A Survey on Model Repair in AI Planning

Pascal Bercher[1] , **Sarath Sreetharan**[2] , **Mauro Vallati**[3]
[1]School of Computing, Australian National University
[2]Department of Computer Science, Colorado State University
[3]School of Computing and Engineering, University of Huddersfield
pascal.bercher@anu.edu.au, sarath.sreedharan@colostate.edu, m.vallati@hud.ac.uk

Abstract

Accurate planning models are a prerequisite for the appropriate functioning of AI planning applications. Creating these models is, however, a tedious and error-prone task – even for planning experts. This makes the provision of automated modeling support essential. In this work, we differentiate between approaches that learn models from scratch (called domain model acquisition) and those that repair flawed or incomplete ones. We survey approaches for the latter, including those that can be used for domain repair but have been developed for other applications, discuss possible optimization metrics (i.e., which repaired model to aim at), and conclude with lines of research we believe deserve more attention.

1 Introduction

AI planning (or: automated planning) allows the autonomous generation of courses of actions – action plans that can serve as basis for autonomous behavior. Planning systems rely on *planning models*, a formal representation of the environment (most notably the available actions). For any application, it is of crucial importance that these models are correct. Incorrect models could lead to suboptimal system behavior (e.g., not even allow to generate certain plans according to the model that would be possible in the real world) or produce incorrect plans (i.e., those that do not work in the real world). In the extreme case, incorrect models might lead to safety issues for the system and the deployment application.

Work on Knowledge Engineering for Planning and Scheduling (KEPS) has thus been a long-standing research area, with various approaches and tools supporting in the domain engineering process (see a recent overview by Vallati and Kitchin [2020]). Most tools directly support writing PDDL [Haslum *et al.*, 2019] (the de-facto standard description language for planning problems), such as the well known online PDDL editor planning.domains [Muise and Lipovetzky, 2020], which offers standard programming capabilities like syntax highlighting, auto-completion, and a few plug-ins for more advanced capabilities by third parties. Similar ideas were implemented in WebPlanner [Magnaguagno *et al.*, 2020], another online PDDL modeling tool that is unfortunately not available anymore. Offline tools also exist, such as MyPDDL [Strobel and Kirsch, 2020], itSIMPLE [Vaquero *et al.*, 2007], or the PDDL plugin for Visual Studio Code. Whereas these tools are extremely useful (and hence extensively used), their *intelligent support* for developing or repairing domains autonomously is limited, as they focus on standard programming environment capabilities and visualizations, and do not explicitly support refinement and maintenance of existing models. Recent work by Grundke *et al.* [2024] is also worth mentioning, although it is purely theoretical. They propose a framework based on PDDL axioms that allows to characterize the set of legal planning problems, which they envision can lead to a domain-independent problem creator with valid problem instances.

Approaches for encoding planning knowledge models can – *as we propose in this survey* – roughly be categorized along two dimensions: *Domain acquisition/learning* and *Domain repair*. The first can be considered as starting from scratch, i.e., with an initially empty/non-existent model plus additional data from which the model can be learned and formulated. This research area is quite prominent — so much so that even a few surveys exist already [Jiménez *et al.*, 2012; Arora *et al.*, 2018; Jilani, 2020]. As a first contribution, here we provide a definition of domain model repair to be, informally, the setting where we are *also* given an initial model to repair – this setting hence generalizes the latter as it is also able to take an initially empty given model into account that is supposed to be repaired.

To the best of our knowledge, no survey on model repair exists as of yet – which is what we provide in this work. Most notably, we do not just survey works that have been specifically developed for intelligent modeling support, but also those that could be used for that purpose (as they internally use model repair techniques), but were developed for different application scenarios, thus making their relevance less apparent within this important research landscape. For the surveyed work, we aimed for completeness within the field, while focusing on publications from top-tier venues (most work is from AAAI, IJCAI, and ICAPS, but we considered all top-tier venues in our area). We specifically excluded workshop publications or other venues that are not top-tier.

We also provide a short discussion on comparing planning models, which is essential in our context as it is a non-trivial

question which repaired model – among all those satisfying the given constraints – should be chosen.

In the end, we summarize the existing works and conclude by stating which research questions in the field of model repair are yet underrepresented and deserve further attention.

2 Planning Formalism

Although model repair is possible for any devisable planning framework (i.e., no matter how expressive it is, e.g., whether it supports time, uncertainty, partial observability, disjunctive preconditions, conditional effects, object creation, hierarchies, or any other extension), most approaches at the time have been developed for the lowest level of expressivity, where these extensions are not present. We hence restrict ourselves to formally explaining the most simplistic version, classical planning, and then extend it to hierarchical planning, as all other extensions can be combined with both of them. For the sake of simplicity, we focus on providing a propositional framework, and later briefly explain how models are specified in practice using standard description languages such as PDDL [Haslum *et al.*, 2019] (for classical/non-hierarchical problems) or HDDL [Höller *et al.*, 2020] (for hierarchical models).

A classical *planning domain model* $D = (F, A)$ consists of the finite set of existing *facts* that encode all possible world properties. The set of collections of facts, 2^F (the power set of facts), is called the state space, and every member $s \in 2^F$ is called a state. We assume the closed world assumption, meaning that all $f \in s$ hold in that state, but all $f' \notin s$ do *not* hold in that state. $A \subseteq 2^F \times 2^F \times 2^F \times 2^F$ is a finite set of *actions*. Actions describe state transitions, specifying the conditions in which states they are applicable and describing how said states change. More specifically, each action is a 4-tuple $a = (pre^+, pre^-, \mathit{eff}^+, \mathit{eff}^-)$, with positive and negative preconditions and effects, respectively. An action a is said to be applicable in state $s \in 2^F$ if and only if $pre^+ \subseteq s$ and $pre^- \cap s = \emptyset$, thus specifying which state features must be true and which are not allowed to be true. If a is applicable in state s, it generates the successor state $s' = (s \setminus \mathit{eff}^-) \cup \mathit{eff}^+$.

Having defined the domain model, we can now define the *planning problem*, formally given as a 3-tuple (D, s_I, g), consisting of its planning domain model, an *initial state* $s_I \in 2^F$, and a *goal description* $g \subseteq F$, which describes all facts that we'd like to have achieved. We call a state goal state if it makes all goals true. Thus, g implicitly defines a set of goal states $\{s \mid s \supseteq g\}$. One could also allow the exclusion of facts (rather than just the inclusion), but that is usually not done in the formalism for the sake of simplicity.

Solutions are action sequences that turn the initial state into a goal state. More formally, an action sequence $a_1, \dots, a_n$ is called a *solution* (or *plan*) if and only if there is a sequence of states $s_0, \dots, s_n$, such that $s_0 = s_i$ and for each $1 \leq i \leq n$, a_i is applicable in state s_{i-1} and generates state s_i, and $s_n \supseteq g$, i.e., the sequence generates a goal state.

We now proceed with *totally ordered Hierarchical Task Network* planning (which we just refer to as HTN planning, for short) [Bercher *et al.*, 2019]. Here, actions are organized in a hierarchical manner, for which we add so-called *compound tasks*, that can be refined into sequences of primitive and compound tasks (where primitive tasks are a new name for actions in the context of HTN planning) using refinement rules (similar to formal languages), called (decomposition) methods. More formally, an *HTN domain model* is a tuple $D = (F, C, A, M)$, where F and A are finite sets of facts and actions as before, C is a finite set of strings, so-called compound tasks (or compound task names), and M is a set of methods. Each method formally is a 2-tuple $(c, \bar{t})$, where $c \in C$ is a compound task and $\bar{t} \in (C \cup A)^*$ is a (possibly empty) task sequence. Now, a task sequence $\bar{t}_1 c \bar{t}_2$ can be refined by a method $(c, \bar{t}_0) \in M$ into the sequence $\bar{t}_1 \bar{t}_0 \bar{t}_2$. An HTN planning problem is given by the tuple $(D, s_I, \bar{t}, g)$, consisting of an HTN domain model D, an initial state s_I, an initial task sequence $\bar{t}$, and a goal description g.

Finally, an HTN solution is simply a classical solution (i.e., action sequence) $\bar{a}$ to the induced classical planning problem that can be obtained from $\bar{t}$ using the decomposition methods. In other words, HTN planning acts as a "filter" on classical planning problems, where some plans that would usually be considered solutions are now ruled out, as only those (applicable, goal-generating) action sequences are considered solutions that can be obtained by adhering to the task hierarchy.

Finally, note that the formalisms presented above were fully propositional, which is not how problems are modeled in practice. Consider a simple Logistics domain, where we have a set of locations (say L_1 to L_n), a number of packages (say P_1 to P_m), and trucks that can drive between locations to deliver packages. To model this, one needs to express at which locations packages may be, giving rise to a predicate $at(?p, ?l)$ encoding that package $?p$ is at location $?l$. Using *facts*, we would require one fact for each instantiation, e.g., facts *at-*P_1*-*L_1 to *at-*P_m*-*L_n, leading to $m \cdot n$ facts instead of just a single predicate and a set of n and m objects. This explosion scales exponentially in the arity of predicates. Likewise, actions are in practice not using facts either, but similarly a representation using variables, such as *drive(*$?t, ?l_1, ?l_2$*)*, which allows a truck ($?t$) to drive from one location ($?l_1$) to another ($?l_2$), using only variables in the problem specification. Such actions are called *action schema*, yet can be turned into the presentation from above by simple "grounding", i.e., instantiating all variables with all possible constants. Such schematic planning languages are called *lifted*, and usually expressed in the modeling languages PDDL [Haslum *et al.*, 2019] (for classical planning and many of its extensions) and HDDL [Höller *et al.*, 2020] (for HTN planning).

3 On Model Repair vs. Model Acquisition

In this survey, we focus on approaches for performing model repair. Informally, model repair can be defined as the process of refining a given model according to a set of constraints that should be satisfied, by making changes to the model (e.g., add new actions, change actions' parameters, or their preconditions and effects). While this notion gives a good intuition of the model repair task, it might not be sufficient to crisply distinguish between model repair and model acquisition for the purposes of this survey, i.e., to give a concise inclusion and

exclusion criterion of the work surveyed.

A large number of automated approaches to model acquisition starts from an initial (partial) model, which is then refined according to some domain knowledge provided under the form of either plan traces or given transition systems. An early example is the work(s) by McCluskey *et al.* [2002], where partial domain information in terms of predicates, invariants, etc., is given (though in a description language different from standard PDDL), and a mixed-initiative process builds the action model based on user-provided action traces, which are action names with their instantiated arguments[1]. Their system also allows the creation of hierarchical models. We refer to existing surveys on domain model learning for further information [Jiménez *et al.*, 2012; Arora *et al.*, 2018; Jilani, 2020].

Usually, initial models are just minimalistic ones that allow the learning to start in the first place (e.g., by providing action names), whereas we regard model repair as having a "more complete model" (which could be created by model acquisition in the first place, or otherwise also modeled by hand) that is potentially either not entirely complete, not correct, or not optimal, and so it is repaired to ensure that it satisfies all those properties that we would like to have satisfied.

To solve the classification conundrum, here we rely on the model development taxonomy introduced by McCluskey and Porteous [1997]. The taxonomy considers four levels of models: *initial*, *compilable*, *runnable*, and *proven*.

- Initial models are not syntactically correct and encode only some of the aspects of the domain at hand.
- Compilable models are syntactically correct, according to the target language, and can be parsed by planning engines or syntax validators, but they do not encode the full dynamics of the domain.
- Runnable models are syntactically correct and can be used to solve at least a subset of problems from the target domain, when an appropriate planning engine is used.
- Proven models satisfy the requirements of the target application and can be used in production. They embody the properties of accuracy, consistency, completeness, adequacy, and operationality [McCluskey *et al.*, 2017].

On the basis of this taxonomy, we can define model acquisition approaches as those that extend or refine input models that are either initial, or (more common) compilable. Please note that we consider an empty model as initial. This means that they start in the early stages of model creation. In contrast, we do not have such a requirement for model repair, meaning that they also could also start in later stages, i.e., to refine runnable and proven models. In the following we provide additional details on the two classes and the proposed rough categorization.

[1] At first glance this might appear as in instance of a repair problem, since partial models are already available to build on. However, given that preconditions and effects are not specified yet, these models can not be "runnable" yet. Further, the model given can be regarded similar to additional information in learning approaches, such as transition systems. Also note that their technology seems to easily work as well with existing preconditions and effects given, especially since the process is not fully automated. So, one could also regard it suitable for repair problems.

3.1 Model Acquisition

Model acquisition, also referred to as model formulation, articulation, or model learning in the literature [McCluskey *et al.*, 2017; Vallati and McCluskey, 2021], is the process of encoding domain knowledge into a model that can be provided as input to a planning engine to generate solutions. Please note that, as presented in Section 2, the domain model is the set of predicates/facts and actions in case of non-hierarchical planning, and also compound tasks and decomposition methods in case of hierarchical planning.

Let us now define this process more formally, according to the considered model development taxonomy.

Definition 1. *The **model acquisition process** is the process where an initial or compilable model is refined according to available domain knowledge.*

Please note that the definition does not fully characterize the level of the resulting refined model – since this depends on the quality and characteristics of the provided domain knowledge. There are no restrictions on the way in which the domain knowledge can be provided.

There are a number of surveys already focusing on the model acquisition process, hence this is not the target of this work. The interested reader is referred to overviews of the field [Jiménez *et al.*, 2012; Arora *et al.*, 2018; Jilani, 2020]. The recent works by Aineto *et al.* [2018] and Balyo *et al.* [2024], although not surveys, have comprehensive related work sections that give excellent overviews of the field.

3.2 Model Repair

The process of model repair can encompass different traditional knowledge engineering processes, such as maintenance and evolution [McCluskey *et al.*, 2017; Vallati and McCluskey, 2021], according to the underlying cause of the repair and the extent of the modifications.

Definition 2. *The **model repair process** is the process where a given model is refined, according to a set of provided constraints into a model that satisfies these constraints.*

Note that also in this definition, we do not specify the resulting level of the refined model. Whereas in practice we assume that often one starts with a model that is already runnable or proven and hence has to also result into a similar or higher level, we also regard it conceivable to get supported in early development stages, where further manual modeling might be required to finish the model.

Further, note that in the definition on model acquisition the input is *domain knowledge*, whereas here the input are *constraints*. In principle, they can be the same, though we used different terminologies since domain model acquisition is usually done in earlier stages, where also more fundamental domain knowledge has to be provided, such as entire state transition systems or large amounts of plan traces that would allow to derive a model from it. In contrast, we expect model repair to be deployed in later development stages,

usually even for runnable or proven models. Hence, the domain knowledge provided here might be less comprehensive and only point to properties that we would like to be satisfied, rather than providing large amounts of data that allow the model to be derived in the first place.

One can also observe that according to our classification, it follows that *domain model repair* can be regarded a generalization of model learning, as it can take an initial (more complete) model into account, whereas typical domain model learning approaches start much earlier and do not aim at making changes, but provide the required information (e.g., preconditions and effects) in the first place.

Also, we are deliberately abstract in what these "constraints" could be, as their definition is limited only by creativity and the specific requirements of the given application domain. In fact, one of the purposes of this survey – on top of providing this classification and pointing to recent developments in model repair – is to make aware of the huge amount of model repair techniques that are conceivable, as there is almost no limit on which properties could be posed as constraints. We are going to see several of these constraints that have been used in literature for model repair, but provide further examples here, although not yet implemented by any work we are aware of. Constraints that could be provided might encompass: *Plan traces:* One or more action sequences could be provided, and required to be solutions in the respective planning problems. Likewise, such plan traces could be demanded to *not* be solutions (e.g., if observed to not work in the environment). *Properties of state transition systems:* One could demand that all executable action sequences, or just all solutions (or optimal ones, etc.) show specific properties. A well-known one is mutex relations. So, we could demand that certain facts never occur at the same time in any produced state. For example, demanding that in a logistics domain that deals with the transportation of packages, no truck can ever be at two locations at the same time, i.e., no $at(?p, ?l_1)$ and $at(?p, ?l_2)$ can be true in the same state for any two different $?l_1$ and $?l_2$. Mutex relations can be *inferred* already [Helmert, 2006; Alcázar and Torralba, 2015; Fišer and Komenda, 2018] from a model or problem specification, but as of yet, to the best of our knowledge, no approach allows to *establish* them by a repair.

The above are just some examples to show that any property a planning domain model could possess could also be posed as constraint, which a respective model repair system then had to *achieve* by changing the model. An important question is which properties would help a domain modeler to ensure that the encoded model does possess the required properties – where many could be envisioned. Also, the examples given above could easily be generalized: finitely many given plan traces could be generalized to infinitely many by using a regular expression over action names or grammar-like structures like in hierarchical planning. Even more general properties are conceivable, like "there is at least one solution" or solutions of certain length or cost. Likewise, mutex relations are just a special case of more general properties as can be expressed by Linear Temporal Logic (LTL) [Pnueli, 1977; Lin and Bercher, 2022]. Linear Temporal Logic (LTL) is a formalism to express temporal properties over sequences of states, such as "no package is ever in two places at once" or "every package is eventually delivered". It is worth noting that there are approaches that enforce that LTL (or related) constraints are respected by all solutions by applying only actions in line with the constraints [Bacchus and Kabanza, 1998; Chrpa *et al.*, 2020], which is, however, different from making sure that the actions by themselves (i.e., by changing their preconditions/effects) make these LTL formulae true, which would be the idea behind using them for model repair.

Finally, note that there are usually *many* (repaired) planning models that satisfy the given constraints. The question which of those is to be preferred is hence an important one and a research question on its own (yet still underrepresented). We show existing work and discuss it in Section 5.

4 Model Repair Approaches

It is worth noting that while some works explicitly mention the application domain *modeling support*, not all of them do. In the following, we review the work that explicitly performs a *model repair process* as per our definition – even if they do not envision the application of modeling support. We organize this section by what constraints are posed onto the model/problem to satisfy.

4.1 Constraint: Make Problem(s) Solvable

One of the most prominent constraints posed onto planning models is to demand that a given (set of) problem(s) is solvable. I.e., in the works reviewed below, the input is always a planning problem that does not admit a solution (or if they do, then no repair will be required).

The first and oldest work we review is also the least related. The reason is that this survey focuses on *model repair*, which implies that an incomplete or flawed domain model is the cause of the issue, thus requiring changes. The first work that fixes unsolvable (classical) planning problems, by Göbelbecker *et al.* [2010], does however make problems solvable by making changes to the initial state, not by changing action definitions. Although this work does thus not quite fit our inclusion criterion, we list it here due to its importance and potential for incorporation into approaches that also allow changing action definitions. Their approach finds solutions, i.e., changes to the initial states that make the respective problem solvable, by compiling this "repair problem" into another classical planning problem. They define different kinds of "repairs", called excuses in their work (a repair is an "excuse" that justifies the unsolvability), and provide complexity results for these problems.

A mostly theoretical work, though with potential practical application, shows that the task of making an unsolvable problem solvable – by changing the initial state, adding new actions, or changing the goal description – can be modeled using a variant of Propositional Dynamic Logic [Herzig *et al.*, 2024]. We are not aware of any implementation of their framework, though also argue that only *adding* actions rather than revising the existing ones is more related to model learning than to model repair.

A practical approach that has been widely used in the context of "explaining unsolvability" (or plan invalidity) is state

abstraction [Sreedharan *et al.*, 2021b]. For example, works have looked at using abstractions as a means for identifying counterfactual models where the plans of certain desired properties may be valid [Krarup *et al.*, 2024], or they can be used to identify the simplest abstraction of the problem that is still unsolvable or the desired plans are still invalid. In the former case, the counterfactual model allows the users to see what properties should be relaxed to get desired behaviors. As for the latter case, modeling support tools like D3WA+ [Sreedharan *et al.*, 2020] have shown how focusing on these simpler abstractions allows one to more easily isolate errors in the model. While these approaches do not make the respective unsolvable problem *directly* solvable, the explanations provided can point towards problematic parts.

The work by Gragera *et al.* [2023b], in contrast, does fully fit our definition of model repair, as they attribute unsolvability to flawed action models and provide repairs that fix the issue. That is, their repairs change preconditions and effects to make the respective problem solvable. Their repairs are however not "complete" in the sense that they cannot change action preconditions and cannot delete effects, they only support adding missed effects (positive and negative), but cannot change any. Their approach can further still be extended by allowing to specify multiple unsolvable (or solvable) planning problems at the same time, all for the same underlying model, to find one set of repairs for this single model, turning all problems solvable. Similar to the work by Göbelbecker *et al.* [2010], they also solve the problem by an encoding into a planning problem. They put particular effort into designing a useful metric that specifies which repairs are preferred. They also made their technology available in a user-friendly tool [Gragera *et al.*, 2023a].

Closely related is the earlier work by Aineto *et al.* [2019], which supports the same setting as a special case. Their general setting is domain learning, where they allow start learning from scratch, i.e., they do not require an initial partial model where actions do already show preconditions and effects. However, they also do support such a partial model and allow extending it. Furthermore, their input also allows for (partially) observed state and action traces – but they *also* support the special case where only an initial and final state is being observed, where the action sequence to explain this still needs to be found. This coincides with finding repairs for making the problem solvable. Their work also solves the problem via a compilation to planning problems.

For HTN planning, we are only aware of one approach for repairing unsolvable problems [Xiao *et al.*, 2020]. In their problem definition, the authors neither aim for changing the initial state nor for changing action definitions, but follow an HTN-specific route. As explained in Section 2, only those action sequences can be considered solutions that can be generated from the initial task network by adhering the decomposition methods. Their approach to repair thus aims at making the problem solvable by completing the task hierarchy, i.e., by adding missing actions (from the existing action portfolio) into the existing methods. They do so by interpreting the unsolvable (lifted) HTN problem as a TIHTN problem [Geier and Bercher, 2011; Alford *et al.*, 2015], HTN planning with task insertion, where one task network can not just be turned into another by decomposition as explained in Section 2, but also by inserting actions, thus leading to a larger set of possible solutions. Once they found a solution to the TIHTN problem (that now contains additional actions, those that made the problem solvable), they decide into which decomposition methods to insert them, thus turning the HTN problem solvable. The decision of which method to extend is guided by a user-specified preference model that tells which method is more likely to have missing actions.

4.2 Constraint: Include/Exclude Given Plans

In this section, we review papers where the constraints demand that a given set of plans is turned into a solution, or, likewise, where such a plan is *excluded* from the solution set – by appropriate changes to the domain model.

It is noteworthy that all those approaches that turn input plans into solutions can thus also be used to make unsolvable problems solvable. In contrast to the approaches above, they do however require the additional input of the respective plan(s). Yet, this also gives more explicit control on how the repairs look like. It furthermore makes the problem easier as fixing unsolvable planning problems without input plan is PSPACE-complete [Göbelbecker *et al.*, 2010], but much easier if one is provided (as outlined next).

Based on an incomplete (lifted) STRIPS problem, and a set of plan traces, Zhuo *et al.* [2013] propose a technique exploiting MAX-SAT solving to turn all provided plans into solutions of the repaired model. In their work, incomplete means that preconditions or effects might be missing, so their work can only add those, but not remove any. They also generate additional macro actions.

The work by Aineto *et al.* [2019] was already mentioned in the previous section. In addition to supporting pairs of initial and final state, they also support partial and full observation of state and plan traces. As special case, they therefore also support the provision of plan traces only. They however only support *adding* preconditions and effects, not changing any.

Lin and Bercher [2021] analyzed the computational complexity of two problems: Given a propositional classical planning problem (without negative preconditions[2]) and a sequence of actions, what is the complexity of deciding whether k repairs (adding or removing preconditions and/or effects) exist? And: Given a propositional (total-order) HTN problem and a sequence of actions, do any repairs exist (adding or removing actions to methods) that make that sequence reachable from the initial task network? Both problems have been identified to be NP-complete in general, though special cases were identified in which the problem becomes tractable. Their results for total-order HTN problems were later extended to a partial-order setting (where methods and the initial task network may be partially ordered) [Lin and Bercher, 2023]. Results remain NP-complete, but further sources of hardness were identified, making the problem tractable when eliminated. They now also propose the inclusion of plans that are required to *not* be solutions (i.e., that one should not

[2]Restricting propositional problems to not have negative preconditions is a common assumption as it makes the problem formalization and heuristic design easier.

be able to achieve them using the hierarchy) and proved the problem where both kinds of problems are available to be NP-hard and in Σ_2^p, the next level in the polynomial hierarchy after NP. Coming back to the first question, k-bounded repair for classical problems, Lin and Bercher [2021] investigated the complexity of propositional problems. In their newest work, they looked at the same problem, but first in a ground setting, where there is the additional constraint that preconditions and effects can not be changed arbitrarily any more, as preconditions and effects have to be compatible with the action arguments. For example, the action schema *drive(*$?t, ?l_1, ?l_2$*)* can only use predicates that use variables $?t$, $?l_1$, and $?l_2$. This prevents, for example, to add preconditions or effects involving a truck different from $?t$, any location different from $?l_1$ and $?l_2$, or anything not a truck or location. For these further constrained problems, Lin *et al.* [2025] could show NP-completeness as well. For the fully lifted setting, where only the input plan is ground, but the model is still lifted, tight bounds remain open as only NEXPTIME-membership could be shown on top of NP-hardness.

The group also worked on solving these problems in practice. For classical problems, they proposed an approach based on computing a sequence of hitting set problems [Lin *et al.*, 2023]. They argue this to be a natural fit, given that both problems are NP-complete (noting that the approach aims at finding a minimal number of repairs). Their approach can handle both propositional problems as well as lifted ones. They can repair both positive and negative preconditions and effects, including both additions and deletions (though adding preconditions is never required as this could never make an action applicable). They can furthermore deal with multiple plans or planning problems referring to the same domain model. They integrated their system into a plug-in for Visual Studio Code [Lin *et al.*, 2024b]. Recently, they extended their algorithm to being able to additionally deal with input plans that are supposed to not be solutions (motivated by plans that turned out to not work in the real world) [Lin *et al.*, 2025].

The work by Chen *et al.* [2024] aligns most closely to the one by Lin *et al.* [2025] in that they also correct models based on failed plans. However, rather than using combinatorial optimization to compute a cardinality-minimal repair set, they exploit the power of LLMs to identify why a failed plan might have failed, and then update the model accordingly. They only repair action preconditions (rather than also effects), but can also create new objects, and even predicates.

Sreedharan and Katz [2023] have looked at how the model repair process could be part of a reinforcement learning process. The proposed method assumes a simulator as its input as well as an "optimistic" model (written in STRIPS without negative preconditions), which is one that admits a superset of all solutions of the true model (represented by the simulator). A diverse planner is used to generate multiple potential plans, which are then tested against the simulator to see if they are valid in the true model. When a failure is detected, the planning model is refined by either adding new preconditions or removing positive or negative effects. The changes are made so that the current estimate remains consistent with previously seen failures and successful state transitions.

Whereas all previous works use simplistic models of planning as defined in Section 2, the work by Lindsay *et al.* [2020] builds on PDDL+ [Fox and Long, 2006], a modeling language for mixed discrete-continuous planning domains with complex time-dependent effects. Their approach relies on an initial hybrid planning problem and observation traces from sensors of a simulator. A learning method uses the observations to derive improved expressions for the "processes" in the domain model, replacing inaccurate process descriptions.

Finally, we turn our attention to repair in hierarchical planning. The total-order HTN model repair problem defined and studied theoretically by Lin and Bercher [2021] was later solved in practice by said group [Lin *et al.*, 2024a]. Although that problem was proved to be in NP, their work encodes the repair problem into an HTN planning problem, which is computationally much harder. Using an optimal search algorithm, their approach guarantees to find the minimal number of repairs. However, their work still has significant restrictions: They can only add missing actions to methods (similar to the work by Xiao *et al.* [2020]), and do not yet support lifted problems. They also assume applicability of the input plan, which could be relaxed to allow changing action definitions.

4.3 Constraint: Make Given Plan Optimal

This constraint can be regarded a refinement of the last criterion. Rather than just ensuring that a plan is a solution to the given problem, one could additionally demand that it is *optimal*. This is the usually the requirement behind *model reconciliation explanations* [Sreedharan *et al.*, 2021a].

In this setting, the explainee, i.e., the user receiving the explanation, is expected to have a user model that is different from the model used by the planner to generate the plan being explained. Here, an explanation consists of identifying the set of model updates to be applied, such that in the resulting model, the plan being explained would be optimal. Many variants of model reconciliation have been investigated, including those where the user model is known [Chakraborti *et al.*, 2017], where there might be model uncertainty [Sreedharan *et al.*, 2018], where the model might be completely unknown [Sreedharan *et al.*, 2019], and in cases where there is a set of users with different beliefs [Sreedharan *et al.*, 2018].

The usual application settings of model reconciliation explanations are those where the planner model corresponds to the ground truth. However, these techniques have also been used in settings where this assumption does not hold. These settings illustrate how these methods could be used for model repair, as the model updates provided as part of the explanation could be used as a focal point for the user to provide corrective feedback to the system. This was an approach used by the RADAR decision-support system [Grover *et al.*, 2020] that allows for the possibility that the planner model need not be completely accurate. They assume that the user might have access to information that the system does not, and as such, certain parts of their mental models may be a more accurate representation of the true underlying task. As such, the RADAR system provides its users the ability to override some model components highlighted by a model-reconciliation explanation, thereby allowing them to perform targeted model updates. However, this is not to say the user is always right, as there may be parts of models where the planner has access

to more accurate information. As such, in these cases, the model reconciliation explanation is treated as being part of a dialogue where the user and the planning system go back and forth to establish the validity of different model components.

Mapping model reconciliation back to our definition of the model repair process, the constraints here could involve ensuring the validity of the plan being explained and the invalidity of any plans that may be less costly than the plan in question. Additionally, all model updates are limited to ones that align with the true planning model.

Extremely closely related to model reconciliation is a method to generate so-called "lies", which again are model updates, whose inclusion in the user model will result in an updated model where some observed plan is optimal – but now these updates are differentiated from model reconciliation in that they are identified independently of the agent model that generated the behavior [Chakraborti and Kambhampati, 2019].

4.4 Constraint: Properties of Solution Space

The next kind of constraint is again related to the last two sections. Previously, it was requested that specific plans are excluded from or included into the solution set. Requesting that a plan is optimal (cf. last section) is also an exclusion criterion on the solution space as it demands that no shorter or cheaper solution exist other than the given one. Properties like these could be generalized further, to allow for arbitrary properties every solution allowed by a planning problem should satisfy.

One example where more properties are enforced is environment design [Zhang *et al.*, 2009]. In this case, the model is an accurate representation of the real world. Such settings involve an environment designer who is capable of modifying the world through actions that are not part of the planning model itself. Consider a planning model capturing a warehouse navigation task. Here, the model corresponds to how a robot could navigate through the cluttered warehouse. On the other hand, the environment designer might be the warehouse supervisor, who could change the environment by moving around shelves, something the robot might not be capable of. Here, the environment designer might want to modify the environment so as to allow solutions of certain properties that are currently not possible in the modeled planning problem. Under environment design, the model repair problem thus corresponds to finding a set of model updates that will allow solutions of the desired property.

A popular design problem is that of goal recognition design [Keren *et al.*, 2020]. Under this problem, the environment designer needs to identify a set of potential environment edits that will try to minimize the length of possible shared prefixes between plans for different goals. Environment design has also been used for applications like plan recognition [Mirsky *et al.*, 2017], and to maximize the utility received by an AI agent [Keren *et al.*, 2019]. An additional requirement placed on model repair techniques by environment design would be that any model change identified in these settings needs to have a corresponding change that can be carried out in the environment. As such, in many of these works, the approaches assume access to a set of model updates that can be applied as part of the model repair process.

5 Assessing (Repaired) Domain Models

The problem of assessing the quality of domain models is a long-standing one in the field of knowledge engineering for planning and scheduling. The most general approach is to identify some properties of a domain model that could define its inherent quality. Well-known properties have been introduced by McCluskey *et al.* [2017], but their assessment is challenging and strongly related to the characteristics of the target domain. This is complicated for automated model repair as they require quantitative optimization metrics that can be evaluated automatically. The alternative approach can be to define aspects of low-quality models [Vallati and Chrpa, 2019], but this can be unfeasible in complex domains. In a different fashion, work has been done in defining quality frameworks to support the encoding of high-quality models [Vallati and McCluskey, 2021], based on the underlying idea of knowledge engineering that the quality of the process strongly influences the quality of the final models. When it comes to assessing repaired domain models, however, it is usually more relevant to compare the repaired models against the original one (especially if they were already runnable or proven, and hence implement a lot of user intent already), or against alternative refined ones.

In the International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS), the organizers introduced a range of qualitative (e.g., correctness and readability) and quantitative measures (e.g., number of predicates, arity of predicates) to compare models that address the same target domain [Chrpa *et al.*, 2017], following the ideas introduced by Shah *et al.* [2013].

Chrpa *et al.* [2023] proposed the use of graph edit distance to assess how syntactically different two models are: in the context of assessing repaired models, this metric could be used to prefer repaired models that are as close as possible to the original one, and to help identify re-usable modules from models encoding similar dynamics [Lindsay, 2023]. Following a similar idea, techniques for performing model repair have focused on generating repaired models that minimize the number of modifications with regard to the original model, as we also saw in several works reviewed in the last section [Lin and Bercher, 2021; Lin *et al.*, 2023].

Rather than comparing the models syntactically, one could also compare the solution sets of the respective problems. For example, adding specific solutions could, depending on the chosen repair, also exclude other solutions, thus changing the solution set in potentially unpredicted ways. This makes more precise requirements desirable, such as making sure that the solution set only increases. Comparing solution sets can however be very hard computationally. For example, checking whether one regular language is a superset of another is in general PSPACE-complete [Hovland, 2012], noting that classical planning problems describe regular languages [Höller *et al.*, 2014; Höller *et al.*, 2016; Lin and Bercher, 2022]. For hierarchical planning, it is even harder. For example, checking whether two context-free languages have an infinite intersection is even undecidable [Hopcroft and Ullman, 1979], noting that total-order HTN problems are context-free [Höller *et al.*, 2014].

Finally, Large Language Models can also be considered for assessing the quality of repaired models: Caglar *et al.* [2024] introduced the idea of the likelihood of model updates, which basically embodies the intuition that a modification makes sense according to the issues to be repaired.

6 Conclusion

Knowledge engineering has long been recognized as a challenging and critical task in the AI planning community – especially for bridging the gap between academic research and real-world applications of planning technology.

In this survey, we focused on *model repair*: the task of modifying an existing model to improve its accuracy or completeness so it satisfies certain desired properties. We contrasted this with *model learning*, which typically aims to construct a domain model from scratch using input data such as plan or state traces. While model learning often requires substantial training data, model repair can be applied later in the development process and may work with far less input. We also proposed this distinction between domain model learning and repair, arguing that the former can be viewed as a specialized subset of the latter.

We reviewed existing work that falls within our definition of model repair, including work that might at first glance not be recognized as technology for modeling support. We found that most work focuses on making unsolvable problems solvable or fixing models based on plan traces. We can observe that almost all works have been developed for the simplest language level, not offering repair support for more expressive language features, although several are often used in practice. Generally, repair still seems to be in its early stages: Even for given input plans (where the most research exists), a lot of further research could be envisioned, such as providing a general description of which solutions are allowed (e.g., using regular expressions or formal grammars) rather than explicitly listing finitely many, allowing to infer new actions (bringing it closer to domain learning), or changing action schema parameters. Also, *many* more possible constraints to be satisfied by the model would be conceivable, such as mutex relations or their generalization to LTL constraints – to name just two examples. Finally, research on how to assess repaired planning models, and how to integrate such metrics into the repair process is also still at its infancy. Most repair approaches therefore simply aim at minimizing the number of repairs, which is simple to evaluate and optimize, yet might not lead to the model with the most desired properties.

Acknowledgments

Pascal Bercher is the recipient of an Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA), project number DE240101245, funded by the Australian Government. Sarath Sreedharan's research is supported in part by grant NSF 2303019. Mauro Vallati was supported by a UKRI Future Leaders Fellowship [grant number MR/Z00005X/1].

References

[Aineto *et al.*, 2018] Diego Aineto, Sergio Jiménez, and Eva Onaindia. Learning strips action models with classical planning. In *Proc. of ICAPS*, pages 399–407. AAAI Press, 2018.

[Aineto *et al.*, 2019] Diego Aineto, Sergio Jiménez Celorrio, and Eva Onaindia. Learning action models with minimal observability. *Artificial Intelligence*, pages 104–137, 2019.

[Alcázar and Torralba, 2015] Vidal Alcázar and Álvaro Torralba. A reminder about the importance of computing and exploiting invariants in planning. In *Proc. of ICAPS*, pages 2–6. AAAI Press, 2015.

[Alford *et al.*, 2015] Ron Alford, Pascal Bercher, and David Aha. Tight bounds for HTN planning with task insertion. In *Proc. of IJCAI*, pages 1502–1508. AAAI Press, 2015.

[Arora *et al.*, 2018] Ankuj Arora, Humbert Fiorino, Damien Pellier, Marc Métivier, and Sylvie Pesty. A review of learning planning action models. *The Knowledge Engineering Review*, 33:1–25, 2018.

[Bacchus and Kabanza, 1998] Fahiem Bacchus and Froduald Kabanza. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 22:5–27, 1998.

[Balyo *et al.*, 2024] Tomáš Balyo, Martin Suda, Lukáš Chrpa, Dominik Šafránek, Stephan Gocht, Filip Dvořák, Roman Barták, and G. Michael Youngblood. Planning domain model acquisition from state traces without action parameters. In *Proc. of KR*, pages 812–822. IJCAI, 2024.

[Bercher *et al.*, 2019] Pascal Bercher, Ron Alford, and Daniel Höller. A survey on hierarchical planning – one abstract idea, many concrete realizations. In *Proc. of IJCAI*, pages 6267–6275. IJCAI, 2019.

[Caglar *et al.*, 2024] Turgay Caglar, Sirine Belhaj, Tathagata Chakraborti, Michael Katz, and Sarath Sreedharan. Can LLMs fix issues with reasoning models? towards more likely models for AI planning. In *Proc. of AAAI*, pages 20061–20069. AAAI Press, 2024.

[Chakraborti and Kambhampati, 2019] Tathagata Chakraborti and Subbarao Kambhampati. (when) can AI bots lie? In *Proc. of AIES*, pages 53–59. ACM, 2019.

[Chakraborti *et al.*, 2017] Tathagata Chakraborti, Sarath Sreedharan, Yu Zhang, and Subbarao Kambhampati. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *Proc. of IJCAI*, pages 156–163. IJCAI, 2017.

[Chen *et al.*, 2024] Guanqi Chen, Lei Yang, Ruixing Jia, Zhe Hu, Yizhou Chen, Wei Zhang, Wenping Wang, and Jia Pan. Language-augmented symbolic planner for open-world task planning. In *Proc. of RSS*, 2024.

[Chrpa *et al.*, 2017] Lukáš Chrpa, Thomas L McCluskey, Mauro Vallati, and Tiago Vaquero. The fifth international competition on knowledge engineering for planning and scheduling: Summary and trends. *AI Magazine*, 38(1):104–106, 2017.

[Chrpa *et al.*, 2020] Lukáš Chrpa, Roman Barták, Jindřich Vodrážka, and Marta Vomlelová. Attributed transition-based domain control knowledge for domain-independent planning. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–13, 2020.

[Chrpa *et al.*, 2023] Lukáš Chrpa, Carmine Dodaro, Marco Maratea, Marco Mochi, and Mauro Vallati. Comparing planning domain models using answer set programming. In *Proc. of JELIA*, pages 227–242. Springer, 2023.

[Fišer and Komenda, 2018] Daniel Fišer and Antonín Komenda. Fact-alternating mutex groups for classical planning. *Journal of Artificial Intelligence Research*, 61:475–521, 2018.

[Fox and Long, 2006] Maria Fox and Derek Long. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27(1):235–297, 2006.

[Geier and Bercher, 2011] Thomas Geier and Pascal Bercher. On the decidability of HTN planning with task insertion. In *Proc. of IJCAI*, pages 1955–1961. AAAI Press, 2011.

[Göbelbecker *et al.*, 2010] Moritz Göbelbecker, Thomas Keller, Patrick Eyerich, Michael Brenner, and Bernhard Nebel. Coming up with good excuses: What to do when no plan can be found. In *Proc. of ICAPS*, pages 81–88. AAAI Press, 2010.

[Gragera *et al.*, 2023a] Alba Gragera, Raquel Fuentetaja, Ángel García-Olaya, and Fernando Fernández. PDDL domain repair: Fixing domains with incomplete action effects. In *ICAPS 2023 System Demonstrations*, 2023.

[Gragera *et al.*, 2023b] Alba Gragera, Raquel Fuentetaja, Ángel García Olaya, and Fernando Fernández. A planning approach to repair domains with incomplete action effects. In *Proc. of ICAPS*, pages 153–161. AAAI Press, 2023.

[Grover *et al.*, 2020] Sachin Grover, Sailik Sengupta, Tathagata Chakraborti, Aditya Prasad Mishra, and Subbarao Kambhampati. Radar: automated task planning for proactive decision support. *Human-Computer Interaction*, 35(5-6):387–412, 2020.

[Grundke *et al.*, 2024] Claudia Grundke, Gabriele Röger, and Malte Helmert. Formal representations of classical planning domains. In *Proc. of ICAPS*, pages 239–248. AAAI Press, 2024.

[Haslum *et al.*, 2019] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool, 2019.

[Helmert, 2006] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[Herzig *et al.*, 2024] Andreas Herzig, Viviane Menezes, Leliane Nunes de Barros, and Renata Wassermann. On the revision of planning tasks. In *Proc. of ECAI*, pages 435–440. IOS Press, 2024.

[Höller *et al.*, 2014] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo. Language classification of hierarchical planning problems. In *Proc. of ECAI*, pages 447–452. IOS Press, 2014.

[Höller *et al.*, 2016] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo. Assessing the expressivity of planning formalisms through the comparison to formal languages. In *Proc. of ICAPS*, pages 158–165. AAAI Press, 2016.

[Höller *et al.*, 2020] Daniel Höller, Gregor Behnke, Pascal Bercher, Susanne Biundo, Humbert Fiorino, Damien Pellier, and Ron Alford. HDDL: An extension to PDDL for expressing hierarchical planning problems. In *Proc. of AAAI*, pages 9883–9891. AAAI Press, 2020.

[Hopcroft and Ullman, 1979] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[Hovland, 2012] Dag Hovland. The inclusion problem for regular expressions. *Journal of Computer and System Sciences*, 78:1795–1813, 2012.

[Jilani, 2020] Rabia Jilani. Automated domain model learning tools for planning. In Vallati and Kitchin [2020], pages 21–46.

[Jiménez *et al.*, 2012] Sergio Jiménez, Tomás De La Rosa, Susana Fernández, Fernando Fernández, and Daniel Borrajo. A review of machine learning for automated planning. *Knowledge Engineering Review*, 27(4):433–467, 2012.

[Keren *et al.*, 2019] Sarah Keren, Luis Enrique Pineda, Avigdor Gal, Erez Karpas, and Shlomo Zilberstein. Efficient heuristic search for optimal environment redesign. In *Proc. of ICAPS*, pages 246–254. AAAI Press, 2019.

[Keren *et al.*, 2020] Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design – survey. In *Proc. of IJCAI*, pages 4847–4853. IJCAI, 2020.

[Krarup *et al.*, 2024] Benjamin Krarup, Amanda Coles, Derek Long, and David E Smith. Explaining plan quality differences. In *Proc. of ICAPS*, pages 324–332. AAAI Press, 2024.

[Lin and Bercher, 2021] Songtuan Lin and Pascal Bercher. Change the world – how hard can that be? on the computational complexity of fixing planning models. In *Proc. of IJCAI*, pages 4152–4159. IJCAI, 2021.

[Lin and Bercher, 2022] Songtuan Lin and Pascal Bercher. On the expressive power of planning formalisms in conjunction with LTL. In *Proc. of ICAPS*, pages 231–240. AAAI Press, 2022.

[Lin and Bercher, 2023] Songtuan Lin and Pascal Bercher. Was fixing this *Really* that hard? On the complexity of correcting htn domains. In *Proc. of AAAI*, pages 12032–12040. AAAI Press, 2023.

[Lin *et al.*, 2023] Songtuan Lin, Alban Grastien, and Pascal Bercher. Towards automated modeling assistance: An efficient approach for repairing flawed planning domains. In *Proc. of AAAI*, pages 12022–12031. AAAI Press, 2023.

[Lin *et al.*, 2024a] Songtuan Lin, Daniel Höller, and Pascal Bercher. Modeling assistance for hierarchical planning: An approach for correcting hierarchical domains with missing actions. In *Proc. of SoCS*, pages 55–63. AAAI, 2024.

[Lin *et al.*, 2024b] Songtuan Lin, Mohammad Yousefi, and Pascal Bercher. A visual studio code extension for automatically repairing planning domains. In *ICAPS 2024 Demonstrations*, 2024.

[Lin *et al.*, 2025] Songtuan Lin, Alban Grastien, Rahul Shome, and Pascal Bercher. Told you that will not work: Optimal corrections to planning domains using counter-example plans. In *Proc. of AAAI*. AAAI Press, 2025.

[Lindsay *et al.*, 2020] Alan Lindsay, Santiago Franco, Rubiya Reba, and Thomas L. McCluskey. Refining process descriptions from execution data in hybrid planning domain models. In *Proc. of ICAPS*, pages 469–477. AAAI Press, 2020.

[Lindsay, 2023] Alan Lindsay. On using action inheritance and modularity in PDDL domain modelling. In *Proc. of ICAPS*, pages 259–267. AAAI Press, 2023.

[Magnaguagno *et al.*, 2020] Maurício C. Magnaguagno, Ramon Fraga Pereira, Martin D. Móre, and Felipe Meneguzzi. Web planner: A tool to develop, visualize, and test classical planning domains. In Vallati and Kitchin [2020], pages 209–227.

[McCluskey and Porteous, 1997] T. L. McCluskey and J. M. Porteous. Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence*, 95:1–65, 1997.

[McCluskey *et al.*, 2002] T. L. McCluskey, N. E. Richardson, and R. M. Simpson. An interactive method for inducing operator descriptions. In *Proc. of AIPS*, pages 121–130. AAAI Press, 2002.

[McCluskey *et al.*, 2017] Thomas L McCluskey, Tiago S Vaquero, and Mauro Vallati. Engineering knowledge for automated planning: Towards a notion of quality. In *Proc. of K-CAP*, pages 1–8. ACM, 2017.

[Mirsky *et al.*, 2017] Reuth Mirsky, Roni Stern, Ya'akov (Kobi) Gal, and Meir Kalech. Plan recognition design. In *Proc. of AAAI*, pages 4971–4972. AAAI Press, 2017.

[Muise and Lipovetzky, 2020] Christian Muise and Nir Lipovetzky. KEPS book: Planning.domains. In Vallati and Kitchin [2020], pages 91–105.

[Pnueli, 1977] Amir Pnueli. The temporal logic of programs. In *Proc. of SFCS*, pages 46–57. IEEE, 1977.

[Shah *et al.*, 2013] Mohammad Munshi Shahin Shah, Lukáš Chrpa, Diane E Kitchin, and Mauro Vallati. Exploring knowledge engineering strategies in designing and modelling a road traffic accident management domain. In *Proc. of IJCAI*, pages 2373–2379. AAAI Press, 2013.

[Sreedharan and Katz, 2023] Sarath Sreedharan and Michael Katz. Optimistic exploration in reinforcement learning using symbolic model estimates. In *NeurIPS*, pages 34519–34535. Curran Associates, Inc., 2023.

[Sreedharan *et al.*, 2018] Sarath Sreedharan, Tathagata Chakraborti, and Subbarao Kambhampati. Handling model uncertainty and multiplicity in explanations via model reconciliation. In *Proc. of ICAPS*, pages 518–526. AAAI Press, 2018.

[Sreedharan *et al.*, 2019] Sarath Sreedharan, Alberto Olmo Hernandez, Aditya Prasad Mishra, and Subbarao Kambhampati. Model-free model reconciliation. In *Proc. of IJCAI*, pages 587–594. IJCAI, 2019.

[Sreedharan *et al.*, 2020] Sarath Sreedharan, Tathagata Chakraborti, Christian Muise, Yasaman Khazaeni, and Subbarao Kambhampati. – D3WA+ – a case study of XAIP in a model acquisition task for dialogue planning. In *Proc. of ICAPS*, pages 488–497. AAAI Press, 2020.

[Sreedharan *et al.*, 2021a] Sarath Sreedharan, Tathagata Chakraborti, and Subbarao Kambhampati. Foundations of explanations as model reconciliation. *Artificial Intelligence*, 301:103558, 2021.

[Sreedharan *et al.*, 2021b] Sarath Sreedharan, Siddharth Srivastava, and Subbarao Kambhampati. Using state abstractions to compute personalized contrastive explanations for AI agent behavior. *Artificial Intelligence*, 301:103570, 2021.

[Strobel and Kirsch, 2020] Volker Strobel and Alexandra Kirsch. MyPDDL: Tools for efficiently creating PDDL domains and problems. In Vallati and Kitchin [2020], pages 67–90.

[Vallati and Chrpa, 2019] Mauro Vallati and Lukáš Chrpa. On the robustness of domain-independent planning engines: the impact of poorly-engineered knowledge. In *Proc. of K-CAP*, pages 197–204. ACM, 2019.

[Vallati and Kitchin, 2020] Mauro Vallati and Diane Kitchin, editors. *Knowledge Engineering Tools and Techniques for AI Planning*. Springer, 2020.

[Vallati and McCluskey, 2021] Mauro Vallati and Thomas Leo McCluskey. A quality framework for automated planning knowledge models. In *Proc. of ICAART*, pages 635–644. SciTePress, 2021.

[Vaquero *et al.*, 2007] Tiago Stegun Vaquero, Victor Romero, Flavio Tonidandel, and José Reinaldo Silva. itSIMPLE$_{2.0}$: An integrated tool for designing planning domains. In *Proc. of ICAPS*, pages 336–343. AAAI Press, 2007.

[Xiao *et al.*, 2020] Zhanhao Xiao, Hai Wan, Hankui Hankz Zhuo, Andreas Herzig, Laurent Perrussel, and Peilin Chen. Refining HTN methods via task insertion with preferences. In *Proc. of AAAI*, pages 10009–10016. AAAI Press, 2020.

[Zhang *et al.*, 2009] Haoqi Zhang, Yiling Chen, and David C. Parkes. A general approach to environment design with one agent. In *Proc. of IJCAI*, pages 2002–2014. IJCAI, 2009.

[Zhuo *et al.*, 2013] Hankz Hankui Zhuo, Tuan Nguyen, and Subbarao Kambhampati. Refining incomplete planning domain models through plan traces. In *Proc. of IJCAI*, pages 2451–2457. AAAI Press, 2013.