Planning and Acting with Non-Deterministic Events: Navigating between Safe States

Lukáš Chrpa
Faculty of Electrical Engineering
Czech Technical University in Prague

Jakub Gemrot, Martin Pilát
Faculty of Mathematics and Physics
Charles University

Abstract

Automated Planning addresses the problem of finding a sequence of actions, a plan, transforming the environment from its initial state to some goal state. In real-world environments, exogenous events might occur and might modify the environment without agent's consent. Besides disrupting agent's plan, events might hinder agent's pursuit towards its goals and even cause damage (e.g. destroying the robot).

In this paper, we leverage the notion of *Safe States* in dynamic environments under presence of non-deterministic exogenous events that might eventually cause dead-ends (e.g. "damage" the agent) if the agent is not careful while executing its plan. We introduce a technique for generating plans that constrains the number of consecutive "unsafe" actions in a plan and a technique for generating "robust" plans that effectively evade event effects. Combination of both approaches plans and executes robust plans between safe states. We empirically show that such an approach effectively navigates the agent towards its goals in spite of presence of dead-ends.

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. It is often known what can happen and under which circumstances (e.g. failure to establish communication, or a blocked passage). Such information can be encoded into a domain model as events, so the system that plans and executes the plan can reason with them.

The concept of 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) while providing the most promising action in a given 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 assumes non-deterministic action effects. 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. PRP has been adapted to deal with multi-agent planning problems (Muise et al. 2016) and, in a similar spirit, it can be extended to deal with problems with events. However, considering that any subset of applicable independent events can occur in a single step makes non-deterministic branching exponential with respect to the number of events.

The success of FF-replan (Yoon, Fern, and Givan 2007) in the International Planning Competition 2006 (it was an unofficial winner of the probabilistic track) indicates that the problem can be addressed by the Planning/Execution/Replanning (PER) approach (Komenda, Novák, and Pechoucek 2014) that interleaves planning, plan execution and re-planning if the agent is in an unexpected state or cannot execute the following action. However, in domains with dead-ends such an approach might not be effective (and might even be dangerous). A recent work concerning reasoning about "dangerous states" (those close to dead-ends) has improved the success rate of the PER approach albeit not providing guarantees of dead-end avoidance (Chrpa, Gemrot, and Pilát 2017).

In this paper, we focus on single-agent planning in fully observable environment with deterministic action effects and non-deterministic exogenous events where we, roughly speaking, adapt the PER approach to reason with "safe states" (Cserna et al. 2018) in order to avoid reaching dead-end states. A *safe state*, the notion adopted from Cserna et al. (2018) who use it for online planning, stands for a state that cannot be transformed into a dead-end state by events only. In contrast to "dangerous states" reasoning (Chrpa, Gemrot, and Pilát 2017), we do not quantify "danger", our idea is to allow replanning episodes only in safe states and executing *safely applicable* sequences of actions, that is, after such a sequence is executed (and it is always possible despite event occurrence) the agent is in the safe state. To find safely applicable sequences of actions, or *robust plans* in other words, inspired by Palacios and Geffner (2009), we present a compilation of the problem of finding robust

plans in the presence of non-deterministic events into classical planning problems. On top of that, to generate an initial plan, we propose a compilation of the planning problem that limits the number of consecutive "unsafe" actions in a solution plan. Hence the safe states are close to each other and much more likely to be connected by robust plans. We empirically show that our safe state reasoning technique effectively and safely navigates the agent towards its goals contrary to the traditional PER approach or the dangerous state reasoning (Chrpa, Gemrot, and Pilát 2017).

Preliminaries

Classical planning, in particular, assumes a static, deterministic and fully observable environment; a solution plan amounts to a sequence of actions. Technically, a ***classical planning domain model*** is a tuple $\mathcal{D} = (L, A)$, where L is the set of propositional atoms used to describe the ***state*** of the environment, i.e., a set of propositions from L that are true, and A is the set of actions over L. An ***action*** is a tuple $a = (\mathit{pre}(a), \mathit{del}(a), \mathit{add}(a))$, where $\mathit{pre}(a), \mathit{del}(a)$ and $\mathit{add}(a)$ are sets of atoms from L representing a's precondition, delete, and add effects, respectively. We assume that $\mathit{del}(a) \cap \mathit{add}(a) = \emptyset$. An action a is *applicable* (or executable) in a state s if and only if $\mathit{pre}(a) \subseteq s$. If possible, application (or execution) of a in s, denoted as $\gamma(s, a)$, yields the successor state of the environment $(s \setminus \mathit{del}(a)) \cup \mathit{add}(a)$, otherwise $\gamma(s, a)$ 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 problem*** is a tuple $\mathcal{P} = (\mathcal{D}, I, G)$, where $\mathcal{D}$ is a planning domain model, I is the initial state of the environment, and G is the goal, generally in the form of a set of propositions. A ***solution plan*** (for a planning problem) is a sequence of actions such that their consecutive application starting in the initial state results in a state satisfying the goal (i.e., a goal state). A sequence of states visited during the execution of the solution plan is called ***state trajectory***.

Events Similarly to the definition of an action, an ***event*** is a tuple $e = (\mathit{pre}(e), \mathit{del}(e), \mathit{add}(e))$, where $\mathit{pre}(e)$, $\mathit{del}(e)$ and $\mathit{add}(e)$ are sets of atoms representing e's precondition, delete, and add effects, respectively. We assume that $\mathit{del}(e) \cap \mathit{add}(e) = \emptyset$. Applicability of an event in a state as well as the result of an application (or execution) of an event is defined in the same way as for actions. 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 modifying the state of the environment according to event's effects. A ***planning domain model***, in this case, is a triple $\mathcal{D} = (L, A, E)$, where L is the set of propositions, A and E is the set of actions and events over L, respectively. A planning problem, $\mathcal{P} = (\mathcal{D}, I, G)$, is defined analogously to the classical planning case.

We say that events e_i, e_j are ***independent*** if and only if $\mathit{del}(e_i) \cap (\mathit{pre}(e_j) \cup \mathit{add}(e_j)) = \emptyset$ and $\mathit{del}(e_j) \cap (\mathit{pre}(e_i) \cup \mathit{add}(e_i)) = \emptyset$.

The following assumption simplifies the reasoning by considering single-agent scenario such that actions of the agent and events of the environment alter like in a two-players game. The process hence follow 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. It should be noted that the set of events defined in the problem does not have to contain events that are independent with each other, however, in the environment turn only a subset of independent events (from the set of all events defined in the problem) can be selected. The reason for selecting only events that are independent with each other is to avoid conflicts between preconditions and effects when events are simultaneously applied. Formally:

Assumption 1. *Let $\mathcal{D} = (L, A, E)$ be a planning domain model. Let noop be an action and an event such that $\mathit{pre}(\mathit{noop}) = \mathit{del}(\mathit{noop}) = \mathit{add}(\mathit{noop}) = \emptyset$. In a current state $s \in 2^L$, the agent can apply an action $a \in A \cup \{\mathit{noop}\}$ such that a is applicable in s. After that, a (randomly selected) set of independent events $E^i \subseteq E \cup \{\mathit{noop}\}$, 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 1, denoted as $\delta(s, a)$, is determined as $\delta(s, a) = \{\gamma(s, \langle a, E^i \rangle) \mid E^i \subseteq E \cup \{\mathit{noop}\}, E^i$ is a set of independent events applicable in $\gamma(s, a)\}$. If a is not applicable in s, then $\delta(s, a)$ is undefined. 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 $s' \in 2^L$ is ***reachable*** from a state $s \in 2^L$ with respect to $\mathcal{D}$ and Assumption 1 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.

Analogously to FOND planning, we can define *strong (a)cyclic plans* for solving problems with non-deterministic events (Cimatti et al. 2003). Finding strong (a)cyclic plans give us guarantees that the agent eventually reaches its goal (in the case of strong cyclic plans under fairness assumption that each non-deterministic alternative can occur). However, the "non-deterministic branching", i.e., the number of alternatives that might occur after agent's action, might be exponential with respect to the number of events (assuming most of them are independent). That, in practice, might be feasible only for toy problems. Alternatively, a PER approach (e.g. FF-Replan (Yoon, Fern, and Givan 2007)) can be exploited to deal with non-determinism such that effects of events are ignored during the planning stage. During plan execution re-planning can be triggered after an event occurs, or when the next action becomes inapplicable (Komenda, Novák, and Pechoucek 2014).

Dead-end States Let $\mathcal{P} = (\mathcal{D}, I, G)$ be a planning problem over the planning domain model $\mathcal{D} = (L, A, E)$. We say that a state s is a ***dead-end state*** if and only if every goal state s_G ($s_G \supseteq G$) is unreachable from s.

In other words, our definition of dead-end states says that the agent cannot reach the goal by any means. With dead-

ends, the PER approach becomes unsafe. Practically speaking, the agent (robot) might even get damaged, or destroyed.

Conditional Effects Standard action (and event) definition can be extended by *Conditional Effects* that capture possible state changes if some extra condition is met in the current state. Formally, a **conditional effect** of an action (or event) x is specified as $\mathit{ceff}(x) = (\mathit{cond}(x), \mathit{cdel}(x), \mathit{cadd}(x))$ where $\mathit{cond}(x)$ is a set of atoms representing a condition and $\mathit{cdel}(x)$, $\mathit{cadd}(x)$ are sets of atoms representing conditional delete and add effects respectively. In plain words, if a condition in a current state is met, then the effects take place in the resulting state after action application. Action (or event) definition can be extended as follows $x = (\mathit{pre}(x), \mathit{del}(x), \mathit{add}(x), \mathit{ceff}^1(x), \ldots, \mathit{ceff}^k(x))$, where $\mathit{ceff}^1(x), \ldots, \mathit{ceff}^k(x)$ are conditional effects of x. Applicability of x in a state s is still determined as whether $\mathit{pre}(x) \subseteq s$. The result of application of x in s is

$$s \setminus (\mathit{del}(x) \cup \bigcup_{\mathit{cond}^i(x) \subseteq s} \mathit{cdel}^i(x)) \cup \mathit{add}(x) \cup \bigcup_{\mathit{cond}^i(x) \subseteq s} \mathit{cadd}^i(x)$$

In a nutshell, conditional effects do not influence action (or event) applicability in a given state but modify the resulting state if the conditions are met in the current state. However, conditional effects influence the relation of independence between actions/events (Rintanen 2008). Conditional effects can be compiled away, i.e., an equivalent classical representation can be obtained, however, the representation either grows exponentially or the length of solution plans grows polynomially (Nebel 2000). For the sake of clarity, we will use conditional effects in our compilation to robust plan generation.

In this paper, we leverage conditional effects only for the compilation of the problem finding robust plans into classical planning (described later).

Case Studies

AUV Sampling We introduce a simplified variant of task planning for AUV, inspired by the recent work of Chrpa et al. (2015). It simulates the situation where an AUV has to perform sampling of some 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. Each ship can enter the area at its entry cells, can move to adjacent cells it is allowed to move, and leave the area at its exit cells. A ship can appear in its entry cell, if the ship is not already in the area. A ship can leave the area, if it is in its exit cell. 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 AUV has to avoid being next to any ship that can move onto the cell AUV is at, or in any ship's entry point (if the ship is not yet in the area).

The "Perestroika" domain The Perestroika domain we introduce here is inspired by the well known Perestroika game (also known as a Toppler game)[1]. 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 do not change its size nor disappear. On the contrary, the shrinking platforms can have *large*, *medium* or *small* shape and they can disappear completely.

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 five events. Two events change the shape of the platform from *large* to *medium* and from *medium* to *small*, respectively. Two events make the platform disappear if it has a small 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 in a *large* shape.

The agent cannot move to or stay on small shrinking platforms and must always have an escaping way to a nearby solid platform to avoid being "trapped" on a shrinking platform.

Towards Safe Use of Classical Planning

To achieve its goal, the agent has to avoid reaching a dead-end state during plan execution. Whereas if the environment is static, dead-end states are implicitly avoided during plan execution as plans do not contain actions leading to dead-ends, non-deterministic events might get the agent into dead-end states. We can adopt the concept of *robust plans* from conformant planning (Palacios and Geffner 2009) that if executed guarantee reaching a goal state despite uncertainty of the environment.

However, robust plans might only exist in a very restricted number of cases. By looking at both AUV and Perestroika, we can observe that with an increasing number of steps (executed actions), uncertainty of the environment rises, i.e., ships might possibly be at many locations, or shrinking platforms might have any shape (or not being there). This observation gives us an intuition that robust plans are likely to exist only if they are short.

Inspired by the work of Vernhes, Infantes, and Vidal (2013) concerning splitting classical planning problems

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

into a sequence of smaller problems, we propose an approach that iteratively plans and executes robust plans between *safe states* which cannot be turned to dead-end states by applying only events. To reflect the above observation, safe states should be "close" to each other, so they can be connected by robust plans. To do so, we propose an approach to generate an initial plan in which the lengths of possibly unsafe action sequences are bounded. The proposed approaches are described in detail in the following sections.

Finding Robust Plans

Events are responsible for "plan branching" that is, roughly speaking, considering all alternatives that might be introduced by event occurrences.

The key feature of robust plans is that they *evade event effects*. In plain words, events can occur but their effects will not affect plan execution. Technically speaking, regardless of event occurrences the agent will keep applying the same actions as if no event occurred.

Definition 2. *Let $\mathcal{D} = (L, A, E)$ be a planning domain model and $\mathcal{P} = (\mathcal{D}, I, G)$ be a planning problem. Let $\pi = \langle a_1, \ldots, a_n \rangle$ be a (finite) sequence of actions. If $\forall s \in \delta(I, \pi) : G \subseteq s$, then we say that π is a* **robust plan** *for $\mathcal{P}$.*

Computing robust plans according to the above definition is impractical as all the alternatives (resulting states of action application) have to be considered. Informally said, it will not make much difference than computing strong (cyclic) plans. On the other hand, with a pessimistic assumption how events can modify the environment we are able to compute robust plans in a similar fashion like classical plans.

Proposition 3. *Let $\mathcal{D} = (L, A, E)$ be a planning domain model and $\mathcal{P} = (\mathcal{D}, I, G)$ be a planning problem. Let $\pi = \langle a_1, \ldots, a_n \rangle$ be a (finite) sequence of actions such that $G \subseteq \gamma(I, \pi)$ (i.e., a solution plan for the underlying classical planning problem), and $s_0 = I, s_1 = \gamma(s_0, a_1), \ldots, s_i = \gamma(s_{i-1}, a_i), \ldots, s_n = \gamma(s_{n-1}, a_n) \supseteq G$ be a state trajectory. We define p^i_+ and p^i_- as sets of atoms that in the i-th step might be added, respectively deleted by events that could have occurred before. Then, we define E^i as a set of events that might occur in the i-th step. Atoms p^i_+ and p^i_-, and events E^i are computed as follows:*

- $E^0 = p^0_+ = p^0_- = \emptyset$ *(no event can occur before agent's first action)*
- $E^i = \{e \mid e \in E, \mathit{pre}(e) \subseteq ((s_{i-1} \cup p^{i-1}_+) \setminus \mathit{del}(a_i)) \cup \mathit{add}(a_i)\}$,
- $p^i_+ = (p^{i-1}_+ \setminus \mathit{del}(a_i) \cup \bigcup_{e \in E^i} \mathit{add}(e))$,
- $p^i_- = (p^{i-1}_- \setminus \mathit{add}(a_i) \cup \bigcup_{e \in E^i} \mathit{del}(e))$.

If for each $1 \leq i \leq n$ it is the case that $\mathit{pre}(a_i) \cap p^{i-1}_- = \emptyset$ and $G \subseteq s_n \setminus p^n_-$, then π is a robust plan for $\mathcal{P}$.

Proof (Sketch). The sets of atoms p^i_+ and p^i_- represent how a state in the i-th step might differ from s_i. Optimistically, atoms in $s_i \cup p^i_+$ might be true in the i-th step while, pessimistically, atoms $s_i \setminus p^i_-$ are only true in the i-th step. Events that might be applicable in the i-th step (E^i) are determined from the optimistic assumption. On the other hand, action (a_i) applicability in the i-th step is determined from the pessimistic assumption.

It can hence be derived from the proposition assumption that in each step a_i remains applicable and that after all actions are applied the goal is satisfied. Hence, π is a robust plan for $\mathcal{P}$ (the agent applies actions consecutively regardless of event occurrence) □

The above proposition illustrates how to find a robust plan by "enhancing" a classical planner. An enhanced classical planner keeps track of atoms that might be true or false in the given step (i.e., it computes the p^i_+ and p^i_- sets). Then, the planner determines action applicability by whether its precondition is met in a current state (s_i) minus "might be false" atoms (p^i_-). If a planner reaches a goal state (minus p^n_-), then the found plan is a robust plan. Such an approach is sound, however, not complete as it considers pessimistic assumption of how events can influence action applicability.

Compiling Event Evasion into Classical Planning

In the spirit of works of Palacios and Geffner (2009) and Grastien and Scala (2017) in Conformant planning, generating robust plans that are not affected by non-deterministic events by standard planners is possible if the problem of finding such plans in encoded as a classical planning problem (with conditional effects).

Let $\mathcal{D} = (L, A, E)$ be a domain model and $\mathcal{P} = (\mathcal{D}, I, G)$ be a planning problem. We define $L^{AE} = \{p \mid p \in (G \cup \bigcup_{a \in A} \mathit{pre}(a)) \cap \bigcup_{e \in E} \mathit{pre}(e)\}$ that represents atoms that are required by action(s) or a goal as well as event(s). We define L'^{AE} such that $L \cap L'^{AE} = \emptyset$ and there exists a bijective mapping between L^{AE} and L'^{AE}. In plain words, L'^{AE} consists of "twins" of L^{AE} that will be used to distinguish between atoms required by actions (L^{AE}) and events (L'^{AE}). For the sake of clarity, we denote that $p \in L^{AE}$ maps to $p' \in L'^{AE}$ and vice versa.

We introduce two atoms, (a-tn) and (e-tn) (we assume they are not present in $L \cup L'^{AE}$) for determining action and event "turn". For each action $a \in A$ we construct an action a^C as follows.

$$\mathit{pre}(a^C) = \mathit{pre}(a) \cup \{\text{(a-tn)}\}$$
$$\mathit{del}(a^C) = \mathit{del}(a) \cup \{\text{(a-tn)}\} \cup \{p' \mid p \in L^{AE} \cap \mathit{del}(a)\}$$
$$\mathit{add}(a^C) = \mathit{add}(a) \cup \{\text{(e-tn)}\} \cup \{p' \mid p \in L^{AE} \cap \mathit{add}(a)\}$$

Then, we construct an action a^E that considers possible effects of events (encoded as conditional effects) as follows (let $L^A = \{p \mid p \in G \cup \bigcup_{a \in A} \mathit{pre}(a)\}$).

$$\mathit{pre}(a^E) = \mathit{del}(a^E) = \{\text{(e-tn)}\}$$
$$\mathit{add}(a^E) = \{\text{(a-tn)}\}$$
$$\forall e \in E : \text{construct } \mathit{ceff}^e(a^E) \text{ such that}$$
$$\mathit{cond}^e(a^E) = (\mathit{pre}(e) \setminus L^{AE}) \cup \{p' \mid p \in L^{AE} \cap \mathit{pre}(e)\}$$
$$\mathit{cdel}^e(a^E) = \mathit{del}(e) \cap L^A$$
$$\mathit{cadd}^e(a^E) = (\mathit{add}(e) \setminus L^A) \cup \{p' \mid p \in L^{AE} \cap \mathit{add}(e)\}$$

The action and event turn alternates such that one action can be applied in the action turn while in the event turn all

applicable events are considered (via conditional effects). To simulate the p_- and p_+ sets (as in Proposition 3) that pessimistically influence action applicability and optimistically influence event applicability, respectively, we have to distinguish between atoms that are required by actions (or a goal) L^A and atoms required by both actions (or a goal) and events L^{AE} ($L^{AE} \subseteq L^A$). Atoms in L^A are still considered in event delete effects while they are not considered in event add effects (pessimistically, event can only make an action inapplicable). For atoms in L^{AE}, we consider two "clones", p, present in preconditions of actions, and p', present in preconditions of events. Whereas effects of actions contain both clones (action effects affect both action and event applicability), add effects of events contain only p' atoms (affecting only event applicability) while delete effects of events only p atoms (affecting action applicability). Atoms that are not required by actions (or a goal) are considered only in add effects of events (affecting event applicability).

The classical domain model is then constructed as $\mathcal{D}^C = (L \cup L'^{AE} \cup \{\mathsf{(a\text{-}tn)}, \mathsf{(e\text{-}tn)}\}, \{a^C \mid a \in A\} \cup \{a^E\})$. The classical planning problem is then constructed as $\mathcal{P}^C = (\mathcal{D}^C, I \cup \{p' \mid p \in L^{AE} \cap I\}, G)$. A solution plan for $\mathcal{P}^C$ (if exists) consists of a sequence of actions in which the a^C actions alternate with the a^E action. Removing all occurrences of the a^E action and replacing the a^C actions by the corresponding a actions while keeping the same order results in a robust plan.

Planning with Safe States

The notion of "safe" states has been used by Cserna et al. (2018) to avoid dead-ends in online planning. For planning with events, the notion has been introduced by Chrpa, Gemrot, and Pilát (2017) as a complement to "dangerous states".

Definition 4. *Let $\mathcal{D} = (L, A, E)$ be a planning domain model and $\mathcal{P} = (\mathcal{D}, I, G)$ be a planning problem. We say that a state $s \in 2^L$ (with respect to $\mathcal{P}$) is* **safe** *if and only if there does not exist a sequence of events from E such that their consecutive application in s results in a dead-end state.*

Consequently, only agent's actions applied in safe states can result in unsafe or even dead-end states. However, staying only in safe states might not be possible in order to achieve a given goal and the agent has to traverse unsafe states. In the AUV example, the AUV might have to move through cells ships can enter. Similarly, the Perestroika agent might have to cross shrinking platforms. Unsafe states can turn into dead-end states by sequences of events (e.g., a shrinking platform can eventually disappear killing the agent staying on it). Hence, rather than focusing on the next action it is more informative to consider whether applying a sequence of actions always results in a safe state despite visiting unsafe states in between.

Definition 5. *Let $\mathcal{D} = (L, A, E)$ be a planning domain model, $\mathcal{P} = (\mathcal{D}, I, G)$ be a planning problem and $s \in 2^L$ be a state. Let $\langle a_1, \ldots, a_k \rangle$ be a sequence of actions applicable in s and their application results in s_k. Let p^i_-, p^i_+ be sets of atoms defined as in Proposition 3. If for each $1 \leq i \leq k$ it is the case that $pre(a_i) \cap p^{i-1}_- = \emptyset$ and s_k under a pessimistic assumption reflecting p^k_-, p^k_+ is a safe state, then we say that $\langle a_1, \ldots, a_k \rangle$ is* **safely applicable** *in s.*

Remark 6. *Safe states can be, for instance, represented by DNF formula (similarly to dead-end traps (Lipovetzky, Muise, and Geffner 2016)). Let l^+ and l^- be positive and negative literals that refer to atoms that if true and false, respectively, in a state, then the state is safe. In this case, we understand by "s* being safe under a pessimistic assumption reflecting *p_-, p_+" that $l^+ \subseteq s \setminus p_-$ and $l^- \cap (s \cup p_+) = \emptyset$.*

Definition 7. *Let $\mathcal{D} = (L, A, E)$ be a planning domain model, $\mathcal{P} = (\mathcal{D}, I, G)$ be a planning problem. We say that an action $a \in A$ is* **safe** *if and only if for each $s \in 2^L$ in which a is applicable $\gamma(s, a)$ is a safe state. An action that is not safe is called* **unsafe***.*

Determining whether an action a is safe with respect to the DNF formula representing safe states regardless of a particular state is it applied can be done accordingly to the following lemma (it is a pessimistic assumption that might not identify all safe actions).

Lemma 8. *Let a be an action and a formula containing a conjunction of literals representing a safe state, where l^+ and l^- are positive and negative literals such that $l^+ \subseteq (pre(a) \setminus del(a)) \cup add(a)$ and $l^- \subseteq del(a)$. Then a is safe.*

Proof (Sketch). *It can be immediately seen from the assumption that the l^+ literals are either true prior a's application, or become true after it (and they are not deleted) and the l^- literals are deleted by a.*

The above lemma can be exploited for determining the subset of "safe" actions from the set of actions defined in a given domain model. The remaining actions are considered as "unsafe".

Generating plans such that we limit the number of consecutive unsafe actions (denoted as *unsafeness limit*) can be done by remodelling the planning task as follows (still within classical planning). Given the unsafeness limit d, we extend the language by $d + 1$ atoms representing "unsafeness credits", i.e., (uc-0),(uc-1),. . . ,(uc-d). Then, we modify each safe action a as follows:

- $add(a) = add(a) \cup \{\mathsf{(uc\text{-}d)}\}$
- $del(a) = del(a) \cup \{\mathsf{(uc\text{-}0)}, \ldots, \mathsf{(uc\text{-}d\text{-}1)}\}$

Each unsafe action a is "cloned" d times, and each clone a^k, where $1 \leq k \leq d$, is modified as follows:

- $pre(a^k) = pre(a) \cup \{\mathsf{(uc\text{-}k)}\}$
- $del(a^k) = del(a) \cup \{\mathsf{(uc\text{-}k)}\}$
- $add(a^k) = add(a) \cup \{\mathsf{(uc\text{-}k\text{-}1)}\}$

One can immediately see that at most d unsafe actions can be planned in a row since after that none of the unsafe actions is applicable as the number of unsafeness credits drops to zero (and no unsafe action remains applicable after that) and that the "credit counter" is reset to maximum (d) after a safe action is planned.

Safe states, in contrast to the traditional PER approach, provide a tool for reasoning about how dead-end states can be avoided. From a given plan, we can identify the longest sequence of safely applicable actions that if applied in the

Algorithm 1 Enhancing the PER approach by Safe State reasoning

```
1: Generate a robust plan π
2: if π exists then
3:     Execute π and terminate with success
4: end if
5: d ← 1
6: repeat
7:     Generate π with unsafeness limit of d
8:     d ← d + 1
9:     if d > threshold then
10:        Terminate with failure
11:    end if
12: until π exists
13: s ← I
14: while G ⊈ s do
15:    k ← max{i | ⟨a_1, ... a_i⟩ ∈ π safely applicable in s}
16:    if k > 0 then
17:        Execute (apply) ⟨a_1, ... a_k⟩ in s
18:    else
19:        Generate a robust plan π′ to the next safe state
       (according to π)
20:        Execute π′ if it exists, otherwise execute noop
21:    end if
22:    s ← observe the current state
23:    π ← the rest of the solution plan (a_{k+1} onwards)
24: end while
```

current state (their application always succeed), the resulting state is safe as well. The idea is formalized in Algorithm 1 which provides a high-level routine for incorporating safe state reasoning into the PER approach. Initially, we try to extract a robust plan, if found we can safely execute it, otherwise, we plan with increasing unsafeness limit until we find a plan (or fail if we reach the threshold)[2]. Then, until the goal is achieved, the agent selects the maximum k such that the first k actions from the plan are safely applicable in the current state (Line 15), executes those first k actions (under Assumption 1) or adapts the plan by generating a robust plan to the next safe state (explained below) if $k = 0$ (Line 19), or performs $noop$ if such a robust plan does not exist (Line 20). Then the agent observes the current state after that (Line 22) and continues the loop. From Definition 5 it implies that if the initial state is safe, then after each iteration the agent remains in the safe state, although during the iteration the agent might find and execute a robust plan which transits through unsafe states.

If not possible to extract any safely applicable sequence from the current plan (i.e., $k = 0$ in Line 15), we can adjust the plan by finding a robust plan to the next safe state (Line 19) as follows. Let s be the current state, G be the goal, $\langle a_i, \ldots, a_n \rangle$ be the current plan (from s to G). Then we perform the following steps.

1. Find j ($i < j \leq n$) such that applying $\langle a_i, \ldots, a_j \rangle$ would result in a safe state if the whole sequence was applicable.
2. Construct a planning problem with the same domain model, s as the initial state, and $\bigcup_{i \leq q \leq j}(add(a_q) \setminus \bigcup_{q < r \leq j} del(a_r)) \cap (\bigcup_{j < q \leq n}(pre(a_q) \setminus \bigcup_{j < r < q} add(a_r)) \cup G \setminus \bigcup_{j < r \leq n} add(a_r)))$ as the goal.
3. Generate a robust plan for the new planning problem and merge it with $\langle a_{j+1}, \ldots, a_n \rangle$ (the rest of the current plan).

[2]Alternatively, as proving plan non-existence might be more difficult, we can generate plans with decreasing unsafeness limit (starting at the threshold) until no plan is found or time expires.

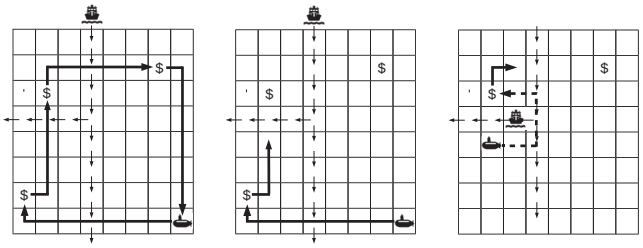

Figure 1: A sample AUV planning problem with a plan (left), a safely applicable part of the plan (middle), and a robust plan between safe states (dashed) connected with another safely applicable part of the plan (right). "$" denotes a cell with resource, thin arrows denote where the ship can move and thick arrows denote AUV's plan (or a part of it).

In Figure 1, a sample planning problem in the AUV domain is depicted in which the AUV (in the bottom right corner) has to sample all resources (denoted by $) and return back to the location of origin. The plan that might be generated (with unsafeneess limit 1) is shown in Figure 1 left. Safe state reasoning as shown in Algorithm 1 leads to selecting a sequence of safely applicable actions from the plan as shown in Figure 1 middle. The unsafe state at the bottom can be safely passed through since the ship cannot be there before or at the same time as the AUV (the ship needs 8 steps to interfere with the AUV's path while the AUV needs 5 steps to cross ship's path). However, the unsafe state on the left cannot be safely passed through since the ship has enough time to interfere with the AUV. Hence, the AUV after executing the safely applicable action subsequence has to observe the current state and perform the safe state reasoning again from the current state. As depicted in Figure 1 right the algorithm finds a robust plan around the ship as it threatens AUV's original plan. As the ship can move only forwards, the robust plan avoid the ship by passing it from behind.

Experimental Evaluation

For our experimental evaluation, we specified 6 problems for each domain. The AUV-1 problem is the same as depicted in Figure 1. AUV-2 has two ships moving "to the cross" while AUV-3 has three ships moving in adjacent columns (in the middle) such that the middle ship moves in the opposite direction than the other two (everything else is the same as in AUV-1). AUV-4-6 are proportionally scaled to 16x16 grid (the number of ships and resources remain the same). Perestroika-1-3 are on 5x5 grid such that in Perestroika-1 and 3 solid platforms are on coordinates that are either

Algorithm	PER-APP			PER-EVENT			DANG			LIMIT		
Problem	PT	ACT	SR	PT	ACT	SR	PT	ACT	SR	PT	ACT	SR
AUV-1	226	37.32	76	3307	36.66	88	1364	41.81	100	410	38.60	100
AUV-2	418	39.16	49	3940	35.91	57	1582	45.79	98	678	32.89	100
AUV-3	352	37.64	50	5078	39.68	47	1620	45.13	80	2198	40.44	100
AUV-4	664	75.12	98	32518	109.56	93	1539	75.99	100	1603	75.39	100
AUV-5	943	75.61	85	39642	101.57	67	2946	78.70	98	2416	77.32	100
AUV-6	848	75.02	83	43604	98.87	61	2681	78.10	97	6982	73.79	100
Perestroika-1	544	21.58	24	2714	24.13	15	6497	60.84	100	600	28.78	100
Perestroika-2	425	22.89	18	2661	23.00	10	7694	68.73	90	699	31.20	100
Perestroika-3	488	26.57	14	3143	27.23	13	7322	72.69	95	577	33.88	100
Perestroika-4	889	40.00	1	N/A	N/A	0	15883	118.54	100	2139	56.49	100
Perestroika-5	1327	42.67	3	6327	46.00	1	21479	153.07	83	2198	57.99	100
Perestroika-6	764	22.21	19	3009	14.63	8	14170	111.78	95	633	15.84	100

Table 1: The results of the compared algorithms over 100 runs on 6 problems from the AUV and Perestroika domains. PT denotes the average time spent by planning (milliseconds), ACT denotes the average number of actions needed to solve the problem, and SR denotes the number of successful runs.

both odd or both even. In Perestroika-2, shrinking platforms are on even rows and even columns. In all problems, the agent starts in a corner. In Perestroika-1-2, resources are located in the other corners and in the middle of the grid. In Perestroika-3, resources are located on all solid platforms. Perestroika-4-6 are proportionally scaled to 9x9 grid[3].

We compare our approach (Algorithm 1) that limits the number of consecutive steps in unsafe states (denoted as LIMIT henceforth) to the more traditional PER techniques of replanning either in case the next actions is not applicable (PER-APP) or whenever an event occurs (PER-EVENT), and to the technique based on state dangerousness (DANG) (Chrpa, Gemrot, and Pilát 2017). In this case, whenever the dangerousness of the state after applying the next action in the current plan would be lower than 2, we replace the current plan by a plan to the closest safe state. Once this state is reached, we again plan to the goal. We used the well known LAMA planner (Richter and Westphal 2010) for plan generation for all the methods.

The results of the experiments are shown in Table 1. We made 100 independent runs of each method for each problem and we show the time spent by planning by each of the methods, the number of actions (including no-op actions), needed to solve the problem (counted only for successful runs) and the number of successful runs. We can see that our LIMIT method is able to solve all the problems successfully in all runs in contrast to other methods. Whereas the DANG method succeeded in four of the problems and in the others the success rate was above 80%, the PER methods failed in the larger Perestroika problems.

Larger planning times of the LIMIT approach in AUV-3 and AUV-6 problems (in contrast to DANG) are the result of attempts to find an initial plan (it can be found for the unsafeness limit of 3). Quality of plans in the AUV domain is comparable. On the other hand, in Perestroika, LIMIT clearly outperforms DANG both in planning time and quality of plans. The reason is that a large shrinking platform produces the "threshold" dangerousness of 2 (after stepping on the platform it can shrink to medium and then it might take 2 steps to disappear).

We have also encoded the problems as FOND planning problems and used the PRP planner (Muise, McIlraith, and Beck 2012) to generate strong cyclic plans. Only AUV-1 was solved (in 987 seconds), the other problems were unsolved (in 1800s) although all the problems are solvable.

Limitations of the Safe State Approach

The Safe State Reasoning (LIMIT) approach relies on whether "unsafe" states can be traversed by "robust" plans. For example, crossing three shrinking platforms in Perestroika cannot be done by a robust plan (the last platform even if large at the beginning might disappear just while the agent is on it). However, a strong cyclic plan exists – the agent might move on the first platform if it is large or medium, then if the second platform is medium or large and the third platform is large, the agent can safely cross them, otherwise the agent waits on the first platform and if it shrinks to a small size, the agent steps back. Noteworthy, REPLAN or DANG might succeed in some cases for this problem as they "take chance".

Also, if there is no way to generate robust plan (Line 19 of Algorithm 1) such as in the above example, the algorithm does not terminate. However, the agent gets stuck in a safe state which has practical benefits (e.g. the AUV does not get destroyed by a ship).

Also, the information about safe states (not necessarily complete) has to be provided up front (by domain experts who already provide domain model specification). A similar assumption was made by Cserna et al. (2018). Determining the unsafeness threshold correlates with "optimistic dangerousness" of unsafe states. For example, for shrinking platforms in Perestroika, it is 3 as it takes 3 steps for a large platform to disappear. The agent can then cross at most 2 shrinking platforms by a robust plan (assuming that events eventually form a viable configuration of the platforms -

[3]Our implementation and benchmark problems are available at https://github.com/martinpilat/jPDDL

large/medium and large).

Conclusions

Planning with non-deterministic events presents a challenge of "navigating" the agent towards its goal in spite of events that might modify the environment without its consent. To reasonably exploit classical planning in such situations, the key aspect is to avoid dead-end states. Inspired by recent works (Chrpa, Gemrot, and Pilát 2017; Cserna et al. 2018), we conceptualized the notion of "safe states" that can be exploited in the PER approach to guarantee that the agent does not fall into a dead-end state (in the worst case the agent gets stuck in a safe state but does not get damaged). The results demonstrate that our approach outperforms the traditional replanning approaches as well as the recent approach of Chrpa, Gemrot, and Pilát (2017).

The introduced concepts are suitable for problems in which safe states are "close enough". As shown in the AUV example, such problems do exist in real-world and hence our concepts can increase robustness and autonomy of such systems. On the other hand, for problems with sparse safe states, it might not be possible to guarantee agent's safety or that the agent reaches its goal in a finite number of steps. For such problems leveraging MDP or MCTS approaches might be more suitable in order to maximize the chance to succeed (or maximize the reward).

In future, we would like to extend introduced concepts to planning with timelines as it would provide a level of expressiveness required by many real-world applications (e.g. Mars Rovers).

Acknowledgements

This Research was funded by the Czech Science Foundation (project no. 17-17125Y), 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

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

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*.

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.

Grastien, A., and Scala, E. 2017. Intelligent belief state sampling for conformant planning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, 4317–4323.

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

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

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.

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.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research* 12:271–315.

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.; 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.

Rintanen, J. 2008. Regression for classical and nondeterministic planning. In *ECAI 2008 - 18th European Conference on Artificial Intelligence*, 568–572.

Vernhes, S.; Infantes, G.; and Vidal, V. 2013. Problem splitting using heuristic search in landmark orderings. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, 2401–2407.

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