M.; Nau, D. S.; and Traverso, P. 2021. Deliberative acting, planning and learning with hierarchical operational models. *Artif. Intell.*, 299: 103523.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research (JAIR)*, 39: 127–177.

Tuisov, A.; Shleyfman, A.; and Karpas, E. 2024. Good Things Come to Those Who Wait: The Power of Sensing in Social Laws. In *ECAI*.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A Baseline for Probabilistic Planning. In *ICAPS 2007*, 352–359.