Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning
Main Track

On Eventual Applicability of Plans in Dynamic Environments with Cyclic Phenomena

Lukáš Chrpa[1] , **Martin Pilát**[2] , **Jakub Med**[1]
[1]Faculty of Electrical Engineering, Czech Technical University in Prague
[2]Faculty of Mathematics and Physics, Charles University
chrpaluk@fel.cvut.cz, Martin.Pilat@mff.cuni.cz, medjaku1@fel.cvut.cz

Abstract

Planning and acting in dynamic environments deals with non-deterministic events that might change the state of the environment without consent of the agent. In the worst case, some events might cause the agent to become "trapped" in a dead-end state, which in practice might mean damage or destruction of the agent. Presence of non-deterministic events often considerably increases the number of alternatives that might occur in a single step and hence traditional non-deterministic planning techniques might not scale. In this paper, we address a class of problems where non-deterministic events represent "cyclic phenomena". If they interfere with the agent, they might be dangerous for it (e.g. ships cruising through the area of AUV operations). We present techniques that initially analyse the problem whether it falls within this class by considering the notion of event reversibility and if so, these techniques generate a plan such that encountered unsafe states, in which the "cyclic phenomena" might interfere with the agent, can be eventually crossed without any risk of "falling" into a dead-end state. Our approach is evaluated in the AUV and Perestroika domains.

1 Introduction

Automated planning seeks to find a sequence of actions transforming an environment from a given initial state to a desired goal state (Ghallab, Nau, and Traverso 2004). Planning and acting in real-world scenarios (Ingrand and Ghallab 2017) poses a challenge as plan execution might not go as planned as, for example, exogenous events might occur during the plan execution (e.g. change of the weather).

The concept of exogenous events in planning is not new (Dean and Wellman 1990) and was used in some systems such as Circa (Musliner, Durfee, and Shin 1993). These systems, however, reason with a very small state space. Markov Decision Process (MDP)-based approaches consider events (Mausam and Kolobov 2012) and provide a policy with the most promising action in each state. Monte-Carlo Tree Search (MCTS) approaches provide similar benefits, however, the success rate tends to drop for problems with dead-ends (Patra et al. 2019).

Fully Observable Non-Deterministic (FOND) planning which assumes non-deterministic action effects (Cimatti et al. 2003) can also be leveraged to tackle problems with non-deterministic events (Chrpa, Pilát, and Gemrot 2019). For instance, the well known PRP planner (Muise, McIlraith, and Beck 2012) handles non-determinism by attempting to "close" states from which there does not yet exist a plan. However, if any subset of applicable independent events can occur in a single step, it makes non-deterministic branching exponential with respect to the number of events. Hence such an approach usually does not scale beyond very small problems (Chrpa, Pilát, and Gemrot 2019).

Addressing the problem by relaxing non-deterministic events, i.e., by leveraging classical planning techniques such as FF-replan (Yoon, Fern, and Givan 2007), where if the agent is in an unexpected state it re-plans, is efficient but prone to dead-ends. Reasoning about "dangerous states" improves the success rate of FF-replan-like approaches, yet it does not guarantee avoiding dead-ends (Chrpa, Gemrot, and Pilát 2017; Chrpa, Gemrot, and Pilát 2020).

To tackle the dead-end proneness, one has to enhance classical planning (and re-planning) techniques by a mechanism that prohibits the agent to get into a dead-end state in spite of event occurrence. Chrpa, Gemrot, and Pilát (2020) adapted the notion of *safe states* (Cserna et al. 2018) for planning tasks with non-deterministic events, where a state is safe if no sequence of events can transform it to a dead-end state. If such an event sequence exists, the state is unsafe. The technique proposed by Chrpa, Gemrot, and Pilát (2020) generates a (reference) plan while minimising the number of consecutive unsafe actions since, following the rule of thumb, the shorter the sequence of unsafe states is the higher chance to safely passing it is. The (reference) plan is then followed such that unsafe states can only be traversed by *robust plans* connecting one safe state to another. Robust plans are guaranteed to always succeed despite event occurrence. The main drawback of the technique is that it tries to find robust plans between safe states online which might not always be possible. If there is no way of transiting an unsafe area via a robust plan, the agent gets stuck forever (albeit in a safe state). Hence, the technique of Chrpa, Gemrot, and Pilát (2020) works only on a subclass of planning tasks and it might not be known up front whether the agent has a chance to reach its goal, or whether it gets stuck somewhere.

In this paper, we define a notion of an *eventually applicable reference plan* that, under the fairness assumption that any event can occur if applicable, guarantees that unsafe states might be eventually safely transited without the need of replanning. Specifically, we focus on problems where

non-deterministic events represent "cyclic phenomena" that might be dangerous for the agent. For example, weather might change in areas where a robot performs a surveillance. The robot might be damaged while being outdoors during the rain. Or ships might be cruising in corridors through an area in which an AUV performs scientific observations. If a ship collides with the AUV, the AUV is destroyed. In a nutshell, cyclic phenomena are embodied by sequences of reversible events. However, if the agent interferes with some cyclic phenomenon, then an irreversible event (damaging the agent) might occur. We introduce techniques for analysing the problem and determining reversible and possibly irreversible events (possibly unsafe ones), that is, identifying cyclic phenomena and possible consequences of agent's interference with them. Then, we show under which conditions a state is safe, so safe states do not have to be specified on top of domain and problem specification (as required by the technique of Chrpa, Gemrot, and Pilát (2020)). We also show under which conditions it is possible that a sequence of actions traversing through unsafe states, denoted as an *unsafe bridge*, might eventually become a robust plan and hence safely applicable. Then, finally, we will present a method that generates *eventually applicable reference plans*, where all unsafe bridges can be eventually passed (under the fairness assumption). Hence, our method alleviates two drawbacks of the approach of Chrpa, Gemrot, and Pilát (2020), i.e., the necessity of specifying safe states upfront and lack of guarantee that the agent can eventually reach its goal. Our approach is evaluated on variants of the AUV and Perestroika domains (Chrpa, Gemrot, and Pilát 2020).

2 Preliminaries

This section introduces the terminology used in the paper.

2.1 Classical Planning

Classical planning, in particular, assumes a static, deterministic and fully observable environment; a solution plan amounts to a sequence of actions. 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., $\gamma(s, \langle a_1, \ldots, a_n \rangle) = \gamma(\ldots \gamma(s, a_1) \ldots, a_n)$.

A ***classical planning task*** is a quadruple $\mathcal{P} = (V, A, I, G)$, where V is a set of variables, A a set of actions, I a complete variable assignment representing the ***initial state*** and G a partial variable assignment representing the ***goal***. A ***solution plan*** (for a classical planning task), $\pi = \langle a_1, \ldots, a_n \rangle$, is a sequence of actions such that $\gamma(s, \pi) \models G$ (i.e., an application of π in the initial state results in a goal state). A sequence of states visited during the execution of the solution plan is called ***state trajectory***.

2.2 Non-deterministic Events

Similarly to the definition of an action, a (non-deterministic) ***event*** is a tuple $e = (pre(e), eff(e))$, where $pre(e)$, $eff(e)$ are partial variable assignments representing e's precondition and effects respectively. Applicability of an event in a state as well as the result of an application of an event is defined in the same way as for actions (we can also extend γ to consider event application). In contrast to actions that are executed by agents, events can occur regardless of agent's consent. Technically, an event can (but does not necessarily have to) occur in a state where event's preconditions are met and modify the state of the environment according to event's effects. A ***planning task***, in this case, is a tuple $\mathcal{P} = (V, A, E, I, G)$, where V is a set of variables, A a set of actions, E is a set of events, I a complete variable assignment representing the initial state and G a partial variable assignment representing the goal.

Next, we define the relation of event independence which consequently allows to apply independent events simultaneously as they do not interfere with each other. Independence of actions can be defined analogously.

Definition 1. *We say that events e_i and e_j are* **independent** *if and only if $vars(eff(e_i)) \cap vars(pre(e_j) \cup eff(e_j)) = \emptyset$, $vars(eff(e_j)) \cap vars(pre(e_i) \cup eff(e_i)) = \emptyset$ and for each $v \in vars(pre(e_i)) \cap vars(pre(e_j))$ it holds that $pre(e_i)[v] = pre(e_j)[v]$.*

Now, we define the *event only Domain Transition Graph (eoDTG)* which differs from the standard definition of Domain Transition Graph given by Jonsson and Bäckström (1998) by considering only events as modifiers of the variables.

Definition 2. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task. For each variable $v \in V$ we define an* **event only Domain Transition Graph** *as a labelled directed graph $G_v^E = (D(v), T_v^E)$, where $D(v)$ is a set of nodes and T_v^E set of labelled edges such that for all $x, y \in D(v)$ and $e \in E$, $(x, e, y) \in T_v^E$ iff $eff(e)[v] = y$ and either $pre(e)[v] = x$ or $v \notin vars(pre(e))$.*

The following assumption introduced by Chrpa, Gemrot, and Pilát (2020) simplifies the reasoning by considering single-agent scenario such that actions of the agent and events of the environment alternate like in two-player games (e.g. Chess). The process hence follows the pattern in which the agent can apply an action (not necessarily has to), then the environment can trigger (apply) a set of independent

events (not necessarily has to), and so on. The reason for selecting only events that are independent with each other is to avoid conflicts between preconditions and effects of events that are simultaneously applied. We consider the *fairness assumption*, i.e., each set of independent events applicable in a given step has a chance to be selected and applied. In order words, we assume that events are acts of nature or actions of non-cooperative agents (that are not adversarial).

Assumption 3. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task. Let noop be an action and an event such that pre(noop) = eff(noop) = $\emptyset$. In a current state s, the agent can apply an action $a \in A \cup \{noop\}$ such that a is applicable in s. After that, a (randomly selected) set of independent events $E^i \subseteq E \cup \{noop\}$, applicable in $\gamma(s, a)$, is applied resulting in a state $s' = \gamma(\gamma(s, a), E_i)$ which will become a new current state. We denote s' as a* **successor state** *of s, a.*

The set of all ***resulting states*** of application of an action a in a state s (a is applicable in s) under Assumption 3, denoted as $\delta(s, a)$, is determined as $\delta(s, a) = \{\gamma(\gamma(s, a), E^i) \mid E^i \subseteq E \cup \{noop\}, E^i$ is a set of independent events applicable in $\gamma(s, a)\}$. If a is not applicable in s, then $\delta(s, a) = \perp$. Note that in $\perp$ no action or event (including noop) is applicable. We can generalise the notion of resulting states for sequences of actions, i.e., $\delta(s, \langle a_1, \ldots, a_{n-1}, a_n\rangle) = \bigcup_{s' \in \delta(s, \langle a_1, \ldots, a_{n-1}\rangle)} \delta(s', a_n)$. We say that a state s' is ***reachable*** from a state s with respect to $\mathcal{P}$ and Assumption 3 if and only if there exists a sequence of actions π such that $s' \in \delta(s, \pi)$. Otherwise, we say that s' is ***unreachable*** from s.

We define a function $dist(s, g)$ representing the minimum number of steps required to reach g from s while the agent does no action, i.e., performs noops. Formally, $dist(s, g) = \min\{k \mid s_g \models g, s_g \in \delta(s, noop^k)\}$ ($noop^k$ represents a sequence of k noop actions).

2.3 FOND Planning

Fully Observable Non-Deterministic (FOND) planning assumes a fully observable and static environment but in contrast to classical planning actions have non-deterministic effects (Cimatti et al. 2003; Ghallab, Nau, and Traverso 2016). In particular, the result of application might be one of the sets of effects. Formally a ***non-deterministic action*** is a tuple $a = (pre(a), eff^1(a), \ldots, eff^k(a))$, where $pre(a)$ its precondition, and $eff^1(a), \ldots, eff^k(a)$ its non-deterministic effects, respectively. Action applicability is the same as for classical planning. The result of applying a in s (if possible) is a set of states with each state corresponding to one of a's effects. The FOND planning problem definition is analogous to the classical planning problem definition.

Plans (or solutions) in FOND planning are in form of policies mapping states to actions, i.e., $\pi : S \rightarrow A$. From these policies we can construct a directed graph $G = (S, E)$ whose nodes S are states and edges are defined as $E = \{(s, s') \mid \pi(s) = a, s'$ is the resulting state of applying a in $s\}$. ***Weak plans*** are policies whose graph has a path from the initial state to some goal state. ***Strong (cyclic) plans*** are policies which form acyclic (resp. cyclic) graphs such that from all states, reachable from the initial states, there is a path to some goal state. In plain words, strong plans always guarantee reaching the goal while strong cyclic plans guarantee eventually reaching the goal under the fairness assumption, i.e., each action outcome has a chance to occur (Cimatti et al. 2003).

Analogously to FOND planning, we can define weak plans and strong (cyclic) plans for tasks with non-deterministic events.

2.4 Relations between Actions and Events

Actions as well as events influence each other by setting values of variables that are required by other actions/events, or, in contrast, by changing values of variables such that other actions/events become inapplicable. Inspired by Chapman (1987) who studied relations between actions such as being an "achiever" (i.e., one action sets a variable for another actions) or being a "clobberer" (i.e., an action changes a variable to a different value than is required by another action) we define the notions for both actions and events.

Definition 4. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task. We say that an action/event $x \in A \cup E$ is an* **achiever** *of a fact (v, val) ($v \in V, val \in D(v)$) for an action/event $y \in A \cup E$ if $(v, val) \in eff(x) \cap pre(y)$. We also say that an action/event $x \in A \cup E$ is a* **clobberer** *of a fact (v, val) for an action/event $y \in A \cup E$ if $(v, val') \in eff(x), val \neq val', (v, val) \in pre(y)$.*

2.5 Dead-end and Safe States

In classical planning, dead-end states are those from which no goal state is reachable. With non-deterministic events in play, we define dead-end states as those from which the agent cannot reach the goal by any means (even if events can maximally help). Practically speaking, if the agent (robot) reaches a dead-end state, it might get damaged, or destroyed.

Definition 5. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task. We say that a state s is a* **dead-end state** *if and only if every goal state s_G ($s_G \models G$) is unreachable from s.*

A specific type of events that we do not want to occur are dead-end events, i.e., those whose application might result in a dead-end state.

Definition 6. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task, S be a set of states reachable from I and $e \in E$ be an event. If there exists a state $s \in S$ such that s is not a dead-end state and $s \models pre(e)$, where $\gamma(s, e)$ is a dead-end state, then e is a* **dead-end event**.

From the opposite perspective, we can define safe states that cannot be transformed into dead-end state only by applying events.

Definition 7. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task. We say that a state s is a* **safe state** *if for each sequence of events π_e from E it is the case that no $\gamma(s, \pi_e)$ is a dead-end state.*

Straightforwardly, if no dead-end event can become eventually applicable in a given state, the state is safe (i.e., if no sequence of events applied in a given state can satisfy a precondition of some dead-end event, the state is safe). The

following proposition considers possible unreachability of dead-end event preconditions, which can be understood as a relaxation of the above (as Definition 6 considers a pessimistic assumption of existence of one non-dead-end state turned into a dead-end one by a dead-end event).

Proposition 8. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task and let $E^d \subseteq E$ be the set of dead-end events. If for a state s it is the case that s is not a dead-end state and for each $e' \in E^d$ it holds that $(s \cup \bigcup_{e \in E \setminus E^d} \mathit{eff}(e)) \not\models \mathit{pre}(e')$, then s is a safe state.*

Proof. From the assumption we know that s is not a dead-end state and hence only the eventual application of a dead-end event might transform s to a dead-end state. The assumption also states that a precondition of any dead-end event cannot be satisfied as some of the facts are neither present in s nor can be achieved by any (non-dead-end) event. Hence, s is a safe state. □

2.6 Reference and Robust Plans

As *reference plans* we consider sequences of actions that under the favourable circumstances lead the agent towards its goal. Note that reference plans are also weak plans.

Definition 9. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task. Let ψ be a sequence of actions and events (from A and E, respectively) such that $\gamma(I, \psi) \models G$. We say that a sequence of actions π obtained from ψ by removing events is a* **reference plan** *for $\mathcal{P}$.*

Robust plans are sequences of actions which if applied, the goal is always reached despite event occurrence. Effectively, the key feature of robust plans is that they *evade event effects*. Note that strong plans can be constructed from robust plans.

Definition 10. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task. Let $\pi = \langle a_1, \ldots, a_n \rangle$ be a sequence of actions. If $\forall s \in \delta(I, \pi) : s \models G$, then we say that π is a* **robust plan** *for $\mathcal{P}$.*

Note that computing robust plans according to the above definition is impractical as all the alternatives (resulting states of action application) have to be considered. On the other hand, with a pessimistic assumption how events can modify the environment[1], robust plans can be computed or verified in a similar fashion like classical plans (Chrpa, Gemrot, and Pilát 2020).

Safe execution of a reference plan accounts for iteratively executing a sequence of actions forming a robust plan and finishing in a safe state from the remainder of the reference plan until the goal is reached. If no such a sequence exists, the agent waits (applies noop) until it does. Here, in contrast to Chrpa, Gemrot, and Pilát (2020), the agent does not try to generate an alternative robust plan to the next safe state.

3 Case Studies

This section describes case studies that were adapted from those introduced by Chrpa, Gemrot, and Pilát (2020).

[1]facts possibly achieved by events are considered as true for events while facts possibly invalidated by events are considered as false for actions

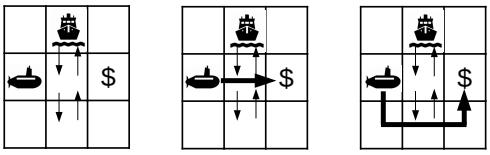

Figure 1: A sample AUV problem (left) and two different reference plans (middle and right).

3.1 The AUV Domain

The AUV domain simulates an AUV operation in which the AUV has to perform sampling of given objects of interest while there might be ships passing by that might endanger the AUV. We have a 4-grid environment, an AUV, ships and several resources. Resources can be found on given cells. Each cell is either free, has the AUV on it, or the ship on it (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 at 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 or might not be present in the area. We consider two variants of problem depending of ship movement.

The *wandering ship* variant considers that each ship moves on specific grid cells. Two "move" events are considered, move-ship-to-free and move-ship-to-auv. Both require that the ship can move to the destination cell. The effect of both events is that the ship moves to the destination 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).

The *cruising ship* variant considers that each ship moves in one direction on a specific corridor. Each ship might be on a cell which is a part of the corridor or be "out of area". A ship can enter the area at its entry cell, can move to adjacent cells of its corridor, and leave the area at its exit cell. The move events are the same as in the wandering ship variant. On top of that, a ship can enter in its entry cell as long as the ship is out of the area, and a ship can leave the area, if it is in its exit cell. The enter event has, analogously, to the move event two variants – one where the entry cell is free and one the entry cell is occupied by the AUV. In the latter case, the AUV gets destroyed.

Figure 1 shows an example of an AUV problem with one wandering ship (for sake of simplicity, the goal is only to sample the resource). The reference plan in the middle, however, does not allow the AUV to safely cross the ship area as the ship either blocks the middle cell or is at most one cell far and thus able to destroy the AUV right after it moves to the middle cell. The reference plan on the right (Fig. 1) allows the AUV to safely cross the ship area such that the AUV might have to wait in the bottom-left corner until the ship is at the top row. Then the ship needs two steps to endanger the AUV at the bottom row, so the AUV can safely pass to the bottom-right corner. Although both reference plans

contain at most one unsafe action in a row, the latter one is "eventually applicable" (under the fairness assumption).

3.2 The "Perestroika" Domain

The Perestroika domain is inspired by the well known Perestroika game (also known as Toppler)[2]. In our domain, an agent has to navigate through a 4-grid of solid and shrinking platforms and collect all resources that can be placed on solid platforms. Solid platforms remain stable, i.e., they neither change its size nor disappear. On the contrary, the shrinking platforms can gradually shrink in shape and then disappear completely. Each shrinking platform can then reappear in its maximum size, which for each platform is constant and between 1 to 5, in contrast to the Perestroika domain introduced by Chrpa, Gemrot, and Pilát (2020), where each shrinking platform had maximum size of 3.

The agent can perform two types of actions. It can move to a neighbouring platform (if it has not disappeared) and/or collect a resource if the resource is on the same platform as the agent. Each shrinking platform is affected by four events. One event changes the shape of the platform by one "step". Two events make the platform disappear if it is in its smallest shape – the difference is whether the platform is empty or the agent is on it. In the former case, the platform only disappears whereas in the latter case it also kills the agent. The last event allows the platform to reappear to its maximum size.

4 Unsafe Bridges

Informally speaking, safe states are those in which the agent has a chance to achieve its goal while not interfering with "phenomena" controlled by events and thus not enabling dead-end events. If the AUV is in a cell into which no ship can enter it is in a safe state. The Perestroika agent if standing on the solid platform is in the safe state as well. Dead-end events, on the other hand, are those, where the ship runs over the AUV and destroys it or where the shrinking platform disappears beneath the agent, who is standing on it.

During plan execution the agent might have to cross an unsafe area in which dead-end events might be enabled. We define a notion of *unsafe bridge* which represents a sequence of actions in which some actions might be achievers for dead-end events while other actions applied later in the sequence are clobberers for these dead-end events. A good example of an unsafe bridge is when the agent crosses one or more shrinking platforms as stepping on a shrinking platform is an achiever of an event "disappear with agent" while leaving the shrinking platform is a clobberer for the "disappear with agent" event.

Definition 11. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task and $E^d \subseteq E$ be the set of dead-end events. Let $\pi = \langle a_1, \ldots, a_n \rangle$ ($a_1, \ldots, a_n \in A$) be an action sequence such that a_1 is an achiever of some fact for some dead-end event. For each $1 \leq i \leq n$ such that a_i is an achiever of some fact f for some dead-end event $e \in E^d$ it holds that there exists a_j, where $i < j \leq n$, such that a_j is a clobberer of f for e. Then, we say π is an* **unsafe bridge**.

[2]https://en.wikipedia.org/wiki/Perestroika_(video_game)

Now, we show that a successful application of an unsafe bridge in a safe state results in another safe state. For example, let an unsafe bridge refer to crossing several shrinking platforms. If the Perestroika agent starts from a solid platform, it finishes on a solid platform if the application of all the actions from the unsafe bridge is successful.

Lemma 12. *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 an unsafe bridge. For each safe state s in which $\langle a_1, \ldots, a_n \rangle$ is applicable, it is the case that $\gamma(s, \pi)$ is also a safe state.*

Proof. As s is a safe state, no dead-end event can eventually become applicable by applying only events. Since the action sequence is an unsafe bridge, it is the case that for each action achieving a fact f required by a dead-end event, there is another action later in the sequence that deletes f. Hence the resulting state is also a safe state. □

The following theorem presents conditions under which an unsafe bridge is a robust plan. In particular, events that might invalidate preconditions of some action from the unsafe bridge have to be far enough, so even in the worst case scenario none of those events can occur before the affected action is applied. Also, dead-end events have to be far enough, so each action that invalidates precondition of some dead-end event has to be applied, even in the worst case, before the dead-end event can occur.

Theorem 13. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task. Let I be a safe state and $\pi = \langle a_1, \ldots, a_n \rangle$ ($a_1, \ldots, a_n \in A$) be an unsafe bridge such that $\gamma(I, \pi) \models G$ and for each $e \in E$ which is not a dead-end event, it holds that $vars(\mathit{eff}(e)) \cap vars(G) = \emptyset$. We define:*

- $c^1(a_i, I) = \min\{dist(I, \mathit{pre}(e) \cap (I \cup \bigcup_{e' \in E} \mathit{eff}(e'))) \mid e \text{ is a clobberer for } a_i\}$
- $c^2(a_j, I) = \min\{dist(I, \mathit{pre}(e') \cap (I \cup \bigcup_{e \in E} \mathit{eff}(e))) \mid a_j \text{ is a clobberer for a dead-end event } e'\}$.

If the following conditions

(1) for each $a_i \in \pi$: $c^1(a_i, I) \geq i - 1$

(2) for each $a_j \in \pi$: $c^2(a_j, I) \geq j$

hold, then π is a robust plan for $\mathcal{P}$.

Proof Sketch. The idea of the proof is in showing that the conditions are sufficient conditions for π being a robust plan from $\mathcal{P}$. The first condition refers to the situation that each action in the sequence (e.g. a_i) has to be applied before events have a chance to invalidate its precondition. The second condition refers to the situation that no dead-end event can become applicable before an action (e.g. a_j) invalidates its precondition. Hence, it is the case that all actions from π are applicable (in the given order), no dead-end event can occur during the execution of π and since no other event can delete a goal atom, the goal will be reached after π is executed. So, π is a robust plan for $\mathcal{P}$. □

Theorem 13, in a nutshell, provides information about whether or not some actions in an unsafe bridge might interfere with events in the worst case scenario. As indicated in the proof sketch, condition (1) accounts for situations in

Algorithm 1 Determining event reversibility from eoDTG

Require: A set of variables V, a set of events E, an event $e_0 \in E$
Ensure: A selected variable v and a partial state s' if e_0 is identified as reversible

1: **function** REVERT(V, E, e_0)
2: Non-deterministically select $v \in \arg\max_{v' \in vars(\mathit{eff}(e_0))} |D(v')|$
3: Construct eoDTG $G_v^E = (D(v), T_v^E)$
4: Let $x, y \in D(v)$ be such that $(x, e_0, y) \in T_v^E$
5: Non-deterministically select a path from y to x in G^E – let $e_1, \ldots, e_n$ be a sequence of labels on the path
6: $s' \leftarrow \{(v', \bot) \mid v' \in V\}$
7: **for** $i \leftarrow 0$ to n **do**
8: **for all** $v' \in \mathit{pre}(e_i)$ with $s'[v'] = \bot$ **do** $s'[v'] \leftarrow \mathit{pre}(e_i)[v']$
9: **if** $s' \not\models \mathit{pre}(e_i)$ **then return** NULL
10: **for all** $v' \in \mathit{eff}(e_i)$ **do** $s'[v'] \leftarrow \mathit{eff}(e_i)[v']$
11: **end for**
12: **if** $s' \not\models \mathit{pre}(e_0)$ **then return** NULL
13: **return** v, s'
14: **end function**

which events have no chance to invalidate precondition of the action (e.g. the shrinking platform cannot disappear before the agent wants to step on it) while condition (2) accounts for situations in which no dead-end event has a chance to become applicable (e.g. no shrinking platform can disappear beneath the agent).

5 Event Reversibility

Action reversibility (or undoability) is the problem of finding a sequence of actions such that it reverses effects of a given action (Daum et al. 2016; Morak et al. 2020). Action reversibility that applies for a restricted set of states S is called S-reversibility (Morak et al. 2020). We can define event S-reversibility analogously.

Definition 14. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning tasks and S be a set of states. We say that an event $e_0 \in E$ is S-***reversible** *if and only if for each $s \in S$, where $s \models \mathit{pre}(e_0)$, there is a sequence of events from E, $\langle e_1, \ldots, e_k \rangle$, such that $s = \gamma(s, \langle e_0, e_1, \ldots, e_k \rangle)$.*

In our domains, we can see that if the AUV does not interfere, the ship movement is reversible (in both wandering and cruising ship variants). Similarly, if the agent does not stand on a shrinking platform, the events modifying the size of their shape are also reversible (as the shrinking platform after it disappears can reappear in its largest shape).

Deciding whether an event is S-reversible in PSPACE-complete (Morak et al. 2020). Event reversibility can be determined by adapting the tool of Daum et al. (2016), which leverages contingent planning.

To provide a tractable method to identify some classes of event S-reversibility we leverage eoDTG. An eoDTG might indicate that some variable value modified an event might no longer be achievable, which compromises reversibility of that events in state having the former value of the variable.

Lemma 15. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task and $e \in E$ be an event. If there exists an eoDTG $G_v^E = (D(v), T_v^E)$ for a variable $v \in V$ such that $(x, e, y) \in T_v^E$ and there is no path from y to x in G_v^E, then e is not S-reversible for such S in which there exists $s \in S$ such that $s \models \mathit{pre}(e)$ and $s[v] = x$.*

Proof. Whenever e is applied, the value of v is changed from x to y, and there does not exist a sequence of events that can change the value of v back to x (otherwise there would be a path from y to x in G_v^E). □

Another way to leverage eoDTGs for identifying a possible event reversibility is to look for cycles in the relevant eoDTGs. Algorithm 1 summarises a method that leverages eoDTGs to identify some reversible events. In particular, for a given event e_0 we non-deterministically select a variable v from $\mathit{eff}(e_0)$ with the largest domain (Line 1). Then, we construct an eoDTG D_v^E (Line 2), identify how e_0 modified the value of v (say from x to y) and after that we non-deterministically select a path in D_v^E reverting the value of v (for y to x) (Lines 3–4). Then, we verify whether a sequence of events on that path is a reverting sequence of e_0 (Lines 6–11). Note that we consider partial states, where variables that are not (or not yet) "touched" by the events are set to a "wildcard" value $\bot$. In particular, in each step the precondition of an event e_i has to be met in the partial state, i.e., each precondition variable has either the same value or the $\bot$ value in the current partial state. If the precondition of e_i is not met, then the algorithm returns NULL (which can be understood as e_0 being deemed as not reversible in any state in which it is applicable). If e_i is applicable in the current partial state, then the partial state is updated by considering effects of e_i. Note that the set of states S, if e_0 is identified as S-reversible, can be constructed from the final partial state s' as $\{s \mid s[v] = s'[v] \vee s'[v] = \bot, v \in V\}$.

6 Potentially Applicable Robust Plans

Whether a sequence of actions can eventually become a robust plan depends on whether there exists a sequence of events which transforms the initial (or current) state, which is safe, into another safe state in which the action sequence becomes a robust plan. In the AUV domain, an AUV might want to cross a few ship cruising corridors. Depending on the current state, the AUV might wait until ships move far enough, so they will not be able to interfere with the AUV after it starts crossing their cruising corridors. Similarly, the Perestroika agent might need to wait until the shrinking platforms it wants to cross become large enough, so there is no chance they will disappear underneath the agent.

Definition 16. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task. We say that a sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ $(a_1, \ldots, a_n \in A)$ is a* **potentially applicable robust plan** *for $\mathcal{P}$ if and only if I is a safe state and there exists a sequence of events $\langle e_1, \ldots, e_k \rangle$ $(e_1, \ldots, e_k \in E)$ such that π is a robust plan for a planning task $\mathcal{P}' = (V, A, E, \gamma(I, \langle e_1, \ldots, e_k \rangle), G)$.*

If the agent does nothing (i.e., applies *noop* actions) in a state s, which is not a dead-end state, and it is the case that

only reversible events can be applied in s or a state reachable from s by applying only events, the agent is in the safe state. Also, the set of reachable states remains the same for states resulting from s by applying events (we consider the fairness assumption).

Proposition 17. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task and s be a state. Let $S^s = \{s' \mid s' \in \delta(s, noop^k), k \geq 0\}$ be a set of states reachable from s by applying only events. If for each $s' \in S^s$ and for each event $e \in E$ it is the case that e is $\{s'\}$-reversible, then (i) $S^{s'} = S^s$ and (ii) if also s is not a dead-end state, then s is a safe state.*

Proof. Let $s' = \gamma(s, e)$ for some event $e \in E$ applicable in s. From the assumption e is $\{s\}$-reversible and thus there exists a sequence of events from E reverting s' back to s. Since s' is reachable by only events from s and s is reachable by only events from s', then $S^{s'} = S^s$. Analogously, we can derive $S^{s'} = S^s$ for s' being the resulting state of application of a sequence of n events in s. From the assumption that s is not a dead-end state and $S^{s'} = S^s$, for any s' reachable from s by applying only events (i.e., $s' \in S^s$), we can immediately get that s is a safe state. □

Even though the agent cannot control event occurrence, if the conditions of Proposition 17 as well as the fairness assumption hold, then a potentially applicable robust plan will stay a potentially applicable robust plan despite any event occurrences. Hence, there is always a chance that a potentially applicable robust plan eventually becomes a robust plan (the agent has to wait until the "right" events occur).

By leveraging Definition 16 and Proposition 17 we define *eventually applicable reference plans* that if "safely" followed the agent eventually reaches its goal. To give an illustration, there are two reference plans in Figure 1 (middle and right). Only the reference plan on the right is eventually applicable as the unsafe bridge in the bottom row (from left to right) is a potentially applicable robust plan since if the ship is in the top row (the AUV might need to wait in the bottom left corner until the ship moves to the top row), the unsafe bridge becomes a robust plan. Note that strong cyclic plans can be constructed from eventually applicable reference plans.

Definition 18. *Let $\mathcal{P} = (V, A, E, I, G)$ be a planning task, where I is safe according to the conditions of Proposition 17. Let $\pi = \langle a_1 \ldots, a_n \rangle$ be a reference plan for $\mathcal{P}$. If for each maximal subsequence of π, $\pi_{i,j} = \langle a_i, \ldots, a_j \rangle$, forming an unsafe bridge it is the case that $\pi_{i,j}$ is a potentially applicable robust plan for $\mathcal{P}' = (V, A, E, \gamma(I, \langle a_1, \ldots, a_{i-1} \rangle), G')$, where $G' = \{(v, val) \mid (v, val) \in pre(a_q) \wedge v \notin vars(eff(a_r)), j < q \leq n, j < r < q,$ or $(v, val) \in G \wedge v \notin vars(eff(a_r)), j < r \leq n\}$, then π is an* **eventually applicable reference plan**.

Note that $\mathcal{P}'$ in the above definition represents a planning task associated with crossing an unsafe bridge. The initial state of $\mathcal{P}'$ takes into consideration only actions preceding the unsafe bridge, since all applicable events alongside the state trajectory are reversible (Proposition 17). Hence, even if some of these events occur, there always exists a sequence of events reverting their effects, so the property of being a potentially applicable robust plan for the unsafe bridge is not affected. The goal of $\mathcal{P}'$, i.e., G', consists of the variable assignments that have to be true after the unsafe bridge is applied in order to successfully execute the rest of the reference plan and achieve the goal (of $\mathcal{P}$).

Algorithm 2 Verifying whether the initial state of a planning task follows the condition for (i) of Proposition 17

Require: A planning task $\mathcal{P} = (V, A, E, I, G)$
Ensure: A set of irreversible events $E^i \subseteq E$, a set of reversible events E^R
$E^i \leftarrow \{e \mid e \in E$ s.t. Lemma 15 holds for $\mathcal{P}$ and $e\}$
$RE = \emptyset$
for all $e \in E \setminus E^i$ **do**
  $v, s^e \leftarrow$ Revert$(V, E \setminus E^i, e)$
  if $v, s^e =$ NULL **then**
    $E^i \leftarrow E^i \cup \{e\}$
  else
    $RE \leftarrow RE \cup \{(e, v, s^e)\}$
  end if
end for
if $\exists e' \in E^i : pre(e') \subseteq (I \cup \bigcup_{e \in E \setminus E^i} eff(e))$ **then**
  return fail
end if
if $\exists (e, v, s^e) \in RE$ and $\exists (e', v', s^{e'}) \in RE$ with $v \notin vars(eff(e'))$ and e' being a clobberer for e **then**
  return fail
end if
$E^r \leftarrow \{e \mid (e, v, s^e) \in RE\}$
for all $v \in \{v \mid (e, v, s^e) \in RE\}$ **do**
  Construct eoDTG $G_v^{E^r} = (D(v), T_v^{E^r})$
  for all $x \in D(v)$ and all paths π_x from $I[v]$ to x in $G_v^{E^r}$ **do**
    if $\exists e'$ s.t. $(x, e', y) \in T_v^{E^r}, (e', v, s^{e'}) \in RE$ and $s^{e'} \not\models \gamma(I, \pi_x)$ **then**
      return fail
    end if
  end for
end for
return E^i, E^r, RE

7 Generating Eventually Applicable Reference Plan

Enumerating a complete set of dead-end events is generally intractable and while some tools such as the "trapper" tool (Lipovetzky, Muise, and Geffner 2016) work in polynomial time, they can identify only a subset of dead-end states, so we cannot directly apply Proposition 8 in practice.

We can, on the other hand, leverage Proposition 17 to identify whether an initial state is a safe state and, consequently, what actions of the agent ensure or compromise safe states. Given a planning task $\mathcal{P} = (V, A, E, I, G)$, Algorithm 2 verifies whether the condition (i) of Proposition 17 is met for the initial state. Initially, as in Lemma 15 we identify events that modify some variable in such a way

Algorithm 3 Generating Eventually Applicable Reference Plan

Require: A planning task $\mathcal{P} = (V, A, E, I, G)$, E^i,E^r
Ensure: Eventually Applicable Robust Plan π
1: $s \leftarrow I$, $\pi \leftarrow \langle\rangle$, $en = \emptyset$
2: **while** $s \not\models G$ **do**
3: **if** $en = \emptyset$ **then**
4: $un \leftarrow 0$
5: non-deterministically select $a \in A \cup E^r$ s.t. $s \models pre(a)$
6: **else**
7: un++
8: non-deterministically select $a \in A$ s.t. $s \models pre(a)$, $c^1(a, s) \geq (un - 1)$ and $c^2(a, s) \geq un$
9: **end if**
10: **if** no a was selected **then return** fail
11: $s = \gamma(s, a)$
12: **if** $a \in A$ **then**
13: $en \leftarrow en \cup \{e_i \mid a$ is an achiever for $e_i\}$
14: $en \leftarrow en \setminus \{e_i \mid a$ is a clobberer for $e_i\}$
15: $\pi \leftarrow \pi.a$
16: **end if**
17: **end while**
18: **return** π

that no other event can restore its previous value (Line 1). Then, the rest of events are sorted according to the result of Algorithm 1 to those possibly irreversible, or those that are reversible (for the latter set we remember the variable which was used to identify reversibility) (Lines 3–10). Then, we check that no possibly irreversible event (from E^i) might become applicable without agent's actions (Line 11) (note that a similar condition is in Proposition 8). The following check rules out a possible interference of two (or more) reversible events that were identified by using different variables (Line 14). Note that such interferences might lead to deadlocks and thus compromise reversibility (e.g. two ships might block each other if they can only move to a location occupied by the other ship). The last check (Lines 18–25) ensures that all reachable reversible events from the initial state are reversible in that state reached from the initial state by applying other (reversible) events.

Potentially unsafe actions are those that are achievers for some irreversible event (that might potentially be a dead-end event) and also those that are clobberers for some reversible event. The latter case, in fact, makes some reversible events irreversible, which might potentially become dead-end events. For the sake of clarity, we will consider scenarios in which the latter case is subsumed by the former one, i.e., actions that are clobberers for reversible events are also achievers for irreversible events and vice versa (it is the case for both AUV and Perestroika domains). Note that considering scenarios in which the latter case does not subsume the former one would require to generalise the definition of unsafe bridge (Def. 11).

To leverage Theorem 13, we need to calculate the value of the $dist$ function. To do so, we exploit eoDTGs $G_v^{E^r} = (D(v), T_v^{E^r})$, constructed for each variable $v \in \{v \mid (e, v, s^e) \in RE\}$ (as in Line 19 of Alg. 2). For a given variable $v \in V$ and its values $x, y \in D(v)$ we define $d_v(x, y)$ as the length of the shortest path from x to y in $G_v^{E^r}$, or 0 if $G_v^{E^r}$ is not defined. Then for a state s and a partial variable assignment g we calculate $dist(s, g)$ as $\max(\{d_v(s[v], g[v]) \mid v \in V\})$. Note that it underestimates the actual value of $dist(s, g)$ as we focus on a projection to a variable v whose values $s[v]$ and $g[v]$ are the most distant.

To generate an eventually applicable reference plan for a planning task $\mathcal{P} = (V, A, E, I, G)$, we run Algorithm 2 and if it does not fail, we run Algorithm 3 (E^i and E^r are passed from Algorithm 2). A standard planning routine is enhanced in two aspects. The minor aspect considers selecting of a reversible event from E^r (Line 5) that can (favourably) change the state of the environment (e.g. reappearing of a shrinking platform), which might be essential for finding a reference plan. The other (major) aspect deals with unsafe bridges. Irreversible events that might possibly become applicable are stored in en (Lines 13 and 14 show how the en set is modified according to the properties of the selected action a). Line 8 ensures that while being on an unsafe bridge the selected action a follows the conditions of Theorem 13. Note that un represents how many actions on the unsafe bridge has been applied so far.

8 Experimental Evaluation

The aim of the experimental evaluation is to compare our approach for finding eventually applicable reference plan (denoted as APP) with the state-of-the-art approach that generates reference plans with minimum number of consecutive unsafe actions (denoted as LIMIT) (Chrpa, Gemrot, and Pilát 2020) and the approach compiling the planning tasks with events to FOND planning tasks (Chrpa, Pilát, and Gemrot 2019). Note that we compiled constraints from Algorithm 3 into the preconditions and effects of affected actions (and events), similarly as the LIMIT approach does. We specified 8 tasks for each domain such that for each task we can find an eventually applicable reference plan. For plan generation, we used the LAMA planner (Richter and Westphal 2010) and for solving the compiled FOND tasks we used the PRP planner (Muise, McIlraith, and Beck 2012; Muise et al. 2016). Time limit for planning was set to 10 minutes. Each reference plan was executed such that only *safely applicable* subsequences of actions were applied, i.e., the (sub)sequences forming robust plans and after they were applied the agent arrived in a safe state (if no irreversible event can become applicable, the agent is in a safe state). If no such subsequence in the current state existed, the agent had to wait (applied noop). The limit for the number of execution steps was set to 1000 to identify situations where the agent is stuck and cannot proceed to the next safe state under any circumstances. For each problem and approach, we simulated plan execution 10 times. The experiments were run on Intel Core-i7 1.8GHz, 16GB RAM.[3]

[3]Our implementation and benchmarks problems can be found here: https://gitlab.com/ctu-fee-fras/public/server-client-simulator-kr-2021

	AUV										
	Structure			APP			LIMIT			FOND	
#	N	#S	#R	Pt	RPL	St	Pt	Pl	St	Pt	St
1	3	1W	1	130	9	17	FAIL - Act			18711	26
2	3	1C	1	129	7	11	400	7	11	13827	31
3	5	3C	3	200	29	136	2210	21	144	FAIL - Plan	
4	10	5C	5	1953	41	61	FAIL - Plan			FAIL - Plan	
5	15	7C	7	59073	93	117	FAIL - Plan			FAIL - Plan	
6	5	2W	2	152	16	29	545	16	30	FAIL - Plan	
7	10	6W	5	909	87	154	229401	41	116	FAIL - Plan	
8	15	12W	7	1231	85	114	FAIL - Act			FAIL - Plan	

Table 1: Results of the comparison between our approach (APP), the approach limiting the number of consecutive unsafe actions (LIMIT) and the (FOND) approach. Structure of the problem consists of N – size of the square grid, $\#R$ – number of resources, and $\#S$ – number of (C)ruising or (W)andering ships. Pt – runtime in ms for generating Reference Plan (including preprocessing), RPL – reference plan length, and St – average number of execution steps.

	Perestroika								
	Structure			APP			LIMIT		
#	N	#S	#R	Pt	RPL	St	Pt	Pl	St
1	5	16E	5	181	17	29	747	17	25
2	5	16E	8	778	32	52	FAIL - Act		
3	9	56E	14	7179	66	118	FAIL - Act		
4	9	56E	24	7709	114	298	FAIL - Act		
5	5	10R	9	161	33	39	FAIL - Act		
6	5	19R	5	1000	17	24	FAIL - Plan		
7	9	40R	22	3257	74	102	FAIL - Act		
8	9	37R	43	7603	167	223	FAIL - Act		

Table 2: Results of the comparison between APP and LIMIT (see Table 1 for details). Structure of the problem consists of N – size of the square grid, $\#R$ – number of resources, and $\#S$ – number of (E)venly (on all even rows and columns) or (R)andomly distributed shrinking platforms. Pt, RPL and St are the same as in Table 1

Tables 1 and 2 show the results of the comparison for the AUV and Perestroika domains, respectively. Note that the FOND approach did not solve any problem in Perestroika, so we excluded its results from Table 2. It can be seen that APP was able in most cases to generate a reference plan within a few seconds (except AUV-5) while LIMIT was generally slower and even failed three times to generate a reference plan in the 10 minutes time limit (see "FAIL - plan" records in Tables 1 and 2). The reason is that LIMIT tries to generate a robust plan first, and then tries to generate reference plans by incrementally increasing the number of allowed consecutive unsafe actions (starting with 1) (Chrpa, Gemrot, and Pilát 2020). Hence, plan generation might fail several times before a reference plan can be generated, which consumes a lot of computational time. Also, LIMIT does not take into account whether some unsafe states are passable or not, which might cause that the agent gets stuck (see "FAIL - act" records in Tables 1 and 2). In Perestroika, it happened six times, since reference plans often went via shrinking platforms with maximum size of 1, which, however, might disappear right after the agent steps on them. In contrary, APP is able to avoid these platforms. The FOND approach managed to solve only the two simplest problems in the AUV domain. Also, the number of execution steps was considerably higher than for both APP and LIMIT.

Note that the difference between reference plan length and the number of execution steps refer to the number of times the agent had to wait (apply noop). The difference is rather large in the AUV-3 problem with unsafe bridges consisting of 5 actions and hence it might take a while before ships are "appropriatelly" arranged so the agent can safely pass.

9 Conclusion

Planning and acting in environments with non-deterministic events poses a challenge, specifically in situations in which events can cause dead-ends that in practice might mean damage or destruction of the agent (or robot). We focused on classes of problems where events represent "cyclic phenomena" that might endanger the agent (e.g. ships passing through the area an AUV performs observations might destroy the AUV if they collide with it). In such problem classes, events can be divided to reversible and irreversible such that the latter ones are considered as "dangerous" as they might possibly cause dead-ends. Also, if irreversible events cannot become applicable in a given state, we know that we are in a safe state (if the goal is still reachable). We introduced methods for dividing events to reversible and irreversible and for identifying whether an initial state is possibly safe. Then, we introduced the concept of "unsafe bridges" that account for sequences of actions crossing unsafe states, in which irreversible events might become applicable. We have shown (in Theorem 13) that unsafe bridges can be safely passed if "problematic" events are far enough. Based on this observation, we proposed a method that generate "eventually applicable" reference plans in which all unsafe bridges can be eventually passed (assuming the fairness assumption). The experimental results have shown that our approach outperforms the state-of-the-art one (Chrpa, Gemrot, and Pilát 2020) both in plan generation time and success rate in plan execution.

In future, we plan to focus on limitations of the method – that sometimes safe states cannot be connected by (possibly applicable) robust plans while there exists a strong (cyclic) plan that can do so. Also, we would like to focus on multi-agent and temporal settings.

Acknowledgements

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

References

Chapman, D. 1987. Planning for conjunctive goals. *Artif. Intell.* 32(3):333–377.

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

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-20)*. in press.

Chrpa, L.; Pilát, M.; and Gemrot, J. 2019. Compiling planning problems with non-deterministic events into FOND planning. In *Proceedings of the RCRA International Workshop*.

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.

Cserna, B.; Doyle, W. J.; Ramsdell, J. S.; and Ruml, W. 2018. Avoiding dead ends in real-time heuristic search. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*.

Daum, J.; Torralba, Á.; Hoffmann, J.; Haslum, P.; and Weber, I. 2016. Practical undoability checking via contingent planning. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016.*, 106–114.

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.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

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

Jonsson, P., and Bäckström, C. 1998. State-variable planning under structural restrictions: Algorithms and complexity. *Artif. Intell.* 100(1-2):125–176.

Lipovetzky, N.; Muise, C. J.; and Geffner, H. 2016. Traps, invariants, and dead-ends. In *ICAPS 2016*, 211–215.

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 *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020)*, 652–661.

Muise, C. J.; Felli, P.; Miller, T.; Pearce, A. R.; and Sonenberg, L. 2016. Planning for a single agent in a multi-agent environment using FOND. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 3206–3212.

Muise, C. J.; McIlraith, S. A.; and Beck, J. C. 2012. Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*.

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.

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

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.

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