One Repair to Rule Them All: Repairing a Broken Planning Domain Using Multiple Instances

Alba Gragera[1] and **Christian Muise**[2]

[1]Computer Science and Engineering Department, Universidad Carlos III de Madrid, Spain
[2]School of Computing, Queen's University
agragera@pa.uc3m.es, christian.muise@queensu.ca

Abstract

AI planners usually require an accurate description of the planning task to obtain a solution that achieves the goals of the problem. However, generating such descriptions can be time-consuming and error-prone, often resulting in unsolvable planning tasks. Planners lack the ability to identify semantic errors or explain how to solve them. Previous works offer repaired initial states/domains to make the planning task solvable. Still, they are limited to fixing things for a single problem instance or rely on input plan traces. In this work, we address the reparation of a flawed domain only based on a set of planning instances viewed holistically. By obtaining individual repairs for each problem, we search in the space of models for a repaired one that covers the full problem set. Our experimental results show the effective application of this approach in repairing a lifted planning domain and establish metrics that quantify the effort required to obtain the ground truth repair through an anytime algorithm.

1 Introduction

Automated Planning (AP) models are typically designed using the Planning Domain Definition Language (PDDL) and separate the planning task definition into two parts: domain and problem (McDermott et al. 1998). The domain describes the object types, predicates and actions that can be used to solve a task, while the problem defines the objects, initial state and goals of the specific task to be solved. Most AI planners assume an accurate task definition specification in PDDL, focusing on sequencing a set of actions that achieve the problem goals when applied to specific initial states. However, formalizing a planning task in PDDL is an error-prone process that requires a broad knowledge of the domain and the PDDL language. Most PDDL models, especially during their early stages of development, are often incomplete, and achieving completeness is the first bottleneck in the development of an AP application. Current planning technology is unable to handle *approximate* models, i.e, models that have some missing details (Kambhampati 2007), leading to unsolvable planning tasks.

Assisting human designers in understanding planning failures and providing solutions is a key challenge for the planning community (Fox, Long, and Magazzeni 2017; Chakraborti, Sreedharan, and Kambhampati 2020). Previous works have explored scenarios in which initial states hinder the achievement of goals (Göbelbecker et al. 2010; Sreedharan et al. 2019), suggesting alternative initial states that render the task solvable. Fixing planning tasks when the error lies within the domain model is not straightforward, given the number of potential changes to the set of actions (Lin and Bercher 2021). Most works assume user guidance, often in the form of a proposed valid plan (Nguyen, Sreedharan, and Kambhampati 2017; Lin, Grastien, and Bercher 2023). More recently, the need for inputs to repair a flawed domain has been eliminated by compiling the unsolvable task into an extended task that incorporates operators to repair itself (Gragera et al. 2023), but this approach only works for a single problem instance, which ultimately can generate reparations that are overly specific to just the given problem, compromising the approach's ability to generalize. Other works in learning action models (Aineto, Jiménez, and Onaindia 2018) can learn a domain when the actions and intermediate states of the plan executions are totally or partially observable. However, their strength is on scenarios where actions and states from the agent's plan executions are observed, with the goal of inferring a domain that accommodates such observations.

Formulated as the *General Domain Repair* problem, our approach assumes a partially specified domain and extends the work by Gragera et al. (2023) to repair it for a set of problem instances, under the assumption of no intermediate action or state observation and partial observable goal states. Similar to model reconciliation (Chakraborti et al. 2017), we perform a search in the space of models, in this case, to find a model that combines repairs tailored to specific problems, resulting valid for the entire problem set. As the approach is data-driven, the obtained reparations might be affected by the variety of input problems. To asses the reliability of the resulting domains, we assume that an oracle equipped with the ground truth is available, so that our experimental results extend beyond merely obtaining any model that works for the set of problems; we also aim to explore the challenges of obtaining the "ground truth reparation" through an iterative process. We find that this iterative approach significantly contributes to generating quality reparations as it helps to filter solutions and facilitates refining them to retrieve new alternatives.

2 Background

2.1 Automated Planning

We use the first-order (lifted) planning formalism, where a classical planning task is a pair $\Pi = \langle D, I \rangle$, where D is the planning domain and I defines a problem. A domain is a tuple $D = \langle Pred, A \rangle$; where $Pred$ is a set of predicates $p(t) \in Pred$, where $t = t_1, \ldots, t_n$ are typed free variables; and A is a set of action schemes. Action schemes $a \in A$ are tuples $a = \langle name(a), par(a), pre(a), add(a), del(a), cost(a) \rangle$, where $name(a)$ is an action name; $par(a)$ is a finite set of free variables; $pre(a) = \langle pre^+(a), pre^-(a) \rangle$, where $pre^+(a)$ and $pre^-(a)$ are sets of atoms representing the *positive* and *negative preconditions* for the action; $add(a)$ and $del(a)$ are the positive and negative changes produced by the application of the action, respectively; and the action $cost(a)$. A problem is a tuple $I = \langle \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{I}$ is the set of ground atoms in the initial state and $\mathcal{G}$ is the set of ground atoms defining the goals.

A ground action $\underline{a}$ is applicable in a state s if $pre^+(\underline{a}) \subseteq s$ and $pre^-(\underline{a}) \cap s = \emptyset$, transitioning to $s' = (s \setminus del(\underline{a})) \cup add(\underline{a})$. A *plan* π is a sequence $\underline{a}_1, \ldots, \underline{a}_n$ such that $\underline{a}_1$ is applicable in $\mathcal{I}$ and the consecutive application of all actions in the plan generates $s_n, \mathcal{G} \subseteq s_n$. A plan's cost is $cost(\pi) = \sum_{\underline{a}_i \in \pi} cost(\underline{a}_i)$.

2.2 Single Instance Domain Repair

Previous works have considered unsolvable planning tasks due to flaws in the model. Specifically, Gragera et al. (2023) considered missing action effects and defined an effect-incomplete domain. We adopt their notation for our work:

Definition 1 (**Effect-incomplete domain).** *A planning domain $D^- = \langle Pred, A^- \rangle$ is effect-incomplete w.r.t. an underlying planning domain $D = \langle Pred, A \rangle$ iff for at least one of the corresponding action schemas $a \in \mathcal{A}$ and $a^- \in \mathcal{A}^-$, $add(a^-) \subset add(a)$ or $del(a^-) \subset del(a)$.*

For an effect-incomplete domain, there always exists at least one $a^- \in \mathcal{A}^-$ whose positive or negative effects are a proper subset of the assumed complete action $a \in \mathcal{A}$. Gragera et al. propose a compilation of the incomplete planning task into a new planning task, equipped with new operators to fix actions in the effect-incomplete domain with additional *add* or *del* effects, obtaining a fixed domain as a result. They define a *repair set* as the total set of fixes applied to the domain. In this work, we want to emphasize each individual *repair* within that set, so we define a *repair* as:

Definition 2 (**Repair).** *We define a **repair** Rep as a pair $(a_i, p(t)^X)$, where $\{p(t) \mid p \in Pred, t \subseteq par(a_i^-)\}$, $X \in \{+, -\}$ extends the positive or negative effects of $a_i^- \in \mathcal{A}^-$.*

Intuitively, a repair is a single "fix" to an action in our incomplete domain. Then, we reformulate their *repair set* and adapt its notation to *reparation*:

Definition 3 (**Reparation).** *Given an effect-incomplete domain $D^- = \langle Pred, A^- \rangle$, a reparation $\hat{R}$ is a set of repairs $\{Rep_1, \ldots, Rep_n\}$ to extend the set of actions $\mathcal{A}^-$.*

Any action can include any subset of repairs as new effects, provided that the terms of the predicates are present in the action parameters. Gragera et al. implement a cost-based approach that penalizes trivial or undesired reparations, such as directly adding the goals as new effects. The repaired domain extends every action a_i with predicates as new positive $(+)$ or negative $(-)$ effects, defined as follows:

```
(stack b3 b4)
(fix on stack)
(add-fix-2par on stack b3 b4 t_block t_block)
(completed_fixed stack)
(pick-up b1)
(fix holding pick-up)
(add-fix-1par holding pick-up b1 t_block)
(completed_fixed pick-up)
```

Figure 1: Excerpt of the solution plan for a single instance.

Definition 4 (**Repaired domain).** *Given an effect-incomplete domain $D^- = \langle Pred, A^- \rangle$ and a reparation $\hat{R}$, the repaired domain is $D^{\hat{R}} = \langle Pred, A^R \rangle$ with the sets:*

$$add(a^{\hat{R}_i}) = add(a_i^-) \cup \{(a_i, p(t)) \mid (a_i, p(t)^+) \in \hat{R}\}$$

$$del(a^{\hat{R}_i}) = del(a_i^-) \cup \{(a_i, p(t)) \mid (a_i, p(t)^-) \in \hat{R}\}$$

The repairing problem for a single problem is defined as:

Definition 5 (**Uninformed repairing problem).** *Given an unsolvable planning task $\Pi^- = \langle D^-, I \rangle$, the uninformed repairing problem consists of determining a reparation $\hat{R}$ such that the planning task $\Pi^{\hat{R}} = \langle D^{\hat{R}}, I \rangle$ is solvable.*

It is named *uninformed* because of the absence of input to repair the domain beyond the problem instance. Unlike other works, it lacks clues about the kind of plans it should generate. Figure 1 shows a part of a plan resulting from solving the uninformed repaired problem for the BLOCKSWORLD domain, where the *stack* and *pick-up* actions lacked the *on* and *holding* effects, respectively. The repair actions are highlighted. Note that the lack of information about the underlying domain leads to estimated reparations and many possible repairing sets for a single planning task.

3 A Method for Joint Repairs

Our approach aims to extend the reparation strategy of a domain for a single problem instance to a set of problems. We draw heavily on the single-instance setting from Gragera et al. (2023). We start by detailing the problem setting formally, which includes an effect-incomplete domain and a set of problem instances (which are assumed to be solvable with the ground-truth domain). The repaired domain is the result of solving the General Domain Repair problem:

Definition 6 (**General Domain Repair problem).** *Given a set of planning problems $P = \{I_1, \ldots, I_K\}$ which all share the same effect-incomplete domain, D^-, and the corresponding set of planning tasks, $\Pi_1^- = \langle D^-, I_1 \rangle, \ldots, \Pi_K^- = \langle D^-, I_K \rangle$, the General Domain Repair task is to compute a general repaired domain $D^{\hat{R}}$, i.e., a valid domain that permits a solution plan π_k for each individual problem instance $I_k \in P, 1 \leq k \leq K$.*

Given this setting, the proposed method to find a general repaired domain $D^{\hat{R}}$ includes the following steps:

1. Solve the uninformed repairing problem for each planning task $\Pi_k^- = \langle D^-, I_k \rangle$ to obtain a myopic repair for each of them (Section 3.1).
2. Search in the space of possible reparations to find one that is valid for the full set of problems (Section 3.2).
3. Extend the algorithm to forbid undesired repairs drawn from step 2, thereby refining the obtained reparations (Section 3.3).

This method not only makes a set of unsolvable planning tasks solvable by achieving the original goals through the same reparation, but it also explicitly points to the changes in the model to fix it. In essence, it serves as a means for generating explanations of unsolvability.

3.1 Pool of Repair Candidates

We solve the corresponding uninformed repairing problem for every $\Pi_k^- = \langle D^-, I_k \rangle$, $I_k \in P, 1 \leq k \leq K$, obtaining a specific reparation $\hat{R}_k$ that renders the current problem instance solvable. A planning task $\Pi_k^- = \langle D^-, I_k \rangle$ may require several repairs to become solvable, i.e., when several action effects are missing from the domain. On the other hand, a planning task might not require a reparation if the flawed actions are not used in the problem configuration considered. This is a natural byproduct of having candidate problems to exhibit the scope of problematic behaviour in the broken domain. When repairing a domain for a single instance, we find one possibility among the many that may exist for the specific problem configuration. Then, when obtaining reparations for a set of problem instances, the variety of repairs that we can obtain is wider. However, some of these repairs may turn out to be identical. For example, if the *stack* action is flawed due to the missing *(on ?x ?y)* effect, problems requiring stacking a block would need to repair some of the actions to stack a block on top of another one. To manage repair repetitions and obtain valuable information about more frequent repairs, we combine the repairs from all problem instances to obtain a *pool of repairs*.

Definition 7 (Pool of repairs). *We define a* ***pool of repairs*** *as* $\mathcal{P} = \{(Rep_1, \{S_{Rep_1}\}), \ldots, (Rep_n, \{S_{Rep_n}\})\}$*, where* Rep_i *is the* i*-th repair and* $\{S_{Rep_i}\}$ *is the subset of problems* $S \subseteq P$ *that include the repair* Rep_i*.*

The following example is a pool of repairs to fix a BLOCKSWORLD domain using 10 problem instances. It shows the repairs (adding new positive effects) and the set of problems that contain the specific repair.

$$
\begin{aligned}
\mathcal{P} = \{&((stack,\ (holding\ ?x)^+),\ \{I_1, \ldots, I_8\}),\\
&((unstack,\ (holding\ ?x)^+),\ \{I_1, \ldots, I_7\}),\\
&((stack,\ (on\ ?x\ ?y)^+),\ \{I_1, \ldots, I_5\}),\\
&((unstack,\ (on\ ?x\ ?y)^+),\ \{I_1, I_2, I_9\}),\\
&((put\text{-}down, (handempty)^+),\ \{I_{10}\})\}
\end{aligned}
$$

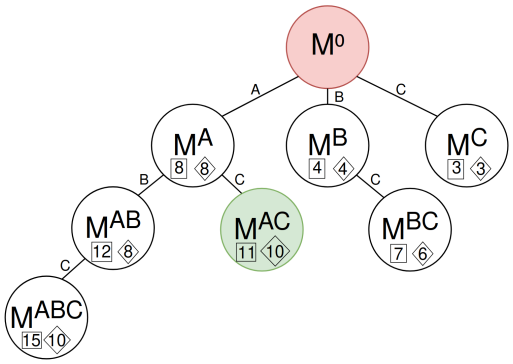


Figure 2: Illustration of model space search for a valid model. Squares and diamonds in each node represent the node priority and the skippability indicator, respectively.

These repair candidates ensure that the problem is solvable for individual problem instances. However, our aim is to obtain from this pool a single reparation (ideally minimal) that works for the entire set of problems.

3.2 Search in Reparation Space

We use the pool of repairs to perform a search in the space of fixed models in order to find one capable of solving the full set of problems. The process consists of a Best-First Search algorithm that receives as input an incomplete domain model D^-, a set of planning tasks $P = \{I_1, \ldots, I_K\}$, and the repair pool $\mathcal{P}$. Figure 2 exemplifies the search process. We denote the initial effect-incomplete domain model D^- as M^0 and the repairs in the repair pool as $\mathcal{P} = \{A, \ldots, Z\}$. Further, M^A represents a domain model repaired with the repair A, M^{AB} represents a model repaired with repairs A and B, and so forth. A node containing a specific model, for example, M^{AB}, is denoted as n^{AB}. Each node n consists of (1) a model fixed with a set of repairs, (2) a priority $\square(n)$, and (3) a goal check skippability indicator $\diamond(n)$ (described below).

The nodes are sorted in the open list first according to their depth, followed by their priority. Processing a node always involves expanding it, although it does not necessarily require checking if it is a goal node (i.e., if the corresponding fixed model solves the set of problems). We now elaborate on the node priority and skippability indicator.

Node priority $\square(n)$. We employ a ranking function to prioritize the processing of nodes whose repairs are more frequent in the pool of repairs. We extract from the pool the count of problems in which a specific repair appears and use that value as the node priority: $\square(n) = |S_{Rep_i}|$. When the node contains more than one repair, the priority is the sum of their individual occurrences. For repairs $\{Rep_1, \ldots, Rep_n\} \in n$, $\square(n) = \sum_{i=1}^{n} |S_{Rep_i}|$. This value is exemplified in Figure 2 by a square in each node. Priority is calculated in the node generation. Nodes with higher priority at the same level will be evaluated first.

Goal check skippability indicator $\diamond(n)$. Checking goal nodes is costly, as it involves verifying if the corresponding model solves the full set of problems. To avoid the need

for constant checking, we compute a *skippability indicator* for each node, defined as the size of the union of all problems where each repair appears for all repairs present in the node. For repairs $\{Rep_1, \ldots, Rep_n\} \in n$, $\diamond(n) = |\bigcup_{i=1}^{n} S_{Rep_i}|$. This value is exemplified in Figure 2 by a diamond in each node. To determine if a node needs to be examined as a potential goal node, its skippability value has to reach a threshold value. This threshold is defined by the number of problems that needed repair during the computation of the repair pool. Following our example, let us consider that we have a set of 10 problems and all of them needed a reparation to make the planning task solvable. The sets of problems where repairs A and C appear are $S_{Rep_A} = \{I_1, \ldots, I_8\}$ and $S_{Rep_C} = \{I_8, I_9, I_{10}\}$. Then, $S_{Rep_A} \cup S_{Rep_C} = \{I_1, \ldots, I_{10}\}$, so $\diamond(n^{AC}) = |S_{Rep_A} \cup S_{Rep_C}| = 10$. It reaches the threshold value, so it is checked as a potential goal. Skippability is calculated in the node evaluation. Nodes whose $\diamond(n)$ does not reach the threshold are expanded but not checked.

We incorporate repairs into the model incrementally, one at a time. Changes in the model are represented as a canonical set, so we do not generate successors like M^{BA} if M^{AB} has already been generated. The solution we are looking for could potentially be a combination of repairs that did not happen when obtaining a reparation for individual problems, so we have to explore every possible combination. The number of nodes at a particular depth is given by the binomial coefficient $\binom{n}{g}$, where n is the size of the pool of repairs and g is the depth. Assuming that the algorithm evaluates every node without skipping any, it would need to assess a total of 2^n nodes. The node priority and the skippability indicator allow us to speed up the search process.

If the node identified as a potential goal solves all of the problems $P = \{I_1, \ldots, I_K\}$, the algorithm halts and returns the model as a valid one (see Def. 6). Through this search, we shift from having a domain $D^{\hat{R}_k}$ for every $I_k \in P, 1 \leq k \leq K$ to having a general valid domain $D^{\hat{R}}$ that can obtain a solution for every problem in the set. If the evaluation result indicates that the model is invalid, the search continues until a valid model is found or the entire search space is explored. If no valid model is found, the algorithm returns the best model found so far, which is the model that solves most problems among those considered.

3.3 Searching for the (Ground) Truth

In the case of exploring the entire search space and not finding a valid model (either because there is no combination of repairs that serve as a valid model or because we skipped all valid models), the best model found solves the majority but not all problems. In addition, an undesired model can be valid. For instance, a model that repairs the *stack* action with the *(holding ?x)* effect can be considered valid but not entirely precise (violates the ground truth).

For that reason, we consider an extension of the algorithm to have access to an oracle that can validate or forbid repairs from the generated solutions, simulating what a user might provide. We are not concerned in this work with usability and user studies of the suggested reparations (though this is an interesting avenue for further research), but rather how difficult it is to iteratively refine the search and compute the ground truth with this limited guidance.

A valid repair is one that appears in the ground truth. Any repairs suggested that do not appear in the ground truth are automatically forbidden by the oracle. To achieve this, we adapted the compilation by introducing a new predicate *(forbidden ?a - action ?p - predicate)*. The repair actions included in the compiled task take an action and a predicate symbol to repair it as a new effect, adding to the state *(fix ?a - action ?p - predicate)*. So we also added the forbidden literal as a negative precondition to every repair action. In this way, if the "stack" action should not add the atom "holding", *(forbidden stack holding)* is added to the initial state, preventing that specific repair. Extending the example shown in Figure 2, let us consider that the node n^{AC} is a goal one, so the model M^{AC} is a valid model. The oracle identifies the repair A as valid, whereas the repair C is forbidden. Now, two possibilities arise: (i) repair A alone is enough to achieve a valid model, or (ii) additional repairs are required, excluding repair C in any scenario.

(i) This scenario could arise when we have problems where A is not the repair found but could have been. In the first level, A was not checked as a goal, it is only in level two that its skippability indicator has obtained coverage to be checked. After the oracles' activation, if it authorizes valid repairs, we always verify if they suffice to hold a valid model. If they prove sufficient and no more repairs are required, the algorithm stops and returns the valid model.

(ii) If further repairs are needed, the model M^A replaces M^0 as the new initial model and a new pool of repairs is computed. Note that the new pool of repairs obtained in this case is different from the previous one since the initial model has been refined, and repair C is not allowed anymore as a possible repair. This promotes new and more desirable solutions. The process iterates in this way until reaching the ground truth.

The search in the model space, as explained in Section 3.2, does not provide a guarantee in two aspects: (1) obtaining a minimal reparation, which can be affected by the skippability criteria implemented, and (2) achieving completeness, as we can skip goal nodes and there is a possibility that a reparation for a valid model may not be present in the initial pool of repairs. The approach in this section, conversely, provides both such guarantees:

Theorem 1 (Completeness)**.** *Given a General Domain Repair problem with a set of unsolvable tasks* $\Pi_1^- = \langle D^-, I_1 \rangle, \ldots, \Pi_K^- = \langle D^-, I_K \rangle$*, sharing an incomplete domain* D^-*, the iterative approach above will eventually find a reparation* $\hat{R}$ *such that* $D^{\hat{R}}$ *makes* $\Pi_k^- = \langle D^-, I_k \rangle$*,* $I_k \in P, 1 \leq k \leq K$ *solvable, whenever such a reparation exists.*

Theorem 2 (Minimal Reparation)**.** *Given a General Domain Repair problem with a set of unsolvable tasks* $\Pi_1^- = \langle D^-, I_1 \rangle, \ldots, \Pi_K^- = \langle D^-, I_K \rangle$*, sharing an incomplete domain* D^-*, the iterative approach above will find a minimal*

reparation (measured in number of repairs) $\hat{R}$ such that $D^{\hat{R}}$ makes $\Pi_k^- = \langle D^-, I_k \rangle$, $I_k \in P, 1 \leq k \leq K$ solvable.

The confirmation steps in this approach enable us to return to the beginning of the algorithm for repeated trials, refining the model in each iteration. This iterative process ensures that we eventually find a minimal reparation. The effort to obtain these properties motivates our evaluation.

4 Evaluation

Our General Domain Repair problem has a range of measurable outputs that are explored in this section to quantify the time and effort required to obtain the ground truth repair, as well as understand how these variables affect the coverage. We conduct the evaluation through the four research questions below and detail our findings.

4.1 Research questions

Q1 – How much time is needed? We compare the time required to obtain a first valid solution with the time needed to obtain the ground truth. We distinguish between the time spent in obtaining the pool of repairs and the time spent in searching for a valid model reparation.

Q2 – How much effort is needed? This effort can be expressed in (1) the number of iterations needed, which represents how many times the model needs to be refined, and (2) the number of corrections made during the refinement process, which is the number of forbidden repairs. The number of iterations is important because it means obtaining a new pool of repairs, which requires additional time. While we could also measure the number of questions, which refers to the total repairs evaluated by the oracle to decide their validity, focusing just on the number of corrections is more representative since the number of questions also depends on the number of missing effects in the domain.

Q3 – What is the impact of the number of missing effects from the domain? In the proposed benchmark, we also consider scenarios where some effects are missing from the domain simultaneously. While the previous questions refer to aggregated solutions over all outputs, this specific question aims to analyze how the number of removed effects influences each of the aforementioned outputs: time, number of iterations, and number of corrections.

Q4 – How do time and effort affect the coverage? Finally, we examine the influence of time and effort on the effectiveness of the process. Ideally, both should be kept minimal, but intuitively, we expect to observe a higher coverage as the time and effort increase. By coverage, we denote the total of times that we obtain the ground truth reparation.

4.2 Experimental Setting

To answer previous questions, we empirically evaluate the proposed approach on a benchmark set consisting of seven planning domains from the International Planning Competition (IPC), increasing in the number of actions (in brackets): TRANSPORT (3), BLOCKSWORLD(4), SATELLITE(5), CHILDSNACK(6), GOLDMINER(7), ROVERS(9) and BARMAN(12). For each domain, we use a PDDL problem generator[1] to create 10 problem instances of increasing difficulty (Seipp, Torralba, and Hoffmann 2022). To generate flawed domains, we randomly removed action effects from the lifted representation of the domain. For each domain, we generated four sets of 10 domains each: the initial set with one effect removed, the second set with two effects removed, and so on, up to four effects removed simultaneously from the domain. In total, we obtained 40 effect-incomplete domains for each domain. For each domain, their effect-incomplete domains are solved by using the same set of 10 problems, which leads to obtaining a specific $\hat{R}_k$ for each $\Pi_k^- = \langle D^-, I_k \rangle, 1 \leq k \leq 10$. We merge these repairs to create the repair pool. We then conduct a search process within this pool to obtain a valid reparation.

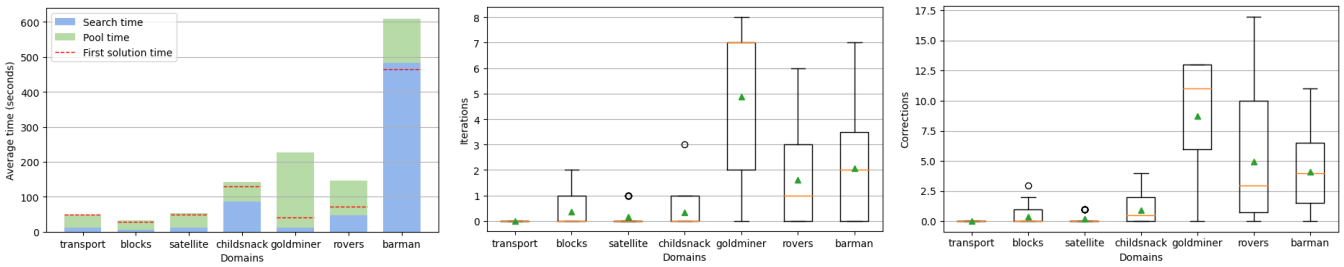
We use the Fast-Downward (Helmert 2006) planning system to obtain a reparation for each $\Pi_k^- = \langle D^-, I_k \rangle$ and form the pool of repairs: we run the LAMA planner (Richter and Westphal 2010) and keep the best plan found in a time window of 5 seconds. If no reparation is found for a specific problem instance, it is not very significant since the repairs found for the other problems may potentially apply to that one. To check potential goal nodes in the search for a model, we use LAMAFIRST, as we are only concerned whether a problem has a solution. The experiments were run on an Ubuntu machine with Intel(R) Core(TM) i7-7700HQ CPU running at 2.80GHz and 16GB memory.

4.3 Results

Q1 – Time results. Figure 3a depicts the average time to obtain the ground truth reparation. We subdivided the aggregated time into the time spent obtaining the pool of repairs (in green) and the time spent searching for a valid model within that pool (in blue). In practice, both processes are interleaved, but the plot shows the aggregated average times for each of them. The red dashed line inside each bar represents the total average execution time for each domain to obtain *any* repair that explains all inputs. In this case, it is the combined searching plus voting time. Since the top of the colored bar represents the time required to obtain the ground truth, the time to achieve a first solution is obviously always lower or equal to that time. When the red line is close to the top of the bar, it indicates that, in most cases, the first solution found corresponds to the ground truth reparation.

Domains are placed in the plot according to their size, with TRANSPORT being the smallest one. The increasing trend in time is evident, as larger domains present more potential ways to be repaired, leading to longer processing times. For example, for the barman domain, the time spent on voting and searching for a first solution is less than the time spent just on searching for the ground truth. A special case is GOLDMINER, where the goal for all problem instances is the 0-arity predicate *holds-gold*. The compilation process applied to obtain repairs penalizes directly adding the goals, but when the action that adds this goal is broken, it needs to add it anyway. In such situations, the cheapest

[1] https://github.com/AI-Planning/pddl-generators



(a) Average times (b) Iterations (c) Corrections

Figure 3: Figure 3a refers to Q1. It shows the averages time per domain to get the ground truth reparation, subdivided in searching and obtaining the pool of repairs. Red dashed lines mean the time to obtain a first solution. Figures 3b and 3c refer to Q2. They represent the distribution of the number of iterations and corrections per domain, respectively.

repair is to simply include *holds-gold* as an effect of any action, even if there are other actions that are broken. The repair pool always includes the goal as an effect of any action, so the search time is minimal. This repair has to be iteratively forbidden by the oracle, computing a new repair pool each time until the ground truth is found.

Q2 – Effort results. Figures 3b and Figure 3c refer to Q2-(1) and Q2-(2), respectively. Orange lines depict the median of the data, and green dots indicate the average value. Individual values outside the boxes significantly deviate from the rest of the data and are considered outliers.

Figure 3b shows the distribution of the number of iterations required to obtain the ground truth reparation across all domains. The represented data does not take into account obtaining the initial pool of repairs, it just focuses on the number of times that new pools are needed to refine the model. Concerning the initial solution (i.e., obtaining a first valid model), the domains TRANSPORT, BLOCKS, SATELLITE, and GOLDMINER always find a first solution in the initial iteration. In the case of CHILDSNACK, new pools of repairs only need to be computed in two instances out of a total of forty (2/40). For ROVERS and BARMAN, these values are (4/40) and (9/40) respectively.

Figure 3c shows the distribution of the number of corrections required to achieve the ground truth reparation. The aforementioned peculiarity of the GOLDMINER domain is evident, where the repairs found render the model valid, but they do not align with the ground truth. In the ROVERS domain, most actions collect samples and communicate data. The only difference lies in the type of material they handle, leading to a semantic distinction. Repairing the domain by communicating *rock* data in the action designated for *images* results in a valid model, but it is still not the desired fix. Such reparations are frequent in this domain, which increases the number of corrections needed.

Q3 – Impact of the effects removed results. Table 1 shows how the number of simultaneous missing effects from the domain (#) influences the time and effort to obtain the ground truth (GT).

As mentioned above, the complexity of repairing a flawed domain typically increases as the domain size grows. The number of missing effects from the domain also plays an important role, with the averages of time, iterations, and corrections increasing along with the number of missing effects. This is not entirely surprising, as a less well-defined domain presents more potential ways to repair it, so finding the ground truth becomes more challenging.

#	Time GT (s)	# Iterations	# Corrections
1	43.6 ± 20.3	0.2 ± 0.4	1.1 ± 1.1
2	111.7 ± 99.2	1.3 ± 2.0	2.6 ± 3.7
3	201.6 ± 200.3	1.8 ± 2.6	3.3 ± 4.5
4	391.1 ± 647.0	2.4 ± 2.9	4.5 ± 5.1

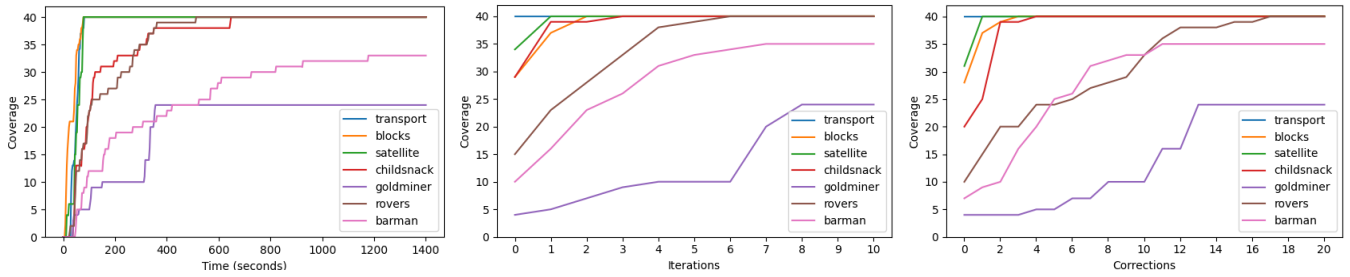
Table 1: Aggregated impact of the number of missing effects (#) to obtain the ground truth over all domains. Each cell represents the average of repairing 70 flawed domains (10 per domain), each with 10 problems.

Q4 – Ratios results The plots presented in Figure 4 provide a clearer perspective on how time, iterations, and corrections impact the coverage of the benchmark. By coverage, we refer to the count of incomplete domains (over 40 per benchmark), each evaluated with 10 problems, for which the ground truth is obtained. For larger domains, reaching the ground truth requires more iterations and corrections, potentially reducing coverage. At the beginning of the process, there exist several potential ways to repair the domain, making reparations easy and quick to find in most cases. As iterations progress, constraints are introduced to the model, limiting the number of valid solutions and adding complexity. This is especially challenging when forbidden solutions are found, and the model is not refined due to the absence of valid solutions. Those cases might lead to an empty repair pool, with no available candidate repairs to fix the model.

4.4 Baseline Comparison

In this subsection, we consider the validity of the initial pool of repairs as a valid model as a basic baseline. This pool is composed of merged repairs from different problem instances. However, one may conjecture that all these repairs together have the potential to offer valid reparations.

Out of a total of 280 domains to repair, the initial repair pool aligns with the ground truth in only 70 cases, which



(a) Time (b) Iterations (c) Corrections

Figure 4: Figures 4a, 4b and 4c refer to Q4. They show the coverage obtained based on time, number of iterations and number of corrections, respectively. Coverage refer to cases when we achieve the ground truth reparation.

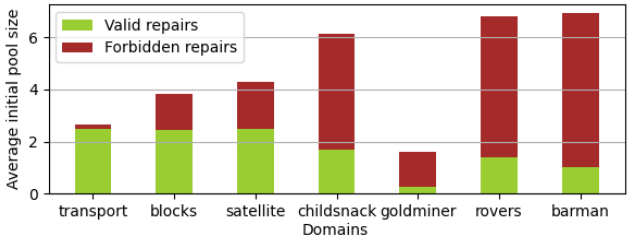


Figure 5: Average distribution of the pool of repairs based on the number of valid/forbidden repairs in it.

means that it achieves a coverage of 0.25% with no search. However, with our proposed approach, this value jumps to 140 by simply applying the suggested search within the existing pool without any corrections.

When completeness and minimal reparation are not a concern, the initial repair pool works as a solution in 257 tasks, meaning that in most cases, repairing the domain using the entire pool of repairs results in a valid model. However, it is worth analyzing the quality of these repairs. Ideally, the size of the pool of repairs (we refer to this size as the count of distinct repairs in the pool) should range between one and four, as the number of missing effects from a broken domain simultaneously is four at most. However, in the majority of cases, these values are greatly exceeded, and we can find initial pools up to size 14 for ROVERS and BARMAN domains.

Figure 5 compares the ratio of valid and forbidden repairs within the average sizes of these pools. In larger domains, around 70% of the repairs in the initial pool do not align with the ground truth and, consequently, do not match the expected repairs that should be incorporated into the model.

To summarize, even if applying the complete repair pool results in a valid model, the quality of that model is relatively low. The search process applied to the pool filters the solutions, which have a reduced number of repairs to validate and guarantee a valid model. In contrast, the initial pool is larger and lacks information about the validity of the solution. Moreover, missing repairs that should be part of the model can only be retrieved through the iterative process.

In this work, we are limited to existing domains and problem instances generated using the problem generator provided by the planning community, which in most cases produces the same set of goals for all problem instances. This lack of variation may not fully capture the domain behaviour and can restrict the pool of repairs available. In the case of GOLDMINER, if all goals involve holding gold, then all fixes will be oriented to achieve that specific goal. This highlights the importance of having a diverse and representative set of problems to effectively fix a domain.

5 Related Work

We focus on works that plan with incomplete models or repair a flawed planning task when no suggested plans are available. Within this framework, Sreedharan et al. (2019) explain the unsolvability of a planning task by identifying unreachable subgoals, derived from abstract and solvable models using planning landmarks (Hoffmann, Porteous, and Sebastia 2004). Counterfactuals theory (Ginsberg 1985) has been used by Göbelbecker et al. (2010) to propose alternative initial states that would make the task solvable. However, both works only apply to the initial state and do not consider changes in the domain model. In planning with approximate domain models, some works try to supply the lack of information using examples of action sequences or annotations about where the specification is incomplete (Garland and Lesh 2002; McCluskey, Richardson, and Simpson 2002; Nguyen, Sreedharan, and Kambhampati 2017). Related work in HTN domains solves the case when there is no valid decomposition tree by inserting actions into the task hierarchy (Xiao et al. 2020). While learning domain approaches present a different problem setting, i.e. inferring a domain model from observations, they can be perceived as a form of model reparation, particularly when the model is partially specified. The closest work that in this field is FAMA (Aineto, Jiménez, and Onaindia 2018), where the problem is compiled into classical planning with conditional effects. Although they prove their approach to be valid for a different range of knowledge input and the ability to build the model without plan traces, their strength is in cases where input plans are available. They focus on obtaining any valid model, but they do not refine it. In contrast, our two-step approach involves a compilation that generates repair candidates, guided to avoid trivial solutions, followed by an iterative search process that refines undesired reparations.

In addition to following a similar approach performing

a search in the space of models, our work can be viewed as an example of *model reconciliation* (Chakraborti et al. 2017), where the solution serves to reconcile the human's mental model (expected to be solvable) and the unsolvable model they employ. Our contribution extends to both *model-lite planning* (Kambhampati 2007) and *explainable planning* (Fox, Long, and Magazzeni 2017). We not only plan using an incompletely specified model (successfully achieving the original goals for all problems), but also, the corrections made in the domain serve as explanations of unsolvability (identifying what was missing in the domain).

6 Conclusion

We presented an approach to repair flawed domains when planning tasks are unsolvable due to missing action effects, extending the existing literature by incorporating multiple problem instances. Our approach generates a pool of repairs derived from individual problem repairs and uses this pool to perform a search in the space of fixed models. The solution is a combination of repairs that, when applied to the flawed domain, render the set of planning tasks solvable. When assuming access to an oracle that can reject proposed repairs, we demonstrated the opportunity of an iterative approach to finding the ground truth repair set. We assessed the time and effort required to obtain the ground truth reparation, both in terms of the number of times that the model needed to be refined and the number of corrections made. Larger domains increase the number of potential reparations, consequently extending the time and effort required to achieve the ground truth reparation. Having a diverse and representative set of problems with a variety of goals can help in the repair process. As a potential future research, we aim to explore whether incorporating semantic knowledge into the repair process could enhance its effectiveness.

Acknowledgments

This work was partially funded by grants PID2021-127647NB-C21 and PDC2022-133597-C43 from MCIN/AEI/10.13039/501100011033, by the ERDF "A way of making Europe" and Next Generation EU/ PRTR and by the Madrid Government under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3M17) in the context of the V PRICIT (Regional Programme of Research and Technological Innovation). This project has been supported by the Doctoral School of UC3M.

References

Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning STRIPS Action Models with Classical Planning. In *Proceedings of ICAPS 2018*, 399–407. AAAI Press.

Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2020. The Emerging Landscape of Explainable Automated Planning & Decision Making. In *Proceedings of IJCAI 2020*, 4803–4811. ijcai.org.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy. In *Proceedings of IJCAI 2017*, 156–163. ijcai.org.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable Planning. *CoRR*, abs/1709.10256.

Garland, A.; and Lesh, N. 2002. Plan Evaluation with Incomplete Action Descriptions. In *Proceedings of AAAI 2002*, 461–467. AAAI Press / The MIT Press.

Ginsberg, M. L. 1985. Counterfactuals. In *Proceedings of IJCAI 1985*, 80–86. Morgan Kaufmann.

Göbelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming Up With Good Excuses: What to do When no Plan Can be Found. In *Proceedings of ICAPS 2010*, 81–88. AAAI.

Gragera, A.; Fuentetaja, R.; Olaya, Á. G.; and Fernández, F. 2023. A Planning Approach to Repair Domains with Incomplete Action Effects. In *Proceedings of ICAPS 2023*, 153–161. AAAI Press.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.*, 26: 191–246.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. *J. Artif. Intell. Res.*, 22: 215–278.

Kambhampati, S. 2007. Model-lite Planning for the Web Age Masses: The Challenges of Planning with Incomplete and Evolving Domain Models. In *Proceedings of AAAI 2007*, 1601–1605. AAAI Press.

Lin, S.; and Bercher, P. 2021. Change the World - How Hard Can that Be? On the Computational Complexity of Fixing Planning Models. In *Proceedings of IJCAI 2021*, 4152–4159. ijcai.org.

Lin, S.; Grastien, A.; and Bercher, P. 2023. Towards Automated Modeling Assistance: An Efficient Approach for Repairing Flawed Planning Domains. In *Proceedings of AAAI 2023*.

McCluskey, T. L.; Richardson, N. E.; and Simpson, R. M. 2002. An Interactive Method for Inducing Operator Descriptions. In *Proceedings of AIPS 2002*, 121–130. AAAI.

McDermott, D.; et al. 1998. The planning domain definition language manual. Technical report, Technical Report 1165, Yale Computer Science, 1998.(CVC Report 98-003).

Nguyen, T.; Sreedharan, S.; and Kambhampati, S. 2017. Robust planning with incomplete domain models. *Artif. Intell.*, 245: 134–161.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.*, 39: 127–177.

Seipp, J.; Torralba, Á.; and Hoffmann, J. 2022. PDDL Generators. https://doi.org/10.5281/zenodo.6382173.

Sreedharan, S.; Srivastava, S.; Smith, D. E.; and Kambhampati, S. 2019. Why Can't You Do That HAL? Explaining Unsolvability of Planning Tasks. In *Proceedings of IJCAI 2019*, 1422–1430. ijcai.org.

Xiao, Z.; Wan, H.; Zhuo, H. H.; Herzig, A.; Perrussel, L.; and Chen, P. 2020. Refining HTN Methods via Task Insertion with Preferences. In *Proceedings of AAAI 2020*, 10009–10016. AAAI Press.