Repair Suggestions for Planning Domains with Missing Actions Effects

Alba Gragera, Ángel García-Olaya, Fernando Fernández

Universidad Carlos III de Madrid, Leganés, 28911, Madrid, Spain
agragera@pa.uc3m.es, {agolaya, ffernand}@inf.uc3m.es

Abstract

In Automated Planning (AP) a proper definition of the domain and problem files is assumed. However, producing complete model descriptions is a time consuming and challenging task, especially for non-experts. It is easy to make mistakes when creating formal models, turning the planning task unsolvable for the planners. This can happen if the initial state is not fully and properly specified, some actions are missing, or some actions are incomplete. Explaining the absence of solution in such cases is essential to help humans in the development of AP tasks. In this paper we focus on repairing planning models where the effects of some actions are incomplete. We propose a compilation of the unsolvable task to a new extended planning task, where the actions are allowed to insert possible missing predicates in their effects. The solution to such task is a plan that achieves the goals of the original problem while also warning about the modifications that were necessary to do so. Experimental results show this approach can be effectively used to repair incomplete planning tasks across different planning domains.

1 Introduction

Automated Planning (AP) is a general problem-solving technique for a wide range of scenarios and goals (Ghallab, Nau, and Traverso 2004). Planning tasks are composed by a domain description, which specifies all available actions and the predicates used to describe the states; and a problem description that contains the initial and goal states. Given a solvable and well-defined task, and assuming infinite memory and time resources, a planner will return a solution. However, there are some scenarios where we can ensure neither completeness nor correctness in the planning task specification (Kambhampati 2007). Flaws in the task model can appear due to a noisy acquisition process, because domain engineers are not experts in the description language or because they do not have deep knowledge of the current task, specially when the domain is tough to represent. This can lead to a wrong or not full specification of the initial state, or an inaccurate actions' definition. Such issues can turn the planning task unsolvable: due to a loss of information, there is no way to achieve the goals from the initial state.

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Previous works have considered initial states from which the goals cannot be achieved (Sreedharan et al. 2019; Göbelbecker et al. 2010). They provide explanations and different alternatives which would turn the task solvable, although they do not consider alterations in the domain and assume a proper specification. But just making changes to the initial state is not enough in some cases. Let us consider the well known barman domain (Linares López, Jiménez Celorrio, and García-Olaya 2015), where a robot prepares drinks using glasses that need to be empty and clean to be filled. Forgetting to include the action effect that cleans the glass (Figure 1) will turn the task unsolvable, but setting the glass as clean in the initial state will not solve the problem, since it will get dirty again. A better option would be to repair the operator and allow the robot to clean the glass.

```
(:action clean-shot
   :parameters (?s - shot ?b - beverage ?h1 ?h2 - hand)
   :precondition (and (holding ?h1 ?s) (handempty ?h2)
                      (empty ?s) (used ?s ?b))
   :effect (and (not (used ?s ?b)) (clean ?s)))
```

Figure 1: Action to clean a shot. If the positive effect *(clean ?s)* is removed, the planner is not able to find a plan.

Due to the number of potential changes to the set of operators, repairing faulty domains is not trivial. Previous works focusing on incomplete domains assume guidelines from the user side to supply the lack of knowledge (McCluskey, Richardson, and Simpson 2002; Simpson, Kitchin, and McCluskey 2007; Nguyen, Sreedharan, and Kambhampati 2017; Garland and Lesh 2002).

This paper introduces an Automated Planning domain reparation procedure for cases where some action effects are missing, receiving no information other than the unsolvable task. Since some of the common mistakes when modelling an AP task are more likely to return an erroneous or undesirable plan (which can be supervised by the user), in this work we consider unsolvable tasks, where no plan is provided by the planner. Therefore we focus on the absence of positive action effects, which can seriously challenge the task solvability. Given a domain with incomplete positive effects for some actions, we compile the unsolvable task into a new extended plannning task, providing operators to repair any

action of the underlying domain with possible missing predicates in their effects. The solution is a plan that achieves the goals while also including the operators warning about what modifications were made to repair the domain model.

In the remainder of the paper we present the background in AP, followed by the problem formulation in Section 3. The compilation of the extended planning task is detailed in Section 4.1, divided into the unsolvable planning task compilation and general operators to repair a planning domain. Completeness and soundness of the compilation is discussed in Section 5. Section 6 contains some problems related to domain reparation identified in the course of this work. We propose a metric to minimize to solve them, detailed in the same section. Sections 7 and 8 include the experiments conducted and a discussion about our approach. Finally, we mentioned the related work and main conclusions of the work.

2 Background

Automated Planning is the task of driving a system from the initial state to the state where the goals are met by applying a set of operators, called plan. A classical planning task can be defined as a tuple $\Pi = \langle F, A, I, G\rangle$, where F is the set of propositions instantiated with typed objects O, A is the set of actions, $I \subseteq F$ defines the initial state and $G \subseteq F$ specifies the goals to achieve. Each action $a \in A$ is described as a tuple $\{pre(a), add(a), del(a)\}$ representing a set of preconditions ($pre(a)$) that must be true in a state to execute the action and a set of effects describing which propositions are added ($add(a)$) or deleted ($del(a)$) from the state after its execution. Additionally, an action can also involve a cost $c(a)$. Each planning task is divided into the *domain* action model and the current *problem* containing the initial state and the goals to achieve. If the planning task is completely and correctly defined, the planner returns the grounded sequence of actions to solve the problem. Then, a plan $\pi = \{a_1, \ldots, a_n\}$ is a solution of Π if there exists a sequence of states $\langle s_1, \ldots, s_n\rangle$, where $s_0 = I$ and for an action a_i to be applicable in a state s_i, $pre(a_i) \subseteq s_i$ and $s_{i+1} = s_i \cup add(a) \setminus del(a)$, ending in a state s_n such as $G \subseteq s_n$. The cost of the resulting plan is defined as $c(\pi) = \sum_{a_i \in \pi} c(a_i)$. When using an optimal planner, it returns the plan with a minimum cost.

3 Problem Formulation

The use of Automated Planning usually relies on properly and complete specified models. However, errors during the modelling of planning tasks may result in faulty domains. In this work we consider errors as incomplete action definitions in the domain model, giving rise to planning tasks where some actions needed to achieve the goal cannot be instantiated due to a missing predicate needed, and none of the other actions in the domain can add it, turning the task unsolvable. We define such unsolvable problem as an *add-incomplete planning* task Π^-. Formally:

Definition 1 (*Add-incomplete* planning task)**.** *From an underlying planning task* $\Pi = \langle F, A, I, G\rangle$*, a planning task* $\Pi^- = \langle F, A^-, I, G\rangle$ *is add-incomplete with respect to* Π *iff:*

(i) $\forall a_i^- \in A^-, \forall a_i \in A$, $pre(a_i^-) = pre(a_i)$, $del(a_i^-) = del(a_i)$, $add(a_i^-) \subseteq add(a_i)$

(ii) $\exists a_i^- \in A^-$ *such that* $add(a_i^-) \subset add(a_i)$

For an *add-incomplete* planning task there always exists at least one $a_i^- \in A^-$ whose positive effects are a proper subset of the original action $a_i \in A$. We will focus just on the cases where the missing effects turn the task unsolvable. In any other case, the found plan may not be semantically correct, but this can be only detected by the domain designer. Following the barman running example, an add-incomplete planning task could have the missing effect *(clean ?s)* for the action *clean-shot*. Since the glass has to be clean to prepare the beverage and no other actions can clean it, the task has no solution. Considering that different flaws can be spread over all the domain, other actions may also have missing positive effects. The solution consist in finding and repairing the add-incomplete planning task with any set of pairs *predicate - action* (denoted as $\mathcal{M}_{\Pi^-}$) that makes the task solvable, provided that $|\mathcal{M}_{\Pi^-}| > 0$. Otherwise, the resulting plan will not include any repairation, since the domain would be completely specified. We define a valid solution as:

Definition 2 (Valid solution of Π^-)**.** *Given a task* Π *and an add-incomplete task* Π^- *with respect to* Π*, any set* $\mathcal{M}_{\Pi^-} = \{\langle f, a_i^-\rangle \mid f \in F, a_i^- \in A^-\}$*, is a valid solution of* Π^- *if* $add(a_i)^- \cup f$ *makes solvable* Π^-*, for all* $\langle f, a_i^-\rangle \in \mathcal{M}_{\Pi^-}$.

Considering the barman domain again and according to that, a valid solution may not fix the action *clean-shot* with the *clean* effect, linking the predicate to another action instead. However, we consider it a solution of the problem since it makes the task solvable. But assuming that the underlying task had been properly specified, we are interested in finding the precise set $\mathcal{M}^*_{\Pi^-}$ desired by the domain designer, such that $\mathcal{M}^*_{\Pi^-} = \{\langle f, a_i^-\rangle \mid f \in add(a_i) \wedge f \notin add(a_i^-)\}$ for $f \in F, a_i \in A, a_i^- \in A^-$. In other words, we want to find the predicates $f \in F$ that would be present in the positive effects of the actions $a_i \in A$ of the underlying domain but not in those of its corresponding action $a_i^- \in A^-$. We define an optimal solution of the *add-incomplete* planning task as follows:

Definition 3 (Optimal solution of Π^-)**.** *Let be* Π^- *an add-incomplete task with respecto to the complete task* Π*, and* $\mathcal{M}_{\Pi^-}$ *as a solution of* Π^-*, such solution is optimal if* $\mathcal{M}_{\Pi^-} = \mathcal{M}^*_{\Pi^-}$*. Consequently,* $\Pi^- = \Pi$.

Therefore, the objective of an *add-incomplete* planning task is finding the set of missing effects $\mathcal{M}_{\Pi^-}$ that makes solvable Π^-, ideally getting the repair to match the domains of the two tasks. It is important to highlight that Π and $\mathcal{M}_{\Pi^-}$ are unknown when solving Π^-, we only use them to compare the results of the proposed method. Our approach to solve the *add-incomplete* planning task is the topic of the next section.

4 Compilation to Classical Planning

The proposed extended task is divided into two parts: a compilation of the unsolvable planning task (Section 4.1) and

general operators to repair any given task (Section 4.2). At the conceptual level, every action is now divided into three different parts, where each of them corresponds to a different operator: (1) the instantiation of the action with its current effects (from the domain compilation), and (2) the reparation of the action and (3) its end (from the general operators), where it performs the reparation if needed and close the current action. Considering the running example, the planner first instantiates the *clean-shot* action. Since it is incomplete and it can not continue without actually cleaning the glass, it will fix it by applying a repair operator and, if no further repairs are necessary for that action, it will close such phase to instantiate a new action. This process is repeated until the goals are achieved. If an action does not need to be repaired, it is instantiated and closed after that, omitting the reparation. So the add-incomplete planning task Π^- is reformulated to a new extended planning task $\Pi' = \langle F', A', I', G' \rangle$, where

- F' is the set of predicates defined for the task reparation, being part of the planning task reformulation (Section 4.1).
- $A' = A^- \cup R$ is the set of the original domain actions, compiled to manage the reparation (Section 4.1), plus the set of general repair operators to fix actions with possible missing effects (explained in Section 4.2).
- I' contains the initial state of the original problem plus extra information about the domain actions (Section 4.1).
- G' is the set of goals of the underlying task in addition to new goals to ensure that all the actions of the domain have been checked for flaws (Section 4.1).

4.1 Unsolvable Planning Task Compilation

In order to repair the unsolvable planning task, we reformulate its planning elements to be managed by the planner in the new extended planning task. Types, constants, predicates and actions constitute the domain, while the initial state and goals are reformulated in the problem. Both are detailed in this section.

Types In Π' **actions** are fixed with **predicates** as new positive effects. We consider these two elements by defining them as new types of the extended domain. The reasoning process also involves the types of the original task, which are defined now as sub-types of the new *itemdomain* type. Figure 2 shows the types involved in the original barman domain and Figure 3 shows their reformulation in the extended planning task.

```
(:types
   hand level beverage dispenser container - object
   ingredient cocktail - beverage
   shot shaker - container)
```

Figure 2: Types definition of the barman original domain.

Constants Predicates and actions reformulated as types will be constant across all problems, so we define their names as domain constants, as shown in Figure 4.

```
(:types
    action predicate itemdomain - object
    hand level beverage dispenser container - itemdomain
    ingredient cocktail  - beverage
    shot shaker  - container)
```

Figure 3: Reformulated types definition in Π'.

```
(:constants
    grasp leave fill-shot clean-shot ... - action
    handempty empty clean used shaked ... - predicate)
```

Figure 4: Action and predicate names of the barman domain reformulated as domain constants in Π'.

Predicates Once an action is repaired in Π', the predicate defined as new positive effect has to be added to the current state. To generalize the compilation for any proposition in the domain, we reformulate them with a new predicate containing the predicate itself, representing if it is present in the current state. We represent predicates with the same arity n (the number of objects $o \in O$ involved in the proposition) through *(in_state(n) ?p - predicate* $\{o_0, \ldots, o_{n-1}\}$ *- itemdomain)*, which will contain the name of the predicate and the objects involved. It creates a predicate $in_state(n)$ for each arity found in F, replacing n by the current arity number. Let us consider the predicates of the barman domain *(clean ?c - container)* and *(empty ?c - container)*, with arity 1. Thanks to this, they can be represented by the same predicate *(in_state1 ?p - predicate ?o - itemdomain)*. Then, if the shot is clean or empty in the current step our new task does not represent *(clean shot01)*, but *(in_state1 clean shot01)* or *(in_state1 empty shot01)* . This lets us generalize our compiled task for any proposition in the domain, allowing the repair operators to add or remove predicates in general from the current state.

Actions The reparation process manages what is the $current_action(a)$ being instantiated, that is $open$ to be repaired, if necessary. The system has to $check(a)$ every action to verify if it lacks on some effects. If the action is repaired, it is also tagged as $patched(a)$ and $fixed(a, f)$, pointing that the action has already been repaired with the predicate f. We emphasize that every action is fixed in its first use and applies its reparation whenever it is used afterwards, so predicates in italic are used to ensure that, once an action has been fixed, it always adds its new effects before its closure. To handle the process, we reformulate each $a_i^- \in A^-$ as $a' = \{pre(a)', add(a)', del(a)'\}$, such that:

$$pre(a)' = \{in_state(n)(f, \{o_0, \ldots, o_{n-1}\}) \mid f \in pre(a^-))\} \cup \neg open$$

$$add(a)' = \{in_state(n)(f, \{o_1, \ldots, o_n\}) \mid f \in add(a^-))\} \cup \{current(a), \bigwedge_{p=1}^{p} used_object(o_p), open\}$$

$$del(a)' = \{\neg in_state(n)(f, \{o_1, \ldots, o_n\}) \mid f \in del(a^-))\}\}$$

For the application of an action, there must not be other *open* phase at that time step. Positive effects now also include a predicate $used_object(o_i)$ for each object in the parameters of the action. It prevents the use of objects that are not involved in the action during the reparation phase: if the clean-shot action is instantiated with the object currently in use *shot01*, the reparation has to be made on the same object. Figure 5 shows the clean shot action as compiled. This conversion is performed for every action present in Π^-, generating a new compiled domain whose actions are part of the new extended planning task Π'.

```
(:action clean-shot
   :parameters (?s - shot ?b - beverage ?h1 ?h2 - hand)
   :precondition (and (in_state2 holding ?h1 ?s)
                      (in_state1 handempty ?h2)
                      (in_state1 empty ?s)
                      (in_state2 used ?s ?b)
                      (not (open))
   :effect (and (not (in_state2 used ?s ?b))
                (current_action clean-shot)
                (used_object ?s) (used_object ?b)
                (used_object ?h1) (used_object ?h2)
                (open)))
```

Figure 5: Clean shot action compiled for the reparation task. It is important to note that the positive effect *clean ?s* is omitted and needs to be added by the planning process.

Initial State and Goals Compiling the problem means to adapt the initial state and goals to the new domain definition. For each $f \in I, f \in G$ with arity n we also reformulate them as *in_state(n)(f,* $\{o_0, \ldots, o_{n-1}\}$*)*. In addition, we extend the problem with information on already existing actions' effects, an input useful to guide the reparation process. Such process has to check every action to verify if it is complete, so we also extend the goals to require it. Then, for every action $a^- \in A^-$ in Π^- we modify the problem state as:

$$I' = \{in_state(f) \mid f \in I\} \cup \{add_eff(f, a) \mid f \in add(a^-)\} \cup \{del_eff(f, a) \mid f \in del(a^-)\}$$

$$G' = \{in_state(f) \mid f \in G\} \cup \{check(a) \mid a^- \in A^-\}$$

Figure 6 shows an example of the problem resulting from the add-incomplete planning task compilation, where information from the domain actions is also added to the problem.

4.2 General Operators to Repair a Planning Task

Since at least one action lacks some positive effect(s), the planning task is extended with general operators to repair them. We split the reparation process into two operators: one action, called `warning`, is in charge of detecting the error and linking an action with a missing predicate, while other action, `repair`, will add such predicate to the state with its proper objects. Following the running example, one action will repair the action *clean-shot* with the predicate *clean* just once, while other will add that predicate to the state with its

```
(:init
    (in_state1 clean shot01)
    (in_state1 empty shot01)
    (in_state2 dispenses dispenser01 ingredient01)
    (del_eff used clean-shot)
    ...)
(:goal (in_state2 contains shot1 cocktail4)
       (check clean-shot)
       ...)
```

Figure 6: Problem compilation example for the barman domain.

corresponding object (*shot01, shot02...*) whenever this action is executed. After that, we also introduce `close` actions to determine that the reparation process has ended for the current action.

Warning action This operator can select any predicate and link it as a new effect of any action. When the process is in the *open* phase, it chooses the $current(a)$ action and fixes it with a new positive effect $f \in F$, adding to the state the corresponding $fix(a, f)$ and $patched(a)$ information. Actions are checked when added to the plan, assuming that they are completely specified or they have been repaired. They have to be repaired in their first use, no action previously added to the plan can be linked to a new effect after that. Then, we define a warning action as $w = \{par(w), pre(w), eff(w)\}$ such that:

$$par_w = \{a - action, f - predicate\}$$
$$pre_w = \{current(a), \neg check(a), open\}$$
$$eff_w = \{fix(a, f), patched(a)\}$$

Repair actions Once the missing effect $f \in F$ has been linked to an action, such predicate has to be added to the state with the proper objects. The number of these objects depends on the arity n of the predicate being added, so a predicate *clean* will be repaired with a single object of type *shot*, whereas other predicates may involve a larger number of objects. Accordingly we need actions with different number of parameters to do so. For each arity n we define a repair action $r(n) = \{par_{r(n)}, pre_{r(n)}, eff_{r(n)}\}$, such that:

$$par_{r(n)} = \{a{-}action, f{-}predicate, \{o_1, \ldots, o_n\}{-}itemdomain\}$$
$$pre_{r(n)} = \{curent(a), fix(f, a), \bigwedge_{i=1}^{n} used_object(o_i)\}$$
$$eff_{r(n)} = \{in_state(n)(f, \{o_1, \ldots, o_n\}), fixed\}$$

Close action To distinguish different phases we introduce close actions, that conclude the reparation stage and change to the next action. They delete the *open* flag and the current action along with all objects used, adding the action as already checked to the current state:

$$par_c = \{a - action\}$$
$$pre_c = \{current(a), open\}$$
$$eff_c = \{check(a), \neg open, \neg current(a), \forall o_i \neg used_object(o_i)\}$$

An example of this operator is shown in Figure 7.

```
(:action close
  :parameters (?a - action)
  :precondition (and (current-action ?a) (open-to))
  :effect (and  (forall (?o - itemdomain)
                    (not (used_object ?o)))
                (check ?a)
                (not (open-to))
                (not (current-action ?a))))
```

Figure 7: Operator to close the reparations process for the current action.

5 Analysis of the compilation

The compiled Π' will be complete and sound if:

Theorem 1 (Completeness)**.** *The extended planning task Π' with respect to Π^- is complete iff there exist reachable actions which involve all the objects present in the goals.*

Proof. There is always the possibility to repair an action by adding goals, but to do so the object present in the goal has to be previously involved in the action to be repaired. If there exist any reachable action which involve such object, it can be currently in use to be managed by the repair operators. □

Theorem 2 (Soundness)**.** *The extended planning task Π' with respect to Π^- returns in the solution a set of tuples $\mathcal{M}_\Pi$ such that for all $\langle f, a_i\rangle \in \mathcal{M}_{\Pi^-}, add(a_i)^- \cup f$ makes solvable Π^-.*

Proof. For any state s_i, if an action a_i cannot be applicable due to $pre(a_i) \not\subset s_i$ and $\nexists a_k \in A$ such that $s_i \cup add(a_k) = pre(a_i)$, this means that $pre(a_i)$ contains some unreachable set of propositions that makes the task unsolvable. Π' has operators to repair actions a_k such that $add(a_k) \cup \{X\} = pre(a_i)$. In the worst case, it will achieve the final state by simply repairing $add(a_k) \cup X = G$. Then, for any plan as a solution of Π', is a plan achieving G and making the task solvable. □

6 Heuristics for Domain Reparation

There are a huge number of ways in which a planning domain can be repaired to make the task solvable, leading the process to fall into undesirable reparations. We have identified several problems related to the planning task reparation:

- **Goals**: the simplest reparation is to add the goal predicates to any of the actions, making the problem solvable by applying just such operator.
- **Domain invariants**: In a complete domain, invariants are implicitly defined in the action effects, i. e, an object can not be in two places at the same time, or a shot is empty or full of beverage, defining the properties of the task. Since we deal with incomplete actions to be repaired, such peculiarity has to be taken into account for the reparation.
- **Repair effects in the same action**: the planner is likely to repair the set of missing effects in the same action, without considering the rest of the planning operators.
- **Unused actions**: closely related to the above, it is possible that the plan repairs several effects in the same action or in an incorrect action that saves it from the application of another action.

In order to avoid such situations we extend the general operators to define different costs that penalize their application. We establish an optimization criteria which minimizes the total cost of the reparations made plus the cost of leaving actions out of the resulting plan. The use of an optimal planner to solve the task guarantees the solution with lower cost. In the remainder of the section we show the different solutions implemented to solve the mentioned issues.

6.1 Goals

To avoid the reparation by simply adding the set of goals, we duplicate every repair action with arity n for the case where the proposition is a goal, but penalizing this reparation with a cost $\mathcal{C}$ for its application, such that:

$$pre_{goal_r(n)} = pre_{r(n)} \cup goal(f, \{o_0, \ldots, o_{n-1}\})$$
$$eff_{goal_r(n)} = eff_{r(n)} \cup \mathcal{C}_{goal_r(n)}$$

Conversely, common repaired actions specify that the predicate involved is not a goal. Considering the barman domain, if the goal were *(contains cocktail01 shot01)*, a plan with lower cost will be to repair the *clean-shot* action and allow the plan to prepare the cocktail, instead of directly adding the goal to any action.

6.2 Domain invariants

Invariants are usually strongly related to the negative effects of the actions. Looking at the action that fills the shot, it involves a *(not (clean ?s))* predicate as effect. An undesirable reparation would be to link the missing effect *clean* in such action, since it contains the same predicate in negative. For this reason we include information about current action effects in the problem of the extended task, as shown in Figure 6. We penalize such option but do not forbid it, since some actions may present this feature, specially in domains where the location of objects is important, being necessary to delete the previous position and update it in the same action. Then, we redefine the warning action as:

$$pre_w = pre_w \cup \neg del_eff(f, a)$$
$$eff_w = eff_w \cup \mathcal{C}_w$$

forcing its application only when the predicate repaired is not part of the delete effects. But to allow this situation, we also create a warning action where the predicate already appears as negative effect:

$$pre_{w_inv} = pre_w \cup del_eff(f, a)$$
$$eff_{w_inv} = eff_w \cup \mathcal{C}_{w_inv}$$

where the cost of applying the latter operator is higher, $\mathcal{C}_{w_inv} > \mathcal{C}_w$.

6.3 Repair effects in the same action

As seen above, we control if the predicate being repaired also appears as negated effect in the action. But in addition, to avoid the planner adding all the missing predicates to the

same action and promote the use of the rest of the actions, the set of warning actions are finally divided to cover the next cases, where the predicate used to repair the action:

- Is not a negated effect and the action does not have more reparations:

  $pre_{w1} = pre_w \cup \{\neg del_eff(f,a), \neg patched\}$
  $eff_{w1} = eff_w \cup \mathcal{C}$

- Is a negated effect and the action does not have more reparations:

  $pre_{w2} = pre_w \cup \{del_eff(f,a), \neg patched\}$
  $eff_{w2} = eff_w \cup \mathcal{C}$

- Is not a negated effect and more predicates are linked to the action:

  $pre_{w3} = pre_w \cup \{\neg del_eff(f,a), patched\}$
  $eff_{w3} = eff_w \cup \mathcal{C}$

- Is a negated effect and the action has more reparations:

  $pre_{w4} = pre_w \cup \{del_eff(f,a), patched\}$
  $eff_{w4} = eff_w \cup \mathcal{C}$

where $\mathcal{C}_{w4} > \mathcal{C}_{w3} > \mathcal{C}_{w2} > \mathcal{C}_{w1}$. This distinction is useful in cases where several action effects are missing from the domain. The planner may tend to fix them all in a single action, creating a kind of "macro-operator". By breaking the operators in a set with different cost we want to prevent such situations and promote the use of all available domain actions.

6.4 Unused actions

All actions have to be checked for incompleteness in Π', specifying such purpose in the set of goals. Actions are checked in the *close* action when they are added to the plan, so we are indirectly forcing the planner to include all the domain actions in the plan. The planner may not return any plan under these conditions, so we transform such constraints to soft goals following the compilation proposed by Keyder and Geffner (2009), including a *forgo* action that can directly achieve goals, but at high cost.

Therefore we aim to minimize the number of reparations made in the domain, while also the penalizing the reparations depending on where they are made. The following plan is part of the result obtained for the extended planning task for the barman domain, where the warning and repair actions are highlighted:

```
...
13:(warning-adding-different clean clean-shot)
14:(repair1 clean clean-shot shot1 o)
15:(close-fixed clean-shot)
16:(fill-shot shot1 ingredient2 left right dispenser2)
17:(close-nofixed fill-shot)
18:(pour-shot-used-shaker shot1 ingredient2 shaker1 left l1 l2)
19:(close-nofixed pour-shot-to-used-shaker)
20:(clean-shot shot1 ingredient2 left right)
21:(repair1 clean clean-shot shot1 o)
22:(close-fixed clean-shot)
...
```

It is important to note that if we perform the compilation on a complete task, the resulting plan will achieve the goals, but without including any repair action.

Domain	$\|A\|$	$\|F\|$
TRANSPORT	3	5
BLOCKSWORLDS	4	5
ROVERS	9	25
BARMAN	12	15

Table 1: Benchmark summary

7 Experiments

To empirically evaluate the feasibility of our approach, we selected the domains TRANSPORT, BLOCKSWORLD, ROVERS and BARMAN from the IPC. They were chosen to verify how the approach works as the number of actions and propositions increases. Table 1 summarises their main characteristics. We generated[1] a set of 10 problems associated to each domain by increasing the number of objects involved, being the first problem the smaller one. Each domain and problem conform a complete planning task Π_i, being $i = \{1, \ldots, 10\}$ the number of the problem to which is associated. In order to test our approach, we created a set of add *add-incomplete* tasks with respect to each Π_i by randomly deleting positive effects of the domain actions, provided that such changes made the task unsolvable. For instance, from the blocksworld domain we deleted the *holding* effect when it picks up a block or we removed a shot as *cleaned* from the *clean-shot* action for the barman domain.

We follow an iterative process in which we generate a set of tasks $\Pi^-_{i\mathcal{D}}$ where $\mathcal{D} = \{1, \ldots, 5\}$ is the number of removed positive effects with respect to Π_i. Therefore we have a total of 50 *add-incomplete* planning tasks for each domain. They are compiled following the explained approach and solved to obtain the set of new effects $\mathcal{M}_{\Pi^-}$ used in the reparation. We have tested the aforementioned planning tasks using the `seq-opt-lmcut` and `lama` configurations of Fast-Downward (Helmert 2006), using the LMCUT admissible heuristic (Helmert and Domshlak 2009) and an anytime planner that reports the best plan found in a given time window LAMA (Richter and Westphal 2010), respectively. The planning times for both configurations were set to 900 seconds.

The results are shown in tables Figure 2 and 3. We compared the set $\mathcal{M}_{\Pi^-}$ obtained as solution with the actual removed $\mathcal{M}^*_{\Pi^-}$, representing in $\%$ the percentage of the domain repaired and in t the planning time in seconds. Empty fields mean that no plan has been found in the given time window. In general, the proposed approach performs well for smaller instances of domains and problems, although the scalability decreases in larger tasks. However, LAMA seems to be a better option since its use increases significantly the number of solved tasks with similar or shorter planning times. In most of the cases, we do not need to wait until the optimal solution of the planner to achieve the desired repair suggestions, intermediate plans already contains them. Rapid responses are also critical in the context of user sup-

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

		$\Pi_{1\mathcal{D}}$		$\Pi_{2\mathcal{D}}$		$\Pi_{3\mathcal{D}}$		$\Pi_{4\mathcal{D}}$		$\Pi_{5\mathcal{D}}$		$\Pi_{6\mathcal{D}}$		$\Pi_{7\mathcal{D}}$		$\Pi_{8\mathcal{D}}$		$\Pi_{9\mathcal{D}}$		$\Pi_{10\mathcal{D}}$	
Domain	$\mathcal{D}$	%	t	%	t	%	t	%	t	%	t	%	t	%	t	%	t	p9	t	p10	t
TRANSPORT	1	100	0,00	100	0,01	100	0,06	100	0,05	100	1,27	100	18,15	100	68,59	-	-	-	-	-	-
	2	100	0,05	100	0,24	100	0,74	100	0,07	100	5,80	100	12,91	-	-	-	-	-	-	-	-
	3	100	0,01	100	16,18	100	1,45	70	0,31	-	-	-	-	-	-	-	-	-	-	-	-
	4	100	0,20	100	21,25	100	2,55	100	96,01	-	-	-	-	-	-	-	-	-	-	-	-
	5	80	0,20	80	0,59	-	-	80	92,99	-	-	-	-	-	-	-	-	-	-	-	-
BLOCKS WORLD	1	100	0,01	100	0,01	100	0,01	100	0,03	100	0,01	100	0,11	100	0,06	100	0,03	100	0,32	100	3,07
	2	50	0,01	100	0,01	100	0,01	50	0,01	100	0,01	100	0,18	50	0,39	50	0,01	100	0,14	50	2,94
	3	70	0,00	70	0,05	70	0,15	100	0,02	35	0,00	70	1,01	70	1,69	35	3,71	70	114,52	100	32,30
	4	50	0,01	100	0,12	50	0,11	100	0,20	75	0,01	100	5,48	75	2,36	25	0,01	75	91,80	-	-
	5	80	0,01	100	0,03	80	0,23	60	0,66	80	0,03	100	16,80	60	385,66	60	0,13	100	34,76	40	798,06
ROVERS	1	100	48,00	-	-	-	-	100	418,63	100	697,40	-	-	-	-	-	-	-	-	-	-
	2	50	67,77	-	-	-	-	-	-	100	133,26	-	-	-	-	-	-	-	-	-	-
	3	35	808,46	-	-	-	-	65	560,42	65	275,03	-	-	-	-	-	-	-	-	-	-
	4	25	205,27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	5	40	709,13	-	-	-	-	40	348,30	-	-	-	-	-	-	-	-	-	-	-	-
BARMAN	1	100	1,75	0	11,99	100	3,97	100	725,80	0	807,99	-	-	-	-	-	-	-	-	-	-
	2	50	16,87	50	11,71	100	27,60	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	3	-	-	-	-	65	589,28	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	5	-	-	40	550,38	40	387,40	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 2: Results of the extended planning tasks using the LMCUT configuration of Fast-Downward. Parameters shown the initial number of removed effects ($\mathcal{D}$), the percentage ($\%$) of reparation achieved compared to the initial number of removed effects and the planning time (t) spent to find the solution.

		$\Pi_{1\mathcal{D}}$		$\Pi_{2\mathcal{D}}$		$\Pi_{3\mathcal{D}}$		$\Pi_{4\mathcal{D}}$		$\Pi_{5\mathcal{D}}$		$\Pi_{6\mathcal{D}}$		$\Pi_{7\mathcal{D}}$		$\Pi_{8\mathcal{D}}$		$\Pi_{9\mathcal{D}}$		$\Pi_{10\mathcal{D}}$	
Domain	$\mathcal{D}$	%	t	%	t	%	t	%	t	%	t	%	t	%	t	%	t	%	t	%	t
TRANSPORT	1	100	0,01	100	0,02	100	0,03	100	0,03	100	1,04	100	0,08	-	-	100	197,55	-	-	-	-
	2	100	0,01	100	0,01	150	0,08	100	17,72	100	5,17	100	47,40	150	63,53	-	-	-	-	-	-
	3	100	0,10	100	0,02	100	178,80	70	51,35	100	1,93	100	1,51	-	-	-	-	-	-	-	-
	4	100	0,18	100	0,24	100	0,21	100	210,14	100	6,69	125	80,85	-	-	-	-	-	-	-	-
	5	80	0,19	80	0,23	100	0,19	120	37,47	100	6,64	120	77,09	-	-	-	-	-	-	-	-
BLOCKS WORLD	1	100	0,00	100	0,01	100	0,01	100	0,01	100	0,01	100	0,03	100	0,01	100	0,01	100	0,04	100	0,03
	2	100	0,00	100	0,17	100	0,01	100	0,01	100	0,01	100	0,22	100	0,01	50	0,01	100	0,04	50	0,25
	3	70	0,01	70	0,01	70	0,02	100	0,01	35	0,01	70	0,21	70	0,36	70	0,13	70	0,04	100	0,27
	4	75	0,02	100	0,18	75	0,04	-	-	75	0,51	100	16,89	75	21,95	75	0,02	75	0,04	75	0,38
	5	100	0,01	100	0,22	100	0,22	60	0,02	100	0,05	100	2,74	80	0,58	80	0,10	100	6,86	80	16,96
ROVERS	1	0	0,03	200	0,03	200	0,04	-	-	200	0,05	-	-	0	1,28	-	-	200	0,28	300	31,36
	2	50	0,03	50	0,08	150	59,28	-	-	150	0,38	-	-	0	0,21	250	2,83	50	0,27	-	-
	3	0	50,98	70	0,03	70	0,75	70	1,33	35	1,08	-	-	-	-	35	0,30	0	1,47	35	0,73
	4	-	-	50	0,09	25	0,16	50	0,49	50	16,82	-	-	-	-	-	-	-	-	-	-
	5	80	1,03	80	0,06	-	-	20	0,45	40	12,85	-	-	-	-	40	0,19	-	-	-	-
BARMAN	1	0	0,10	200	0,06	100	0,10	-	-	100	264,45	100	447,26	-	-	-	-	-	-	200	1,30
	2	50	0,24	50	3,27	50	0,13	-	-	-	-	250	36,40	-	-	-	-	-	-	-	-
	3	-	-	35	0,08	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	4	-	-	50	3,50	50	4,22	-	-	-	-	-	-	25	10,87	-	-	-	-	-	-
	5	-	-	40	3,59	60	0,93	-	-	-	-	40	4,76	-	-	-	-	80	27,79	-	-

Table 3: Results of the extended planning tasks using the LAMA configuration of Fast-Downward. Parameters shown the initial number of removed effects ($\mathcal{D}$), the percentage ($\%$) of reparation achieved compared to the initial number of removed effects and the planning time (t) spent to find the solution.

port, and although some reparations exceed the repair rate (they include extra reparations besides the desired ones), domain designers can supervised the solutions suggested and discard unwanted reparations, even refining the search to obtains new plans which may show new suggestions.

8 Discussion

In this section we argue the proposed method along with the solutions provided. The main decision of this work relies on the applied method, which may have any other technique based on cost optimization as alternative. However, we implemented an AP compilation of the unsolvable task to keep the goal oriented approach, ensuring that the given solution also achieve the original goals of the problem. It provides a guideline for the reparations, showing those ones which help to achieve the goal.

Reparations are dependent on the problem configuration and the established metric provides an heuristic, presenting several ways to achieve the goals. To exemplified this, let us consider again the clean shot example. To achieve a clean glass it has to repair the clean operator, but the domain contains a very similar operator: empty-shot. This situation can return a solution in which the operator fixed is the latter. The effect is the same and the problem turns solvable anyway, the only difference is semantic. But semantic issues are out of the scope of this work, delegating them to the users' side. By embedding the proposed system in any PDDL editor, users have the opportunity to decide if the proposed solution fits in its domain model or a better option it is to apply the reparation made to another operator. For larger domains and problems with multiple missing effects, were the performance of the method decreases, it is also possible to follow an iterative process. Instead of repair all the effects at once, the user can select some of the recommendations and run the

system again, obtaining more precise suggestions since the model is now more accurate.

This approach also has the advantage of being parameterizable. For further developments, we can offer to users the possibility of choosing the penalizations applied to each operator, in addition to extend the metric with extra parameters as the plan length, for which it would be sufficient to also penalize the close operators.

9 Related Work

Several works addressed the previous problem in a effort to help users in the development of AP tasks. In *Mixed-initiative planning* (Burstein and McDermott 1996), planning is seen as a collaborative activity in which given a domain, a problem and a solution, automated and humans planners need to interact to jointly build a plan to accomplish a certain objective. Such solution can be generated from similar stored past plans (Veloso, Mulvehill, and Cox 1997) or generated by hand by the user (Howey and Long 2003; Howey, Long, and Fox 2004, 2005; Fox, Howey, and Long 2005), where if the plan is flawed it gives advice on how the plan should be fixed. Similarly, *Model reconciliation* (Chakraborti et al. 2017) presents way of bringing the human model closer to the agent's model by explaining the plan when it differs from the expected one.

These works rely on a properly specified domain and problem and also assume an initial plan which is changed or improved in collaboration with humans. However, there may be some cases where there is no suggested plan and the AP task proves to be unsolvable anyway. Techniques involved in such problems try to explain *why* it fails and *how* it could be solved. (Sreedharan et al. 2019) propose to assist the user by identifying unreachable subgoals of the problem. Since is challenging to extract subgoals from a unsolvable problem, they derive them from abstract and solvable models by using planning landmarks (Hoffmann, Porteous, and Sebastia 2004). Going a step further, given an unsolvable task it is also possible to make it solvable (Göbelbecker et al. 2010). Based on counterfactuals theory (Ginsberg 1985), it is able to explain why a plan fails and what should be done to prevent it, creating *excuse states* from which the given task is solvable, although they only consider changes in the initial state.

Planning with incomplete domains or approximate domain models (McCluskey, Richardson, and Simpson 2002; Simpson, Kitchin, and McCluskey 2007; Nguyen, Sreedharan, and Kambhampati 2017; Garland and Lesh 2002) considers not properly specified domains, but all those works assume that the incompleteness in the model is filled with annotations or statements about where the model has been incompletely specified, and providing guidelines to supply the lack of knowledge.

Therefore, all of the works seen so far make crucial assumptions about the domain engineer understanding of the problem or do not deal with partially specified domains without further help of the user. In this paper we work on fixing domains that have some missing actions' effects which make the problem unsolvable, and we have no *a priory* clues about where the error resides.

10 Conclusions

We have presented a novel approach for the reparation of unsolvable planning tasks due to a missing positive effects in its domain, which is based on classical planning techniques. Since the given task is incomplete, we have proposed its compilation to a new extended planning task which introduces general operators to repair any of the actions of the domain by linking them to new positive effects. To test our method, we selected some domains from the IPC to generate complete planning tasks and altering its domain by randomly deleting positive effects, provided that such changes turn the task unsolvable. The resulting incomplete task is compiled to the new extended task and solved by an optimal planner, including in the plan the modification on the actions made while also achieving the original goals. Results show that the approach performs well for a wide range of planning tasks, especially with an anytime planner configuration. We believe that this work has enough potential to be considered as a user support system to develop planning tasks.

References

Burstein, M. H.; and McDermott, D. V. 1996. Issues in the development of human-computer mixed-initiative planning. In *Advances in Psychology*, volume 113, 285–303. Elsevier.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 156–163. ijcai.org.

Fox, M.; Howey, R.; and Long, D. 2005. Validating Plans in the Context of Processes and Exogenous Events. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, 1151–1156. AAAI Press / The MIT Press.

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

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. Elsevier.

Ginsberg, M. L. 1985. Counterfactuals. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence. Los Angeles, CA, USA, August 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 the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, 81–88. AAAI.

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

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*. AAAI.

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

Howey, R.; and Long, D. 2003. VAL's progress: The automatic validation tool for PDDL2. 1 used in the international planning competition. In *Proc. of ICAPS Workshop on the IPC*, 28–37.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004), 15-17 November 2004, Boca Raton, FL, USA*, 294–301. IEEE Computer Society.

Howey, R.; Long, D.; and Fox, M. 2005. Plan validation and mixed-initiative planning in space operations.

Kambhampati, S. 2007. Model-lite Planning for the Web Age Masses: The Challenges of Planning with Incomplete and Evolving Domain Models. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, 1601–1605. AAAI Press.

Keyder, E.; and Geffner, H. 2009. Soft Goals Can Be Compiled Away. *J. Artif. Intell. Res.*, 36: 547–556.

Linares López, C.; Jiménez Celorrio, S.; and García-Olaya, A. 2015. The deterministic part of the seventh International Planning Competition. *Artificial Intelligence*, 223: 82–119.

McCluskey, T. L.; Richardson, N. E.; and Simpson, R. M. 2002. An Interactive Method for Inducing Operator Descriptions. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*, 121–130. AAAI.

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.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007. Planning domain definition using GIPO. *Knowl. Eng. Rev.*, 22(2): 117–134.

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 the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 1422–1430. ijcai.org.

Veloso, M. M.; Mulvehill, A. M.; and Cox, M. T. 1997. Rationale-Supported Mixed-Initiative Case-Based Planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island, USA*, 1072–1077. AAAI Press / The MIT Press.