Calculating Optimal Corrections for Unsolvable Planning Problems

Michael Welt[a], **Alexander Lodemann**[a], **Conny Olz**[a], **Pascal Bercher**[b] **and Birte Glimm**[a]

[a]Ulm University
[b]Australian National University

Abstract. Detecting and resolving unsolvable planning problems is an active research area that has recently received increased attention. Nevertheless, unsolvability remains a significant challenge, particularly when it comes to efficiently identifying potential causes for a problem's unsolvability. To address this challenge, we propose a method that computes modifications to the planning task. Specifically, given an unsolvable planning problem, our approach identifies a cardinality-minimal set of state variables whose removal renders the problem solvable. Existing literature typically relies on subset enumeration to identify such sets. While effective for small variable sets, we find that this approach becomes impractical for larger sets due to its high computational cost. To overcome this limitation, we introduce a novel method based on hitting set duality, a well-established technique for solving various combinatorial problems. Our results show that this new approach consistently outperforms subset enumeration for medium-sized and large result sets. We validate the effectiveness of our method through experiments on modified problems from the 2016 International Planning Competition on Unsolvability.

1 Introduction

Automated planning is a powerful approach for a range of different applications which require the computation of a sequence of actions that transform a given initial state into a goal state [8]. The formal description of a planning problem is often split into a general domain model (e.g., in the famous blocks world domain, we have the notion of blocks and actions for moving them) and a concrete problem instance (e.g., a concrete initial setup of blocks and a goal description that specifies desired block arrangements).

The creation and maintenance of an error-free model is a well-recognized bottleneck in the use of automated planning [9, 14, 16]. For example, errors might be introduced by failing to specify an essential effect necessary for progressing towards the goal state. Alternatively, the modeler may inadvertently constrain an action excessively, thereby hindering the application of an action that is necessary for fulfilling the goal description. Such errors may manifest in an unsolvable planning problem, i.e., no solution can be found for the given problem instance [2, 7, 10]. While unsolvability can be detected by some planning systems, typically not much insight is given as to what are possible causes, let alone the proposal of fixes for regaining solvability.

To that end, this paper is concerned with model repair under the constraint of restoring solvability [4]. Repairing a faulty planning problem, while preserving as much as possible from the original definitions, is computationally challenging since (i) planners are typically optimized towards finding plans rather than towards showing unsolvability and (ii) the number of potential fixes can be huge [10].

The essence of this paper lies in its novel approach, which tackles unsolvable planning problems by computing a cardinality-minimal correction. This correction comprises a set of state variables whose removal renders the problem solvable. Consequently, our method aims to preserve the integrity of the model by making minimal adjustments to the underlying set of variables.

Our method differs significantly from existing literature, which primarily focuses naively on enumerating subsets [2, 23]. In contrast, we propose a novel approach based on the so-called hitting set duality [18, 22]. We provide a formal proof of the complexity associated with our method and illustrate its superior performance compared to subset enumeration, particularly evident on medium-sized correction sets, while demonstrating comparable efficiency on small solutions. These comparisons are conducted using a modification of the dataset from the 2016 Unsolvability International Planning Competition.

The remainder of the paper is organized as follows: Section 2 presents related works, the following Section 3 provides a succinct overview of the technical background and formalizations utilized throughout this paper along with the introduction of our illustrative example. In Section 4, we present our method, discuss its complexity, and the associated algorithms. In Section 5, we then present the outcomes of our empirical evaluation. Finally, Section 6 concludes with some remarks and considerations for future research directions.

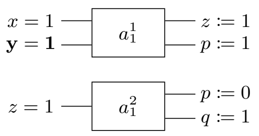
2 Related works

Various approaches have been documented in the existing literature aiming to transform unsolvable planning problems into solvable ones through (optimal) problem transformation. Yang [25] introduced a conflict resolution theory to identify inconsistencies early in the search process using a constraint satisfaction framework.

Another approach, presented by Göbelbecker et al. [9], addresses unsolvable planning problems by proposing optimal modifications to the initial state, thereby rendering the problem solvable. This method involves compiling the optimization problem into a planning instance to derive a solution, without altering the underlying model. In contrast, our work addresses the challenge of fixing unsolvable planning instances by optimally altering the entire task.

Modifying the domain of an unsolvable lifted planning problem has only recently been addressed by Gragera et al. [10]. Their ap-

$$\begin{aligned} V_1 &= \{x, y, z, p, q\} \\ a_1^1 &= (\{(x,1), (\mathbf{y}, \mathbf{1})\}, \{(z,1), (p,1)\}) \\ a_1^2 &= (\{(z,1)\}, \{(p,0), (q,1)\}) \\ I_1 &= \{(x,1), (\mathbf{y}, \mathbf{0}), (z,0), (p,0), (q,0)\} \\ G_1 &= \{(p,1), (q,1)\} \end{aligned}$$



$$\begin{aligned} V_2 &= \{x, z, p, q\} \\ a_2^1 &= (\{(x,1)\}, \{(z,1), (p,1)\}) \\ a_2^2 &= (\{(z,1)\}, \{(p,0), (q,1)\}) \\ I_2 &= \{(x,1), (z,0), (p,0), (q,0)\} \\ G_2 &= \{(p,1), (q,1)\} \end{aligned}$$

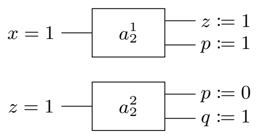


Figure 1: Variables and actions with sketches for Π_1 (left, problematic parts highlighted) and Π_2 (right) for Example 1

proach involves fixing the domain by compiling the problem, akin to Göbelbecker et al. [9], into a planning problem itself. This translation introduces repair actions and utilizes an empirically established cost model to guide the search for a cost-optimal plan, i.e., a fix. Gragera et al. [10] focus, however, solely on missing action effects, identified as a primary cause of errors in planning problems. The resulting plan from the compiled planning problem simultaneously serves as an explanation and a fix to the original domain. Optimality in terms of minimizing repairs is not guaranteed by the approach.

In the realm of Explainable Planning, Sreedharan et al. [23] elucidate to a human why a problem instance, generated by a robot, is unsolvable. They pinpoint the state variables that potentially cause misunderstanding between the robot and the human by employing subset enumeration and problem reduction techniques, which involve removing variables from the unsolvable instance. Bäckström et al. [2] utilize a comparable technique of variable enumeration followed by variable removal from the problem. Their goal is to efficiently identify an unsolvable problem instance in the first place.

Our work builds upon enumeration and strategic variable removal techniques, by incorporating hitting sets, a well-known method in various fields dealing with combinatorial problems. Lin et al. [14, 15] and Bavandpour et al. [3] utilize hitting sets within the context of optimal planning domain repair, albeit with a different overarching premise. In their approach, the authors start with a given planning problem and plan that is not a valid solution for the instance yet. They then 'repair' the problem by executing atomic changes to actions such as removing a precondition or changing an effect to ensure that the plan becomes a valid solution. By compiling their optimization problem into a diagnosis problem akin to Reiter [18] and Slaney [22], an optimal number of repairs can be determined. While this method can be employed to convert an unsolvable planning instance into a solvable one, compared to our proposed solution, it necessitates additional input – the plan intended to become a solution. Finally, it is worth noting that the additional input required by Lin et al. [14] impacts the computational complexity of the problem we tackle. Their repair problem, contingent on additional plan inputs, is NP-complete [13], whereas our presented approach implicitly relies on solving the plan existence problem, which is PSPACE-complete [1, 5].

3 Background

In this paper, we adopt a convention where lowercase letters denote variable names and constants, while capitalized letters signify sets, except when referring to states, maintaining consistency with existing literature. Lowercase Greek letters are employed to represent collections of sets. Furthermore, we will use the *SAS*$^+$ formalism as specified by Bäckström and Nebel [1]. A *SAS*$^+$ *(planning) problem* is a tuple $\Pi = (V, A, I, G)$ with V denoting a finite set of *variables*. Each variable $v \in V$ is associated with a finite *domain* D_v. A *partial state* s is a set of pairs, $s \subseteq \bigcup_{v \in V} \{(v, d) \mid d \in D_v\}$, where each variable in $v \in V$ occurs in at most one pair. A *total state* or just *state* for short is a partial state in which all variables of V are covered. States can also be seen as (partial or total) functions highlighted by the corresponding notation $s[v] = d$ if $(v, d) \in s$. We will use the notation $\mathit{vars}(s) = \{v \mid (v, d) \in s\}$ to refer to all variables that have a corresponding assignment in s. We define the *composition* of two states s, t as $s \oplus t = t \cup \{(v, s[v]) \mid v \in \mathit{vars}(s) \setminus \mathit{vars}(t)\}$. The special total state I is called the *initial state* of Π and the partial state G is referred to as the *goal description*. The set A specifies the *actions*, where each action $a \in A$ consists of a pair of states $(\mathit{pre}(a), \mathit{eff}(a))$, which are called the *precondition* and *effect* of a, respectively. We say that a has a *prevail condition* on a variable v if $v \in \mathit{vars}(\mathit{pre}(a))$ but $v \notin \mathit{vars}(\mathit{eff}(a))$. An action $a \in A$ is *applicable* in a state s if $\mathit{pre}(a) \subseteq s$ and the result of applying such an a in s is the state $s \oplus \mathit{eff}(a)$.

Let s_0 and s_n be two states and $\omega = \langle a_1, \ldots, a_n \rangle$ a possibly empty sequence of actions. We say that ω is a *plan* from s_0 to s_n if either ω is empty and $s_0 = s_n$ or there exists a sequence of intermediate states $\langle s_1, \ldots, s_{n-1} \rangle$ such that a_i is applicable in s_{i-1} and s_i is the result of applying a_i in s_{i-1} for $1 \leq i \leq n$. A plan from s_0 to s_n is a *solution* for Π if $I = s_0$ and $s_n \supseteq G$. A *SAS*$^+$ problem Π is said to be *solvable* if a solution exists and *unsolvable* otherwise. Note that in the special case where $G \subseteq I$, the empty sequence serves as a valid solution for Π.

Given a *SAS*$^+$ problem $\Pi = (V, A, I, G)$ and a set $P \subseteq V$, we define the *(variable) abstraction* [6] of a state s onto P as $s|_P = \{(v, d) \mid (v, d) \in s, v \in P\}$. In other words, an abstraction of a state only retains pairs containing variables in P. An abstraction is also denoted as *projection abstraction* or *projection* for short. The abstraction of the problem Π is defined accordingly as $\Pi|_P = (P, A|_P, I|_P, G|_P)$ such that $A|_P = \{(\mathit{pre}(a)|_P, \mathit{eff}(a)|_P) \mid a \in A, \mathit{eff}(a)|_P \neq \emptyset\}$. This definition suggests that the abstraction process could result in fewer actions compared to the original concrete problem.

The *complement* of a set $P \subseteq V$ is defined relative to V as $\overline{P} = V \setminus P$. If sufficiently clear from the context, we omit the explicit reference to V. Furthermore, by abuse of notation, we also use G ($\overline{G}$) as shorthand for $\mathit{vars}(G)$ ($\overline{\mathit{vars}(G)}$). In this context, we write $\Pi|_G$ and $\Pi|_{\overline{G}}$ as a shorthand for $\Pi|_{\mathit{vars}(G)}$ and $\Pi|_{\overline{\mathit{vars}(G)}}$, respectively.

It is crucial to highlight that in the case of an unsolvable problem $\Pi = (V, A, I, G)$, the abstraction $\Pi|_{\overline{G}}$ is always solvable. This is because the abstracted goal $G|_{\overline{G}}$ becomes the empty set, which is inherently a subset of the initial state. Consequently, the problem $\Pi|_{\overline{G}}$ becomes trivially solvable by employing the empty plan and any executable one.

Example 1. The subsequent examples showcase the formalisms and serve as an illustration of a prevalent modeling error arising from overly restrictive preconditions. Consider the following two planning problems: $\Pi_1 = (V_1, A_1 = \{a_1^1, a_1^2\}, I_1, G_1)$ and $\Pi_2 = (V_2, A_2 = \{a_2^1, a_2^2\}, I_2, G_2)$ where V_1, A_1, V_2 and A_2 are specified in Figure 1, which additionally illustrates the actions. Both planning problems differ only in the variable y, which is used as a precondition for a_1^1 in Π_1, but which does not occur in Π_2.

The planning problem Π_1 is unsolvable: For the initial state I_1 in Π_1, no action is applicable as neither the preconditions of a_1^1 nor the preconditions of a_1^2 are satisfied and $G_1 \not\subseteq I_1$.

The planning problem Π_2, on the other hand, is solvable: For the initial state I_2 in Π_2, the action a_2^1 is applicable, that is $s_1 = I_1 \oplus \mathit{eff}(a_2^1) = \{(x,1),(z,1),(p,1),(q,0)\}$. One can now apply a_2^2 to get $s_2 = s_1 \oplus \mathit{eff}(a_2^2) = \{(x,1),(z,1),(p,0),(q,1)\}$. While this removes one of the goals, action a_2^1 is still applicable and applying it again yields the goal state $s_3 = s_2 \oplus \mathit{eff}(a_2^1) = \{(x,1),(z,1),(p,1),(q,1)\}$. Hence, the sequence $\langle a_2^1, a_2^2, a_2^1\rangle$ is a solution for Π_2.

4 Method

We start by formally describing the optimization problem and discussing its complexity. Then, we introduce two algorithms to tackle the problem. The first algorithm describes a state-of-the-art method based on simply enumerating subsets [23], which we use as a baseline for our experiments. The second algorithm is more complex and is based on hitting sets, a technique we adapt from the work of Slaney [22] to the context of automated planning.

Given an *unsolvable SAS*$^+$ planning problem $\Pi = (V, A, I, G)$, our objective is to identify a set $P \subseteq \overline{G}$ such that the abstraction of Π induced by the complement $\Pi|_{\overline{P}}$ becomes solvable. Additionally, we aim for P to be of minimum cardinality, meaning that for any $P' \subseteq \overline{G}$ where the induced abstraction $\Pi|_{\overline{P'}}$ is solvable, it must hold that $|P'| \geq |P|$. We call such a set P a *minimal correction* for Π.

Example 2. We illustrate our formalization of the problem using Example 1. It is evident that the difference between Π_1 and Π_2 lies in the over-restriction of action a_1^1 caused by the variable y. Removing y from Π_1 yields Π_2. Consequently, we observe that $\Pi_2 = \Pi_1|_{\overline{\{y\}}}$, indicating that Π_2 represents the abstraction of Π_1 induced by the complement of $\{y\}$. The solution $\langle a_2^1, a_2^2, a_2^1\rangle$ for Π_2 from above demonstrates the solvability of the corresponding abstraction of Π_1. Furthermore, only the empty set has a smaller cardinality than $\{y\}$. For the empty set, it holds that $\Pi_1|_{\overline{\emptyset}} = \Pi_1$, which, by assumption, is unsolvable. Additionally, $\{y\} \subseteq \overline{G}$ satisfies our formal requirements. Therefore, we can conclusively affirm that $\{y\}$ indeed serves as a solution for our formalized optimization problem and consequently acts as a cardinality-minimal correction for Π_1.

Before proceeding with an analysis of the complexity of the optimization problem at hand, we address the necessity of our formal requirement that a correction needs to be *goal-avoidant*, i.e., to be a subset of $\overline{G}$. As outlined in Section 3, removing the goal description from an unsolvable planning problem renders it trivially solvable. In addition to the formal rationale, it is customary to preserve the original intent of the modelers as much as possible, which typically precludes modifications to the goal description. Consequently, the existence of any correction is not necessarily given, as it might happen that a correction needs to abstract away all or at least some part of the goal. Therefore, our method has no trivial solutions.

A straightforward test, however, enables us to determine the existence of a solution within $2^{\overline{G}}$. Intuitively, if the problem, reduced solely to its goals, has no solution, then a solution containing more variables than the goal variables cannot exist. This is because a finer abstraction never contradicts unsolvability. A more formal proof is provided below.

Lemma 1. Let $\Pi = (V, A, I, G)$ be an unsolvable *SAS*$^+$ planning problem. If $\Pi|_G$ is unsolvable, then there exists no cardinality-minimal goal-avoidant correction $P \subseteq \overline{G}$.

Proof. If $\Pi|_G$ is unsolvable, then, for any $P \subseteq \overline{G}$, $\overline{P} \supseteq G$. Thus, $\Pi|_{\overline{P}}$ must also be unsolvable, since it represents a finer abstraction than the one induced by G alone (cf. Bäckström et al. [2]). $\square$

4.1 Complexity

We establish that the optimization problem we address (or, to be more precise, the related decision problem) is PSPACE-complete. In describing the complexity of optimization problems, it is often customary to introduce a parameter $k \in \mathbb{N}$ to create a corresponding bounded decision problem. In our scenario, systematically reducing the value of k to pinpoint the threshold at which the decision switches from true to false would determine the solution to the original minimization problem.

Theorem 1. Let $\Pi = (V, A, I, G)$ be an unsolvable *SAS*$^+$ problem and $k \in \mathbb{N}$. Deciding whether there exists a set $P \subseteq \overline{G}$ with $|P| \leq k$ s.t. $\Pi|_{\overline{P}}$ is solvable, is PSPACE-complete.

Proof. Membership: Our problem falls within NPSPACE because (i) we can guess a set $P \subseteq \overline{G}$ with $|P| \leq k$ and (ii) verify whether $\Pi|_{\overline{P}}$ is solvable, where (ii) essentially addresses the plan existence problem which is PSPACE-complete [1]. PSPACE membership follows from NPSPACE = PSPACE [19].

Hardness: We reduce from the general plan existence problem. Let $\Pi = (V, A, I, G)$ be a *SAS*$^+$ problem. We transform Π to an instance of our problem: Let $\Pi' = (V', A', I', G')$ be the *SAS*$^+$ problem, where $V' = V \cup \{x, y\}, x, y \notin V, D_x = D_y = \{0, 1\}$ with $I' = I \cup \{(x,0),(y,0)\}$ and $G' = G \cup \{(y,1)\}$, respectively. Additionally, let $A' = A \cup \{a'\}$, with $a' \notin A, a' = (\{(x,1)\},\{(y,1)\})$. Clearly, Π' is unsolvable since $(y,1)$ can never be established without adding $(x,1)$ first and then apply a' in a potential plan. However, $(x,1)$ can never be established in the first place, since there is no action to accomplish that. Choosing the set $P = \{x\}$ ($|P| = 1 \leq k$) is the only way to render $\Pi'|_{\overline{\{x\}}}$ solvable if Π is solvable. Thus, Π is solvable if and only if there exists a set $P \subseteq V' \setminus G'$ with $|P| \leq 1$ such that $\Pi'|_{\overline{P}}$ is solvable, which shows hardness. $\square$

4.2 Enumeration

To find a set with minimal cardinality, one can undertake a straightforward process of exhaustively enumerating all possible elements within the powerset $2^{\overline{G}}$ of $\overline{G}$ in ascending order of their respective cardinalities. This approach resembles a similar process outlined by Sreedharan et al. [23]. This listing process starts with individual singleton subsets, progresses to pairs of variables, then expands to subsets with three elements, and so on. For each enumerated set P, the abstracted problem induced by its complement needs to be tested for solvability. Upon encountering an abstraction that leads to a solvable problem, the procedure halts, and the solution is returned. This solution must be of minimal cardinality, as all potentially smaller solutions have been explored in the prior enumeration.

This procedural approach is sound and complete as it finds a minimal goal-avoidant correction if and only if one exists within $2^{\overline{G}}$. Importantly, the two trivial solutions, namely $\Pi|_{\overline{V}}$ (which is the empty problem, containing no variables) and $\Pi|_{\overline{G}}$ (which has an empty goal description), remain undetected, as they are excluded initially by selecting $P \subseteq \overline{G}$. The procedure terminates either upon discovering a correction or after exhaustively enumerating all elements within $2^{\overline{G}}$. In the worst case, the procedure enumerates $2^{|\overline{G}|}$ subsets and tests them for solvability.

4.3 Hitting set duality

Another method commonly found in the literature to address combinatorial optimization problems is the utilization of the so-called hitting set duality [14, 18, 22]. Given a collection of sets δ with its associated domain of elements $D_\delta = \bigcup_{M \in \delta} M$, a subset $H \subseteq D_\delta$ qualifies as a *hitting set* for δ if for every set $M \in \delta$ the intersection of M and H contains at least one element, i.e., $M \cap H \neq \emptyset$.

In specific discrete optimization scenarios, a notable phenomenon arises where two sets mutually act as hitting sets for each other. This property, initially recognized by Reiter [18], was first leveraged within the context of optimal diagnosis. Subsequently, this property has been recurrently employed in discrete optimization, including tasks such as identifying an optimal set of constraints in constraint satisfaction or determining a minimal set of axioms entailing a specified formula. Eventually, Slaney [22] introduced a generalized representation of this optimization problem using set-theoretic notation, along with a corresponding generalized algorithm.

To employ the duality-based approach to our problem, we establish two distinct collections of sets. Firstly, θ encompasses all subsets for which the abstraction induced by its complement is solvable:

$$\theta = \{P \subseteq \overline{G} \mid \Pi|_{\overline{P}} \text{ is solvable } \}$$

This set, θ, constitutes our target set, comprising all potential corrections, which may not necessarily be minimal. Note that in our constrained scenario where $P \subseteq \overline{G}$, θ could possibly be empty.

The second set we define, is called the *dual set* θ^* containing subsets whose complements relative to $\overline{G}$ are not part of θ:

$$\theta^* = \{P \subseteq \overline{G} \mid \overline{G} \setminus P \notin \theta\}$$

With this formalization, we can now redefine our optimization problem as the search for a minimal-cardinality element within θ.

We proceed by establishing two crucial properties: Firstly, we demonstrate the superset monotonicity of θ. Secondly, we establish the relationship between hitting sets for θ^* and elements of θ. Subsequently, we utilize these properties to suggest an algorithm for identifying minimal elements of θ by seeking minimal hitting sets for θ^* without necessarily fully computing θ^*. We begin by proving the superset monotonicity of θ:

Theorem 2. Let $\Pi = (V, A, I, G)$ be an unsolvable planning problem. Then, for the set θ it holds that

$$\forall P \subseteq P' \subseteq \overline{G} \colon P \in \theta \Longrightarrow P' \in \theta$$

Proof. Let $P \subseteq P' \subseteq \overline{G}$. To the contrary of what is to be shown, assume $P \in \theta$ and $P' \notin \theta$. This implies that $\Pi|_{\overline{P}}$ is solvable, but $\Pi|_{\overline{P'}}$ is unsolvable. By definition of complements and since $P \subseteq P'$, $\overline{P'} \subseteq \overline{P}$. Hence, a plan for $\Pi|_{\overline{P}}$ exists, but for the coarser abstraction $\Pi|_{\overline{P'}}$, the established plan no longer holds. The contradiction arises because coarser abstractions can never be more restrictive as shown by Bäckström et al. [2]. □

Next, we establish the hitting set connection between elements of θ^* and θ:

Lemma 2. Let $\Pi = (V, A, I, G)$ be an unsolvable planning problem and let $H \subseteq \overline{G}$ be such that, for every $P \in \theta^*$, $H \cap P \neq \emptyset$. Then, $H \in \theta$.

Proof. As all potential elements from H are subtracted from $\overline{G}$ before intersecting, we have $H \cap (\overline{G} \setminus H) = \emptyset$. Furthermore, H is such that $H \cap P \neq \emptyset$ for all $P \in \theta^*$ by assumption. Hence, $(\overline{G} \setminus H) \notin \theta^*$. By definition of θ^*, we infer $\overline{G} \setminus (\overline{G} \setminus H) = H \in \theta$. □

Theorem 2 and Lemma 2 allow us to reformulate our optimization problem once more as the search for a minimal hitting set for θ^*. Algorithm 1 outlines a method for identifying a minimal hitting set for θ^* given an unsolvable problem Π. Before invoking the algorithm, Lemma 1 can be used to check whether a solution exists. On the initial call, $\kappa = \emptyset$ is passed to the function. The set κ represents the subset of θ^* computed so far. The algorithm operates recursively (cf. Line 9). Each recursive step first computes a minimal-cardinality hitting set H for the current set κ. For the initial call, with $\kappa = \emptyset$, the function MINHS returns an empty set by convention. The solvability of the abstraction induced by the complement of H is then assessed. If solvable, a correction is found and returned. Otherwise, if H is not a member of θ, its complement relative to $\overline{G}$ must belong to θ^*. However, the complement of H with respect to $\overline{G}$ is not directly added to κ. Instead, κ is expanded with the complement of a maximized superset of H relative to $\overline{G}$. The intuition behind this is, to produce minimal elements in κ in order to potentially speed up the search. To this end, H can be expanded with elements from $\overline{G} \setminus H$ as long as the planning problem induced by the complement of the expanded set H remains unsolvable. The exact procedure is described in Algorithm 2. In a loop, H is iteratively expanded until the abstraction induced by its complement becomes solvable. Without this maximization step, Algorithm 1 would be equivalent to subset enumeration. Thus, in the worst-case scenario, this approach necessitates $2^{|\overline{G}|}$ tests [22].

The underlying concept of the approach lies in avoiding a full computation of θ^*. Instead, the method focuses on finding a subset $\kappa \subseteq \theta^*$ for which a minimal hitting set also serves as a minimal correction. Consequently, if a hitting set is minimal for a subset of θ^*, it is inferred to be minimal for the entire set θ^* [22].

Example 3. We utilize the unsolvable problem Π_1 of Example 1 to demonstrate the functionality of Algorithm 1. The process begins with evaluating the problem reduced to its objectives $\Pi_1|_G$ for solvability: $\Pi_1|_G = (V_1|_G, A_1|_G = \{a_1^1|_G, a_1^2|_G\}, I_1|_G, G|_G)$ with

$$\begin{array}{llll} V_1|_G & = \{p, q\} & a_1^1|_G & = (\emptyset, \{(p,1)\}) \\ I_1|_G & = \emptyset & a_1^2|_G & = (\emptyset, \{(p,0),(q,1)\}) \\ G_1|_G & = \{p, q\} & & \end{array}$$

A plan for $\Pi_1|_G$ is $\langle a_1^1|_G, a_1^2|_G, a_1^1|_G \rangle$. Consequently, we can run Algorithm 1. Figure 2 shows a trace for the algorithm including its calls to Algorithm 2. Next to the line numbers, the left-hand side shows the expressions that are evaluated during the run, while the right-hand side shows relevant current values for the expressions.

In Line 1 of Figure 2, the computation of a cardinality-minimal correction for Π_1 starts by calling Algorithm 1 for Π_1 and an initially empty set κ. As per convention, MINHS$(\emptyset) = \emptyset$, which causes H to be initialized with the empty set (Line 2). Since $\Pi_1|_{\overline{\emptyset}} = \Pi_1$ is (obviously still) unsolvable (Line 3), Algorithm 2 is called to maximize H (Line 4). We assume that the variable x is considered in the first loop iteration (Line 5). Since the abstraction $\Pi_1|_{\overline{\{x\}}}$ remains unsolvable (Line 6), H is extended with x (Line 7) and the maximization process continues. In the next loop iteration, we assume that y taken (Line 8), yielding a solvable abstraction (Line 9). Hence, the last unsolvable $H = \{x\}$ is returned (Line 10).

Back in Algorithm 1, κ is extended to $\{\{y, z\}\}$ (the complement of H relative to $\overline{G}$) in Line 11. Then, MINCORR is called recursively for Π_1 and the extended κ (Line 12). The next steps depend on the employed hitting set algorithm; for the sake of our example, we assume that the algorithm returns $\{z\}$ (Line 13), which is used as new hitting set H. Note that the algorithm could also have equally returned the cardinality-minimal hitting set $\{y\}$, terminating

Algorithm 1 Find a cardinality-minimal correction

```
 1: function MinCorr($\Pi = (V, A, I, G), \kappa$)
 2:     $H \leftarrow$ MinHS($\kappa$)
 3:     if $\Pi|_{\overline{H}}$ is solvable then
 4:         return $H$
 5:     else
 6:         $M \leftarrow$ Maximize($\Pi, H$)
 7:         $\kappa \leftarrow \kappa \cup \{\overline{G} \setminus M\}$
 8:     end if
 9:     return MinCorr($\Pi, \kappa$)
10: end function
```

Algorithm 2 Maximize Hitting Set

```
 1: function Maximize($\Pi = (V, A, I, G), H$)
 2:     for $v \in \overline{G} \setminus H$ do
 3:         if $\Pi|_{\overline{H \cup \{v\}}}$ is unsolvable then
 4:             $H \leftarrow H \cup \{v\}$
 5:         else
 6:             return H
 7:         end if
 8:     end for
 9:     return H
10: end function
```

1	MinCorr(Π, κ)	$\Pi = \Pi_1$, $\kappa = \emptyset$
2	$H \leftarrow$ MinHS(κ)	MinHS($\emptyset$) $= \emptyset \rightsquigarrow H = \emptyset$
3	$\Pi\|_{\overline{H}}$ is unsolvable	$\overline{H} = \overline{\emptyset} = V_1$
4	$M =$ Maximize(Π, H)	$\Pi = \Pi_1$, $H = \emptyset$
5	take $v \in \overline{G} \setminus H$	$\overline{G} \setminus H = \{x, y, z\}$, take $v = x$
6	$\Pi\|_{\overline{H \cup \{v\}}}$ is unsolvable	$\overline{H \cup \{v\}} = \{y, z, p, q\}$
7	$H \leftarrow H \cup \{v\}$	$\rightsquigarrow H = \{x\}$
8	take $v \in \overline{G} \setminus H$	$\overline{G} \setminus H = \{y, z\}$, take $v = y$
9	$\Pi\|_{\overline{H \cup \{v\}}}$ is solvable	$\overline{H \cup \{v\}} = \{z, p, q\}$
10	return H	$H = \{x\} \rightsquigarrow M = \{x\}$
11	$\kappa \leftarrow \kappa \cup \{\overline{G} \setminus M\}$	$\overline{G} \setminus M = \{y, z\}$
		$\rightsquigarrow \kappa = \{\{y, z\}\}$
12	MinCorr(Π, κ)	$\Pi = \Pi_1$, $\kappa = \{\{y, z\}\}$
13	$H \leftarrow$ MinHS(κ)	MinHS($\{\{y, z\}\}$) $= \{z\}$
		$\rightsquigarrow H = \{z\}$
14	$\Pi\|_{\overline{H}}$ is unsolvable	$\overline{H} = \{x, y, p, q\}$
15	$M =$ Maximize(Π, H)	$\Pi = \Pi_1$, $H = \{z\}$
16	take $v \in \overline{G} \setminus H$	$\overline{G} \setminus H = \{x, y\}$, take $v = x$
17	$\Pi\|_{\overline{H \cup \{v\}}}$ is unsolvable	$\overline{H \cup \{v\}} = \{y, p, q\}$
18	$H \leftarrow H \cup \{v\}$	$\rightsquigarrow H = \{x, z\}$
19	take $v \in \overline{G} \setminus H$	$\overline{G} \setminus H = \{y\}$, take $v = y$
20	$\Pi\|_{\overline{H \cup \{v\}}}$ is solvable	$\overline{H \cup \{v\}} = \{p, q\}$
21	return H	$H = \{x, z\} \rightsquigarrow M = \{x, z\}$
22	$\kappa \leftarrow \kappa \cup \{\overline{G} \setminus M\}$	$\overline{G} \setminus M = \{y\}$
		$\rightsquigarrow \kappa = \{\{y, z\}, \{y\}\}$
23	MinCorr(Π, κ)	$\Pi = \Pi_1$, $\kappa = \{\{y, z\}, \{y\}\}$
24	$H \leftarrow$ MinHS(κ)	MinHS($\{\{y, z\}, \{y\}\}$) $= \{y\}$
		$\rightsquigarrow H = \{y\}$
25	$\Pi\|_{\overline{H}}$ is solvable	$\overline{H} = \{x, z, p, q\}$
26		$\rightsquigarrow$ recursive ascent: $H = \{y\}$

Figure 2: A full trace of Algorithm 1 on Π_1 from Example 1

the search.

The set $H = \{z\}$ does again not belong to θ due to the continued inapplicability of a_1^1, rendering the goal $(p, 1)$ unachievable (Line 14). Hence, $H = \{z\}$ is again maximized by calling Algorithm 2 (Line 15) yielding the extension $\{x, z\}$ (Lines 16–21). The complement of this set with respect to $\overline{G}$ is again added to κ upon the return to Algorithm 1, resulting in $\kappa = \{\{y, z\}, \{y\}\}$ (Line 22). The next recursive call of MinCorr (Line 23) finds that the set κ admits only one cardinality-minimal hitting set (Line 24), namely $\{y\}$. Since $\Pi_1|_{\overline{y}} = \Pi_1|_{\{x,z,p,q\}}$ is solvable (Line 25), $H = \{y\}$ is returned as cardinality-minimal correction.

In summary, the process showcases the gradual expansion of κ by utilizing the identified hitting sets until a set H belonging to θ is discovered, indicating the solvability of $\Pi_1|_{\overline{H}}$.

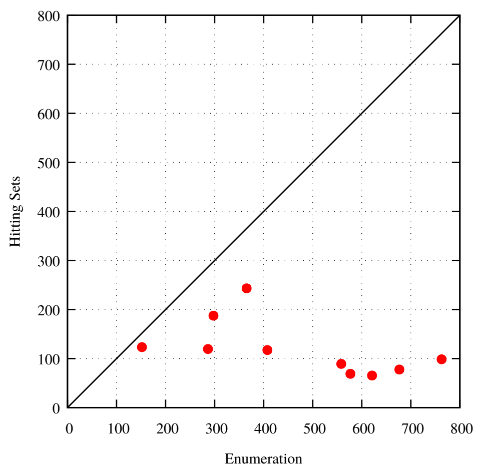
5 Evaluation

We conducted an extensive comparison between our hitting set-based approach and the baseline enumeration strategy. Both methods were implemented in C, including our own implementation of the hitting set solver. For all experiments, we used the Fast-Downward planner in version 23.96+ [11]. Although we set a timeout of 1800 seconds for individual planner calls, our overall runtime budget per instance was extended to 2000 seconds. Since the majority of the inputs we hand over to the planner are expected to be unsolvable planning tasks, we chose Eager Greedy Search in combination with the `hmax` heuristic, which offers acceptable pruning properties at relatively low computational cost. To the best of our knowledge, these parameters strike a good balance, when compared to the often more elaborate setups used in the IPC 2016 planners [17]. The complete source code[1] as well as the used dataset is available as executable Docker file for easy reproducibility [24].

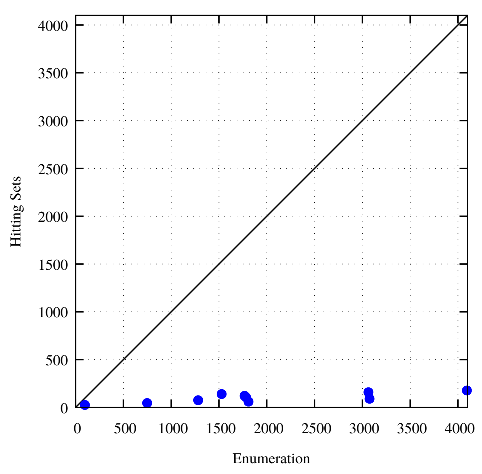
For our experiments, we used the dataset from the IPC 2016 Unsolvability Contest [17] as a starting point and introduced additional modifications. Specifically, we generated multiple new instances from each original problem to control the size of the required corrections. We included those domains and problem instances for which the Fast-Downward planner was able to confirm unsolvability within one minute, allowing for the generation of harder variants that can potentially be shown unsolvable within the overall time limits. Domains for which problem instances are particularly difficult to classify as unsolvable, such as the tetris domain, are, therefore, not included in our dataset. In total, this resulted in a dataset of 475 problem instances across nine different domains. All experiments were conducted on an Intel(R) Xeon(R) CPU E5-2440 at 2.40GHz, using one core per instance and a total memory capacity of 144GB.

Our main findings are summarized in Figures 3 and 4. The left plot of Figure 3 compares the runtimes[2] of both approaches, with the baseline on the x-axis and the hitting set-based method on the y-axis. The values are grouped by correction size, meaning that each point represents the average value for a specific correction size. The baseline approach is shown on the x-axis and the hitting set approach on the y-axis. Since all points lie below the diagonal, it is evident that our approach consistently outperforms the baseline. The right

[1] https://gitlab.uni-ulm.de/ptc22/mincorr.git
[2] In Figure 3, runtimes are averaged by correction size. Only those sizes are shown, for which both methods succeeded, which limits the upper bound.

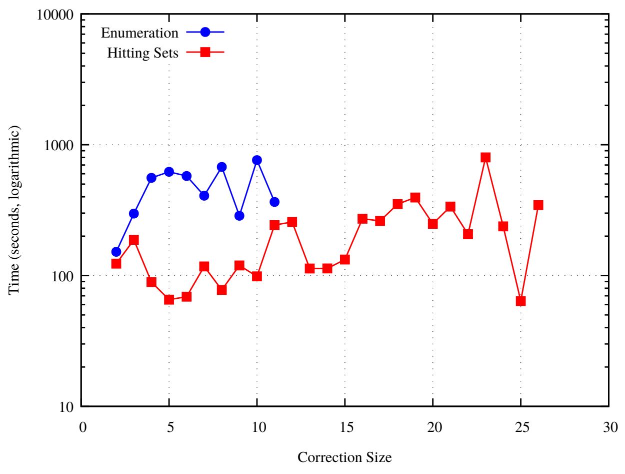


(a) Comparison of total runtimes in seconds

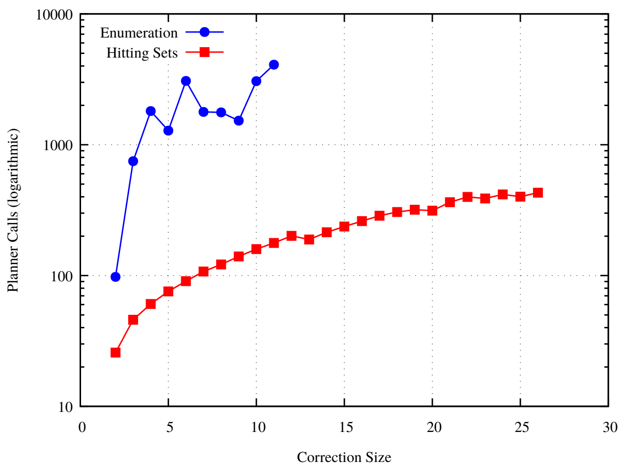


(b) Comparison of total number of planner calls

Figure 3: Direct comparisons baseline versus hitting set approach on the modified IPC 2016 data set



(a) Total runtimes



(b) Total number of planner calls

Figure 4: Baseline versus hitting set approach on the modified IPC 2016 data set

plot of Figure 3 compares the number of planner calls required by each method. Again, the points lie significantly below the diagonal, indicating the superiority of our approach in terms of efficiency.

In contrast, Figure 4 shows how the correction sizes affect the runtime and the number of planner calls, where we observe that the baseline (shown using blue circles) fails to deliver results beyond a correction size of roughly 12 within the given time constraints, whereas our method (shown in red squares) continues to succeed. While the logarithmic scale of the y-axes in both sub-figures clearly highlights the substantial improvement in runtime and number of planner calls, it is worth noting that for very small correction sizes, the baseline sometimes yields faster results than the hitting-set approach.

A more detailed breakdown of our findings is provided in Table 1. The table lists the number of solved instances per domain for each approach. For each entry, the average number of variables and the average correction size, respectively, are shown in parentheses. In addition, the table reports the average number of planner calls and the average runtime per domain for both approaches. The overview once again demonstrates the superiority of the hitting set-based approach over the baseline, regardless of the domain. An additional point of interest is the average accumulated runtime spent in the minimal hitting set solver, which is shown in parentheses alongside the total runtimes. While in most domains this overhead is almost negligible, there are cases, such as chessboard-pebbling and over-tpp, where the share of hitting set computation in the total average runtime becomes significant. However, since these are mean values, which can be heavily influenced by outliers, the observed proportions should not be overgeneralized even within individual domains. Please also note that in cases where the enumeration approach appears to perform better on average (e.g., bag-gripper), this is based on a single solved instance, whereas the hitting set-based method was able to solve 16 instances in the same domain.

Table 1: Evaluated domains with number of instances, number of solved instances with average number of variables (avg. $|V|$) and average correction size (avg. |correction|) in braces, average numbers of planner calls and average runtime per domain with average accumulated runtime for the hitting set solver in braces; best value highlighted in bold

		Instances Solved (avg. $\|V\|$, avg. |correction|)				Average Planner Calls		Average Runtimes in Seconds		
Domain	Instances	Enumeration		Hitting Sets		Enumeration	Hitting Sets	Enumeration	Hitting Sets (Solver)	
document-transfer	24	4	(16.75, 3.75)	**19**	**(24.00, 11.00)**	1880.50	**173.00**	236.42	**138.30**	(10.43)
bag-gripper	18	1	(12.00, 2.00)	**16**	**(20.25, 10.25)**	**50.00**	161.88	**375.81**	640.83	(11.00)
chessboard-pebbling	14	2	(26.50, 2.50)	**11**	**(31.00, 7.00)**	1150.00	**279.00**	391.35	**179.67**	(112.5)
over-rovers	58	8	(25.25, 2.88)	**49**	**(32.27, 9.94)**	1219.25	**196.98**	762.15	**267.92**	(65.18)
over-nomystery	29	10	(16.50, 6.50)	**18**	**(20.56, 10.56)**	815.40	**96.06**	**182.98**	271.35	(0.11)
pegsol-row5	56	7	(13.57, 5.14)	**48**	**(18.23, 6.31)**	1295.43	**128.54**	403.90	**63.55**	(31.00)
over-tpp	66	11	(18.45, 3.45)	**54**	**(26.07, 10.52)**	1646.27	**199.78**	398.41	**138.28**	(69.86)
sliding-tiles	58	0	(–, –)	**57**	**(20.00, 11.00)**	–	**93.00**	–	**45.42**	(0.05)
bottleneck	65	7	(16.57, 7.00)	**57**	**(25.00, 15.00)**	2137.86	**236.33**	522.81	**159.04**	(27.34)

6 Discussion

The area in which our procedure currently has the most potential for improvement, lies in the maximization step for new elements of κ. One could use an over-approximating algorithm for this step, as long as false positives are only generated for solvable cases [22]. Another aspect worth exploring is the structural analysis of the problem to assess the impact of specific variables on the abstractions. Some abstractions take significantly more time to solve than others, even when the size of the problem is smaller, i.e., the abstraction is coarser. This observation raises interesting questions and warrants additional investigation. It may imply the idea of finding heuristics for variable selection.

The problem instances in the Unsolvability IPC 2016 dataset, which we used in the evaluation, do not necessarily mimic modeling errors made by human designers and, to the best of our knowledge, no such real-world problem set has been collected so far. Hence, establishing a more realistic problem set for evaluating model assistance approaches would be a desirable effort. The IPC problem set rather consists of computationally demanding instances. Consequently, the planning systems often exhibit long runtimes to solve the abstracted subproblems.

To further minimize the impact of our approach to the model, it may prove beneficial to leverage existing research to achieve greater precision in identifying the root causes of unsolvability. For instance, Sreedharan et al. [23] demonstrate the incorporation of landmarks to more accurately pinpoint the causes of unsolvability. While Lin et al. [14] require a desired plan to guide the repair of a flawed planning problem, their method achieves more precise results due to their notion of atomic repairs.

Introducing variable weights and implementing a cost function to enable the search for a minimal cost solution is another interesting idea. By introducing variable weights and a cost function in our approach, we would be able to integrate the structural analysis discussed earlier. Moreover, this method would enable us to explore solutions that contain goal variables when alternative corrections cannot be found. Corrections involving goal variables would, consequently, carry higher costs. However, this would necessitate modifications to the hitting set algorithm and would require a structural analysis of the problem description in advance.

Lastly, exploring alternative notions of abstractions and associated methodologies, such as domain abstractions, CEGAR [20], and Merge-and-Shrink [12, 21], could offer valuable insights. It could further be valuable to explore similar approaches in lifted planning to achieve a closer alignment with the perspective of modelers.

7 Conclusion

This paper presented a method for identifying cardinality-minimal corrections in unsolvable SAS^+ planning problems by removing a minimal set of variables to restore solvability. We introduced two algorithmic approaches: a baseline using subset enumeration and a novel method based on hitting set duality.

Our formal analysis established the computational complexity of the problem and justified the use of hitting sets through theoretical insights. We showed that while enumeration is feasible for small corrections, it scales poorly. In contrast, our hitting set-based approach consistently outperforms enumeration in both runtime and planner calls, especially for medium and large correction sizes, across all tested domains. The underlying technique leverages recursive construction of partial conflict sets to avoid full enumeration, providing significant performance advantages.

An evaluation on a modified IPC 2016 Unsolvability dataset confirms its practical effectiveness, solving more instances and scaling beyond the limits of enumeration.

Future directions include structural analysis, integration with weight models, and extending to other abstraction formalisms. Our method offers a solid foundation for diagnosing and repairing unsolvable planning models with minimal changes.

Acknowledgements

Birte Glimm and Michael Welt acknowledge the funding of the project 2LIKE (16DHBKI001) funded by the Federal Ministry of Research, Technology and Space (BMFTR) and the Ministry for Science, Research and Arts Baden-Württemberg within the funding line Artificial Intelligence in Higher Education. 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. Alexander Lodemann acknowledges funding from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – GRK 3012 – 520750254 (kemai.uni-ulm.de).

References

[1] C. Bäckström and B. Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[2] C. Bäckström, P. Jonsson, and S. Ståhlberg. Fast detection of unsolvable planning instances using local consistency. In *Proceedings of the 6th International Symposium on Combinatorial Search (SoCS)*, pages 29–37. AAAI Press, 2013.

[3] N. K. Bavandpour, P. Lauer, S. Lin, and P. Bercher. Repairing planning domains based on lifted test plans. In *Proceedings of the 28th European Conference on Artificial Intelligence (ECAI 2025)*. IOS Press, 2025.

[4] P. Bercher, S. Sreedharan, and M. Vallati. A survey on model repair in AI planning. In *Proceedings of the 34th International Joint Conference on Artificial Intelligence (IJCAI 2025)*. ijcai.org, 2025.

[5] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[6] J. C. Culberson and J. Schaeffer. Searching with pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[7] S. Eriksson, G. Röger, and M. Helmert. Unsolvability certificates for classical planning. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS 2017)*, pages 88–79. AAAI Press, 2017.

[8] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.

[9] M. Göbelbecker, T. Keller, P. Eyerich, M. Brenner, and B. Nebel. 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)*, pages 81–88. AAAI, 2010.

[10] A. Gragera, R. Fuentetaja, A. García-Olaya, and F. Fernández. A planning approach to repair domains with incomplete action effects. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS 2023)*, pages 153–161. AAAI Press, 2023.

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

[12] R. Kreft, C. Büchner, S. Sievers, and M. Helmert. Computing domain abstractions for optimal classical planning with counterexample-guided abstraction refinement. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS 2023)*, pages 221–226. AAAI Press, 2023.

[13] S. Lin and P. Bercher. Change the world – how hard can that be? on the computational complexity of fixing planning models. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI 2021)*, pages 4152–4159. ijcai.org, 2021.

[14] S. Lin, A. Grastien, and P. Bercher. Towards automated modeling assistence: An efficient approach for repairing flawed planning domains. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI 2023)*, pages 12022–12031. AAAI Press, 2023.

[15] S. Lin, A. Grastien, R. Shome, and P. Bercher. Told you that will not work: Optimal corrections to planning domains using counter-example plans. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI 2025)*, pages 26596–26604. AAAI, 2025.

[16] A. Lindsay, S. Franco, R. Reba, and T. L. McCluskey. Refining process descriptions from execution data in hybrid planning domain models. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS 2020)*, pages 469–477. AAAI Press, 2020.

[17] C. Muise and N. Lipovetzky. Unsolvability IPC track. In *Proceedings of the 2015 Workshop on the International Planning Competition*, 2015.

[18] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[19] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2): 177–192, 1970.

[20] J. Seipp and M. Helmert. Counterexample-guided cartesian abstraction refinement for classical planning. *Journal of Artificial Intelligence Research*, 62:535–577, 2018.

[21] S. Sievers and M. Helmert. Merge-and-shrink: A compositional theory of transformations of factored transition systems. *Journal of Artificial Intelligence Research*, 71:781–883, 2021.

[22] J. Slaney. Set-theoretic duality: A fundamental feature of combinatorial optimisation. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, pages 843–848. IOS Press, 2014.

[23] S. Sreedharan, S. Srivastava, D. Smith, and S. Kambhampati. Why can't you do that HAL? explaining unsolvability of planning tasks. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pages 1422–1430. ijcai.org, 2019.

[24] M. Welt, A. Lodemann, C. Olz, P. Bercher, and B. Glimm. Experimental setup for the ECAI 2025 paper: "Calculating Optimal Corrections for Unsolvable Planning Problems", 2025. URL https://doi.org/10.5281/zenodo.16761001.

[25] Q. Yang. A theory of conflict resolution in planning. *Artificial Intelligence*, 58(1-3):361–392, 1992.