The Knowledge Engineering Review (2025), **40**, e7, pp. 1–23
doi:10.1017/S0269888925100076

RESEARCH ARTICLE

Optimally stable plan repair

Alessandro Saetti and Enrico Scala

Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia, Brescia, Italy
Corresponding author: Alessandro Saetti; Email: alessandro.saetti@unibs.it

Received: 24 February 2025; **Revised:** 14 October 2025; **Accepted:** 29 October 2025

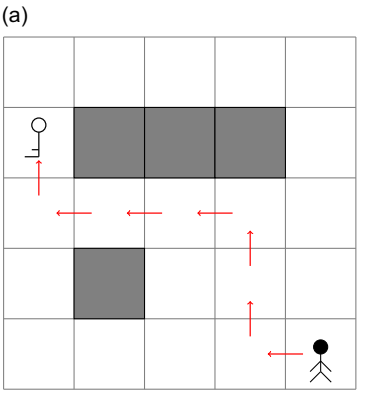
Abstract
Plan repair is the problem of solving a given planning problem by using a solution plan of a similar problem. This paper presents the first approach where the repair has to be done optimally, that is, we aim at finding a minimum distance plan from an input plan; we do so by introducing a number of compilation schemes that convert a classical planning problem into another where optimal plans correspond to plans with the minimum distance from an input plan. We also address the problem of finding a minimum distance plan from a set of input plans, instead of just one plan. Our experiments using a number of planners show that such a simple approach can solve many problems optimally and more effectively than replanning from scratch for a large number of cases. Also, the approach proves competitive with LPG-adapt, a state-of-the-art approach for the plan repair problem.

1. Introduction

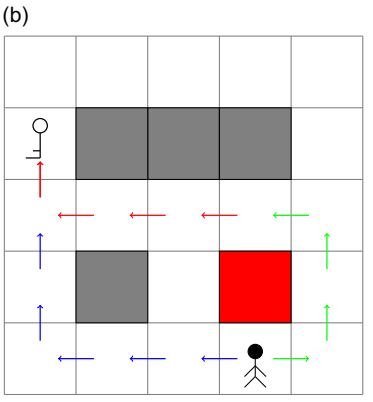
Agents acting in real worlds have to deal with uncertainty; therefore, any plan can become invalid at some point. An approach to address such a problem is to either anticipate all possible contingencies at planning time (e.g., through conformant or contingent planning models Smith & Weld, 1998; Bonet, 2010) or deal with them as soon as something disruptive actually arises. When, however, there is no model about the uncertainty or the number of unexpected situations is not bounded, it is nevertheless necessary to have some mechanism to come up with a new course of actions. A solution to such a problem is replanning. That is, as soon as the agent recognizes that its plan does not work anymore, it formulates a new problem and computes new plans from that point onward.

Although replanning can be effective in certain scenarios (Yoon *et al*., 2007; Ruml *et al*., 2011) and plan reuse can be even more challenging (Nebel & Koehler, 1995), it is widely recognized that repairing the current course of action is often significantly more effective in practice (Gerevini & Serina, 2010; Scala & Torasso, 2014). Not only can the plan be more easily recoverable so limiting the computational burden, but, also, the number of modifications to apply can be limited, therefore optimizing the stability of the system. Plan stability is important when humans have already validated the planning activities under execution, and the effort required for such a validation is considerable. In this case, stable plans reduce the cognitive load on human observers by ensuring coherence and consistency of behaviors (Fox *et al*., 2006). Consider for instance the example of Figure 1. We have an agent who needs to get a key somewhere in an environment. Unfortunately, while executing the plan shown in Figure 1(a), the agent is found to be in front of a wall that was not visible at planning time, as depicted in Figure 1(b). How we recover from that situation? Of course, the agent can compute a new plan (the blue arrows in Figure 1b), but such a plan has no guarantee to account for what the agent decided before. A more stable plan is instead the one starting with the green arrows (the part of the plan which was not computed before) followed by the red one (the part of the plan which was computed offline).

Cite this article: A. Saetti and E. Scala. Optimally stable plan repair. *The Knowledge Engineering Review* 40(e7): 1–23. https://doi.org/10.1017/S0269888925100076



First plan is in red.



Red obstacles are observed only when the agent gets to position (3,0). Green arrows are those allowing the agent to reconnect to the path implied by the first plan. Blue arrows achieves the key ignoring the path implied by the first plan.

Figure 1. *An agent moves in the cardinal directions trying to get to the key. (0,0) is the left most cell on the bottom, so the agent starts at position (4,0) and is tasked to go to position (0,3).*

Previous solutions to the problem of repairing a plan have however no guarantees that the recovered plan is the most stable plan that can be computed (Koenig & Likhachev, 2002; van der Krogt & de Weerdt, 2005; Fox *et al*., 2006; Gerevini & Serina, 2010; Garrido *et al*., 2010; Scala, 2014; Scala & Torasso, 2015; Höller *et al*., 2018; Goldman *et al*., 2020). Following on this line of research, in this paper, we take plan stability as the primary objective of the plan repair problem. In particular, we present a compilation-based approach that, given a planning problem and a plan, solves the plan repair problem with the guarantee of finding plans that are at the minimum distance from the input plan. We do so by making heavy use of a cost function that captures the implications of using actions that do not belong to the input plan, using extra occurrences of actions in the input plans, and that captures whenever the agent is not using actions in the input plan. More specifically, we focus our attention on using the plan distance metric defined by Fox *et al*. (2006) as cost function. This distance is useful in all scenarios where the presence/absence of an action is a good proxy to understand the behavior of an agent; other scenarios can benefit from more powerful notions of distance, but we leave the study on how to deal with them computationally as a future investigation. The compilation we propose produces a classical planning problem that can be handled by any cost-sensitive planner.

In many applications, there may be multiple plans already validated by human experts, and all the accumulated experience stored in these plans is considered equally valuable and worth reusing. This is analogous to case-based planning, where a plan library can store multiple plans that solve the same problem (Muñoz-Avila & Cox, 2008; Gerevini *et al*., 2013). So, in this paper we also tackle a new variant of the repair problem with the aim of finding a plan with minimum distance from a *set* of input plans. Like for the stability problem from a single plan, we addressed this problem by a compilation-based approach that computes the minimum distance plan w.r.t. any plan in the set of input plans.

To understand the practicability of our compilation, we report on an extensive experimental analysis comparing various optimal and satisficing planners with and without our compilation, over the planning problems from the 2018 edition of the planning competition, as well as with LPG-adapt, a state-of-the-art system that natively repairs plans. Our results show that optimal plan repair is not only feasible but also competitive with greedy approaches, such as LPG-adapt. Furthermore, it serves as an effective alternative to replanning from scratch across a wide range of domains.

This paper extends our initial work on the topic, that is, Saetti and Scala (2022). In particular, we provide: (i) a complete running example that helps understanding the problem and the compilation solution

that we study starting from the background, (ii) two variants of the compilation scheme that we initially proposed, one of which is a simplification of our initial scheme, while the other one works at a lifted level, (iii) an extension of the problem and compilation to the multi-plan case, that is, when we want to find the optimal stable plan trying to repair from a set of input plans, (iv) extended proofs showing the soundness, completeness, and optimality of our initial compilation scheme, as well as new proofs for the proposed variants of our initial scheme, (v) novel experiments, and (vi) a deeper related work analysis.

The paper starts with some background in Section 2 and provides the necessary definitions for the optimal repair problem in Section 3. Then, Sections 4 and 5 provide the compilation methods for the single and multi-plan repair case, together with soundness and completeness proofs. We end our paper with experiments (Section 7) and related work (Section 8).

2. Background

A *planning domain* $\mathcal{D}$ is a pair $\langle P, O\rangle$, where P is a set of predicates with associated arity of a first-order language, and O is a finite set of operators with associated arity. Predicates and operators of arity n are called n-ary predicates and n-ary operators. In the rest of the paper, we used both terms 'lifted action' and 'operator' interchangeably.

We define an n-ary *operator* op as a tuple $\langle \mathsf{par}(op), \mathsf{pre}(op), \mathsf{eff}(op)\rangle$, where $\mathsf{par}(op)$ is a tuple of n distinct variables $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\mathsf{pre}(op)$ is a formula over P, and $\mathsf{eff}(op)$ is a consistent set of positive and negative literals over P. $\mathsf{par}(op)$, $\mathsf{pre}(op)$, and $\mathsf{eff}(op)$ are called the parameters, preconditions, and effects of the operator op, respectively.

The *ground predicate* $p(c_1, \ldots, c_n)$ of a predicate $p \in P$ that applies to n variables $\boldsymbol{x} = (x_1, \ldots, x_n)$ is obtained by replacing x_i with constant c_i for $i \in \{1, \ldots, n\}$. Similarly, the *ground action* $a = op(c_1, \ldots, c_n)$ of an n-ary operator op w.r.t. constants $c_1, \ldots, c_n$ is the pair $\langle \mathsf{pre}(a), \mathsf{eff}(a)\rangle$, where $\mathsf{pre}(a)$ (resp. $\mathsf{eff}(a)$) is obtained by replacing the i-th parameter of $\mathsf{par}(op)$ in $\mathsf{pre}(op)$ (resp. $\mathsf{eff}(op)$) with c_i for $i \in \{1, \ldots, n\}$. In the following, with $\mathsf{eff}^+(a)$ and $\mathsf{eff}^-(a)$ we indicate the positive and the negative effects of a. Moreover, we use the term *object* to refer to the constants used for grounding, refer to a ground predicate as a *fact*, and, for brevity, refer to a ground action simply as an *action*.

Let F be a set of facts, a *state* s is a subset of F with the meaning that if $f \in s$ then f is true in s, otherwise it is false. An action a is applicable in s if and only if $s \models \mathsf{pre}(a)$. The execution of an action a in s, denoted by $s[a]$, generates a new state s' such that $s' = (s \setminus \mathsf{eff}^-(a)) \cup \mathsf{eff}^+(a)$.

A *classical planning problem* $\mathcal{P}$ of a planning domain $\mathcal{D}$ and a set of constants $\mathcal{C}$ is the tuple $\langle F, A, I, G, c\rangle$, where F is the set of facts obtained from $\mathcal{C}$ and the set of predicates P of $\mathcal{D}$, A is a set of actions obtained from $\mathcal{C}$ and the set of operators O of $\mathcal{D}$, $I \subseteq F$ is a state called the initial state, G is a formula over F, $c : A \mapsto \mathbb{R}_{\geq 0}$ is a function associating non-negative costs to actions. We will also refer to $\mathcal{P}^l = \langle \mathcal{D}, \mathcal{C}, I, G, c^l\rangle$ with $c^l : O \times \mathcal{C} \mapsto \mathbb{R}_{\geq 0}$ as the lifted version of a classical planning problem (before grounding); intuitively, function c^l differs from c in that we need to consider the mapping starting from operators and some constants in order to obtain the non-negative cost of a concrete ground action.

Let $\hat{\Pi}_{\mathcal{P}}$ be the set of all possible plans for a problem $\mathcal{P}$. A *plan* $\pi \in \hat{\Pi}_{\mathcal{P}}$ is a sequence of actions of $\mathcal{P}$. The application of $\pi = \langle a_1, \ldots, a_n\rangle$ in a state s_0 gives the sequence of states $s_0[\pi] = \langle s_0, \ldots, s_n\rangle$ where $s_i = s_{i-1}[a_i]$ for all $1 \leq i \leq n$. Plan π is said to be a *solution* for $\mathcal{P}$ from $I = s_0$ if and only if, let $I[\pi] = \langle s_0, \ldots, s_n\rangle$ be the sequence of states obtained applying the plan by starting from the initial state, for all $1 \leq i \leq n$ we have that $s_{i-1} \models \mathsf{pre}(a_i)$ and $s_n \models G$. Function $cost : \hat{\Pi}_{\mathcal{P}} \to \mathbb{R}$ measures the cost of a plan for $\mathcal{P}$. The *cost* of a plan π is the sum of the costs of all the actions in π, that is, $cost(\pi) = \sum_{a \in \pi} c(a)$. A plan π is said to be *optimal* if it is a solution and there is no plan π' such that π' is a solution for $\mathcal{P}$ and $cost(\pi) > cost(\pi')$. Let $\pi = \langle a_1, \ldots, a_i, \ldots, a_j, \ldots, a_n\rangle$ with $1 \leq i \leq j \leq n$. In the rest of the paper, $\pi^{i,j}$ denotes the action sequence $\langle a_i, \ldots, a_j\rangle$, that is, the (intermediate) actions between the i-th action and the j-th action in π.

Running Example. As an example of a classical planning problem, we can model the navigation task shown in Figure 1 as follows. Let n and m be the boundaries of our grid. One can use n objects to represent x-coordinates and m objects for the y-coordinates. Then, predicate $at(x,y)$ models whether the

agent is in position (*x*,*y*); predicate $conn(x_0, y_0, x_1, y_1)$ models whether the two cells (x_0, y_0) and (x_1, y_1) are connected or not. Given all these predicates, we can build an operator $move(x_0, y_0, x_1, y_1)$ that models the movement from a cell to another. The precondition of the operator simultaneously ensures that $conn(x_0, y_0, x_1, y_1)$ and $at(x_0, y_0)$ are true; the effect instead models that $at(x_0, y_0)$ will be false at the end of the execution, and that $at(x_1, y_1)$ will instead be true. The initial state and the goals are also quite trivial. The initial state contains as many facts as connections in the grid (the obstacles are implicit), and a fact representing the initial position of the agent. As a goal we simply require the agent to be in the position where the key is. For the grid in Figure 1(a), this can be represented by having $\{at(4, 0)\} \subset I$ and $G = \{at(0, 3)\}$. Note that here we use numbers for objects for convenience, while they are treated just as strings. A plan π_1 solving such a task is the sequence $\langle move(4, 0, 3, 0), move(3, 0, 3, 1), move(3, 1, 3, 2), move(3, 2, 2, 2), move(2, 2, 1, 2), move(1, 2, 0, 2), move(0, 2, 0, 3)\rangle$, represented by the red arrows in the grid of Figure 1(a). For the problem depicted in Figure 1(b), $\{at(3, 0)\} \subset I$ and the goal G is the same as for Figure 1(a). A plan π_2 solving such a task is the sequence $\langle move(3, 0, 2, 0), move(2, 0, 1, 0), move(1, 0, 0, 0), move(0, 0, 0, 1), move(0, 1, 0, 2), \pi_1^{7,7}\rangle$, represented by the blue arrows and a red arrow in the grid of Figure 1(b). Of course, there might exist more than one plan solving a planning problem. Another solution is a plan, π_3, consisting of $\langle move(3, 0, 4, 0), move(4, 0, 4, 1), move(4, 1, 4, 2), move(4, 2, 3, 2), \pi_2^{4,7}\rangle$, represented by the green arrows and the red arrows in the grid of Figure 1(b).

3. The repair planning problem

A repair planning problem combines a planning problem and a plan attempting to solve such a problem. We formally define a repair planning problem as follows.

Definition 1 (Repair problem). *A repair planning problem is the pair $\langle \mathcal{P}, \pi \rangle$, where $\mathcal{P}$ is a planning problem and π is some sequence of actions from actions in $\mathcal{P}$.*

A plan solves the repair problem if and only if it solves $\mathcal{P}$, too. What differs from the planning problem is the notion of optimality.

Definition 2 (Optimal plan repair). *A plan π^* is said to be optimal for a repair problem $\langle \mathcal{P}, \pi \rangle$ if it is the one that minimizes the distance from the input plan π and solves $\mathcal{P}$. Formally, let $D : \hat{\Pi}_{\mathcal{P}} \times \hat{\Pi}_{\mathcal{P}} \to \mathbb{R}$ be a function measuring the distance between two plans for problem $\mathcal{P}$. Then:*

$$\pi^* = \underset{\pi' \text{ solves } \mathcal{P}}{\arg\min} D(\pi, \pi').$$

Intuitively, given the input plan π and any solution of the replair planning problem we are looking for the most similar plan, and do so by characterizing the distance between any two plans by the function D which is defined in the next paragraph. The input plan π might be a plan that was already used and validated by humans and, as mentioned before, finding a solution plan that is similar to π would guarantee coherence and consistency with already used behaviors and would reduce the cognitive load on humans.

In this work, we use a simple notion of *plan distance*, which considers a plan more stable if it is closer to the input plan in terms of number of different actions they contain, while we ignore the action ordering in plans. This is the same notion of plan distance proposed by Fox *et al*. (2006). Let π and π' be two plans. For the definition of plan distance, we treat π and π' as multisets of actions. We use multisets rather than simple action sets because a plan may contain multiple instances of the same action. Formally, the distance D between π and π' is defined as

$$D(\pi, \pi') = |\pi \setminus \pi'| + |\pi' \setminus \pi| \tag{1}$$

Note here that we are operating over multisets, that is, repetition of the same action does matter. For this reason, operator $\setminus$ is defined so that $\pi \setminus \pi'$ contains $m - l$ instances of action a iff π and π', respectively, contain m and l instances of a and $m > l$; $\pi \setminus \pi'$ contains 0 instances of a otherwise. The smaller

the distance between the new plan and the input plan, the more *stable* the new plan is with respect to the input plan.

Running Example (cont.) As for the plans depicted in Figure 1, let π_1 be the plan represented by the red arrows in the grid of Figure 1(a); π_2 be the plan represented by the blue arrows and a red arrow in the grid of Figure 1(b); π_3 be the plan represented by the green and red arrows in the grid of Figure 1(b). The distance between π_1 and π_2 is 11, the sum between 5 and 6, because the 5 actions in $\pi_2^{1,5}$ are not in π_1 (the blue arrows in the figure), and the 6 actions in $\pi_1^{1,6}$ are not in π_2. The distance between π_1 and π_3 is 7, the sum between 4 and 3, because the 4 actions in $\pi_3^{1,4}$ are not in π_1 (the green arrows in the figure), and the 3 actions in $\pi_1^{1,3}$ are not in π_3. Consider the repair planning problem defined as the grid-based navigation problem instance depicted in Figure 1(b) and plan π_1. Plan π_3 is the optimal repair plan for this instance of the problem because there exists no plan with a distance from π_1 lower than the distance between π_3 and π_1.

In general, there might be more than one plan that were already successfully used to solve problems similar to the new encountered planning problem $\mathcal{P}$. Thereby, the goal of ensuring coherence and consistency with already used behaviors is achieved if the solution of the new encountered problem is similar to any already successfully used plan. For this reason, we define an extension of the repair planning problem that combines a planning problem and a set of input plans.

Definition 3 (Multi-repair planning problem). *A multi-repair planning problem is the pair* $\langle \mathcal{P}, \Pi \rangle$, *where* $\mathcal{P}$ *is a planning problem and* $\Pi \subseteq \hat{\Pi}_{\mathcal{P}}$ *is a set of plans for* $\mathcal{P}$, *each of which is some sequence of actions from actions in* $\mathcal{P}$.

Like for the single repair planning problem, we say that a plan solves the multi-repair planning problem if and only if it solves $\mathcal{P}$. Differently from the single repair problem, the notion of optimality for the multi-repair planning problem takes into account all the input plans in the problem definition.

Definition 4 (Optimal multi-repair plan). *A plan is said to be optimal for a multi-repair problem if it is the one that minimizes the distance from any input plan* $\pi \in \Pi$ *and solves* $\mathcal{P}$. *Formally, let* $D : \hat{\Pi}_{\mathcal{P}} \times \hat{\Pi}_{\mathcal{P}} \rightarrow \mathbb{R}$ *be a function measuring the distance between two plans for problem* $\mathcal{P}$. *Then:*

$$\pi^* = \underset{\pi' \text{ solves } \mathcal{P}}{\arg\min} \left(\min_{\pi \in \Pi} D(\pi, \pi') \right).$$

4. Solving the optimal repairing problem

This section presents a compilation that takes in input a repair problem and generates a novel classical problem whose optimal solutions are optimal solutions for the repair planning problem. Our compilation creates a number of copies for each action in the plan and customises the cost function for keeping track of those actions undermining the optimality of the solution. The compilation also makes use of a number of additional, dummy predicates, whose purpose is to monitor the already executed actions. Specifically, the new set of predicates, denoted F', extends the original set F with

- a dummy atom w, whose truth value distinguishes two stages: the first stage constructs the solution, while the second stage evaluates its quality;
- a set of predicates that serve as proxies for the actions in the input plan, and
- a set of predicates that, for each action in the input plan, track the number of times that action has already been executed in the new plan.

The new initial state, I', extends the original initial state I by including w, indicating that the planner starts at the first stage, and by adding special atoms that specify, for each action a in the input plan, that zero occurrences of a have been executed initially. The new set of actions, A', is divided into four subsets:

- A_0, action instances that appear in the input plan,
- A_1, actions not present in the input plan,
- A_2, actions from the input plan that have already been used in the new plan, and
- A_3, action instances from the input plan that are not considered for the new plan.

The new goal state, G', extends the original goal set by adding literals that force the planner to either include the actions of the input plan in the new plan or account for them when defining the plan cost. Finally, the compilation scheme defines a cost function c' that encourages the use of actions from the input plan, while discouraging the use of actions not in the input plan, the omission of input-plan actions, as well as an excessive reuse of input-plan actions. In this compilation, we work at the grounding level. That is, we assume that all actions have been instantiated against the objects of our problem. Grounding can be achieved using well-known reachability based techniques (Helmert, 2006). Then, in the next section, we also show a compilation that does not require grounding upfront.

In what follows, we formally explain the grounded compilation, that we call Resa (REpair for StAbility). We use two functions to simplify notation. Function $B(a, i, \pi)$ counts the number of occurrences of input action a *Before* step i in plan π; formally, $B : A \times \mathbb{N} \times 2^{A^*} \to \mathbb{N}$. Function $M : A \times 2^{A^*} \to \mathbb{N}$ returns the number of repetitions of an action in a plan.

Definition 5 (Resa Compilation). *Let $\mathcal{P} = \langle F, A, I, G, c\rangle$ be a planning problem, and $\pi = \langle a_1, \ldots, a_n\rangle$ be a sequence of n actions in A.* Resa *takes in input $\mathcal{P}$ and π, and generates a new planning problem $\mathcal{P}' = \langle F', A_0 \cup A_1 \cup A_2 \cup A_3 \cup \{switch\}, I', G', c'\rangle$ such that:*

$$F' = F \cup \{w\} \cup \bigcup_{i\in\{1,\ldots,n\}} d_i \cup \bigcup_{a\in\pi} \{p_a^i \mid 0 \le i \le M(a,\pi)\}$$

$$I' = I \cup \{w\} \cup \bigcup_{a\in\pi} p_a^0$$

$$A_0 = \bigcup_{a_i\in\pi} \langle \mathsf{pre}(a) \wedge w \wedge p_a^{B(a,i,\pi)}, \mathsf{eff}(a) \cup \{p_a^{B(a,i,\pi)+1}, \neg p_a^{B(a,i,\pi)}, d_i\}\rangle$$

$$A_1 = \bigcup_{a\in A\setminus set(\pi)} \langle \mathsf{pre}(a) \wedge w, \mathsf{eff}(a)\rangle$$

$$A_2 = \bigcup_{a\in set(\pi)} \langle \mathsf{pre}(a) \wedge w \wedge p_a^{M(a,\pi)}, \mathsf{eff}(a)\rangle$$

$$A_3 = \bigcup_{i\in\{1,\ldots,n\}} \langle \neg w \wedge \neg d_i, \{d_i\}\rangle$$

$$switch = \langle w, \{\neg w\}\rangle$$

$$G' = G \wedge \bigwedge_{i\in\{1,\ldots,n\}} d_i$$

$$c'(a) = \begin{cases} 0 & \text{if } a \in A_0 \cup \{switch\} \\ 1 & \text{if } a \in A_1 \cup A_2 \cup A_3 \end{cases}$$

Intuitively, the compilation reshapes the planning problem in such a way that all actions that do not contribute in increasing the distance from the previous plan are given cost 0. Action instances that instead were not in the plan (the actions set A_1) or that were in the plan but we have already used them (set A_2), or that were in the plan but are not going to be considered for the new plan (set A_3) are given cost 1. Indeed, the goal formula requires that, besides achieving the problem goals, all actions of the input plan are processed. This is achieved by formulating a fact d for each action within the starting plan. Such a fact can either be made true by actions from set A_0 whose cost is equal to 0 – indeed that corresponds to the case in which we did replicate what the plan was before, or actions from A_3 whose cost is equal to 1. Basically, actions from A_3 are the *give-up* actions, that is, they emulate whether the planner gave up in trying to pick actions from the input plan and for that it pays an extra cost of 1 for each one of them.

These dummy actions share some analogy with previous work to compile soft goals away (Keyder & Geffner, 2009).

In order to consider whether some action instance has been already considered we make use of the aforementioned functions B and M. These functions make it possible to create as many p_a predicates as needed to keep track of the number of instances of used action a, and to monitor whether limit $M(a, \pi)$ has been hit. The initial state contains an atom p_a^0 for each action a used in the input plan. As mentioned above, this special atom indicates that the number of occurrences of action a contained in the (partial) plan under construction is initially zero. The first instance of action a added to the partial plan is an action in A_0 that makes p_a^0 false while achieves p_a^1. In general, atom p_a^i represents the fact that the number of occurrences of action a in the plan under construction is i. The last instance of action a that is part of set A_0 and is added to the plan under construction makes $p_a^{M(a,\pi)}$ true. This means that all the occurrences of action a in the input plan have been added to the plan under construction. This way, precondition $p_a^{M(a,\pi)}$ of the copy of action a in set A_2 is true, any extra occurrence of action a that will be eventually added to the plan under construction will be such an action copy of a in A_2 and will increase the distance from the starting plan. The *switch* action is what marks the end of the first stage and the beginning of the second, finalizing the search of the plan and starting the collection of the *give-up* actions from A_3. Indeed, none of the actions is executable after *switch* but those in A_3. This is guaranteed by dummy literal w, which is true in the initial state, required by any action in sets A_0, A_1, and A_2, and made false by action *switch*; in addition, any action in set A_3 requires that w is false.

Running Example (cont.) In order to see how the compilation works in practice, let us take a look at our grid-based navigation example. Consider the repair planning problem defined as the grid-based navigation problem instance depicted in Figure 1(b), and the input plan defined as the plan represented by the red arrows in the grid of Figure 1(a). First, we augment the set of predicates to include those representing the various action instances. Therefore, we define $F' = F \cup \{d_1, \cdots, d_7\} \cup \{w\} \cup \bigcup_{i\in\{0,1\}} \{p^i_{m_{4,0,3,0}}, p^i_{m_{3,0,3,1}}, p^i_{m_{3,1,3,2}}, p^i_{m_{3,2,2,2}}, p^i_{m_{2,2,2,1}}, p^i_{m_{2,1,2,0}}, p^i_{m_{2,0,3,0}}\}$. Here, the subscripts denote the names of the move actions as introduced earlier, with parameters written as subscripts for clarity. The initial state is updated to include the switch phase predicate w, and the predicates indicating that the number of occurrences of the actions in the input plan is initially zero in the plan under construction. That is, we define $I' = I \cup \{w\} \cup \{p^0_{m_{4,0,3,0}}, p^0_{m_{3,0,3,1}}, p^0_{m_{3,1,3,2}}, p^0_{m_{3,2,2,2}}, p^0_{m_{2,2,2,1}}, p^0_{m_{2,1,2,0}}, p^0_{m_{2,0,3,0}}\}$. In this plan, there is only one instance of each action, that is, $M(a, \pi) = 1 \ \forall a \in \pi$, so function B will only be used to differentiate whether the action has been used or not—and this is also the reason why we have only two predicates per action.

For simplicity, we will now consider the encoding of two actions *move*(3, 1, 3, 2) and *move*(2, 1, 3, 2), symbolically denoted as $a = m_{3,1,3,2}$ and $b = m_{2,1,3,2}$. Action a is in the input plan, while b is not part of such a plan. For action a, we create an instance in set A_0 with the conjunction between the precondition formula of a and atom $p^0_{m_{3,1,3,2}}$ as precondition, and the set of effects of a extended by atoms $\neg p^0_{m_{3,1,3,2}}$, $p^1_{m_{3,1,3,2}}$, and d_3 as set of effects. Atom d_3 in the set of effects of a indicates that, when action a is executed, the third action in the input plan is included into the plan under construction. Additionally, we need a copy of a, labeled a', which belongs to set A_2, where $\mathsf{pre}(a') = \mathsf{pre}(a) \wedge w \wedge p^1_{m_{3,1,3,2}}$ and $\mathsf{eff}(a') = \mathsf{eff}(a)$. Another difference between these two instances is the cost: the interpretation in A_0 has a cost of 0 (as it reuses part of the previous plan), while the interpretation in A_2 has a cost of 1, representing the extra cost of executing the action a number of times greater than in the input plan.

For action b, which is not part of the plan, we need only one interpretation, stored in set A_1. Such an instance is the same as b but its cost is 1 and w is part of its precondition formula. Indeed, this action must be executed before the predicate w becomes false, as w tracks whether the plan is complete and when the cost of the repair should be calculated.

Finally, set A_3 for this problem consists of 7 actions, one for each instance of each action in the plan. That is: $A_3 = \{\langle \neg w \wedge \neg d_1, d_1\rangle, \cdots, \langle \neg w \wedge \neg d_7, d_7\rangle\}$. The actions in A_3 have cost equal to 1, because each of them represents the absence of an action of the input plan from the solution for the repair planning problem. The formulation of the goal is straightforward. Indeed, what we would only need is to conjoin

the goal formula G with all atoms d_i. This way we know for sure that either we used an action instance from the plan or we add some cost for its absence.

4.1. Properties of RESA

In this section, we study some properties of RESA. In particular, we prove that RESA is sound, complete and always generates optimal solutions for the repair problem it is encoding. Moreover, RESA size is polynomial in the input task.

Theorem 1 (Soundness). *Let $\mathcal{R} = \langle \mathcal{P}, \pi \rangle$ be a plan repair problem.* RESA *transforms $\mathcal{R}$ into a problem $\mathcal{P}'$ such that if $\mathcal{P}'$ is solvable so is $\mathcal{R}$.*

Proof. Each plan computed by RESA has one of the following two forms, for $k \geq 0, m \geq 0$:

$$\langle \underbrace{a_1, \ldots, a_k}_{\pi'} \rangle,$$

$$\langle \underbrace{a_1, \ldots, a_k}_{\pi'}, \mathit{switch}, \underbrace{a_{k+1}, \ldots, a_m}_{\pi''} \rangle$$

The RESA plan has the same form as the first plan if all the occurrences of the actions in the input plan are in π', has the latter form otherwise. Each action in the first part of the plan, π', is in sets A_0, A_1, or A_2, since actions in A_3 has $\neg w$ as precondition, w is true in the initial state, and the only action that made w false is *switch*. Similarly, each action in the second part of the plan, π'', is in set A_3, since every other action has w as precondition and w becomes false after the execution of the *switch* action. For construction, each action in A_0, A_1, or A_2 has the same preconditions and effects in set F as an action of $\mathcal{P}$, plus some additional preconditions and effects in $F' \backslash F$. In practice, all such actions require the preconditions of the original actions to hold before their execution, and the original effects are preserved; the remaining preconditions and effects only monitor whether the action instance is playing the role of an action that was in the input plan, or not. The goal of $\mathcal{P}'$ that are in F are the same as in $\mathcal{P}$. Action *switch* and the action in π'' do not change the truth value of facts in F, therefore the goals in F are achieved at the end of π'. It follows that the sequence of original actions from which we derived the action in π' is a plan solving $\mathcal{P}$. □

Theorem 2 (Completeness). *Let $\mathcal{R} = \langle \mathcal{P}, \pi \rangle$ be a plan repair problem.* RESA *transforms $\mathcal{R}$ into a problem $\mathcal{P}'$ that is solvable if $\mathcal{R}$ is solvable.*

Proof. For any valid plan $\pi_1 = \langle a_1, \ldots, a_m \rangle$ of $\mathcal{R}$ there is (at least) one plan π_2 in the compiled problem $\mathcal{P}'$ which can be constructed as follows. Plan π_2 has the first m actions that are copies of actions in π_1 from set A_0, A_1, or A_2. We show that the j-th action of π_2 is applicable in $\pi_2^{1,j-1}[I']$ for $1 \leq j \leq m$. For each action a in π_1, we distinguish three cases: (i) a is an instance of an action that is not part of the input plan; (ii) a is an instance of an action that has k occurrences in the input plan, is the i-th occurrence of the action in π_1 and $i \leq k$; (iii) a is an instance of an action that has k occurrences in the input plan, is the i-th occurrence of the action in π_1 and $i > k$. Let a be the j-th action in π_1. For case (i), a has a copy in set A_1 that is applicable in $\pi_2^{1,j-1}[I']$ because the action preconditions and effects that are in F are the same as in $\mathcal{R}$. The only difference between a and its copy is that the copy has w as a precondition. Such a precondition is true in the state where the a's copy is executed, because w is true in the initial state and the copies of the actions in π_2 executed before the copy of a are in sets A_0, A_1,or A_2, and therefore have not $\neg w$ in the effects. For case (ii), we distinguish two sub-cases. a is the first occurrence of an action in π_1; a is the i-th occurrence of an action in π_1 and $i > 0$. Then, a has a copy in set A_0 that is applicable in $\pi_2^{1,j-1}[I']$ because the only preconditions different from the original version of a are w and $p_a^{B(a,i,\pi)}$; w is true in the initial state and is not an effect of the actions in A_0, A_1, and A_2; if a is the first occurrence in π_1, $B(a, i, \pi) = 0$, and p_a^0 is true in the initial state and is not an effect of the action in π_2 preceding the copy of a; if a is the i-th occurrence in π_1, then $p_a^{B(a,i,\pi)}$ is achieved by the copy of the $(i-1)$-th occurrence of action a in π_2, and is not falsified by any action between the $i-1$-th and the

i-th occurrence of a in π_2. For case (iii), a has a copy in set A_2 that is applicable in $\pi_2^{1,j-1}[I']$, because the only preconditions different from the original action are w and $p_a^{M(a)}$; w is true for the same reason mentioned before; $M(a) = k$, and p_a^k is achieved by the copy of the k-th occurrence of action a in π_2 and not falsified by any other action.

The difference between the goals of the compiled problem and the original ones is a set of n atoms d_i, where n is the number of actions in the input plan. If these extra goals have been achieved in π_2 by the copies of the actions of π_1, then π_2 is a solution for the compiled problem. Otherwise, π_2 will have a number of extra actions executed after the copies of the actions in π_1. Such actions consist of action *switch* followed by actions in set A_3. Indeed, if after the execution of the copies of the actions in π_1 an atom d_i was false, it could be achieved by an action in A_3, since the preconditions of such an action are $\neg d_i$ and $\neg w$, and $\neg w$ is achieved by the *switch* action. □

Theorem 3 (Optimality). *Let $\mathcal{R} = \langle \mathcal{P}, \pi \rangle$ be a plan repair problem.* RESA *transforms $\mathcal{R}$ into a problem $\mathcal{P}'$ such that the cost of the optimal solution for $\mathcal{P}'$ equates that of the solution for $\mathcal{R}$.*

Proof. Let π_1 be an optimal plan for $\mathcal{R}$. We can construct a solution π_2 for $\mathcal{P}'$ by choosing for each action in π_1 its copy from $A_0 \cup A_2$ according to the prefix of π_1 if the action is also in π, its copy from A_1 otherwise; subsequently, adding action *switch* and one action from A_3 for each action instance in π that is not in π_1. According to Equation (1), the cost of plan π_1 is the sum between the number of actions that are in π but not in π_1 plus the number of actions that are in π_1 but not in π.

As for the cost of π_2, each action from sets A_1, A_2, and A_3 has cost 1, any other action has zero cost. Therefore, the cost of π_2 is the sum between the number of actions in π_2 that are from either A_1, A_2 or A_3. An action in π_2 is from A_1 iff the action is in π and not in π_1. An action in π_2 is from A_2 iff π_1 contains a number of occurrences of such an action greater than π. Therefore, the number of actions in π_2 that are from A_1 and A_2 is equal to the number of actions that are in π_1 but not in π.

Assume that the input plan π contains n actions. Then, the goals of the compiled problem contains a dummy literal d_i for $1 \leq i \leq n$. Such a literal is achieved by an action from either A_0 or A_3. π_2 contains an action from A_0 achieving d_i iff plan π_1 contains the i-th action of π; it contains an action from A_3 achieving d_i, otherwise. Therefore, the number of actions from A_3 contained in π_2 is equal to the number of actions that are in π but not in π_1. It follows that the cost of π_2 is equal to the cost of π_1.

Let us assume for contradiction that there exists another solution for the compiled problem with cost lower than π_2. Then, we could derive a plan for $\mathcal{P}$ from such a solution formed by the original actions from which we derived the action copies in π_2. Such a plan would have a cost lower than π_1. Of course, this is not possible because π_1 is an optimal plan for $\mathcal{P}$. Therefore, we can state that π_2 is an optimal plan for the compiled problem. Analogously for the case in which we assume a larger cost. □

Theorem 4 RESA *is linear on the size of $\mathcal{R} = \langle \mathcal{P}, \pi \rangle$.*

Proof. Set F' of the compiled problem contains two extra atoms for each action in π. Similarly, set I' and the goal formula G' of the compiled problem contains an extra atom for each action in π. The number of actions in A_0 and A_3 is still equal to the number of action in π. The sum between the number of actions in A_1 and A_2 is equal to the number of actions of the repairing problem. It follows that the compiled problem contains only $O(\pi)$ new atoms and actions. □

4.2. An alternative, simplified yet optimality preserving compilation

In RESA, we devise two set of actions A_1 and A_2 to distinguish the case where the agent uses some action which was not in the plan at all (set A_1) from the case where the action was in the plan, but the agent has already executed it a number of times that equates the number of occurrences of such an action in the plan (set A_2). Whenever the agent executes one of these actions, there is a cost to be paid at the end. It turned out, however, that we can avoid making such a distinction and design a compilation that only considers one set of actions. Intuitively, the agent can choose to use either an action that is in the plan, and therefore there is not going to be any cost associated to it, or execute an action from this set

and therefore incur in a cost. This gives rises to a new compilation that we call Simple-Resa, in short S-Resa. S-Resa is as Resa but without set A_2 and with set A_1 involving all actions from the original set of actions A, that is, $A_1 = \bigcup_{a \in A} \langle \mathsf{pre}(a) \wedge w, \mathsf{eff}(a) \rangle$.

Interestingly, where for any solution π of the plan repair problem $\mathcal{R} = \langle \mathcal{P}, \pi \rangle$, there is a solution for Resa with the same cost, we can construct more solutions in S-Resa mapping π. Indeed, each action a in π has a copy in sets A_0 and a copy in set A_1 for S-Resa and, differently from Resa, both these copies might be applicable. For instance, take a plan π' for $\mathcal{R}$ and suppose, for simplicity, that $\pi' = \pi$, and includes only one action, say a. Let a_0 and a_{12} the copies of a in sets A_0 and $A_{1,2}$, respectively. Then, $\langle a_0, \mathit{switch} \rangle$ is a valid plan for Resa, while both $\langle a_{12}, \mathit{switch} \rangle$ and $\langle a_0, \mathit{switch} \rangle$ are valid plans for S-Resa.

The cost of the Resa solutions reflects the costs of the solutions for the original repair problem, while this is not always true for the solutions of S-Resa. However, S-Resa optimal solution still has the same cost of the optimal solution for the original repair problem. Consider our example for which $\pi' = \pi$ is a solution for $\mathcal{R}$. Then, the optimal solution of the repair problem is plan π' with zero cost; the cost of plan $\langle a_{12}, \mathit{switch} \rangle$ is 1, while the cost of plan $\langle a_0, \mathit{switch} \rangle$ is 0, the same as of π'.

Theorem 5 (S-Resa is sound, complete and optimal). *Let $\mathcal{R} = \langle \mathcal{P}, \pi \rangle$ be a repair planning problem.* S-Resa *transforms $\mathcal{R}$ into a problem $\mathcal{P}'$ that is solvable if and only if so is $\mathcal{R}$. Moreover, the optimal solution for $\mathcal{P}'$ equates to that of the solution for $\mathcal{R}$ that minimizes the distance from π.*

Proof. (Soundness) The fact that the solvability of $\mathcal{R}$ is implied by the solvability of $\mathcal{P}'$ follows directly from Theorem 1. Indeed, like the actions in A_1 and A_2 for Resa, the actions in A_1 for S-Resa have the same preconditions and effects in set F as the actions of $\mathcal{P}$. (Completeness) The fact that the solvability of $\mathcal{P}'$ is implied by the solvability of $\mathcal{R}$ follows directly from Theorem 2, because the solutions of Resa are a subset of the solution of S-Resa. Indeed, set A_1 of S-Resa is substantially the union of sets A_1 and A_2 of Resa (without precondition $p_a^{M(a)}$ for the actions in A_2); therefore, all the valid plans for the Resa compilation scheme are also valid plan for S-Resa. (Optimality) The set of valid plans for S-Resa includes the valid plan for Resa. By Theorem 3, the cost of the optimal plan for Resa equates that of the solution for $\mathcal{R}$, and hence there exists also such an optimal plan for S-Resa. □

Resa and S-Resa are quite similar but offer complementary strengths. Resa generates fewer plans because actions in A_2 have an additional precondition, while S-Resa results in slightly lighter representations. We will compare the two empirically.

5. Solving the optimal multi-repair problem

This section presents another compilation transforming a multi-repair planning problem into a classical problem such that the optimal solution of the classical problem are also solution for the original multi-repair problem. The compilation scheme is similar to that used by Resa; we call such a compilation scheme M-Resa.

Definition 6 (M-Resa Compilation). *Let $\mathcal{P} = \langle F, A, I, G, c \rangle$ be a planning problem, and $\Pi = \{\pi_1, \ldots, \pi_n\}$, where each plan $\pi_i = \langle a_1, \ldots, a_{n_i} \rangle$ is a sequence of n_i actions in A.* M-Resa *takes in input $\mathcal{P}$ and Π, and generates a new planning problem $\mathcal{P}' = \langle F', A_0 \cup A_{1,2} \cup A_3 \cup A_4 \cup A_5 \cup \{\mathit{switch}\}, I', G', c' \rangle$ such that:*

$$F' = F \cup \{w,\ q,\ r\} \cup \bigcup_{\substack{\pi_i \in \Pi \\ j \in \{1,\ldots,n_i\}}} d_{ij} \cup \bigcup_{\substack{\pi_i \in \Pi \\ a \in \pi_i}} \{p_a^j \mid 0 \leq j \leq M(a, \pi_i)\} \cup \bigcup_{\pi_i \in \Pi} p_{\pi_i}$$

$$I' = I \cup \{w,\ r\}$$

$$A_0 = \bigcup_{\substack{\pi_i \in \Pi \\ a_j \in \pi_i}} \langle \mathsf{pre}(a) \wedge w \wedge p_{\pi_i} \wedge p_a^{B(a,j,\pi_i)}, \mathsf{eff}(a) \cup \{p_a^{B(a,j,\pi_i)+1}, \neg p_a^{B(a,j,\pi_i)}, d_{ij}\} \rangle$$

$$A_{1,2} = \bigcup_{a \in A} \langle \mathsf{pre}(a) \wedge w, \mathsf{eff}(a) \rangle$$

$$A_3 = \bigcup_{\substack{\pi_i \in \Pi \\ j \in \{1,\dots,n_i\}}} \langle \neg w \wedge \neg d_{ij} \wedge p_{\pi_i}, \{d_{ij}\} \rangle$$

$$A_4 = \bigcup_{\pi_i \in \Pi} \langle r, \bigcup_{a \in \pi_i} p_a^0 \cup \{p_{\pi_i}, \neg r\} \rangle$$

$$A_5 = \bigcup_{\pi_i \in \Pi} \langle p_{\pi_i} \wedge \bigwedge_{j \in \{1,\dots,n_i\}} d_{ij}, q \rangle$$

$$switch = \langle w, \{\neg w\} \rangle$$

$$G' = G \wedge q$$

$$c'(a) = \begin{cases} 0 & \text{if } a \in A_0 \cup A_4 \cup A_5 \cup \{switch\} \\ 1 & \text{if } a \in A_{1,2} \cup A_3 \end{cases}$$

Set $A_{1,2}$ for the M-Resa compilation is exactly the same described in the previous chapter. For each plan π_i in Π we have a set of actions in A_0 and A_3 for M-Resa, which are very similar to the respective sets for Resa. Sets A_0 and A_3 still represent the actions contained in any plan π_i, and the give-up actions for π_i, respectively. Similar to Resa, each action in sets A_0 and A_3 makes true a fact d_{ij}, representing the action executed at time step j of plan π_i. The difference w.r.t. the respective sets for Resa is that the actions in these sets have an additional precondition p_{π_i}. Such a dummy literal represents the intent to adapt plan π_i. Moreover, the M-Resa compilation has two more action sets than Resa, A_4 and A_5. Set A_4 contains an action for each plan π_i in Π, which substantially represents the decision of adapting plan π_i. Indeed, for each plan $\pi_i \in \Pi$, set A_4 contains an action that makes p_{π_i} true, enabling the possible execution of the actions in A_0 and A_3 with p_{π_i} as precondition. Moreover, for each plan π_i of Π, set A_5 contains an action that requires that each action in π_i or its corresponding give-up action is part of the solution for M-Resa.

Theorem 6 (M-Resa is sound, complete and optimal). *Let $\mathcal{R} = \langle \mathcal{P}, \Pi \rangle$ be a multi-repair planning problem.* M-Resa *transforms $\mathcal{R}$ into a problem $\mathcal{P}'$ that is solvable if and only if so is $\mathcal{R}$. Moreover, the optimal solution for $\mathcal{P}'$ equates to that of the solution for R that minimizes the distance from a plan π in Π.*

Proof. The soundness and completeness of M-Resa follows directly from the soundness and completeness of S-Resa (proof of Theorem 5), because for any plan π_i in Π there exists a solution of $\mathcal{P}'$ such that its first action is in A_4, its last action is in A_5, the sequence of its intermediate actions would be a solution $\hat{\pi}$ of the problem compiled by S-Resa for $\mathcal{R} = \langle \mathcal{P}, \pi_i \rangle$. Moreover, the optimality also follows from Theorem 5 because the cost of the actions in A_4 and A_5 is zero, and hence the cost of the solution of $\mathcal{P}'$ is equal to the cost of $\hat{\pi}$. □

6. Lifted optimal plan repair and compilation

We observed that all the compilations analyzed so far tend to result in quite large problems because they rely on the grounded representation. An alternative formulation of the plan repair problem becomes possible if all descriptions are kept in a 'lifted' form. In this lifted formulation, it is also feasible to define a variant that avoids grounding the entire domain upfront. Instead, it operates predominantly at the lifted level, grounding only the actions present in the plan. In the following, we will first define the optimal plan repair problem using lifted terms and then present a lifted compilation.

Definition 7 (Lifted Optimal Plan Repair). *Let $\mathcal{D} = \langle P, O \rangle$ be a domain and $\mathcal{C}$ a set of constants. A Lifted Plan Repair problem is $\mathcal{R}^l = \langle \mathcal{D}, \mathcal{C}, I, G, \pi \rangle$, where I and G are the initial state and the goals such that all literals in set I and those used for formula G are legit groundings w.r.t. P and $\mathcal{C}$, and similarly π is a plan such that all actions in π are legit groundings w.r.t. O and $\mathcal{C}$. A plan π' is a solution for $\mathcal{R}^l$ iff it*

is a solution for $\mathcal{P}^l = \langle \mathcal{D}, \mathcal{C}, I, G \rangle$*, and is optimal if there does not exist a plan* π'' *solving* $\mathcal{P}^l$ *such that* $D(\pi, \pi'') < D(\pi, \pi')$*.*

We define a compilation that takes in input a generic $\mathcal{R}^l$ and returns a lifted planning problem whose optimal solution corresponds to an optimal solution for $\mathcal{R}^l$. The compilation is very inspired by RESA, but in order to make it work, we need to have a way to intercept when the grounding of an operator leads to an action which is present in the plan. In order to do so, first, we extend $\mathcal{P}$ with set $\bigcup_{op \in O} q_{op}$, and make all atoms in set $\bigcup_{op(\boldsymbol{c}) \in \pi} q_{op}(\boldsymbol{c})$ true in I. Then, we substitute set A_1 with set O_1^ℓ, where O_1^ℓ is a set of operators defined as follows:

$$O_1^\ell = \bigcup_{op \in O} \langle \boldsymbol{x}, \mathsf{pre}(op) \wedge w \wedge \phi, \mathsf{eff}(op) \rangle,$$

and ϕ is a special precondition formula defined as:

$$\forall \boldsymbol{y},\ q_{op}(\boldsymbol{y}) \rightarrow \neg \Big(\bigwedge_{x_i \in \boldsymbol{x}} x_i = y_i \Big).$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are two sequences of variables having the same size. For example, take a binary operator $op(x_1, x_2)$ and assume that action $op(c_1, c_2)$ is in the input plan. In PDDL, precondition ϕ of $op(x_1, x_2)$ is encoded by the following expression:

$$\texttt{(forall } (y_1\ \ y_2) \texttt{ (imply } (q_{op}(y_1\ \ y_2)) \texttt{ (not (and } (=\ \ x_1\ \ y_1)\ (=\ \ x_2\ \ y_2)))).$$

In the initial state atom $q_{op}(\boldsymbol{c})$ with $\boldsymbol{c} = (c_1, c_2)$ indicates that action $a = op(\boldsymbol{c})$ is part of the input plan π, while precondition ϕ makes non-executable any action that is obtained by grounding the lifted actions in O_1^ℓ and that is part of the input plan. Substantially, this precondition prohibits any action of an operator *op* from being performed with its parameters grounded with the same constants that label *op* in the original plan. Together with lifted actions, we also take the grounded actions as for the sets A_0, A_2, and A_3 of the RESA compilation and lift them up to operators with empty parameters (denoted later with '[]'). We call this lifted variant L-RESA, which we summarize below.

Definition 8 (L-RESA Compilation). *Let* $\mathcal{R}^l = \langle \mathcal{D}, \mathcal{C}, I, G, \pi \rangle$ *be a plan repair problem as defined above with* $\mathcal{D} = \langle P, O \rangle$ *and* $\pi = \langle op_1(\boldsymbol{c}_1), \cdots, op_n(\boldsymbol{c}_n) \rangle$*.* L-RESA *takes in input* $\mathcal{R}^l$ *and generates a new lifted planning problem* $\mathcal{P}^l = \langle \mathcal{D}', C, I', G', c^{l'} \rangle$ *where* $\mathcal{D}' = \langle P', O_0 \cup O_1 \cup O_2 \cup O_3 \cup switch \rangle$ *such that:*

$$P' = P \cup \{w\} \cup \bigcup_{i \in \{1,\ldots,n\}} d_i \cup \bigcup_{op(\boldsymbol{c}) \in \pi} \{p^i_{op(\boldsymbol{c})} \mid 0 \leq i \leq M(op(\boldsymbol{c}), \pi)\} \cup \bigcup_{op(\boldsymbol{c}) \in \pi} q_{op}(\boldsymbol{c})$$

$$I' = I \cup \{w\} \cup \bigcup_{op(\boldsymbol{c}) \in \pi} p^0_{op(\boldsymbol{c})} \cup \bigcup_{op(\boldsymbol{c}) \in \pi} q_{op}(\boldsymbol{c})$$

$$O_0 = \bigcup_{op_i(\boldsymbol{c}) \in \pi} \langle [], \mathsf{pre}(op(\boldsymbol{c})) \wedge w \wedge p^{B(op(\boldsymbol{c}),i,\pi)}_{op(\boldsymbol{c})}, \mathsf{eff}(op(\boldsymbol{c})) \cup \{p^{B(op(\boldsymbol{c}),i,\pi)+1}_{op(\boldsymbol{c})}, \neg p^{B(op(\boldsymbol{c}),i,\pi)}_{op(\boldsymbol{c})}, d_i\} \rangle$$

$$O_1 = \bigcup_{op(\boldsymbol{x}) \in O} \langle \boldsymbol{x}, \mathsf{pre}(op(\boldsymbol{x})) \wedge w \wedge \phi, \mathsf{eff}(op(\boldsymbol{x})) \rangle$$

$$O_2 = \bigcup_{op(\boldsymbol{c}) \in set(\pi)} \langle [], \mathsf{pre}(op(\boldsymbol{c})) \wedge w \wedge p^{M(op(\boldsymbol{c}),\pi)}_{op(\boldsymbol{c})}, \mathsf{eff}(op(\boldsymbol{c})) \rangle$$

$$O_3 = \bigcup_{i \in \{1,\ldots,n\}} \langle [], \neg w \wedge \neg d_i, \{d_i\} \rangle$$

$$switch = \langle [], w, \{\neg w\} \rangle$$

$$G' = G \wedge \bigwedge_{i \in \{1,\ldots,n\}} d_i$$

$$c^{l'}(o) = \begin{cases} 0 & \text{if } o \in O_0 \cup \{switch\} \\ 1 & \text{if } o \in O_1 \cup O_2 \cup O_3 \end{cases}$$

It is easy to see that this compilation is mimicking RESA and only keeps all actions abstract. This way, we do not need to ground the problem before compilation, and, as showed below, we can still be sound, complete and optimal.

Theorem 7 (L-RESA is sound, complete and optimal). *Let $\mathcal{R} = \langle \mathcal{P}, \Pi \rangle$ be a multi-repair planning problem.* L-RESA *transforms $\mathcal{R}$ into a problem $\mathcal{P}'$ that is solvable if and only if so is $\mathcal{R}$. Moreover, the optimal solution for $\mathcal{P}'$ equates to that of the solution for $\mathcal{R}$ that minimizes the distance from a plan π in Π.*

Proof. This follows directly from Theorems 1, 2, and 3, because for each action a in A_1 of RESA there exists one and only one action copy in set A_1^ℓ of L-RESA, and a is executable in a state $s \subseteq F'$ if and only if its copy is executable in s augmented with set $\bigcup_{op(\boldsymbol{c}) \in \pi} q_{op}(\boldsymbol{c})$. Indeed, these additional atoms are true in the initial state and are not in the effects of any other action, the special precondition formula ϕ of the copy of a in A_1^ℓ constraints such a lifted copy to have contradicting preconditions when the grounded actions is in the input plan, and the actions in A_1 are not in the input plan. □

Just as RESA was extended into a lighter version (L-RESA) and a multi-plan repair variant (M-RESA), other variations of this compilation are also possible. However, we leave these straightforward extensions outside the scope of this paper.

7. Experimental analysis

Our experimental analysis evaluates the performance of repairing a plan by RESA, S-RESA, L-RESA, and M-RESA w.r.t. replanning from scratch (hereinafter RS). With this analysis, we aim at shedding some light on the cost of the optimal-stability repair, as well as on its benefit in terms of stability. We implemented the different versions of RESA on the top of the UP planning framework (Micheli *et al.*, 2024). Our implementation is available from https://github.com/hstairs/resam.

Our comparison against RS uses both optimal and satisficing planners. With optimal planners, we evaluate the coverage of doing optimal plan repair against just doing optimal planning. The goal of this comparison is evaluating how harder/simpler the problem gets when we aim at finding plans of minimum distance from the input plan. Moreover, we also collect the actual plan distance found by the planners to show how different the solutions found by RESA, S-RESA, L-RESA, and M-RESA w.r.t. just ignoring the input plan are. The aim of this experiment is to assess the benefit of the plan repair in terms of stability. For this experiment, we compared the evaluated systems in pairs, and considered only the problems solved by both the compared systems. We used the same metrics for the satisficing planners. Also for the satisficing setting, we aim at understanding how harder/simpler the satisficing problem gets when the search is driven by minimizing the distance from an input plan, and the effectiveness of the satisficing planners in computing solutions closer to the input plan. For optimal planning, we used A* with a simple blind heuristic, and Delfi1 (Katz *et al.*, 2018), the winner of the 2018 edition of the International Planning Competition (IPC-18). For the satisficing setting we use Lama (Richter & Westphal, 2010) and BFWS (Lipovetzky & Geffner, 2017). For both systems we run only the first cost-sensitive iteration.

We also compared RESA w.r.t. LPG-adapt, which is the state-of-the-art approach to plan repair; LPG-adapt adapts plans through a refinement approach in the plan spaces. It is designed to take plan stability into account (Fox *et al.*, 2006). Specifically, we compared coverage and plan distance of LPG-adapt w.r.t. RESA using the optimal planner Delfi1 and the anytime version of Lama. In the rest of the section, RESA[X], S-RESA[X], L-RESA[X] denote planner X using the planning problem definition obtained by RESA, S-RESA, and L-RESA, respectively.

We take all the domains and problems from the optimal and satisficing track of the IPC-18. For each problem instance we generate three plan repair problems modifying the initial state by randomly executing 1, 2 and 5 actions in sequence. As an input plan for RESA, S-RESA, and L-RESA, we used the shortest plan among those generated during the competition. Ties among shortest plans are broken randomly. As for the sets of input plans for M-RESA, we used the 2 shortest, and the 5 shortest competition

Table 1. *Coverage Analysis. Each entry of the table corresponds to the number of problems solved by the system identified by the column: replanning from scratch (RS), or using* Resa, S-Resa, *and* L-Resa. *D-{1,2,5} is a regrouping of the instances that considers all instances computed by injecting 1, 2, or 5 random actions in sequence. The number of problems is in parenthesis. Bolds are for best performers*

Optimal	A*(Blind)				Delfi			
Domain	RS	Resa	S-Resa	L-Resa	RS	Resa	S-Resa	L-Resa
Agricola (48)	7	**9**	**9**	**9**	**39**	13	10	16
Caldera (45)	27	**45**	**45**	**45**	39	**45**	**45**	**45**
Data-Net (48)	24	**48**	**48**	**48**	39	**48**	**48**	**48**
Nurikabe (42)	36	**38**	37	15	**36**	18	18	26
Settlers (30)	22	**26**	24	–	25	**27**	**27**	–
Spider (51)	**29**	16	16	19	29	34	**36**	35
Termes (54)	**36**	11	11	11	36	37	37	**38**
D-1 (106)	60	**75**	**75**	56	82	**83**	81	75
D-2 (106)	62	**65**	**65**	49	**82**	75	73	68
D-5 (106)	**59**	53	50	42	**79**	71	67	65
Total	181	**193**	190	147	**243**	222	221	208

plans. Ties among shortest plans are again broken randomly. This gives us a grand total of 694 instances. All experiments ran up to 1800 seconds with a memory cut of 8GB. Tests are performed on a single core of a 4 cores Intel(R) Xeon(R) 2.30 GHz. In our comparison against RS, each experiment consists in launching each instance and each planner either with the original formulation of the problem or with that obtained by our compilations.

RS vs Resa, Optimal Planning. Table 1 shows the coverage of A* ran with the blind heuristic, and Delfi1 over both the original problem and the problems compiled by Resa, S-Resa, and L-Resa. As it is possible to observe, we have mixed results in terms of number of solved problems w.r.t. RS. RS proves to be more beneficial than repairing for only two domains; on the contrary, repairing is more beneficial for five domains over seven. This shows that in general our compilation does not make the optimal repair planning problem harder than replanning from scratch. L-Resa does not work with settlers because the domain contains universal quantifiers, which was not supported by the version of the UP framework at the time of our experiment.

As expected, the larger the difference from the problem solved by the input plan, the harder solving the optimal repair problem. Indeed, the number of solved problems obtained by changing the initial state executing 5 random actions is always smaller than executing 2 random actions, which in turn is smaller than executing only 1 random action. While repair proves beneficial on average when the divergence from the problem solved by the input plan is limited (D-1 and D-2), our findings indicate that when such a divergence becomes high (at D-5) replanning from scratch becomes the most effective approach. We conjecture that as divergence increases, the performance gap widens: repair mechanisms attempt to reuse actions that progressively lose relevance for solving the problem, whereas replanning from scratch avoids such bias and can fully exploit the available choices.

Table 2 shows the distance between the input plan and the plans obtained by repairing w.r.t. the plans obtained by replanning. It comes with no surprise that the plan distance obtained by repairing using optimal planners is (almost) always much better than replanning. For only the termes domain and A*, replanning from scratch performs as well as repairing. Left side of Figure 2 shows a more detailed picture of the performance gap. The results in figure indicate that repairing leads to plans that are very often at least one order of magnitude less distant from the input plans than RS.

Table 2. *Plan Distance Analysis. Each entry of the table corresponds to an average of the plan distances across problems solved by replanning from scratch (RS) or using* Resa, S-Resa, L-Resa. *Bolds are for best performers*

Optimal	A*(Blind)						Delfi					
Domain	RS	Resa	RS	S-Resa	RS	L-Resa	RS	Resa	RS	S-Resa	RS	L-Resa
Agricola	35.0	**3.00**	35.0	**3.00**	35.0	**3.00**	28.5	**5.18**	25.8	**2.00**	32.9	**6.71**
Caldera	3.81	**0.00**	3.81	**0.00**	3.81	**0.00**	3.97	**0.00**	3.97	**0.00**	3.97	**0.00**
Data-Net	2.88	**1.75**	2.88	**1.75**	2.88	**1.75**	8.85	**1.82**	8.85	**1.82**	8.85	**1.82**
Nurikabe	9.74	**2.29**	9.15	**2.21**	4.65	**0.78**	14.92	**3.67**	14.9	**3.67**	13.2	**6.50**
Settlers	8.71	**1.95**	7.42	**1.74**	–	–	8.88	**2.08**	8.88	**2.08**	–	–
Spider	29.3	**1.73**	29.3	**1.73**	29.3	**1.36**	37.5	**10.4**	36.4	**8.48**	35.8	**7.96**
Termes	**0.55**	**0.55**	**0.55**	**0.55**	**0.55**	**0.55**	28.9	**0.67**	28.9	**0.67**	28.9	**0.67**

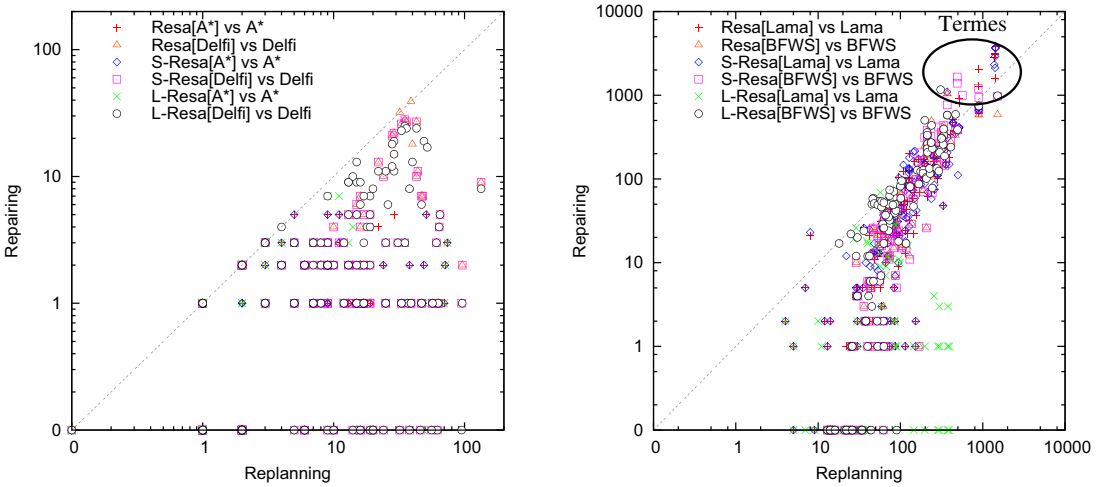


Figure 2. *Scatter plotting the plan distances across problems solved by replanning from scratch w.r.t. repairing by* Resa *using the optimal (left) and the satisficing (right) planners.*

RS vs Resa, Satisficing Planning. Table 3 reports on the coverage of Lama and BFWS with or without our compilations. Note that our compilations let us improve on the coverage of Lama over three domains, gets the same coverage as Lama for the caldera domain, and degrades the coverage over agricola, spider and termes. For these domains, the problems have solution plans definitely longer than for other domains. Our conjecture is that longer plans make the compilation more difficult, nullifying the guidance that such plans can provide in solving the problem. The results of Resa with BFWS are similar.

Interestingly, differently from the optimal setting, the satisficing repair planning problem does not become more difficult as the number of differences w.r.t. the problem solved by the input plan increases. Indeed, the number of solved problems obtained by changing the initial state executing 1, 2, or 5 random actions is similar.

The total number of problems solved by L-Resa is considerably lower than that achieved by RS, as well as by repairing with Resa and S-Resa. Our findings nevertheless show some complementarity among the compilations. For example, using Lama, L-Resa solves 2 problems more than Resa in Data-Net, 1 more problem in termes, and 10 problems more in Agricola. Similarly, using BFWS, L-Resa solves more problems for the Agricola, Data-Net and Spider domains. This is due to the fact that both Lama and BFWS seem more efficient, at times, when we submit a lifted representation of the operators instead of a grounded one. It is also interesting to note that the problems compiled by L-Resa are much

Table 3. *Coverage Analysis. Each entry of the table corresponds to the number of problems solved by the system identified by the column: replanning from scratch (RS), or using* Resa, S-Resa, *and* L-Resa. *D-{1,2,5} is a regrouping of the instances that considers all instances computed by injecting 1, 2, or 5 random actions in sequence. The number of problems is in parenthesis. Bold is for best performers*

Satisficing	Lama				BFWS			
Domain	RS	Resa	S-Resa	L-Resa	RS	Resa	S-Resa	L-Resa
Agricola (48)	**37**	14	13	24	16	12	16	**23**
Caldera (27)	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**
Data-Net (57)	15	51	51	**53**	28	23	22	**30**
Nurikabe (57)	31	**56**	55	8	31	**57**	**57**	27
Settlers (57)	47	**54**	**53**	–	10	**22**	15	–
Spider (54)	**48**	21	20	15	35	34	33	**36**
Termes (54)	**42**	27	28	29	**26**	11	12	12
D-1 (116)	84	**86**	85	53	56	61	**62**	50
D-2 (116)	83	**84**	81	53	56	**65**	61	51
D-5 (116)	80	80	**81**	50	**61**	60	59	54
Total	247	**250**	247	156	173	**186**	182	155

Table 4. *Plan Distance Analysis. Each entry of the table corresponds to an average of the plan distances across problems solved by replanning from scratch (RS) or using* Resa, S-Resa, L-Resa. *Bolds are for best performers*

Satisficing	Lama						BFWS					
Domain	RS	Resa	RS	S-Resa	RS	L-Resa	RS	Resa	RS	S-Resa	RS	L-Resa
Agricola	46.1	**6.31**	48.0	**9.54**	57.3	**8.71**	55.2	**10.5**	59.3	**13.3**	59.5	**10.1**
Caldera	21.5	**0.00**	21.5	**0.00**	21.5	**0.00**	24.1	**0.78**	24.1	**0.78**	24.1	**0.78**
Data-Net	123.2	**28.8**	123.2	**28.3**	107.2	**12.5**	99.0	**44.5**	95.5	**44.4**	90.7	**43.9**
Nurikabe	65.5	**8.61**	65.5	**10.7**	47.8	**46. 1**	83.4	**22.7**	83.4	**22.7**	67.6	**49.2**
Settlers	111.3	**71.9**	108.1	**69.0**	–	–	81.9	**19.0**	89.2	**15.7**	–	–
Spider	315.1	**227.0**	332.7	**227.1**	287.3	**1.07**	270.8	**23**3.4	271.6	**249.3**	271.5	**240.7**
Termes	**683.2**	1028	**672.1**	950.0	**647.9**	1055	**57**5.0	589.9	**602.3**	831.5	**548.3**	632.9

shorter than those compiled by Resa and S-Resa. Indeed, for our benchmark the problem representation of L-Resa is up to more than two orders of magnitude shorter.

Table 4 shows that for all domains but termes the plans computed by Resa with both Lama and BFWS are less distant from the input plan than RS. In termes, very long input plans give no fruitful information to the search. The results for S-Resa and L-Resa are similar to those for Resa. The right side of Figure 2 shows a more detailed picture about the plan distance. Repairing leads to plans that are usually less distant from the input plans and often at least one order of magnitude less distant than replanning from scratch.

LPG-Adapt vs Resa. This experiment evaluates the effectiveness of repairing using our approach w.r.t. using LPG-adapt, the state-of-the-art for the plan repair. For our approach, we used the problems compiled using Resa, as it generally works better than S-Resa and L-Resa, and ran Delfi1 and Lama, the best performers among the used optimal and satisficing planners. Specifically, for this analysis we take the best solution found by LPG-adapt and Resa using Lama in anytime modality run up to 1800 seconds.

Table 5. *Coverage and plan distance between* LPG-adapt *and* RESA *using* Delfi1 *and* Lama *over the problems derived from the optimal track of IPC-18. Bolds are for best performers; "–" for unsupported*

	Solved		Distance		Solved		Distance	
Domain	LPG	RESA[Delfi1]	LPG	RESA[Delfi1]	LPG	RESA[Lama]	LPG	RESA[Lama]
AGRICOLA	**16**	13	110.0	**18.00**	6	**14**	**5.00**	**5.00**
CALDERA	–	**45**	–	–	–	**27**	–	–
DATA-NET	**48**	**48**	4.94	**1.85**	**57**	51	**6.33**	28.6
NURIKABE	–	**18**	–	–	–	**56**	–	–
SETTLERS	–	**27**	–	–	–	**54**	–	–
SPIDER	–	**34**	–	–	–	**21**	–	–
TERMES	**54**	37	0.81	**0.65**	**45**	27	**0.74**	1028

Table 6. *Coverage Analysis. Each entry of the table corresponds to the number of problems solved by the system identified by the column: replanning from scratch (RS), or using* M-RESA *with 1 (1P), 2 (2P), or 5 (5P) input plans. D-{1,2,5} is a regrouping of the instances that considers all instances computed by injecting 1, 2, or 5 random actions in sequence. The number of problems is in parenthesis. Bolds are for best performers*

Optimal	A*(Blind)			Delfi		
	RS	M-RESA		RS	M-RESA	
Domain		2P	5P		2P	5P
Agricola (48)	7	**8**	5	**39**	12	12
Caldera (45)	27	**45**	**45**	39	**45**	**45**
Data-Net (48)	24	**48**	**48**	39	**48**	**48**
Nurikabe (42)	36	**37**	34	**36**	15	15
Settlers (57)	22	**24**	23	25	**27**	**27**
Spider (51)	**29**	17	11	**29**	28	24
Termes (54)	**36**	11	7	**36**	21	10
D-1 (96)	60	**74**	70	**82**	75	69
D-2 (96)	62	**66**	59	**82**	64	59
D-5 (96)	**59**	50	44	**79**	57	53
Total (288)	181	**190**	173	**243**	196	181

The results in Table 5 show that, for all domains but TERMES, RESA with Delfi1 is competitive with LPG-adapt in terms of coverage, while guaranteeing the optimality of the solution for the repair problem. The plans computed by RESA with Delfi1 are up to one order of magnitude closer to the input plans (i.e., in AGRICOLA). RESA with Lama solves much more problems than LPG-adapt overall, but is not effective in TERMES where LPG-adapt was able to solve many more instances. Missing results of Table 5 are due to LPG-adapt not supporting conditional effects.

RS vs M-RESA, Optimal Planning In the rest of this section, we focus on evaluating the effectiveness of M-RESA. For this experiment, we considered 2 and 5 input plans.

Table 6 shows the coverage of M-RESA with a different number of input plans w.r.t. replanning from scratch. With 2 input plans, M-RESA using $A^\star$ solves more problems than RS for five domains over seven, while with 5 input plans M-RESA using $A^\star$ solves more problems for only three domains. The total

Table 7. *Plan Distance Analysis. Each entry of the table corresponds to an average of the plan distances across problems solved by replanning from scratch (RS) or using* M-RESA *with 1 (1P), 2 (2P), or 5 (5P) input plans. Bolds are for best performers*

Optimal	A*(Blind)				Delfi			
setting	2P		5P		2P		5P	
Domain	RS	M-RESA	RS	M-RESA	RS	M-RESA	RS	M-RESA
Agricola	35.0	**3.00**	–	–	22.3	**2.70**	18.5	**2.70**
Caldera	3.81	**0.00**	3.81	**0.00**	3.05	**0.00**	1.97	**0.00**
Data-Net.	2.46	**1.75**	2.46	**1.75**	7.10	**1.82**	5.85	**1.82**
Nurikabe	7.45	**2.15**	5.30	**1.77**	3.11	**2.22**	3.11	**2.11**
Settlers	6.58	**1.74**	5.95	**1.63**	7.28	**2.08**	6.56	**2.00**
Spider	20.73	**1.73**	22.29	**1.43**	39.9	**3.59**	29.1	**3.15**
Termes	**0.55**	**0.55**	**0.57**	**0.57**	17.8	**0.67**	7.60	**0.60**

number of problems solved by M-RESA[$A^\star$] with 2 or 5 input plans is comparable than RS. The picture is different with Delfi1 in place of A*, as M-RESA[Delfi1] solves more problems for only 3 domains over 7. In general, the results in Table 6 indicates that the optimal multi-repair planning problem is more difficult for Delfi1 than RS, while the hardness is comparable using A*. Moreover, as expected, the greater the number of input plans, the harder the optimal multi-repair planning problem.

As for the comparison in terms of plan distance, the distance of M-RESA is computed as in Definition 4; similarly, the distance of a plan obtained by replanning is the minimum between the distances from any input plan. Table 7 shows that, as expected, repairing using M-RESA with an optimal planner is always better than RS in terms of plan distance, and M-RESA sometimes computes plans that are at least one order of magnitude more similar to the input plan than RS. Replanning by $A^\star$ computes plans that are as far from the input plan as M-RESA only for TERMES domain. For the AGRICOLA domain, $A^\star$ and 5 input plans, we do not have data because the problems solved by replanning are different from those solved by repairing with M-RESA. It is not surprising to observe that, both by replanning and repairing, the distances tends to decrease w.r.t. an increasing number of input plans. Indeed, the higher the number of input plans, the more the chance to be stable with at least one of them. Left side of Figure 3 shows a more detailed picture of the plan distances obtained by repairing and replanning. Like for a single input plan, solving the optimal multi-repair problem leads to plans that are very often at least one order of magnitude less distant from the input plans than replanning from scratch.

RS vs M-RESA, Satisficing Planning Table 8 shows the coverage obtained by repairing with M-RESA w.r.t. replanning with either Lama or BFWS. The results in the table show that solving the multi-repair planning problem by using Lama is slightly harder than RS, while it becomes significantly more difficult using BFWS. As for the hardness of solving the multi-repair planning problem with a different number of input plans, we have mixed results: using Lama, solving the problems with 2 input plans is approximately as hard as with 5 input plans, while using BFWS the hardness tends to increase with the number of input plans.

Finally, Table 9 shows the evaluation between M-RESA and RS in terms of plan distance. M-RESA with every considered number of input plans is (almost) always better than RS for all domains but TERMES and SPIDER using BFWS and TERMES using Lama. As mentioned before, the input plans for TERMES are very long, and we conjecture they cannot provide good guidance for the heuristics of Lama and BFWS. As for the distance from the input plans w.r.t. an increasing number of input plans, the picture with the satisficing planners is less clear than with the optimal planner. We conjecture that the larger set of actions with cost greater than zero only sometimes confuses the heuristics used by Lama and BFWS. Right side of Figure 3 shows a more detailed picture of the plan distances obtained by repairing w.r.t. replanning from scratch. The conclusion we draw from this analysis is the same as for repairing a single

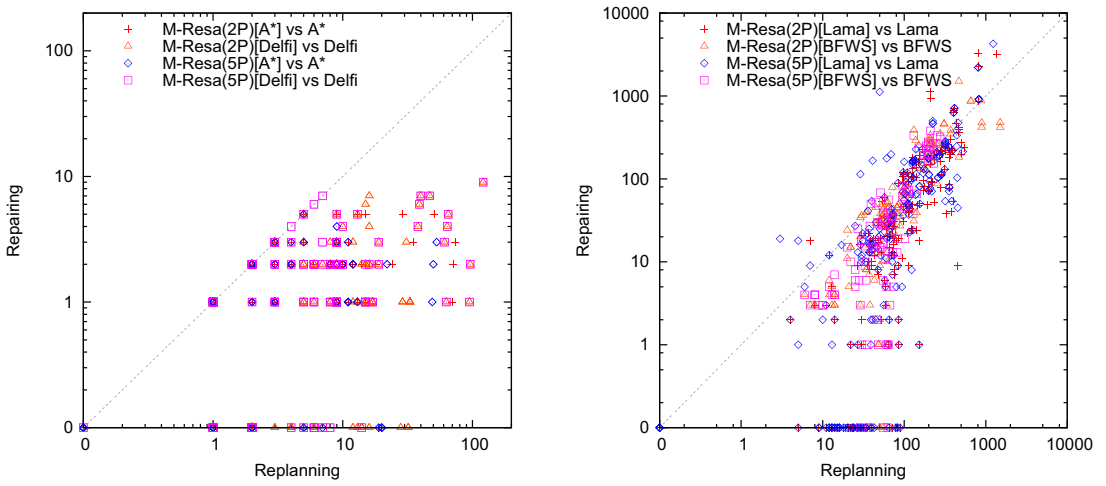


Figure 3. *Scatter plotting the plan distances across problems solved by replanning from scratch w.r.t. repairing by* M-Resa *using the optimal (left) and the satisficing (right) planners.*

input plan: solving the multi-repair planning problem leads to plans that are usually less distant from the input plans, and often at least one order of magnitude less distant than replanning from scratch.

8. Related work

Sooner or later plans will fail regardless on how precise the models are. This observation has led many researchers to question on how to build intelligent agents that can deal with this issue. One possibility is to anticipate everything upfront and therefore build on models that take the uncertainty into account from the very beginning, assuming or not observation capabilities of the agent (Cimatti *et al*., 2003; Rintanen, 2004; Grastien & Scala, 2020), and whether the agent has some probability distribution over the state space, and action effects (Kaelbling *et al*., 1998). However, it is in many cases much more practical to revise the plan as soon as the actual discrepancy from the plan being executed is observed (Ingrand & Ghallab, 2017).

One possibility is replanning (e.g., desJardins *et al*., 1999; Yoon *et al*., 2007; Borrajo & Veloso, 2021; Cardellini *et al*., 2023; Pozanco *et al*., 2023), the other is trying to re-use as much as possible the old plan by repairing it (e.g., van der Krogt & de Weerdt, 2005; Fox *et al*., 2006; Garrido *et al*., 2010; Gerevini & Serina, 2010; Scala, 2014; Scala & Torasso, 2014; Percassi *et al*., 2023). This paper focuses on repair, and because of this in what follows we provide a small outlook to those works which we believe are relevant to what we studied here. van der Krogt and de Weerdt (2005) define repair as a two-step process: first, the flawed actions are removed (unrefinement), then new actions are added to fix the gaps. The algorithm searches for solutions in the plan space and backtracks to the unrefinement stage if no solution is found during refinement. Gerevini and Serina (2010) look for 'windows' of plans that need to be fixed, and runs planning off-the-shelf only over those portions. Other approaches, somehow closer to our approach, look instead at compilation techniques. Scala and Torasso (2014) compile the repair problem as a Constraint Satisfaction Problem over the space of action modalities. This work also is motivated by stability to some extent, though stability is controlled by focusing the search on a specific portion of the state space. Other compilation-based approach is presented by Percassi *et al*. (2023) and Scala (2014). Specifically, Percassi *et al*. (2023) propose a repair technique based on fixing the timing of the actions and does so by reformulating a time-discrete PDDL+ problem. Scala (2014) instead proposes the use of macro-operators as a means to re-use previous portions of the plans. Scala and Torasso (2015) extend their previous work to extract macros from plan decompositions obtained through a process of deordering.

Table 8. *Coverage Analysis. Each entry of the table corresponds to the number of problems solved by the system identified by the column: replanning from scratch (RS), or using* M-Resa *with 1 (1P), 2 (2P), or 5 (5P) input plans. D-{1,2,5} is a regrouping of the instances that considers all instances computed by injecting 1, 2, or 5 random actions in sequence. The number of problems is in parenthesis. Bolds are for best performers*

Satisficing	Lama			BFWS		
	RS	M-Resa		RS	M-Resa	
Domain		2P	5P		2P	5P
Agricola (48)	**37**	9	12	**16**	6	7
Caldera (27)	**27**	**27**	**27**	**27**	11	6
Data-Net (57)	15	48	**53**	**28**	16	18
Nurikabe (57)	31	**38**	37	**31**	18	20
Settlers (57)	47	**55**	52	10	**13**	4
Spider (51)	**48**	25	29	**35**	8	9
Termes (51)	**42**	24	20	**26**	7	3
D-1 (97)	**84**	75	78	**56**	26	24
D-2 (97)	**83**	78	78	**56**	28	21
D-5 (97)	**80**	73	74	**61**	25	22
Total (291)	**247**	226	230	**173**	79	67

Table 9. *Plan Distance Analysis. Each entry of the table corresponds to an average of the plan distances across problems solved by replanning from scratch (RS) or using* M-Resa *with 1 (1P), 2 (2P), or 5 (5P) input plans. Bolds are for best performers*

Satisficing	Lama				BFWS			
	2P		5P		2P		5P	
Domain	RS	M-Resa	RS	M-Resa	RS	M-Resa	RS	M-Resa
Agricola	45.0	**4.22**	33.9	**7.00**	59.3	**13.3**	49.2	**12.0**
Caldera	20.9	**0.00**	20.0	**0.00**	13.6	**5.55**	11.7	**3.50**
Data-Net.	121.7	**21.0**	103.6	**24.3**	79.1	**36.1**	69.0	**30**.1
Nurikabe	59.9	**18.2**	57.8	**19.5**	47.3	**28.7**	47.3	**28.6**
Settlers	105.3	**67.4**	88.3	**70.5**	78.5	**28.5**	51.0	**14.5**
Spider	349.5	**231.5**	295.5	**221.2**	**212.0**	232.9	**206.1**	245.2
Termes	**407.0**	815.8	**429.6**	668.0	**307.9**	303.9	**196.7**	248.3

The state-of-the art approach to plan repair prior to this work was LPG-adapt (Fox *et al*., 2006). LPG (Gerevini *et al*., 2011) is a planner that natively supports plan repair, as it does plan search by refinement, making repair a byproduct of the planner's search process. The extension into LPG-adapt (Fox *et al*., 2006) makes the repair more central, and also introduces the plan distance metric that we use in this paper. All these works do not however focus on producing optimal stable plans; this paper presents the first approach to bridge this gap systematically by compilation.

The compilation path that we use is similar in the spirit to the compilation proposed by Keyder and Geffner (2009), in which one can see the process of repair as an oversubscription problem where the goals of the problem remain the same but we seek for those plans which minimize the distance to the previous one(s). Similar is also the technique that attributes cost only to those actions which signal the

fact that some over-subscribed goal is not reachable. In our case this over-subscribed goals are the actions from the original plan that we did not cover, or those actions that we added to achieve the (sub)goals but were not in the original plan.

Researchers also looked at the problem of repairing plans using domain-specific knowledge, like hierarchical task networks. A research direction about plan repair via HTN focuses on the plan generation process. That is, several approaches propose algorithms that directly revise plan generation by first retracting the development steps responsible for the failed plan fragments, for example, Bidot *et al*. (2008), Boella and Damiano (2002), Warfield *et al*. (2007). The most recent approach among those we are aware of is ShopFixer (Goldman *et al*., 2020), which uses a graph of causal links and task decomposition to identify the portion of the plan to repair. In particular, it finds the minimal subtree of the plan tree that contain an unsupported precondition; then, it restarts the search by backjumping to the corresponding entry in the search tree, and updates the world state with the effects of the disturbance. Another research direction is based on the compilation of the HTN problem to enforce HTN plans to start with a prefix of actions (e.g., Bercher *et al*., 2014). A recent compilation approach was proposed by Höller *et al*. (2020). In addition to new propositions and actions, their compilation introduces new primitive and abstract taks into the problem definition, so that the HTN planning system is forced to generate a solution that starts with the prefix of the plan executed before an unexpected change. Our approach is different in the scope, as it does not require a task decomposition network and can be used to compute plans as stable as possible w.r.t. an input plan.

9. Conclusion

This paper faces the problem of optimal plan repair, which consists in finding a plan that solves the problem and also minimizes the distance between the solution plan and the input plan. We solved this problem through a compilation into classical planning. On top of that, we showed that it is possible to obtain an easy extension of this approach that takes multiple plans into account, therefore minimizing the distance over such a set. For the case with a single plan, we also provide some additional compilation schemes, including a lifted encoding – no need to ground upfront which provides some further complementary behavior.

Our results show that not only is this approach possible but also competitive with LPG-adapt, the state-of-the-art system for repairing plans, while can also guarantee that the plan stability is optimised. As a future work we want to investigate different metrics for stability and adaptations of Resa to such metrics. Moreover, it would be interesting to use this approach for other settings, such as numeric or temporal planning (Fox & Long, 2003). Numeric planning problems could easily leverage this compilation schema, particularly since the use of numerical functions would significantly reduce the number of actions required by the compilation, while temporal planning would require a more profound restructuring of the encoding.

Competing interests. The authors declare none.

References

Bercher, P., Biundo, S., Geier, T., Hoernle, T., Nothdurft, F., Richter, F. & Schattenberg, B. 2014. Plan, repair, execute, explain - how planning helps to assemble your home theater. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21–26, 2014*. AAAI.

Bidot, J., Schattenberg, B. & Biundo, S. 2008. Plan repair in hybrid planning In *KI 2008: Advances in Artificial Intelligence*, 169–176. Springer Berlin Heidelberg.

Boella, G. & Damiano, R. 2002. A replanning algorithm for a reactive agent architecture. In *Artificial Intelligence: Methodology, Systems, and Applications, 10th International Conference, AIMSA 2002, Varna, Bulgaria, September 4–6, 2002, Proceedings*, Scott, D. (ed.). Lecture Notes in Computer Science 2443, 183–192. Springer.

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

Borrajo, D. & Veloso, M. 2021. Computing opportunities to augment plans for novel replanning during execution. In *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021*, *Guangzhou, China (virtual), August 2–13, 2021*, Biundo, S., Do, M., Goldman, R., Katz, M., Yang, Q. & Zhuo, H. H. (eds), 51–55. AAAI Press.

Cardellini, M., Dodaro, C., Galatà, G., Giardini, A., Maratea, M., Nisopoli, N. & Porro, I. 2023. Rescheduling rehabilitation sessions with answer set programming. *Journal of Logic and Computation* **33**(4), 837–863.

Cimatti, A., Pistore, M., Roveri, M. & Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* **147**(1–2), 35–84.

desJardins, M., Durfee, E. H., Jr., C. L. O. & Wolverton, M. 1999. A survey of research in distributed, continual planning. *AI Magazine* **20**(4), 13–22.

Fox, M., Gerevini, A., Long, D. & Serina, I. 2006. Plan stability: Replanning versus plan repair. In *ICAPS*, 212–221. AAAI.

Fox, M. & Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* **20**, 61–124.

Garrido, A., C., G. & Onaindia, E. 2010. Anytime plan-adaptation for continuous planning. In Proc. of P&S Special Interest Group Workshop *(PLANSIG-10)*.

Gerevini, A., Roubcková, A., Saetti, A. & Serina, I. 2013. On the plan-library maintenance problem in a case-based planner. In *Case-Based Reasoning Research and Development - 21st International Conference, ICCBR 2013, Saratoga Springs, NY, USA, July 8–11, 2013, Proceedings*, Delany, S. J. & Ontañón, S. (eds), Lecture Notes in Computer Science 7969, 119–133. Springer.

Gerevini, A., Saetti, A. & Serina, I. 2011. An empirical analysis of some heuristic features for planning through local search and action graphs. *Fundamenta Informaticae* **107**(2–3), 167–197.

Gerevini, A. & Serina, I. 2010. Efficient plan adaptation through replanning windows and heuristic goals. *Fundamenta Informaticae* **102**(3–4), 287–323.

Goldman, R. P., Kuter, U. & Freedman, R. G. 2020. Stable plan repair for state-space htn planning. In *Proceedings of the ICAPS-20 Workshop on Hierarchical Planning (HPlan 2020)*, 27–35.

Grastien, A. & Scala, E. 2020. CPCES: A planning framework to solve conformant planning problems through a counterexample guided refinement. *Artificial Intelligence* **284**, 103271.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* **26**, 191–246.

Höller, D., Bercher, P., Behnke, G. & Biundo, S. 2018. Htn plan repair using unmodified planning systems. In *Proceedings of the 1st ICAPS Workshop on Hierarchical Planning (HPlan)*, 26–30.

Höller, D., Bercher, P., Behnke, G. & Biundo, S. 2020. HTN plan repair via model transformation. In *KI 2020: Advances in Artificial Intelligence - 43rd German Conference on AI*, *Bamberg, Germany*, *September 21–25, 2020, Proceedings*, Lecture Notes in Computer Science 12325, 88–101. Springer.

Ingrand, F. & Ghallab, M. 2017. Deliberation for autonomous robots: A survey. *Artificial Intelligence* **247**, 10–44.

Kaelbling, L. P., Littman, M. L. & Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* **101**(1-2), 99–134.

Katz, M., Sohrabi, S., Samulowitz, H. & Sievers, S. 2018. Delfi: Online planner selection for cost-optimal planning. *IPC-9 planner abstracts*, 57–64.

Keyder, E. & Geffner, H. 2009. Soft goals can be compiled away. *Journal of Artificial Intelligence Research* **36**, 547–556.

Koenig, S. & Likhachev, M. 2002. D*lite. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28–August 1, 2002, Edmonton, Alberta, Canada*, 476–483. AAAI Press/The MIT Press.

Lipovetzky, N. & Geffner, H. 2017. Best-first width search: Exploration and exploitation in classical planning. In *AAAI*, 3590–3596. AAAI Press.

Micheli, A., et al. 2024. Unified planning framework.

Muñoz-Avila, H. & Cox, M. T. 2008. Case-based plan adaptation: An analysis and review. *IEEE Intelligent Systems* **23**(4), 75–81.

Nebel, B. & Koehler, J. 1995. Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence* **76**(1-2), 427–454.

Percassi, F., Scala, E. & Vallati, M. 2023. Fixing plans for PDDL+ problems: Theoretical and practical implications. In *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling*, *Prague, Czech Republic*, *July 8–13, 2023*, 324–333. AAAI Press.

Pozanco, A., Borrajo, D. & Veloso, M. 2023. Generating replanning goals through multi-objective optimization in response to execution observation. In *ECAI 2023 - 26th European Conference on Artificial Intelligence*, *September 30 - October 4, 2023, Kraków, Poland*. Frontiers in Artificial Intelligence and Applications 372, 1898–1905. IOS Press.

Richter, S. & Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* **39**, 127–177.

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

Ruml, W., Do, M. B., Zhou, R. & Fromherz, M. P. J. 2011. On-line planning and scheduling: An application to controlling modular printers. *Journal of Artificial Intelligence Research (JAIR)* **40**, 415–468.

Saetti, A. & Scala, E. 2022. Optimizing the stability in plan repair via compilation. In *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling*, *ICAPS* 2022, *Singapore (virtual), June 13–24, 2022*, 316–320. AAAI Press.

Scala, E. 2014. Plan repair for resource constrained tasks via numeric macro actions. In *ICAPS*. AAAI.

Scala, E. & Torasso, P. 2014. Proactive and reactive reconfiguration for the robust execution of multi modality plans. In *ECAI*. Frontiers in Artificial Intelligence and Applications **263**, 783–788. IOS Press.

Scala, E. & Torasso, P. 2015. Deordering and numeric macro actions for plan repair. In *IJCAI*, 1673–1681. AAAI Press.

Smith, D. E. & Weld, D. S. 1998. Conformant graphplan. In *AAAI/IAAI*, 889–896. AAAI Press/The MIT Press.

van der Krogt, R. & de Weerdt, M. 2005. Plan repair as an extension of planning. In *Proc. of International Conference on Automated Planning and Scheduling (ICAPS-05)*, 161–170.

Warfield, I., Hogg, C., Lee-Urban, S. & Muñoz-Avila, H. 2007. Adaptation of hierarchical task network plans. In *Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference*, *May 7–9, 2007, Key West, Florida, USA*, 429–434. AAAI Press.

Yoon, S. W., Fern, A. & Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In *ICAPS*, 352. AAAI.