Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning
Main Track

On Verifying and Generating Robust Plans for Planning Tasks with Exogenous Events

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 and acting under the presence of exogenous events brings a number of challenges as events might modify the environment without the consent of the acting agent. Consequently, the agent's plan might get disrupted, agent's goals might no longer be achievable, or, worse, the agent might suffer some damage (e.g. damage to the robot). Although policies, mapping states to appropriate actions to take, can describe, in theory, how the agent should act, they might be difficult to explain and understand for humans in the loop.

In this paper, we describe the concept of *robust plans* that are sequences of actions that can be successfully executed regardless of event occurrence. Robust plans are easier to understand (than policies). We present two methods for verifying whether a sequence of actions is a robust plan, one based on compilation to classical planning, and the other based on leveraging delete-relaxation. We also present a method for generating robust plans that is derived from the "relaxation" verification method. The methods are evaluated on three domains.

1 Introduction

Planning and acting in real-world scenarios (Ingrand and Ghallab 2017), such as planetary rovers (Ai-Chang et al. 2004), or autonomous underwater vehicles (AUVs) (Chrpa et al. 2015), poses a challenge as during plan execution the environment might change by exogenous events that are not under the control of the agent. Events might render the plan invalid, make the agent's goal no longer achievable, or, worse, they might cause damage to the agent. Therefore, during planning the agent has to take into account possible outcomes of event occurrence.

The concept of planning with the presence of exogenous events has been studied for some time (Dean and Wellman 1990; Musliner, Durfee, and Shin 1993; Iocchi, Nardi, and Rosati 2000) and to address planning tasks with events usually requires to reason with the whole state space or a large portion of it. Methods that reason with Markov Decision Process (MDP) models can be leveraged to tackle events (Mausam and Kolobov 2012) and aim to generate a policy with the most promising action in each state. Policies, however, might not be easy to interpret and explain to human operators, for instance, who might oversee the acting agent. Monte-Carlo Tree Search (MCTS) approaches provide similar benefits; however, the success rate tends to drop for problems with dead-ends (Patra et al. 2021).

Fully-observable non-deterministic (FOND) planning considers actions with non-deterministic effects, i.e., when an action is applied, the result of its application might have different outcomes (Cimatti et al. 2003). Exogenous events, however, might not necessarily be triggered by an action the agent had just applied. Also, taking into account that multiple events can occur between the agent's actions, the number of non-deterministic alternatives that might occur after the agent applies an action might be exponential (with respect to the number of events). Alternatively, the specification of the FOND planning problem can be encoded in Linear Temporal Logic (Pnueli 1977), and a controller can be automatically synthesised via FOND planning (Camacho et al. 2018) or using other techniques, e.g., (De Giacomo, Parretti, and Zhu 2023).

The traditional planning-acting-replanning loop (see e.g. (Yoon, Fern, and Givan 2007)) deals with non-determinism by relaxing it, i.e., assuming that it will not have an impact on the plan. If during acting the non-determinism of the environment (e.g. events) disrupts the plan, the agent generates (or tries to generate) a new plan. Such a lazy approach for dealing with non-determinism can not guarantee safety and could be dangerous for the agent as it might get damaged during the process (e.g. a ship might run over an AUV).

Safe planning and acting under the presence of exogenous events (with non-deterministic occurrence) have been recently studied (Chrpa, Gemrot, and Pilát 2020; Chrpa, Pilát, and Med 2021). Both works assume that only a set of independent events[1] can occur between any two of the agent's actions and that event occurrence adheres to the fairness assumption. These assumptions are strong; the agent's actions and events might (in reality) take different amounts of time and fairness might not always be guaranteed.

In this paper, we describe the concept of *robust plans* that are sequences of the agent's actions whose execution is guaranteed to succeed under any circumstances caused by event occurrence. The concept of robust plans is inspired

[1]Independent events do not interfere with each other and can be applied in any order yielding the same resulting state (Blum and Furst 1997)

by conformant planning that, in a nutshell, deals with the problem of generating linear plans in partially-observable 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). Limiting the number of alternatives that a solution (policy or plan) has to consider has a positive impact on explainability as more complex policies (or plans) might be much more difficult to understand for humans in the loop. Limited or branching-bound contingency planning aims at the direction of making solutions simple by limiting the number of alternatives (Meuleau and Smith 2003; Bonet 2010). Although the requirement to generate robust plans might be too strong (we alleviate any branching), robust plans are easier to explain and visualise for planning non-experts, and, on top of that, full situational awareness of the environment is not required during the execution (e.g. an AUV might only need to be aware of its surrounding for successful execution of its actions). In contrast to Chrpa, Gemrot, and Pilát (2020), who also discussed the concept of robust plans, we assume that any finite and valid sequence of events can occur between the agent's actions. This assumption is derived from the work on social laws in multi-agent planning (Karpas, Shleyfman, and Tennenholtz 2017) to account for possibly different durations of actions and events, without explicitly specifying them. Perhaps the most closely related work is our recent one on linear execution strategies (Chrpa and Karpas 2024). These are different from the robust plans we consider here, as a linear execution strategy has the power to wait and decide when to execute the next action according to the state of the world, while the robust plans we consider in this paper do not even control when the next action in the plan will be executed.

Our contribution is twofold. Firstly, we consider verifying whether a sequence of actions (for example, generated by a classical planner) is a robust plan. In practice, such action sequences might be, for example, emergency instructions for which we want to verify that they are robust against given circumstances (e.g. power outage). We show that the problem of deciding whether a sequence of actions is a robust plan is PSPACE-complete and that the problem can be compiled into a problem of plan non-existence of a classical planning task. We propose another verification method that leverages the concept of delete-relaxation, well studied in planning (Bonet and Geffner 2001; Hoffmann and Nebel 2001), that runs in polynomial time at the cost of sacrificing completeness (some action sequences might not be recognised as robust plans). Then, we focus on the problem of robust plan existence and provide an upper bound of its minimum length that can be double exponential. We then propose a method for robust plan generation that (again) leverages the concept of delete-relaxation in the progressive state space search. To evaluate the introduced methods we have specified three domains — Plain AUV, Extended AUV, and ServiceRobot. The results of the experiments are then thoroughly discussed.

2 Preliminaries

Classical planning deals with the problem of finding a sequence of actions that transforms the state of the environment from a given initial state to a state satisfying a given goal condition (Ghallab, Nau, and Traverso 2004).

To represent the environment, we use Finite Domain Representation (FDR) (Helmert 2009). Let V be a set of ***variables*** where each variable $v \in V$ is associated with its domain $D(v)$. An ***assignment*** of a variable $v \in V$ is a pair (v, val), where its value $val \in D(v)$. Hereinafter, an assignment of a variable is also denoted as a ***fact***. A (partial) ***variable assignment*** p over V is a set of assignments of individual variables from V, where $vars(p)$ is a set of all variables in p and $p[v]$ represents a value of v in p. 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 $p \models q$, if and only if $vars(q) \subseteq vars(p)$ 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 $s \models pre(a)$. The ***result*** of applying a in s, denoted as $\gamma(s, a)$, is a state s' such that for each variable $v \in V$, $s'[v] = eff(a)[v]$ if $v \in vars(eff(a))$ 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., $\gamma(s, \langle a_1, \dots, a_n\rangle) = \gamma(\dots\gamma(s, a_1)\dots, a_n)$.

A ***classical planning task*** is a tuple $\mathcal{P} = (V, A, I, G)$, 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*** for $\mathcal{P}$ is a sequence of actions $\pi = \langle a_1, \dots, a_n\rangle$ such that $\gamma(I, \pi) \models G$.

2.1 Exogenous Events

An exogenous ***event*** is a pair $e = (pre(e), eff(e))$, where $pre(e)$ and $eff(e)$ have analogous meaning as in the definition of an action. The notions of applicability and the result of the application of an event are also analogous to the same notions concerning actions. We can hence extend the definition of the transition function γ to take into account the result of the application of an event in a state (if possible).

Semantically, events are not under the control of the agent, in contrast to actions, and can occur randomly if their preconditions are met.

Let $\delta_E : 2^S \rightarrow 2^S$ be a mapping that returns a set of reachable states by any valid sequence of events from E from a given set of states (from S), i.e., $\delta_E(S') = \{s'' \mid s'' = \gamma(s, \langle e_1 \dots, e_k\rangle), s' \in S', k \geq 0, e_1, \dots, e_k \in E\}$.

We can extend the notion of classical planning tasks by taking into account exogenous events. We say that $\mathcal{P} = (V, A, E, I, G)$, where V is a set of variables, A a set of actions, E a set of exogenous events, I a complete variable assignment representing the initial state and G a partial variable assignment representing the goal, is a ***planning task with exogenous events*** (or a ***planning task*** for short).

3 Case Studies

In this section, we provide case studies used to explain and evaluate our approaches.

3.1 AUV Domain - Plain

The AUV domain, initially introduced by Chrpa, Gemrot, and Pilát (2020), simulates AUV operations in which AUVs have to perform sampling of given resources while there might be ships passing by that might endanger AUVs. We have a 4-grid environment such that AUVs and ships can move between neighbouring cells while resources are placed into given cells. Each cell is either free, has the AUV on it, or has the ship on it (the presence of a resource does not interfere with any cell status). The AUV can move to an adjacent cell if the cell is free. The AUV can sample a resource if it is in the same cell. The task for the AUV is to sample the resources and return back to the place of origin. Ships, however, are not controlled by the agent, i.e., ships are controlled by the environment. Ships can move only on some cells from the grid. We consider two "move" events, move-ship-to-free and move-ship-to-auv. Both require that the ship can move to the neighbouring cell and the effect of both events is that the ship moves to that cell. If the ship moves to a free cell, then besides the cell becoming not free for a moment, nothing else happens. However, if the ship moves to the cell with the AUV, then the AUV is destroyed (and can no longer perform any action).

3.2 AUV Domain - Extended

We propose a variant of the AUV domain that differs from the "plain" one by explicitly considering that AUVs can be (and move) either on the surface or underwater. Grid cells can have either shallow or deep water. Each AUV can descend if it is in a cell with deep water, and ascend if nothing is on the surface of a given cell (i.e., the cell is free). Note that a descending AUV will make the cell it is in free while an ascending AUV makes the cell not free. Moving to a neighbouring cell, if the AUV is in depth, is possible only if the cell has deep water. If the AUV is on the surface, moving is the same as in the "plain" domain. Ships also move in the same way as in the "plain" domain, however, they can destroy the AUV only if the AUV is on the surface.

3.3 Service Robot Domain

We propose a Service Robot domain in which we have robots, each with two hands. The robots work in an environment in which rooms are connected by a corridor. Each robot can move from a room to the corridor or from the corridor to a (different) room. There are items (or objects) that need to be delivered from their rooms of origin to the required rooms without being damaged during the transport. Each robot can grab an item by an empty hand and leave the object it holds (by some hand) in a room. Items can be either solid or fragile. We specify two types of events. One event – crack – can be triggered if the robot holds a fragile item (in any hand) and both robot's hands carry some item (i.e., no robot's hand is empty), then the fragile item gets damaged. The other event – interfere – can be triggered if there are two or more robots in the corridor and some robot carries a fragile item, then the fragile item also gets damaged.

4 Robust Plans

The notion of *robust plan* has been defined by Chrpa, Gemrot, and Pilát (2020) as a sequence of actions that always achieves the goal regardless of event occurrence. In contrast to the assumption of Chrpa, Gemrot, and Pilát (2020), which allows only sets of independent events between agent's actions, we assume that an action of the agent can be followed by any applicable sequence of events (including the empty sequence). This assumption follows the reasoning of Karpas, Shleyfman, and Tennenholtz (2017) that without the explicit notion of time in the model, different actions and events might have different execution duration in reality.

Hence, we provide a stronger definition of *robust plans*, i.e., a plan is *robust* if no sequence of events can invalidate the precondition of the following action or invalidate the goal if all the actions were applied.

Definition 1. *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 $\forall s \in S^{i-1} : s \models pre(a_i)$ $(1 \leq i \leq n)$ and $\forall s \in S^n : s \models G$.*

We show that each robust plan for a given planning task is a plan for the underlying classical planning task (without events). That also means that the non-existence of a plan for a classical planning task yields the non-existence of a robust plan for any planning task (with exogenous events) that shares the same set of actions, initial state, and goal. We formalize the claims in the following lemma.

Lemma 1. *Let $\mathcal{P}_c = (V, A, I, G)$ be a classical planning task. For every planning task $\mathcal{P} = (V, A, E, I, G)$, where E is a set of events over V it is the case that (i) a robust plan for $\mathcal{P}$ is a plan for $\mathcal{P}_c$ and (ii) if $\mathcal{P}_c$ is unsolvable (does not have a plan), then $\mathcal{P}$ does not have a robust plan.*

Proof. We can see that each state is reachable by events from itself, i.e., $s \in \delta_E(\{s\})$, in $\mathcal{P}$. Hence $S \subseteq \delta_E(S)$. If $\pi = \langle a_1, \ldots, a_n \rangle$ is a robust plan for $\mathcal{P}$, then we can derive from Definition 1 and the above claim that $I \in \delta_E(\{I\})$, $\gamma(I, a_1) \in \delta_E(\{\gamma(I, a_1)\}), \ldots, \gamma(I, \pi) \in \delta_E(\{\gamma(I, \pi)\})$ yielding that π is a plan for $\mathcal{P}_c$. Note that claim (ii) is a transposition of the implication of claim (i). □

To give an example, in the "plain" AUV domain plans in which none of the AUVs crosses any ship corridor are robust. In the "extended" AUV domain, AUVs can cross ship corridors only if they are deep in the water in order not to compromise plan robustness. In the Service Robot domain, a plan is robust if a robot never carries two items at the same time if at least one of the items is fragile, as well as if two or more robots never meet on the corridor if any of the robots carries a fragile item.

Paraphrasing the meaning of the conditions in Definition 1, we can observe that events can invalidate the precondition of an action by modifying at least one of its variables.

We define sets of *affected variables* that events might modify within an action sequence.

Definition 2. *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 1. We define $V^0, \ldots, V^n$ 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 $0 \leq i \leq n$.*

We can hence show that for a sequence of action to be a robust plan, in each step, none of the variables of the precondition of the next action can be affected.

Lemma 2. *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 $V^0, \ldots, V^n$ be the sets of affected variables. Then, π is a robust plan if and only if $V^{i-1} \cap vars(pre(a_i)) = \emptyset$ ($1 \leq i \leq n$) and $V^n \cap vars(G) = \emptyset$.*

Proof. The claim directly implies from Definitions 1 and 2 as it is the case that there exists $v \in (V^{i-1} \cap vars(pre(a_i))$ if and only if there exists $s \in S^{i-1}$ such that $s[v] \neq pre(a_i)$ yielding $s \not\models pre(a_i)$. $\square$

5 Verification of Robust Plans

The above examples provide an intuition that a plan has to adhere to additional constraints in order to be robust. In a general sense, it might not always be straightforward to intuitively identify such constraints.

To verify that a sequence of actions is a robust plan, we have to check whether the conditions of Definition 1 hold. 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 from A. We define a dummy action a_{n+1} that has the goal in its precondition, i.e., $pre(a_{n+1}) = G$, and that its effects are empty, i.e., $\mathit{eff}(a_{n+1}) = \emptyset$ (meaning that $\mathit{eff}(a_{n+1})$ is an empty variable assignment). Note that introducing the dummy action a_{n+1} unifies the conditions for action preconditions and the goal in Definition 1.

To invalidate the conditions of plan robustness, we have to show that for some i with $1 \leq i \leq n + 1$ there exists a state $s \in \delta_E(S^{i-1})$ such that $s \not\models pre(a_i)$ (as well as $V^{i-1} \cap pre(a_i) \neq \emptyset$). In other words, there have to be sequences of events (from E) in between the actions from π invalidating the precondition of the next action (or the goal). Formally speaking, let $\pi^\perp = \langle e_{0_1}, \ldots, e_{0_k}, a_1, e_{1_1}, \ldots, e_{1_k}, a_2, \ldots, a_{i-1}, e_{(i-1)_1}, \ldots, e_{(i-1)_k} \rangle$ such that $\gamma(I, \pi^\perp)$ is defined and $\gamma(I, \pi^\perp) \not\models pre(a_i)$. On the other hand, if no such $\pi^\perp$ exists (for any action or the goal), then we can conclude that π is a robust plan for $\mathcal{P}$.

The above idea indicates that we can formulate the problem of whether π is (not) a robust plan for $\mathcal{P}$ as a classical planning task whose plans are the $\pi^\perp$ sequences invalidating preconditions of some of the actions (including the dummy action a_{n+1} representing the goal).

Definition 3. *$\mathcal{P} = (V, A, E, I, G)$ be a planning task and $\pi = \langle a_1, \ldots, a_n \rangle$ be a sequence of actions from A and $a_{n+1} = (G, \emptyset)$ be a "dummy goal" action.*

Let goal and applied$_j$ with $0 \leq j \leq n + 1$ be variables (without loss of generality not being present in V), each with a domain $\{\top, \bot\}$.

Let a^*_i with $1 \leq j \leq n + 1$ be actions such that

$$pre(a^*_i) = pre(a_i) \cup \{(applied_{i-1}, \top), (applied_i, \bot)\}$$
$$\mathit{eff}(a^*_i) = \mathit{eff}(a_i) \cup \{(applied_i, \top)\}$$

Then, we define sets of actions A_{goal_i} ($1 \leq i \leq n + 1$) as

$$A_{goal_i} = \bigcup_{v \in vars(pre(a)), val \neq pre(a)[v]} a^{(v,val)}_{goal_i}$$
$$\begin{aligned} pre(a^{(v,val)}_{goal_i}) =&\{(applied_{i-1}, \top), (applied_i, \bot), (goal, \bot), \\ &(v, val)\}, \\ \mathit{eff}(a^{(v,val)}_{goal_i}) =&\{(goal, \top)\} \end{aligned}$$

Now, we create a classical planning task $\mathcal{P}^\pi = (V^\pi, A^\pi, I^\pi, G^\pi)$, called **invalidating task** *for π and $\mathcal{P}$, as follows.*

$$\begin{aligned} V^\pi =&V \cup \bigcup_{j=0}^{n+1} \{applied_j\} \cup \{goal\} \\ A^\pi =&E \cup \{a^*_1, \ldots, a^*_{n+1}\} \cup \bigcup_{j=1}^{n+1} A_{goal_j} \\ I^\pi =&I \cup \{(goal, \bot), (applied_0, \top), \\ &(applied_1, \bot), \ldots, (applied_{n+1}, \bot)\} \\ G^\pi =&\{(goal, \top)\} \end{aligned}$$

We would like to note that the "applied" variables are used to keep the right ordering of applying actions from π and to keep track of which action was applied last. Actions from A_{goal_i} can be triggered if the action a_{i-1} has been applied and some of the precondition of a_i is not met. If that happens, we have found the $\pi^\perp$ plan whose existence invalidates the robustness of π.

Theorem 1. *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 from A. The invalidating task for π and $\mathcal{P}$, $\mathcal{P}^\pi$ (see Definition 3), is unsolvable if and only if π is a robust plan for $\mathcal{P}$.*

Proof. As we elaborate below, a plan for $\mathcal{P}^\pi$ is a sequence of actions (from A^π) in the form of $\pi^\pi = \langle e_{0_1}, \ldots, e_{0_k}, a^*_1, e_{1_1}, \ldots, e_{1_k}, a^*_2, \ldots, a^*_{i-1}, e_{(i-1)_1}, \ldots, e_{(i-1)_k}, a^l_{goal_i} \rangle$, where $1 \leq i \leq n$, $a^l_{goal_i} \in A_{goal_i}$, and $e_x \in E$ (with $x \in \{0_1, \ldots, 0_k, 1_1 \ldots, 1_k \ldots, (i-1)_1, \ldots, (i-1)_k\}$).

Actions from E are applicable if their preconditions are met and modify the environment according to its effects in the same way as events defined in $\mathcal{P}$. Actions $a^*_1, \ldots, a^*_n$ modify the environment in the same way as actions $a_1, \ldots, a_n$ defined in $\mathcal{P}$ and on top of that they set to $\top$ the corresponding variables $applied_i$. Actions $a^*_1, \ldots, a^*_{i-1}$ are applicable if their counterparts defined in $\mathcal{P}$ are applicable and if $applied_{i-1} = \top$ and $applied_i = \bot$ (meaning that

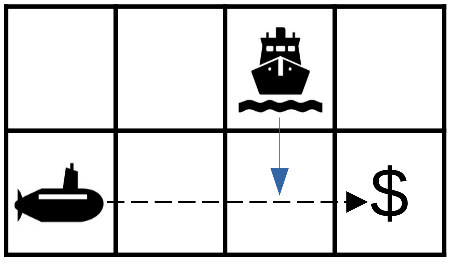

Figure 1: An AUV domain example. The dashed arrow indicates the AUV's plan to collect the resource (the $ symbol). The blue arrow indicates the possible ship movement.

a^*_{i-1} has been applied while a^*_i has not yet been applied). The "goal" action $a^l_{goal_i}$ is applicable if a^*_{i-1} has been applied, some variable v (other than $applied_{i-1}$ and $applied_i$) from $vars(pre(a^*_i))$ is set to a different value than required by a^*_i and $goal = \bot$ (note that it also proves that $v \in V^{i-1}$). Applying $a^l_{goal_i}$ changes $goal$ to $\top$ which is the goal of $\mathcal{P}^\pi$.

Hence, we can observe that $\gamma(I^\Pi, \pi^\pi) \not\models pre(a^*_i)$. By removing the "goal" action $a^l_{goal_i}$ from π^π and replacing actions $a^*_1, \dots, a^*_{i-1}$ by their corresponding counterparts $a_1, \dots, a_{i-1}$, we obtain a sequence $\pi^\bot = \langle e_{0_1}, \dots, e_{0_k}, a_1, e_{1_1}, \dots, e_{1_k}, a_2, \dots, a_{i-1}, e_{(i-1)_1}, \dots, e_{(i-1)_k}\rangle$. Similarly, we can observe that $\gamma(I, \pi^\bot) \not\models pre(a_i)$ (in $\mathcal{P}$). By definition of δ_E, where e.g. $\gamma(I, \langle e_{0_1}, \dots, e_{0_k}\rangle) \in \delta_E(\{I\})$, and the recursive definition of S^{i-1} (in Definition 1), we can observe that $\gamma(I, \pi^\bot) \in \delta_E(S^{i-1})$. That implies that if $\mathcal{P}^\pi$ is solvable, then π is not a robust plan for $\mathcal{P}$.

If none of the "goal" actions defined in $\mathcal{P}^\pi$ can become applicable (and thus $\mathcal{P}^\pi$ is unsolvable), there do not exist sequences of events (interleaving the actions in π) that invalidate the precondition of any action from π (in a given step) as well as the goal of $\mathcal{P}$. Hence, in such a case π is a robust plan for $\mathcal{P}$. □

Example 1. *Let ⟨move(a,l-1-1,l-1-2),move(a,l-1-2,l-1-3),move(a,l-1-3,l-1-4),collect(a,r,l-1-4)⟩ be a sequence of actions in the AUV domain representing that the AUV moves from location l-1-1 to location l-1-4 (via l-1-2 and l-1-3) and collects resource r there (see Figure 1 for illustration). There is a ship s, initially at location l-2-3, that can move to location l-1-3 (e.g., by move-ship-free(s,l-2-3,l-1-3) event). Intuitively, the above action sequence is not a robust plan as the ship can block the AUV at location l-3-1. A solution for the invalidating task can be, in this case, a sequence ⟨move*(a,l-1-1,l-1-2), move-ship-free(s,l-2-3,l-1-3),* $a^{(free(l\text{-}3\text{-}1),\bot)}_{goal_2}$ *⟩.*

Theorem 2. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task and $\pi = \langle a_1, \dots, a_n\rangle$ be a sequence of actions from A. Deciding whether π is a robust plan for $\mathcal{P}$ is PSPACE-complete*

Proof. It is known that deciding plan existence in classical planning is PSPACE-complete (Bylander 1994) and since co-PSPACE=PSPACE, the problem of deciding whether an action sequence is a robust plan is in PSPACE according to Theorem 1 (as we need to decide non-existence of a plan for the corresponding invalidating task).

PSPACE-hardness of the problem can be shown by polynomially reducing the problem of plan (non)existence for a classical planning task $\mathcal{P}^c = (V^c, A^c, I^c, G^c)$ to it. Let $goal$ be a variable with domain $\{\bot, \top\}$ (W.l.o.g. $goal \notin V^c$). Let e_g be an event such that $pre(e_g) = G$ and $eff(e_g) = \{(goal, \top)\}$. Then, we define a planning task $\mathcal{P}'$ as $\mathcal{P}' = (V^c \cup \{goal\}, \emptyset, A^c \cup \{e_g\}, I^c \cup \{(goal, \bot)\}, \{(goal, \bot)\})$ (that is constructed from $\mathcal{P}^c$ in a constant time). We can observe that there does not exist a plan for $\mathcal{P}^c$ if and only if an empty action sequence is a robust plan for $\mathcal{P}'$. Only the event e_g can invalidate $(goal, \bot)$, which is the goal of $\mathcal{P}'$, and it can occur only if there exists a plan for $\mathcal{P}^c$, which satisfies the precondition of e_g.

Hence deciding whether π is a robust plan for $\mathcal{P}$ is PSPACE-complete. □

Algorithm 1 Verifying whether a sequence of actions is a robust plan

```
Require: Planning task $\mathcal{P} = (V, A, E, I, G)$, a sequence of actions $\pi = \langle a_1, \dots, a_n\rangle$
Ensure: Check whether $\pi$ is a robust plan for $\mathcal{P}$.
 1: $a_{n+1} \leftarrow (G, \emptyset)$
 2: $f \leftarrow I$
 3: for $i = 1$ to $n + 1$ do
 4:     $f \leftarrow$ExpandRelaxed$(E, f)$
 5:     if $f \not\models pre(a_i)$ or $\exists v \in vars(pre(a_i)) : |\{val \mid (v, val) \in f\}| \geq 2$ then
 6:         return fail        ▷ Events might invalidate the precondition of $a_i$
 7:     end if
 8:     $f \leftarrow f \setminus \{(v, val) \mid v \in vars(eff(a_i))\} \cup eff(a_i)$
 9: end for
10: return success

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

5.1 Relaxed Verification

Verifying whether a sequence of actions is a robust plan by means of compiling the problem into an invalidating task, which is a classical planning task, is generally intractable as classical planning is PSPACE-complete (Bylander 1994). On top of that, we have to prove that the invalidating task is unsolvable which is believed to be harder (albeit not computational complexity wise) than finding a plan, as unsolvability requires proving that *every* possible sequence of actions is not a valid plan. Furthermore, only one competition focused on unsolvability of planning tasks[2] in contrast to nine editions of the International Planning Competitions[3], which is partly the reason that most planners are geared towards solving planning problems rather than towards proving unsolvability.

To mitigate the computational demands of robust plan verification, we leverage the concept of delete-relaxation, which is well established in (classical) planning (Bonet and Geffner 1999; Hoffmann and Nebel 2001). In particular, delete-relaxation assumes that actions (or events) do not delete variable assignments even though the effects of actions (or events) would have changed the values of these variables.

In the context of robust plan verification, rather than computing exact sequences of events that might compromise the conditions of plan robustness, we track events that might occur in a given step (between the actions) and what impact these events might have on variable assignments. Inspired by Chrpa, Gemrot, and Pilát (2020), we leverage delete-relaxation to make an optimistic assumption about what events might occur in a given step and a pessimistic assumption about how much these events might interfere with actions (and the goal) by any means. If the events do not interfere with actions (and the goal), the plan is robust.

Algorithm 1 summarises the process of robust plan verification by delete-relaxing effects of events. In an intermediate step (the for loop on Lines 3–9), we expand the set of facts f (initially set to I) by facts possibly achieved by events (starting from f). The process is summarised in the ExpandRelaxed routine that constructs a relaxed planning graph (from events) until the fixed point (no new events are applicable in a given step). The expanded set of facts f is used to check whether a current action a_i can be applied regardless of events (Line 5). That means that f must entail $\mathit{pre}(a_i)$ to ensure the applicability of a_i and for each variable contained in $\mathit{pre}(a_i)$, f must not contain more than one value. We can observe that if f contains facts representing multiple values of a single variable, the variable is affected (i.e., some event could have deleted the "right" value). If the check fails, we conclude that we were unable to verify the robustness of the action sequence. Otherwise, we "apply" a_i on f by removing all facts representing the values of variables present in $\mathit{eff}(a_i)$ and adding facts in $\mathit{eff}(a_i)$ (Line 8). If we successfully check all the actions (including the dummy action a_{n+1} representing goal achievement), then we can conclude that the action sequence is a robust plan.

Theorem 3. *Algorithm 1 is sound, i.e., if for a given planning task $\mathcal{P}$ and an action sequence π it returns "success", then π is a robust plan for $\mathcal{P}$.*

Proof. The ExpandRelaxed routine constructs a Relaxed Planning Graph (RPG) from events (for E), starting with the set of facts f, until the fixed point (i.e., no more new events are applicable in a given iteration). As known from the literature (e.g. (Blum and Furst 1997; Hoffmann and Nebel 2001)), the resulting set of facts contains all the facts that can be potentially achieved by events without deleting any fact in the process.

Starting with $f = I$, the RPG expansion at Line 4 expands the set of facts, in a given step of the iteration, by all delete-reachable facts by events. The expanded set of facts f is used to check the applicability of a_i (in the i-th step) as if $f \not\models \mathit{pre}(a_i)$, then a_i is not applicable in the i-th step (and such a sequence might not be robust). If $\exists v \in \mathit{vars}(\mathit{pre}(a_i)) : |\{val \mid (v, val) \in f\}| \geq 2$, then v might be affected as there exists values val and val' ($val \neq val'$) such that $(v, val) \in f$ and $(v, val') \in f$. Wlog. let us assume that $\mathit{pre}(a_i)[v] = val$. However, in order to satisfy $(v, val') \in f$ there is an event $e \in E$ such that $\mathit{eff}(e)[v] = val'$ that might have occurred before a_i. Without delete-relaxation e deletes any other value of v than val', when it occurs (including (v, val)). That might compromise robustness as $\mathit{pre}(a_i)$ might be invalidated. On the other hand, for v such that $|\{val \mid (v, val) \in f\}| = 1$, we can guarantee that v is not affected as no event could change its value. Since it is required that $v \in \mathit{vars}(\mathit{pre}(a_i)) : |\{val \mid (v, val) \in f\}| = 1$ and that $f \models \mathit{pre}(a_i)$, then we can observe that no event can invalidate $\mathit{pre}(a_i)$. That said $V^{i-1} \cap \mathit{pre}(a_i) = \emptyset$ and $f \subseteq \bigcup_{s \in \delta_E(S^{i-1})} s$.

After a_i is applied (in the i-th step), we know that the values of the variables in $\mathit{eff}(a_i)$ will have only the values specified in $\mathit{eff}(a_i)$ (so the other values of these variables are removed from f). That said, $f \subseteq \bigcup_{s \in S^i} s$. □

It can be seen that each step in Algorithm 1 takes linear time with respect to the size of the representation of $\mathcal{P}$, and the algorithm does at most $n + 1$ steps (in the case of successful verification).

Note that delete-relaxing invalidating tasks (see Definition 3) might not yield the same results as Algorithm 1 since the algorithm does not (delete-)relax the effects of actions.

Example 2. *Let us have the same settings as in Example 1. The set of facts f after expanding (by the ExpandRelaxed routine) will contain facts representing that the variables free(l-2-3), free(l-1-3) are both true and false. The verification will fail on action move(a,l-1-2,l-1-3) as the set f contains two values of the variable free(l-1-3) that is required in the action's precondition to be true.*

5.2 Limitations of the Relaxed Approach

Delete-relaxation, as it does not consider "deletion" of facts affected by change of the value of a variable, can overestimate what events might become applicable, what (sets of) facts are reachable, and, in consequence, what variables might become affected. This source of incompleteness might result in false negatives, i.e., not recognising a sequence of actions as a robust plan, as illustrated by the following example.

Example 3. *We extend the AUV domain by introducing* fuel *for ships. For each ship, we define a variable fuel(s) determining its fuel level -* high,low,empty. *Then we can modify the move-ship event by introducing two variants, move-ship-high requiring fuel(s)=high, and move-ship-low requiring fuel(s)=low. After applying the former variant, the*

[2]https://unsolve-ipc.eng.unimelb.edu.au/
[3]http://ipc.icaps-conference.org

fuel level decreases to fuel(s)=low, for the latter variant, the fuel level becomes fuel(s)=empty. That said, a ship s having initially fuel(s)=high can do at most two moves. In such a case, the AUV can safely cross the ship corridor at locations that not are reachable by ship in two or fewer moves since the free variables for those locations will not be affected. Delete-relaxation, however, does not "delete" fuel(s)=high (as well as fuel(s)=low) yielding an incorrect assumption that the ship can reach all the locations on its corridor (and affect all the respective free variables), hence it might fail to "certify" some correct robust plans.

6 Generation of Robust Plans

Given a sequence of actions (generated by a classical planning engine, for instance) the verification approach can certify that the sequence is a robust plan. However, if the verification fails we might need to generate another sequence of actions and try to verify it.

Robust plan existence is a decidable problem. To see that, first note that it is enough to limit the length of a robust plan to $2^{\prod_{v\in V} D(v)}$, i.e., the size of the power set of states, as we show in the following lemma.

Lemma 3. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task. If $\mathcal{P}$ has a robust plan, then there exists a robust plan π such that $|\pi| \leq |2^{\prod_{v\in V} D(v)}|$.*

Proof. Let $\pi' = \langle a_1, \dots, a_{i-1}, a_i, \dots, a_j, \dots, a_n\rangle$ and $S^0, \dots, S^n$ be sets of states defined as in Definition 1. We can observe that if $S^{i-i} \subseteq S^{j-1}$, then $\pi'' = \langle a_1, \dots, a_{i-1}, a_j, \dots a_n\rangle$ is also a robust plan. The claim follows from the fact that if $\forall s' \in S^{j-1} : s' \models pre(a_j)$ (otherwise π cannot be a robust plan), then $\forall s \in S^{i-1} : s \models pre(a_j)$, and $\delta_E(S^{i-1}) \subseteq \delta_E(S^{j-1})$.

Hence, we need only to look for action sequences such that for their sets of states $S^0, \dots, S^n$ (determined as in Definition 1), we require that $S^i \not\subseteq S^j$ for all $0 \leq i < j \leq n$. This implies that $n \leq |2^{\prod_{v\in V} D(v)}|$, i.e., the length of an action sequence for determining the existence of a robust plan is at most the power set of the set of all states (over V). □

With the upper bound for the robust plan length, we can prove decidability of the problem of robust plan existence as well as an upper bound for the complexity (2-EXPSPACE).

Theorem 4. *The problem of robust plan existence is decidable and in 2-EXPSPACE.*

Proof. According to Lemma 3, we can non-deterministically select a sequence of actions that is at most $|2^{\prod_{v\in V} D(v)}|$ long. The sequence can be verified by constructing a corresponding invalidating task and proving its unsolvability (see Theorem 1) that is PSPACE-complete with respect to the length of the plan (see Theorem 2). The length of the action sequence can be double exponential with respect to the size of the representation. Hence the problem of robust plan existence is in 2-EXPSPACE (which means that it is also decidable). □

To give an intuition how to generate a robust plan, we non-deterministically select an action that complies with the applicability condition (as in Definition 1) and the resulting sets of states emerging from the application of a and occurrence of subsequent events are not a superset of some set of states already visited (as it would be unnecessary to explore as indicated in the proof of Lemma 3). We keep (non-deterministically) selecting actions until we satisfy the goal condition (as in Definition 1), then we have a robust plan, or until we are unable to select the next action.

6.1 "Unguarded" Variables in Action Effects

We can observe that only actions with "unguarded" variables in effects (i.e., variables that are in the effects but not in the precondition) can eliminate effects of the events (occurring before such an action is applied) on these variables. This observation is formally summarised in the following lemma.

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

(a) If $vars(eff(a_i)) \subseteq vars(pre(a_i))$, then $V^{i-1} \subseteq V^i$

(b) $(V^{i-1} \setminus V^i) \subseteq (vars(eff(a_i)) \setminus vars(pre(a_i)))$

Proof. Let $S^0, \dots, S^n$ be the sets of states as in Definition 1. For every i such that $1 \leq i \leq n$ we do the following reasoning. We compute a set of states S' such that $S' = \{\gamma(s, a_i) \mid s \in S^{i-1}\}$ (because π is a robust plan a_i is applicable in all states in S^{i-1}). From Lemma 2, it holds that $V^{i-1} \cap vars(pre(a_i)) = \emptyset$ (otherwise π cannot be a robust plan).

We can hence observe that (i) if $v \in V^{i-1}$, then $v \notin vars(pre(a_i))$, (ii) if $v \in vars(eff(a_i))$, then $\forall s \in S' : s[v] = eff(a_i)[v]$, and (iii) for each variable $v' \in V^{i-1} \setminus vars(eff(a_i))$ and for each state $s \in S^{i-1}$, there exists a state $s' \in S'$ such that $s[v'] = s'[v']$.

Regarding claim (a), we can derive that if $v \in V^{i-1}$, then $v \notin vars(eff(a_i))$ because of (i) and $vars(eff(a_i)) \subseteq vars(pre(a_i))$. That together with (iii) gives us that for each $v' \in V^{i-1}$ and for each state $s \in S^{i-1}$, there exists a state $s' \in S'$ such that $s[v'] = s'[v']$. Since $S' \subseteq S^i$ (as we have to consider cases of no event occurrence), we get $V^{i-1} \subseteq V^i$ that proves claim (a).

Regarding claim (b), we can observe that (i) does not make V^{i-1} and $vars(eff(a_i))$ necessarily disjoint, in particular, $V^{i-1} \cap (vars(eff(a_i)) \setminus vars(pre(a_i)))$ might not be empty. Because of (ii), we can derive that for each $v \in V^{i-1} \cap (vars(eff(a_i)) \setminus vars(pre(a_i)))$ it is the case that for each $s \in S'$ it holds that $s[v] = eff(a_i)[v]$. Since $S' \subseteq S^i$, we have no guarantee that V^i contains variables from $vars(eff(a_i)) \setminus vars(pre(a_i))$, proving claim (b). □

The claims of the above lemma include an inability to eliminate the effects of events on variables that are not "unguarded" in the effects of some action. Thus, once a variable becomes affected, no action having it in the precondition can be applied at any point afterwards as well as the goal containing such a variable can no longer be achieved.

Algorithm 2 Delete-relaxed generation of a robust plan

```
Require: Planning task $\mathcal{P} = (V, A, E, I, G)$
Ensure: $\pi$ being a robust plan for $\mathcal{P}$.
 1: $f \leftarrow I$; $\pi \leftarrow \langle\rangle$; $\tau \leftarrow \langle\rangle$
 2: $\nu \leftarrow \bigcup_{a \in A}(vars(eff(a)) \setminus vars(pre(a)))$
 3: while $f \not\models G$ or $\exists v \in vars(G) : |\{val \mid (v, val) \in f\}| \geq 2$ do
 4:     $f \leftarrow$ExpandRelaxed$(E, f)$
 5:     if $\exists f' \in \tau : f' \subseteq f$ or $\exists v \in (vars(G) \setminus \nu) : |\{val \mid (v, val) \in f\}| \geq 2$ then
 6:         return fail
 7:     end if
 8:     non-deterministically select $a \in A$ s.t. $f \models pre(a)$ and $\forall v \in vars(pre(a)) : |\{val \mid (v, val) \in f\}| = 1$
 9:     if no such $a$ can be selected then
10:         return fail
11:     end if
12:     $\pi$.append$(a)$
13:     $\tau$.append$(f)$
14:     $f \leftarrow f \setminus \{(v, val) \mid v \in vars(eff(a))\} \cup eff(a_i)$
15: end while
16: return $\pi$
```

Corollary 1. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task and $\nu = \bigcup_{a \in A}(vars(eff(a)) \setminus vars(pre(a)))$ be the set of "unguarded" variables. Then for every robust plan $\pi = \langle a_1, \ldots, a_n\rangle$ with its sets of affected variables $V^0, \ldots, V^n$, it holds that $vars(pre(a_i)) \cap (\bigcup_{j=0}^{i-1} V^j) \setminus \nu) = \emptyset$ $(1 \leq i \leq n)$ and $vars(G) \cap (\bigcup_{j=0}^{n} V^j \setminus \nu) = \emptyset$.*

Proof. It immediately follows from the claims of Lemma 4 and Lemma 2. □

Lemma 4 and its Corollary 1 give us an intuition that "unguarded" variables in action effects can make the problem of finding a robust plan (or proving its non-existence) harder. The absence of "unguarded" variables in action effects, on the other hand, maintains the monotonic growth of the sets of affected variables during the search for a robust plan that, in consequence, restrict the number of actions that can be possibly applied afterwards.

6.2 Relaxed Robust Plan Generation

As indicated before, robust plans can be constructed non-deterministically according to Definition 1, which is computationally demanding. Alternatively, we might leverage the verification such that we check whether a (partial) action sequence in each step might be a part of the robust plan (by checking unsolvability of the corresponding invalidating task). Such an approach is still impractical due to large computational demands. Leveraging the concept of delete-relaxation, similar to what we did for the verification approach, mitigates computational demands at the cost of losing completeness.

Algorithm 2, in a nutshell, incorporates the delete-relaxed verification from Algorithm 1 into the generic forward-search plan generation method. In principle, Algorithm 2 admits expanding only states that have not yet been visited before (are in τ), do not prohibit goal achievement (according to Corollary 1) and selecting only those actions that can pass the delete-relaxed verification (in a given step).

	1	2	3	4	5	6
AUV Plain						
Time (C)	0.18	0.18	0.23	0.23	0.29	0.29
Succ (C)	✓	✓	✗	✓	✓	✓
Time (R)	0.19	0.14	0.17	0.17	0.19	0.19
Succ (R)	✓	✓	✗	✓	✓	✓
Length	8	8	18	18	30	28
AUV Extended						
Time (C)	0.18	0.20	0.23	0.24	0.27	0.28
Succ (C)	✗	✗	✓	✗	✗	✗
Time (R)	0.14	0.15	0.17	0.17	0.19	0.19
Succ (R)	✗	✗	✓	✗	✗	✗
Length	10	15	16	20	28	30
ServiceRobot						
Time (C)	0.17	0.17	0.18	0.17	0.18	0.18
Succ (C)	✓	✗	✗	✓	✗	✗
Time (R)	0.13	0.13	0.13	0.13	0.14	0.14
Succ (R)	✓	✗	✗	✓	✗	✗
Length	8	10	16	10	12	12

Table 1: Results for robust plan verification. Runtimes are in seconds."Succ" means whether the verification succeeded, and "Length" represents the number of actions in a generated plan. (C) and (R) means verifying by compilation to invalidating task, or by Algorithm 1, respectively.

Theorem 5. *Algorithm 2 is sound, i.e., if for a given planning task $\mathcal{P}$ it returns a sequence of actions π, then π is a robust plan for $\mathcal{P}$.*

Proof. Algorithm 2 admits selecting as the next action (Line 8) only actions satisfying the "verification" condition (Line 5 of Algorithm 1). Similarly, the goal check condition is derived from Algorithm 1. According to Theorem 3, an action sequence generated by Algorithm 2 is a robust plan for $\mathcal{P}$. Although pruning conditions in Line 5 of Algorithm 2 trivially cannot affect its soundness (as they cannot cause generating an incorrect action sequence), the conditions do not "overprune". The latter condition follows Corollary 1 (as once the variable that is not part of any "unguarded" action effect becomes affected it stays affected). The former condition follows the observation that f' admits (at least) the same valid action sequences as f (since $f' \subseteq f$). □

We implemented Algorithm 2 as Breadth-First Search (in principle, Algorithm 2 can accommodate any type of forward-search technique).

7 Experimental Results

The experimental evaluation evaluates the introduced robust plan verification methods as well as the introduced robust plan generation method. For that purpose, we generated 6 problem instances for each domain, introduced in Section 3, such that for each instance there exists a robust plan.

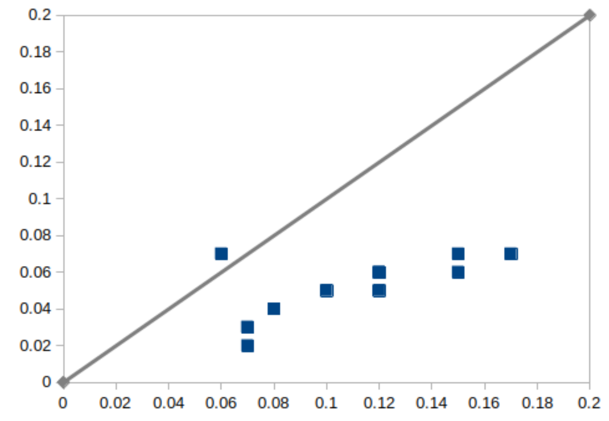

Figure 2: Comparison of runtimes for the compilation verification (x-axis) and for the relaxed verification (y-axis), excluding plan generation times. Data points refer to runtimes per problem instance. Note that some data points overlap as runtimes for both methods were identical in a few cases.

	1	2	3	4	5	6
AUV Plain						
Time	0.05	0.05	17.11	16.20	862.82	844.56
Length	8	8	20	18	30	26
AUV Extended						
Time	0.04	0.09	0.62	0.69	15.90	13.90
Length	14	17	14	22	30	34
ServiceRobot						
Time	0.03	0.04	0.11	0.08	1.50	57.98
Length	8	12	20	8	14	14

Table 2: Results for robust plan generation. Runtimes are in seconds. “Length” represents the number of actions in a robust plan.

In the plain AUV domain, we designed 6 problem instances, each instance considers two AUVs starting (and finishing) in the bottom-left and bottom-right corner respectively. The grid size ranges from 3x3 to 6x6, the number of resources from 2 to 6, and the number of ships from 1 to 2. Each ship has a single column as its corridor (distinct from other ships' corridors). The 6 problem instances we defined for the extended AUV domain differ (number-wise) in considering only one AUV (starting and finishing in the bottom-left corner), the number of ships ranged from 1 to 3, and the deep water is in grid rows (ranging from 1 to 3 rows of deep water, not necessarily adjacent). In the Service Robot domain, we designed 6 problem instances, each instance considers 2 rooms, the number of robots ranges from 1 to 2 (each robot has 2 hands), and the number of items from 2 to 4, where some of the items might be “fragile”.

For generating plans as an input to our robust plan verification methods, we used LAMA (Richter and Westphal 2010) (these plans were generated considering only actions and ignoring events). LAMA was also used for solving invalidating tasks (only if LAMA proved that no solution exists, we consider the verification as successful). 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.[4]

7.1 Results

Table 1 summarises the results of robust plan verification. The runtimes consist of both plan generation (by LAMA) and plan verification. As can be seen, all runtimes are below 0.3 seconds, and most of the runtimes for the compilation verification (the (C) rows) are about $30\% - 50\%$ higher than for the relaxed verification (the (R) rows). Excluding plan generation times (as they are the same for both verification methods) emphasizes the difference between the verification times of both methods as depicted in Figure 2. As can be seen, the relaxation verification is generally faster than the compilation verification (roughly $2-3$ times). In all cases, in which the compiled invalidating task was unsolvable, the unsolvability was proved by unreachability of the goal (so the runtime difference was not that large). This aspect contributed to the alignment of verification successes (and failures) of both compilation and relaxed verification methods. In cases, where the invalidating task was solvable, the (invalidating) plan was shorter than the verified plan.

The verification success rate of generated plans (by LAMA) was high in the plain AUV domain because the instances were designed with ship corridors in the middle columns of the grid, so each AUV can collect resources on “its side” of the grid. Plans usually followed that pattern, except in the third instance, when one of the AUVs collected all resources, meaning that it had to cross ship corridors. On the other hand, most plans generated in the extended AUV domain failed the verification as the AUV was not forced to move deep in the water to prevent interference with ships. The AUV in the plan for the third instance did not have to cross ship corridors and hence such a plan was successfully verified as robust. In the ServiceRobot domain, LAMA-generated plans are verified as robust in two cases. In instance 1, there is no fragile item, and hence every plan is robust. In instance 4, each item is initially in a different room and the plan delivers the items one after another. In the other instances, the plans fail the verification because at some point some robot carries two items, where at least one is fragile (and might break).

Table 2 summarises the results of robust plan generation. Since we used uninformed Breadth-First Search, the runtime heavily depends on the resulting length of found robust plans as well as on the branching factor. The Branching factor is larger in the plain AUV domain (as the instances consider two AUVs) and in the ServiceRobot domain that introduces symmetries as the robot might use either hand to carry an item. Also, the ServiceRobot domain is designed such that undesirable effects of events only affect goal achievability and do not affect the applicability of actions, and hence early dead-end detection helps to prune undesirable parts of the search space. It can be seen that generated robust plans are longer (albeit by a small margin) than LAMA-generated plans whose verification failed.

[4]Our source code and benchmark data are provided at: https://github.com/lchrpa/Robust-Plans-KR24

8 Conclusion

In this paper, we have investigated *robust plans* that are sequences of actions that can always be successfully applied despite the presence of exogenous events that might modify the environment without the consent of the agent. In particular, we considered that any finite and valid sequence of events might occur between the actions of the agent. We have introduced two methods for verifying that a sequence of actions is a robust plan for a given planning task. The first method, which is complete, compiles the verification problem into the problem of plan non-existence of an *invalidating task* (which is a classical planning task). The second method leverages the concept of delete relaxation for pessimistically assuming the impact of events on the applicability of the agent's actions. We have proved that the "relaxed" method is sound, albeit incomplete, and have shown that it runs in polynomial time. Then, we have introduced the method for generating robust plans that adopts the concepts introduced in the "relaxed" method for admitting only action sequences that might form robust plans. Our evaluation on three domains has shown that generating a plan (by a classical planner) and validating it is not very time consuming, however, it might not often be a successful strategy. Generating robust plans provides a remedy for a low success rate of "generate and test" approach albeit at the cost of higher computational demands.

In the future, we plan to develop heuristics to guide the search for robust plans (we plan to leverage the findings of Lemma 4). We also plan to investigate dependencies between actions and events that would help to determine whether a (partial) sequence of actions is admissible as a part of a robust plan. To prevent inadmissible sequences of actions (for being a part of a robust plan) we plan to investigate a conflict-directed approach (de Haro et al. 2022). We also plan to revisit the complexity of robust plan existence, since we believe the result in Theorem 4 can be tightened – we do not believe this is an inherently more difficult problem than LTL synthesis. To do so, we will take inspiration from the complexity results in partially-observable and non-deterministic planning (Rintanen 2004) and conformant planning (Bonet 2010) as those classes of planning are related to our concept.

Acknowledgements

This research is supported by the Czech Science Foundation (project no. 23-05575S) 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.

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

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *Proceedings of ECP*, 360–372.

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

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

Bylander, T. 1994. The computational complexity of propositional strips planning. *Artificial Intelligence* 69:165–204.

Camacho, A.; Baier, J. A.; Muise, C. J.; and McIlraith, S. A. 2018. Finite LTL synthesis as planning. In *ICAPS 2018*, 29–38. AAAI Press.

Chrpa, L., and Karpas, E. 2024. On verifying linear execution strategies in planning against nature. In *ICAPS 2024*. AAAI Press.

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.

Chrpa, L.; Gemrot, J.; and Pilát, M. 2020. Planning and acting with non-deterministic events: Navigating between safe states. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, 9802–9809. AAAI Press.

Chrpa, L.; Pilát, M.; and Med, J. 2021. On eventual applicability of plans in dynamic environments with cyclic phenomena. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021*, 184–193.

Cimatti, A., and Roveri, M. 2000. Conformant planning via symbolic model checking. *J. Artif. Intell. Res.* 13:305–338.

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.

De Giacomo, G.; Parretti, G.; and Zhu, S. 2023. Ltl_f best-effort synthesis in nondeterministic planning domains. In *ECAI 2023*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, 533–540. IOS Press.

de Haro, J. O.; Karpas, E.; Katz, M.; and Toussaint, M. 2022. A conflict-driven interface between symbolic planning and nonlinear constraint solving. *IEEE Robotics Autom. Lett.* 7(4):10518–10525.

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

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning, theory and practice*. Morgan Kaufmann Publishers.

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 2000, Principles of Knowledge Representation and Reasoning Proceedings of the Seventh International Conference*, 678–689.

Karpas, E.; Shleyfman, A.; and Tennenholtz, M. 2017. Automated verification of social law robustness in STRIPS. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017*, 163–171.

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

Meuleau, N., and Smith, D. E. 2003. Optimal limited contingency planning. In *UAI '03, Proceedings of the 19th Conference in Uncertainty in Artificial Intelligence*, 417–426.

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.

Pnueli, A. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, 46–57.

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.

Rintanen, J. 2004. Complexity of planning with partial observability. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, 345–354. AAAI.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A baseline for probabilistic planning. In *ICAPS 2007*, 352–359.