Action-Failure Resilient Planning

Alberto Rovetta

PhD Advisors: Alfonso Emilio Gerevini, Diego Aineto, Enrico Scala, Ivan Serina
Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia, Italy
alberto.rovetta@unibs.it

1 Introduction

This work introduces the task of "Resilient Planning," a novel approach that seeks to handle action failures during plan execution and ensure robust goal attainment. It considers scenarios where the execution of a plan in the real world can be thwarted by external factors that make planned actions infeasible. The traditional classical planning models often fail to account for such unforeseen action failures, as they merely verify the correctness of plans in an abstract world. Anticipating and addressing all possible execution failures using detailed planning models can result in a overly complex state and action models, making the planning process cumbersome. In automated planning, a common strategy for handling action failures involves alternating between plan execution and replanning from the state where the failure occurs [5]. In some cases, this can be done by repairing the existing plan rather than starting over with a new one (e.g., [2, 3, 4]). However, this approach does not always ensure that the plan can be fixed in a way that still achieves the original goal. To tackle this issue, we propose a complementary method focused on generating plans that come with repair guarantees in the event of action failures during execution. We define the task of finding solutions to classical planning problems that can withstand action failures as *Resilient Planning*, and we refer to the resulting plans as *K-resilient plans*. These plans ensure that an agent can always reach its goal (potentially by replanning alternative sequences of actions) as long as no more than K action failures occur on the way to the goal.

2 Background on Classical Planning

A classical planning problem is a tuple $\Pi = \langle F, A, s_0, G\rangle$ whose components are defined as follows. F is a finite set of positive literals inducing a set S of states. A state $s \in S$ is a subset of F. If an element $f \in F$ is in a state s then f is true in s; otherwise f is false in s by the closed world assumption. s_0 is the initial state. $G \subseteq F$ is the problem goal consisting of a set of literals over F that should hold in any goal state. A is a set of actions; each action $a \in A$ is specified by the pair $a = \langle \mathsf{pre}(a), \mathsf{eff}(a)\rangle$ where $\mathsf{pre}(a) \subseteq F$ is the precondition set of a, and $\mathsf{eff}(a)$ the effect set of a formed by subsets of positive and negative literals over F, that are denoted with $\mathsf{eff}(a)^+$ and $\mathsf{eff}(a)^-$, respectively. An action a is applicable in state s iff $\mathsf{pre}(a) \subseteq s$, and we denote the set of actions applicable in state s with $A(s)$. The application of an action $a \in A(s)$ in s generates a state $s' = s[a]$ such that, for every fluent $f \in F$, f is in s' iff $f \in (s \setminus \mathsf{eff}(a)^-) \cup \mathsf{eff}(a)^+$.

A plan π is a sequence of actions in A, i.e., $\pi = (a_1, \ldots, a_n)$. Given a planning problem $\Pi = \langle F, A, s_0, G\rangle$, $(s_0, s_1, \cdots, s_n)$ is the trajectory of states induced by applying π in s_0, i.e., $s_i = s_{i-1}[a_i]$ for $i = 1, \cdots, n$. A plan $\pi = (a_1, \ldots, a_n)$ is a solution for $\Pi = \langle F, A, s_0, G\rangle$ iff the induced trajectory of states $(s_0, s_1, \cdots, s_n)$ is such that for all $i \in [1, n]$ it holds that $\mathsf{pre}(a_i) \subseteq s_{i-1}$ and $G \subseteq s_n$.

3 Resilient Solutions and Algorithm

Resilient planning [1] describes the world through a (classical) planning problem but explicitly considers at planning time that actions can fail at execution time; if an action fails, that failures do not modify the state of the world and cannot be reapplied. Our formalisation of resilient planning and its solutions rely on the following notion of resilient states.

Definition 1 (k-Resilient State). *Let $\Pi = \langle F, A, s_0, G\rangle$ be a planning problem, S the state space induced by F, and k a non-negative integer.*

(i) A state $s \in S$ is 0-resilient in Π iff there is a plan from s that achieves G (i.e. $\langle F, A, s, G\rangle$ is solvable);
(ii) A state $s \in S$ is k-resilient in Π if $s \models G$;
(iii) A state $s \in S$ such that $s \not\models G$ is k-resilient in Π for $k \geq 1$ iff there exists an action $a \in A(s)$ such that (1) $s[a]$ is k-resilient in Π and (2) s is $(k-1)$-resilient in $\langle F, A \setminus \{a\}, s_0, G\rangle$.

A state s is k-resilient in a planning problem $\langle F, A, s_0, G\rangle$ if the goal G can be achieved from s even after k action failures occur along the way of a plan from s to G. Definition 1 formalizes this notion by considering that the execution of an action a applicable in s can either success or fail. The definition of k-resilient state implies that there exists a trajectory from the state to the goal where all states are k-resilient.

A *resilient planning problem* is a pair $\langle \Pi, K\rangle$ where Π is a planning problem and K is an non-negative integer (classical planning is a special case of resilient planning with $K = 0$).

Definition 2 (k-Resilient Plan). *Given a planning problem $\Pi = \langle F, A, s_0, G\rangle$, a solution plan for Π that induces a state trajectory $(s_0, s_1, \ldots, s_n)$ is k-resilient for Π if, for all $0 \leq i < n$, it holds that s_i is k-resilient in Π.*

A *solution* for $\langle \Pi, K\rangle$ is a K-resilient plan for Π. Following, we present a novel algorithm for Resilient Planning called ResPlan that computes resilient plans for classical planning problems by iteratively using a classical planner to prove whether a state is resilient or not. ResPlan algorithm takes as input a resilient planning problem $\langle\langle F, A, s_0, G\rangle, K\rangle$, and outputs a K-resilient plan π^K for $\langle F, A, s_0, G\rangle$ if it exists and $unsolvable$ otherwise. The recursive nature of the definition of k-resilient state means that, in order to achieve this, we will have to prove resilience of many other states. That is, to prove that a state s is k-resilient, we need to find a successor state $s' = s[a]$ that is also k-resilient, and we also need to show that s is still $(k-1)$-resilient without using action a. The RESPLAN algorithm does this by performing a search over an augmented state space of nodes of the form $\langle s, k, V\rangle$ where s is a state, $0 \leq k \leq K$, and $V \subseteq A$ are faulty actions that cannot be used again. Implicitly, a node $\langle s, k, V\rangle$ represents the problem of deciding whether s is a k-resilient state in $\langle F, A \setminus V, s_0, G\rangle$.

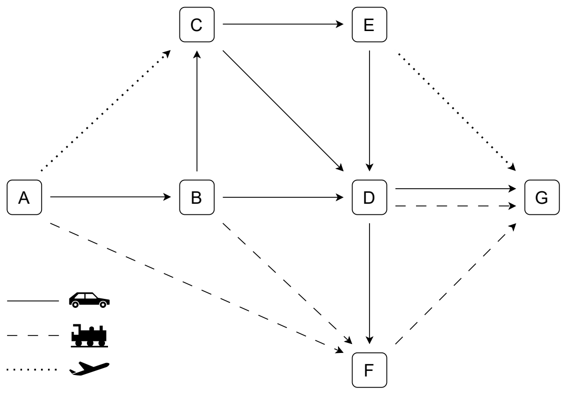

Figure 1. Navigation problem with seven locations (squares) and three types of connections: road (solid), railway (dashed), and flight (dotted).

Now, we present a quick walk-through of what could be a possible execution of RESPLAN. We consider as inputs the problem of Figure 1 and $K = 2$, so the $Open$ list will be initialized with $\langle \texttt{A}, 2, \emptyset\rangle$. Assume that, in the first iteration, $ComputePlan$ returns the plan (`car(A,B)`,`train(B,F)`,`train(F,G)`) so, after pushing the generated nodes, $\langle \texttt{F}, 1, \{\texttt{train(F,G)}\}\rangle$ will occupy the last position in $Open$. In the next iteration, RESPLAN will pop $\langle \texttt{F}, 1, \{\texttt{train(F,G)}\}\rangle$ and it will find out that `F` is not 1-resilient, since $ComputePlan$ will not be able to find a plan from `F` to `G` that does not use `train(F,G)`. It will then call $UpdateNonResilient$ and add $\langle \texttt{F}, 1, \{\texttt{train(F,G)}\}\rangle$ to $\mathcal{R}_\downarrow$ alongside $\langle \texttt{F}, 2, \emptyset\rangle$. At this point, the plan (`car(A,B)`,`train(B,F)`,`train(F,G)`) is not 2-resilient since `F` is not 2-resilient, and this will be reflected in the algorithm by failing the $RCheck$ after it pops $\langle \texttt{B}, 2, \emptyset\rangle$. RESPLAN will then try to compute a plan to the goal starting from `B`. Recall that $\langle \texttt{F}, 2, \emptyset\rangle$ belongs to $\mathcal{R}_\downarrow$ and, therefore, $S_\downarrow$ contains `F` which will be in turn not visited by $ComputePlan$. Let us assume that the computed plan is (`car(B,D)`,`car(D,G)`) It is worth noticing that, at this stage, the algorithm discarded the suffix (`train(B,F)`,`train(F,G)`) of the first computed plan (`car(A,B)`,`train(B,F)`,`train(F,G)`), and is now considering (`car(A,B)`,`car(B,D)`,`car(D,G)`). Next, RESPLAN will pop $\langle \texttt{D}, 1, \{\texttt{car(D,G)}\}\rangle$ and compute a plan from `D` to `G` without using `car(D,G)`. One possibility is to use `train(D,G)` instead, so, in the next iteration, it will pop $\langle \texttt{D}, 0, \{\texttt{car(D,G)}, \texttt{train(D,G)}\}\rangle$. At this point it will once again compute a plan from `D` to `G`, but this time without using neither `car(D,G)` nor `train(D,G)`, and the only solution here will be (`car(D,F)`,`train(F,G)`). Since we are at $k = 0$, the nodes $\langle \texttt{D}, 0, \{\texttt{car(D,G)}, \texttt{train(D,G)}\}\rangle$ and $\langle \texttt{F}, 0, \{\texttt{car(D,G)}, \texttt{train(D,G)}\}\rangle$ will be directly added to $\mathcal{R}_\uparrow$. In the next two iteration, RESPLAN will pop first $\langle \texttt{D}, 1, \{\texttt{car(D,G)}\}\rangle$ and then $\langle \texttt{D}, 2, \emptyset\}\rangle$, and both times the $RCheck$ will return True; so these nodes will be moved to $\mathcal{R}_\uparrow$. At this point, RESPLAN has proven that `D` is 2-resilient. The following iterations, all the way to the end of the algorithm, will follow a pattern similar to the one described for `D` but for states `B` and `A`. Once RESPLAN terminates, it will return the 2-resilient plan (`car(A,B)`,`car(B,D)`,`car(D,G)`).

		K=1			K=2			K=3			K=4		
Domain	**Sol**	**MN**	**FS**	**RP**	**MN**	**FS**	**RP**	**MN**	**FS**	**RP**	**MN**	**FS**	**RP**
Total-IPC	S	10	2	**56**	2	1	**12**	0	0	0	0	0	0
(#106)	U	**8**	-	7	**9**	-	**9**	**10**	-	8	**15**	-	9
Total-Res	S	9	11	**29**	1	2	**29**	0	0	**21**	0	0	**9**
(#30)	U	0	-	0	0	-	0	0	-	0	0	-	0
Total	S	19	13	**85**	3	3	**41**	0	0	**21**	0	0	**9**
(#136)	U	**8**	-	7	**9**	-	**9**	**10**	-	8	**15**	-	9

Table 1. Coverage results for RESPLAN (RP), compilation into FOND using MyND (MN) or FOND-SAT (FS) across benchmarks from the IPCs and the newly generated instances (Res)

In light of these results, we are currently studying ways to improve the performance of RESPLAN. In particular, we are working on a technique that exploits the landmarks [6] of the planning problem to identify an upper bound of the resilience of a state that can be used for early detection of unsolvable instances.

As future work, we plan to make our approach more flexible and general through a number of extensions that consider richer planning formalisms. In this regard, we believe that defining the notion of resilience with respect to time and resources could be of great interest.

4 Conclusion and Future Work

We addressed the challenge of creating plans within classical planning that maintain robustness during execution. These plans are required to pass through states that satisfy a specified resilience threshold. The RESPLAN algorithm generates plans that are resilient, ensuring they can be repaired if a limited number of planned actions fail to execute or have no impact, leaving the planning model's current state unaffected. An experimental comparison with another approach, which relies on compilation into strong FOND planning, shows that RESPLAN is competitive in coverage for unsolvable instances, but is significantly more effective in terms of both coverage and runtime on solvable instances. In future work, we aim to optimize RESPLAN's performance through various methods, particularly by implementing new pruning techniques that could enhance its efficiency on unsolvable instances, such as by leveraging landmarks [6]. Furthermore, we plan to explore alternative definitions of resilient states and additional strategies to model and manage action failures. Resilient planning may serve as a tool to assess the quality of such models by evaluating the level of resilience they allow in their plans, prioritizing models that offer greater resilience.

References

[1] Diego Aineto, Alessandro Gaudenzi, Alfonso Gerevini, Alberto Rovetta, Enrico Scala, and Ivan Serina, ‘Action-failure resilient planning’, in *ECAI 2023*, 44–51, IOS Press, (2023).

[2] Mohannad Babli, Óscar Sapena, and Eva Onaindia, ‘Plan commitment: Replanning versus plan repair’, *Engineering Applications of Artificial Intelligence*, **123**, (2023).

[3] Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina, ‘Plan stability: Replanning versus plan repair’, in *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS-2006)*, pp. 212–221. AAAI, (2006).

[4] Alfonso Gerevini and Ivan Serina, ‘Efficient plan adaptation through replanning windows and heuristic goals’, *Fundamenta Informaticae*, **102**(3-4), 287–323, (2010).

[5] Malik Ghallab, Dana S. Nau, and Paolo Traverso, *Automated Planning and Acting*, Cambridge University Press, 2016.

[6] Jörg Hoffmann, Julie Porteous, and Laura Sebastia, ‘Ordered landmarks in planning’, *J. Artif. Intell. Res.*, **22**, 215–278, (2004).