Computing Minimal Unsolvable and Maximal Solvable Abstractions of Planning Problems via Hitting Set Trees

Michael Welt,[1] **Milene Santos Teixeira,** [1] **Birte Glimm** [1]

[1] Ulm University

michael.welt@uni-ulm.de, milene.santos-teixeira@uni-ulm.de, birte.glimm@uni-ulm.de

Abstract

The unsolvability of planning problems is frequently associated with errors on the representation of the domain. Knowledge engineers often struggle to understand the underlying conditions of why a planning problem cannot be solved and which changes might be needed to reach a solution. In this work, we discuss theoretical foundations and algorithms that can be used to calculate minimal parts, called abstractions, of the planning problem that preserve the unsolvability. Additionally, our approach provides minimal sets of facts that repair the planning problem by rendering it solvable when removed. Both together can point to the source of the problem and give a clear focus on which parts of the planning specification are problematic. In particular, our proposed method can be used to debug planning problems that are unsolvable due to ill-modeled effects or over-restricted preconditions. Furthermore, we show that sets of facts inducing minimal unsolvable abstractions and repairs are related by diametrical hitting set properties, which allows us to exploit hitting set tree data structures for a goal-directed computation of these sets.

Introduction

Automated planning is a powerful approach to be employed in a range of different applications that require an initial state to be transformed into a goal state (Ghallab, Nau, and Traverso 2004). Given the representation of a domain in planning terms, here called a planning problem, an automated planner searches for a (best) path to achieve such a goal. However, for different reasons, it is possible that a solution cannot be found, e.g. the planning problem is unsolvable (Eriksson, Röger, and Helmert 2017; Bäckström, Jonsson, and Ståhlberg 2013). Frequently, unsolvable problems are associated with shortcomings in the domain representation, i.e. the knowledge engineer fails to implement some aspect that is required by further actions in order to get closer to the goal state. While planners inform the knowledge engineer about the unsolvability of the problem, typically not much insight as to what went wrong and what are possible fixes for the problem are given.

In this work, we present the theoretical foundations for the identification of minimal unsolvable and maximal solvable model abstractions of an unsolvable planning problem. While minimal unsolvable abstractions point the knowledge engineer to the core parts of the planning problem that are responsible for the unsolvability, maximal solvable model abstractions highlight parts of the planning problem that are unproblematic. Motivated by existing works that aim at explaining the unsolvability of planning problems (Herzig et al. 2014; Sreedharan et al. 2020), our method lays ground to augment and help in situations where the knowledge engineer may struggle to comprehend the underlying conditions of why a planning problem has no solution.

Our generated reductions of the input problem (i.e. generated abstractions) provide insights that may point to ill-defined effects or over-restricted preconditions, which may be responsible for the unsolvability of the planning problem at hand. Additionally, the demonstrated theoretical properties for sets that induce minimal and maximal abstractions, allow us to adapt established data structures and algorithms to enumerate minimal unsolvable abstractions, paving the way for an iterative and user-interactive debugging procedure. In this process, the knowledge engineer would be able to additionally guide the enumeration by selecting or excluding propositions for the continuation of our algorithm.

This work shows, to the best of our knowledge, for the first time, that subset minimal sets of propositions inducing unsolvable abstractions and super-set maximal sets of propositions inducing solvable abstractions are related by diametrical hitting set properties. These qualities allow us to exploit hitting set trees and respective algorithms to not only enumerate minimal unsolvable abstractions, but to calculate maximal solvable abstractions "en passant" as a by-product. This allows us to enumerate both sets of abstractions by calculating only one of them.

The remainder of this paper is organized as follows: In the next section we discuss works related to explaining unsolvability as well as works that utilize hitting sets in the realm of automated planning. Subsequently, we introduce necessary preliminaries, followed by the introduction of our theoretical propositions including the main theorem of this paper. The adapted algorithms alongside with examples of hitting set trees for minimal unsolvable abstractions are then presented. Finally, we demonstrate the inversion of the hitting set trees, followed by a brief discussion and conclusion in the last two sections of this paper.

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Related Work

Automatically fixing unsolvability of a planning problem at hand is explored on several occasions throughout the literature. The work of Yang (1992), for example, presents a theory of conflict resolution for the early identification of inconsistencies in the search process utilizing a constraint satisfaction framework. The work by Göbelbecker et al. (2010) on "good excuses" for unsolvablity focuses instead on finding a suitable and, in their words, desirably "perfect" excuse, by utilizing counterfactual explanations, such that a set of appropriate modifications of the input state eventually renders the planning problem solvable. A similar endeavor is the main concern of Herzig et al. (2014), who provide a framework based on propositional dynamic logic in order to modify the input state and, thus, find a solvable model.

More recently, with the field of explainable artificial intelligence planning (XAIP), the process and interaction with a human designer and respective feedback during the modeling phase of a planning problem came into focus. Escorting the modeling process and pointing out mistakes by the modeler raises the need for explaining the potential reasons for a failure to the model's author. The work of Sreedharan et al. (2020), for example, adopts XAIP ideas to support designers in the process of model acquisition of dialogue systems. From abstractions of a yet unsolvable model, their approach iteratively illustrates to domain designers where the plan first fails. This approach reduces the burden from the designers since they can find fixes for smaller parts of the domain, instead of debugging the whole specification while looking for a solution. However, in this approach, the designer is in charge of reflecting and finding a solution from the unsolvable point. While Sreedharan et al.'s approach also points to minimal unsolvable and maximal solvable abstractions, the finding of these sets is left as an open question. This matter is closed in our work by providing (iterative) algorithms for computing both minimal unsolvable and maximal solvable abstractions. An important difference to their proposal is that we use subset minimal and super-set maximal notions of abstractions, whereas their definitions for minimal and maximal abstractions are cardinality-based. The ideas presented in our approach are inspired by the aforementioned works. However, instead of focusing on the iterative refinement process from an initial model to a final model that matches the designer's ideas, we only focus on one step of this end-to-end process by explaining only a single failing model in an iterative and user-interactive manner. In a similar context, the work of Käser et al. (2022) describes the software "Machetli", a debugging tool that calculates a minimal planning task, that still reproduces the error, i.e. unsolvability. However, the tool only produces a single minimal planning task, which is minimal with respect to the input size and actively removes actions and predicates in order to prune the search space.

Our main propositions and respective algorithms are based on finding minimal hitting sets, a well-established research question in the communities of error diagnosis (Reiter 1987), constraint satisfaction and program verification (Bailey and Stuckey 2005), logic (Glimm and Kazakov 2019) as well as database analysis and data-mining (Gunopulos et al. 2003). The problem is also closely related to computing minimal transversals of hyper-graphs (Bailey, Manoukian, and Ramamohanarao 2003). The planning community also has explored the notion of hitting sets on various occasions. Koehler et al. (1997), for example, mention hitting sets in the introduction of an extension to their GRAPHPLAN system and also in the RIFO planning system (1999), which was used in the AIPS-98 Planning Competition (Long et al. 2000). The authors refer to the complexity of the hitting set problem to explain why generating a set of initial facts to their fact generation tree is NP-complete. Haslum, Slaney, and Thiébaux (2012), on the other hand, use the notion of minimum-cost hitting sets for disjunctive sets of action landmarks to provide an efficient way of calculating a h^+-heuristic for deletion-free planning problems. However, none of the above mentioned works covers the idea of subset minimal and super-set maximal abstractions w.r.t. the sets of the abstraction inducing facts.

Preliminaries

We base our theoretical propositions on the notion of unsolvability in classic STRIPS planning. In STRIPS planning, an automated planner is given a description of an initial state in terms of propositional variables that are assumed to hold. A goal state is also provided to the planner as a set of propositional variables. A plan is then to be computed, consisting of a sequence of actions that transform the initial state, step by step, into the goal state. The available actions to the planner are typically specified by a set of preconditions (propositional variables) that specify when the action is applicable to a state. By applying an action to a state, the state is modified by removing propositional variables listed in the action's delete effects and by adding the propositions of the action's add effects. We formally define these ideas as follows:

Definition 1. A *STRIPS planning problem*, short planning problem or planning task Π, is a quadruple $\langle V, A, I, G\rangle$. The set V is a (finite) set of propositions (facts) and we call any subset $s \subseteq V$ a *state*. The (finite) set A of *actions* is such that each action $a \in A$ is a triple $\langle pre(a), del(a), add(a)\rangle$ with $add(a) \cap del(a) = \emptyset$ and where $pre(a), del(a), add(a) \subseteq V$ denote the *preconditions*, *delete effects*, and the *add effects* of the action, respectively. The sets $I \subseteq V$ and $G \subseteq V$ describe the *initial* and the *goal* state, respectively.

We say that an action $a \in A$ is *applicable* in a state s, denoted $s \models a$, if $pre(a) \subseteq s$. The *application* of such an a to s is $appl(s, a) = (s \setminus del(a)) \cup add(a)$.

Note that the above introduced notation for a state is an abbreviation in that, strictly speaking, a state s is a function $s\colon V \to \{true, false\}$, which we choose to denote as $s = \{v \in V \mid s(v) = true\}$.

Definition 2. The *state space* of a STRIPS planning problem $\Pi = \langle V, A, I, G\rangle$ is a tuple $\langle S, A, T, I, S_G\rangle$ where the (world) states $S = 2^V$ are the subsets of V, the transitions T are $\{s \xrightarrow{a} s' \mid s \models a, s' = appl(s, a)\}$, and the *goal states* S_G are $\{s \in S \mid G \subseteq s\}$.

A sequence of actions $\vec{a} = \langle a_1, \dots, a_n\rangle$ is a *solution* for a state $s \in S$, if $s \xrightarrow{a_1} s_1, \dots, s_{n-1} \xrightarrow{a_n} s' \in T$ and $s' \in$

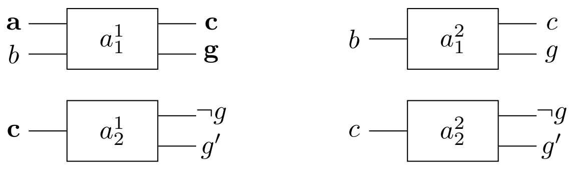

Figure 1: Sketches for Example 1, with Π_1 (left, problematic parts highlighted), and Π_2 (right), where the delete effect g is denoted as $\neg g$

S_G. A solution for I is a *plan* for Π. A planning problem $\Pi = \langle V, A, I, G\rangle$ is *solvable* if a plan for I exists and it is *unsolvable* otherwise.

Example 1. As (running) examples, consider the planning problems $\Pi_1 = \langle V_1, A_1 = \{a_1^1, a_1^2\}, I_1, G_1\rangle$ and $\Pi_2 = \langle V_2, A_2 = \{a_2^1, a_2^2\}, I_2, G_2\rangle$ (cf. Figure 1) with

$$\begin{aligned} V_1 &= \{a, b, c, g, g'\} & V_2 &= \{b, c, g, g'\} \\ a_1^1 &= \langle\{a, b\}, \emptyset, \{c, g\}\rangle & a_2^1 &= \langle\{b\}, \emptyset, \{c, g\}\rangle \\ a_1^2 &= \langle\{c\}, \{g\}, \{g'\}\rangle & a_2^2 &= \langle\{c\}, \{g\}, \{g'\}\rangle \\ I_1 &= \{b\} & I_2 &= \{b\} \\ G_1 &= \{g, g'\} & G_2 &= \{g, g'\} \end{aligned}$$

Both planning problems differ only in the proposition a, 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, i.e. $I_2 \models a_2^1$, and $s_1 = \mathit{appl}(I_2, a_2^1) = \{b, c, g\}$. One can now apply a_2^2 to get $s_2 = \mathit{appl}(s_1, a_2^2) = \{b, c, g'\}$. While this removes one of the goals, action a_2^1 is still applicable and applying it again yields the goal state $s_3 = \mathit{appl}(s_2, a_2^1) = \{b, c, g, g'\}$. Hence, $\langle a_2^1, a_2^2, a_2^1\rangle$ is a solution for Π_2.

In the remainder, it will be convenient to restrict planning problems to certain subsets of facts, which are or should be in the focus of the knowledge engineer. Hence, we introduce the notion of abstractions of planning tasks:

Definition 3. Let $\Pi = \langle V, A, I, G\rangle$ be a planning problem. For $P \subseteq V$, the *abstraction* of Π w.r.t. the *projection* P, denoted $\Pi|_P$, is the planning problem $\langle P, A', I \cap P, G \cap P\rangle$ with $A' = \{\langle \mathit{pre}(a) \cap P, \mathit{del}(a) \cap P, \mathit{add}(a) \cap P\rangle \mid \langle \mathit{pre}(a), \mathit{del}(a), \mathit{add}(a)\rangle \in A\}$. We say that the abstraction *respects the goal* if $G \subseteq P$.

We introduce the notion of goal-respecting abstractions since preserving goal states might sometimes be important to knowledge engineers, although it is not required for our method.

Example 2. Considering again our planning problems Π_1 and Π_2 from Example 1. We see that the projection $\{b, c, g, g'\}$ w.r.t. Π_1 yields exactly $\Pi_2 = \Pi_1|_{\{b,c,g,g'\}}$, i.e. Π_2 abstracts away the proposition a. Note again that while Π_1 is unsolvable, its abstraction Π_2 is solvable.

Hitting Set Properties of Solvable and Unsolvable Abstractions

As previously mentioned, debugging an unsolvable planning problem without tool support can be very hard. In order to support a knowledge engineer with this task, we now present our approach of computing minimal unsolvable and maximal solvable abstractions for an unsolvable planning problem. While, on the one hand, the minimal unsolvable abstractions provide the core of the planning problem that is unsolvable, the maximal solvable abstractions, on the other hand, highlight which parts of the planning problem are "free of problem", i.e. solvable as desired.

Definition 4. Let $\Pi = \langle V, A, I, G\rangle$ be a planning problem and $P \subseteq V$ a projection. We call $\Pi|_P$ a *solvable (unsolvable) abstraction* if $\Pi|_P$ is solvable (unsolvable). An unsolvable abstraction $\Pi|_P$ is a *minimal unsolvable abstraction*, if, for each proper subset P' of P, $\Pi|_{P'}$ is solvable. A solvable abstraction $\Pi|_P$ is a *maximal solvable abstraction* if, for each proper super-set P' of P, $\Pi|_{P'}$ is unsolvable.

Note that for finding a minimal unsolvable abstraction, we can minimize the projection P. We can, however, equivalently phrase the problem of finding maximal solvable abstractions as a minimization problem (which we find convenient in the remainder) by finding a solvable abstraction $\Pi|_{V\setminus P}$ such that P is minimal, i.e. for any $P' \subsetneq P$, $\Pi|_{V\setminus P'}$ is unsolvable. We formalize this complementary notion of abstractions as follows:

Definition 5. Let $\Pi = \langle V, A, I, G\rangle$ be a planning problem. For a projection $P \subseteq V$, we call $\Pi|_{V\setminus P}$ the *complement abstraction* of Π w.r.t. P and V. If the complement abstraction $\Pi|_{V\setminus P}$ is a maximal solvable abstraction, we simply refer to $\Pi|_{V\setminus P}$ as *maximal solvable complement abstraction*. A projection P that induces a (maximal) solvable complement abstraction $\Pi|_{V\setminus P}$ is also called a (minimal) *repair* for Π.

Following the notion of explaining why a planning problem is unsolvable, the intuition behind repairs is that such sets tell us which facts need to be removed from the set of facts in order to obtain a (maximal) solvable abstraction. In that sense the notion of a repair seems to be justifiable.

If no confusion is likely to arise, we also just refer to a complement abstraction $\Pi|_{V\setminus P}$ without saying that it is the complement abstraction *of* Π *w.r.t.* P.

Example 3. Consider again the planning problem Π_1 from Example 1, which overall has 32 abstractions for all combinations of propositions in V (including V and $\emptyset$). Table 1 shows Π_1 and its abstractions that are minimal unsolvable or maximal solvable including $\Pi_2 = \Pi_1|_{\{b,c,g,g'\}} = \Pi_1|_{V\setminus\{a\}}$ in row (2), which is a maximal solvable abstraction. That is, $\{a\}$ is a minimal repair for Π_1. Since also $\Pi_1|_{V\setminus\{c,g\}}$ in row (4) and $\Pi_1|_{V\setminus\{g,g'\}}$ in row (5) are maximal solvable abstractions, we have that also $\{c, g\}$ and $\{g, g'\}$ are minimal repairs for Π_1.

Since the repair $\{g, g'\}$ drops all goals, it is unlikely a desired repair. The repair $\{c, g\}$ still drops one goal, but it might be taken to indicate some problems with the effects of a_1^1. The repair $\{a\}$, however, makes it clear that a is causing

		V	A	I	G	solvability
(1) $\Pi_1\vert_{\{a,b,c,g,g'\}}$	$=\Pi_1\vert_{V\setminus\emptyset}$	$=\langle\{a,b,c,g,g'\},$	$\{\langle\{a,b\},\emptyset,\{c,g\}\rangle,\langle\{c\},\{g\},\{g'\}\rangle\},$	$\{b\},$	$\{g,g'\}\rangle$	unsolvable
(2) $\Pi_1\vert_{\{b,c,g,g'\}}$	$=\Pi_1\vert_{V\setminus\{a\}}$	$=\langle\{b,c,g,g'\},$	$\{\langle\{b\},\emptyset,\{c,g\}\rangle,\langle\{c\},\{g\},\{g'\}\rangle\},$	$\{b\},$	$\{g,g'\}\rangle$	max. solvable
(3) $\Pi_1\vert_{\{a,c,g'\}}$	$=\Pi_1\vert_{V\setminus\{b,g\}}$	$=\langle\{a,c,g'\},$	$\{\langle\{a\},\emptyset,\{c\}\rangle,\langle\{c\},\emptyset,\{g'\}\rangle\},$	$\emptyset,$	$\{g'\}\rangle$	min. unsolvable
(4) $\Pi_1\vert_{\{a,b,g'\}}$	$=\Pi_1\vert_{V\setminus\{c,g\}}$	$=\langle\{a,b,g'\},$	$\{\langle\{a,b\},\emptyset,\emptyset\rangle,\langle\emptyset,\emptyset,\{g'\}\rangle\},$	$\{b\},$	$\{g'\}\rangle$	max. solvable
(5) $\Pi_1\vert_{\{a,b,c\}}$	$=\Pi_1\vert_{V\setminus\{g,g'\}}$	$=\langle\{a,b,c\},$	$\{\langle\{a,b\},\emptyset,\{c\}\rangle,\langle\{c\},\emptyset,\emptyset\rangle\},$	$\{b\},$	$\emptyset\rangle$	(trivially) max. solvable
(6) $\Pi_1\vert_{\{a,g\}}$	$=\Pi_1\vert_{V\setminus\{b,c,g'\}}$	$=\langle\{a,g\},$	$\{\langle\{a\},\emptyset,\{g\}\rangle,\langle\emptyset,\{g\},\emptyset\rangle\},$	$\emptyset,$	$\{g\}\rangle$	min. unsolvable

Table 1: Planning problem Π_1 from Example 1 with its minimal unsolvable and maximal solvable abstractions

a problem. Either a is used as an over-restrictive precondition for a_1^1 or is missing in the initial state. Which fix is the preferred one (add a to the initial state or remove a as precondition) is to be decided by the knowledge engineer according to the domain requirements.

While the repairs point to the source of the problem, a knowledge engineer can further look at the minimal unsolvable abstractions as they highlight the core parts that are responsible for the unsolvability. For example, by looking at $\Pi|_{\{}a, g\}$ in row (6), one can clearly see that the planning problem for just a and the single goal g is unsolvable. Assuming that one wants to stick to the goal g, this additionally indicates the problematic use of the proposition a. This is further substantiated by looking at the other minimal unsolvable abstraction $\Pi|_{\{}a, c, g'\}$ in row (3), which shows that also the goal g' cannot be reached in the presence of a and c.

The core idea of our approach hinges on the monotonicity w.r.t. solvability under the removal of propositions, i.e. by abstracting propositions away from a solvable planning problem, we always again get a solvable planning problem:

Lemma 1. Let $\Pi = \langle V, A, I, G\rangle$ be a planning problem such that $P' \subseteq P \subseteq V$. If $\Pi|_P$ is solvable, then $\Pi|_{P'}$ is solvable.

Proof. Since $\Pi|_P$ is solvable, there is a solution $\langle a_1, \dots, a_n\rangle$ such that the sequence $I \cap P = s_0 \xrightarrow{a_1} s_1, \dots, s_{n-1} \xrightarrow{a_n} s_n \in T$ and s_n is a goal state, i.e. $s_n \in S_{G\cap P}$. We consider two cases: (i) $\Pi_{P'}$ is not goal respecting and (ii) $\Pi_{P'}$ is goal respecting.

For case (i), we have $G \cap P' = \emptyset$ and, hence, $\Pi_{P'}$ is trivially solvable.

For case (ii), we have $G \cap P' \neq \emptyset$. We construct $\langle a_1', \dots, a_n'\rangle$ by setting, for $1 \leq i \leq n$, $a_i' = \langle pre(a_i) \cap P', del(a_i) \cap P', add(a_i) \cap P'\rangle$ and show that $\langle a_1', \dots, a_n'\rangle$ is a solution for $\Pi|_{P'}$, i.e. there is a transition $I \cap P' = s_0' \xrightarrow{a_1'} s_1', \dots, s_{n-1}' \xrightarrow{a_n'} s_n'$ and $s_n' \in S_{G\cap P'}$. First, we have $s_0' = s_0 \cap P'$ since $P' \subseteq P$ and, hence, $s_0 \cap P' = I \cap P \cap P' = I \cap P'$. We have that applicability is preserved for states under set intersection, i.e. we have that $pre(a_i) \subseteq s_i$ implies $pre(a_i) \cap P' \subseteq s_i \cap P'$. In particular, a_1' is applicable to the initial state $s_0' = s_0 \cap P'$ of $\Pi|_{P'}$. We next show that $s_i' = s_i \cap P'$ also for $1 \leq i \leq n$: Since $s_i = (s_{i-1} \setminus del(a_i)) \cup add(a_i)$, we have $s_i \cap P' = ((s_{i-1} \setminus del(a_i)) \cup add(a_i)) \cap P' = ((s_{i-1} \setminus del(a_i)) \cap P') \cup (add(a_i) \cap P') = ((s_{i-1} \cap P') \setminus (del(a_i) \cap P')) \cup add(a_i') = (s_{i-1}' \setminus del(a_i')) \cup add(a_i') = s_i'$. Hence, $s_n' = s_n \cap P'$ and, since $G \cap P' \neq \emptyset$ and $G \cap s_n \neq \emptyset$, we have $s_n' \in S_{G\cap P'}$. $\square$

Sets inducing minimal unsolvable abstractions and repairs can better support the knowledge engineer if the two sets are somehow connected such that comparing the minimal unsolvable abstractions and the repairs (or the induced maximal solvable complement abstractions) can give insights into where exactly the problem is. The following theorem shows that this is indeed the case:

Theorem 1. Let $\Pi = \langle V, A, I, G\rangle$ be an unsolvable planning problem. For each $P_u, P_s \subseteq V$ such that $\Pi|_{P_u}$ is a minimal unsolvable abstraction of Π and $\Pi|_{V\setminus P_s}$ is a maximal solvable complement abstraction of Π, we have $P_u \cap P_s \neq \emptyset$.

Proof. To the contrary of what is to be shown, assume that P_u, P_s exist such that $P_u \cap P_s = \emptyset$. This implies $P_u \subseteq V \setminus P_s$. Given that $\Pi|_{V\setminus P_s}$ is solvable by assumption and since $P_u \subseteq V \setminus P_s$, Lemma 1 gives us that $\Pi|_{P_u}$ is solvable, which gives us the desired contradiction to our assumption that $\Pi|_{P_u}$ is unsolvable. $\square$

Based on Theorem 1, we next show that sets inducing minimal unsolvable abstractions and repairs are even further related, which allows us to come up with an incremental algorithm for computing these sets based on Reiter's minimal hitting set algorithm (Reiter 1987). For this, we show that every repair is a minimal hitting set for the set that contains all sets that induce minimal unsolvable abstractions, where a hitting set is defined as follows:

Definition 6. Let $\mathcal{P}$ be a set of sets of some elements. A set H is a *hitting set* for $\mathcal{P}$ if $H \cap P \neq \emptyset$ for each $P \in \mathcal{P}$. A hitting set H for $\mathcal{P}$ is *minimal* if every $H' \subsetneq H$ is not a hitting set for $\mathcal{P}$.

Lemma 2. Let $\Pi = \langle V, A, I, G\rangle$ be an unsolvable planning problem, $\mathcal{U}$ the set of projections that induce minimal unsolvable abstractions, and $\mathcal{S}$ the set of projections that induce maximal solvable complement abstractions of Π. Then each $P_s \in \mathcal{S}$ is a minimal hitting set for $\mathcal{U}$.

Proof. The desired hitting set property follows directly from Theorem 1. To show that elements of $\mathcal{S}$ are indeed minimal hitting sets for $\mathcal{U}$, let $P_s \in \mathcal{S}$. Furthermore, let $P'_s \subsetneq P_s$ be a proper subset of P_s. It necessarily follows that $\Pi|_{V \setminus P'_s}$ is unsolvable, since $\Pi|_{V \setminus P_s}$ is already maximal solvable. Hence, a set $P_u \subseteq V \setminus P'_s$ exists, such that $P_u \in \mathcal{U}$. It is now easy to see, that $P_u \cap P'_s = \emptyset$ must hold, which violates the hitting set property for P'_s, and therefore P_s is indeed a minimal hitting set for the set $\mathcal{U}$. □

The next lemma shows that the hitting set property is in fact diametrical, such that every set inducing a minimal unsolvable abstraction is also a minimal hitting set for the set that contains all repairs.

Lemma 3. Let $\Pi = \langle V, A, I, G \rangle$ be an unsolvable planning problem, $\mathcal{U}$ the set of projections that induce minimal unsolvable abstractions, and $\mathcal{S}$ the set of projections that induce maximal solvable complement abstractions of Π. Then each $P_u \in \mathcal{U}$ is a minimal hitting set for $\mathcal{S}$.

The proof of Lemma 3 is analogous to that of Lemma 2.

Computing Minimal Solvable and Maximal Unsolvable Abstractions via Hitting Set Trees

With Algorithm 1, we introduce a first and naive way of calculating a single projection that induces a minimal unsolvable abstraction of an unsolvable planning problem $\Pi = \langle V, A, I, G \rangle$. The procedure starts by initializing the set P_u with V. It is then checked, for each proposition $v \in V$, whether $\Pi|_{P_u \setminus \{v\}}$ is still unsolvable. If that is the case, v is removed from P_u as $P_u \setminus v$ gives us a smaller and, eventually, a minimal projection that induces an unsolvable abstraction.

Lemma 4. Let $\Pi = \langle V, A, I, G \rangle$ be an unsolvable planning problem. Given Π as input, the output P_u of Algorithm 1 is such that $\Pi|_{P_u}$ is a minimal unsolvable abstraction.

Proof Sketch 1. Since Π is unsolvable by assumption, P_u is initialized such that $\Pi|_{P_u}$ is unsolvable. Since we only remove a proposition v from P_u, if the unsolvability of $\Pi|_{P_u \setminus \{v\}}$ is still given in Line 4, the unsolvability of the abstraction is preserved. We can further show, that P_u is indeed minimal, since every proposition, that is unnecessary for $\Pi|_{P_u}$ and its induced abstraction to remain unsolvable, is removed consecutively until each further removal would turn the abstraction $\Pi|_{P_u}$ into a solvable abstraction, where the required monotonicity follows from the contra-positive form of Lemma 1.

Algorithm 1 returns only a single projection that induces a minimal unsolvable abstraction. The resulting set depends on the sequence of propositions in the initial P_u, i.e. on the sequence of the input V. However, the algorithm can implicitly be used to calculate all desired projections that induce a minimal unsolvable abstraction by permutating the input V exhaustively:

Lemma 5. For each set P_u that induces a minimal unsolvable abstraction of an unsolvable planning problem $\Pi = \langle V, A, I, G \rangle$, there exists some order of propositions in V such that Algorithm 1 returns P_u for the input $\Pi = \langle V, A, I, G \rangle$.

Algorithm 1: Finding a single minimal unsolvable abstraction

MinUnsolvAbstr compute a single P_u s.t. $\Pi|_{P_u}$ is a minimal unsolvable abstraction of Π
Input: $\Pi = \langle V, A, I, G \rangle$: an unsolvable planning problem
Output: P_u: a projection s.t. $\Pi|_{P_u}$ is a minimal unsolvable abstraction of Π

```
1: $P_u \leftarrow V$;
2: for $v \leftarrow V$ do
3:     if $\Pi|_{P_u \setminus \{v\}}$ is unsolvable then
4:         $P_u \leftarrow P_u \setminus \{v\}$;
5:     end if
6: end for
7: return $P_u$;
```

Proof Sketch 2. Assume, to the contrary of what is to be shown, that a set P_u inducing a minimal unsolvable abstraction $\Pi|_{P_u}$ of $\Pi = \langle V, A, I, G \rangle$, is not returned by Algorithm 1. We can show that by processing first propositions in $V \setminus P_u$ and then propositions from P_u leads to a contradiction as desired.

Instead of exhaustively trying each possible permutation of input facts, it is possible to exploit that after finding a projection P_1 inducing a minimal unsolvable abstraction, it is known that the next projection P_2 inducing a minimal unsolvable abstraction (assuming its existence) differs in at least one element from P_1. Otherwise P_2 would be equal to P_1 (or smaller than P_1, which contradicts the assumption that P_1 is minimal). This observation facilitates the use of Algorithm 1 in a more goal-directed way to produce another solution based on an already known solution P_1, by selecting an arbitrary fact $v \in P_1$ and alter the input for the subsequent run of Algorithm 1 to $\Pi|_{V \setminus \{v\}}$. This idea can be continued inductively yielding a tree data structure.

Example 4. Consider again the planning problem Π_1 from Example 1. If we assume a lexicographic ordering of the propositions, the initial run of Algorithm 1 results in the projection $\{a, c, g'\}$, inducing a minimal unsolvable abstraction. We can take this set to form the root node n_0 of a tree structure as depicted in Figure 2. For each proposition of this set, a branch is created. Note that the node n_1 ($\Pi_1|_{V \setminus \{a\}}$) equals the problem Π_2 from Example 1, which is known to be solvable. Thus, as it cannot be rendered unsolvable by minimization (Lemma 1), the node is marked with the label $\perp$, without the need for running Algorithm 1, rendering n_1 as a leaf-node of the tree. The labels for the subsequent nodes, n_2 and n_3, are calculated by running Algorithm 1 with $\Pi_1|_{V \setminus \{c\}}$ and $\Pi_1|_{V \setminus \{g'\}}$, respectively, as input. Both are still unsolvable sub-problems and yield the labels for their following sub-nodes. The procedure is then repeated for each child node of n_2 and n_3. In the case of node n_2, branches are then created for a and g, where the branch for a leads to the sub-problem $\Pi_1|_{V \setminus \{c,a\}}$. By looking at the path back to the root, each branch corresponds to one proposition that is taken out of the initial set of facts V of the original problem Π_1. It happens that this sub-problem is also already solvable, preventing the run of Algorithm 1 for the node,

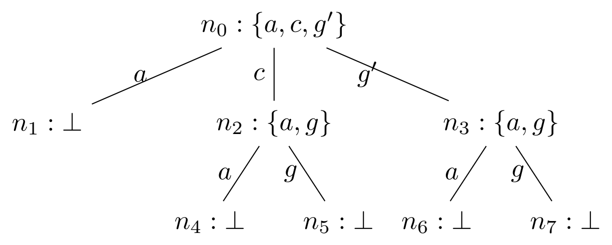

Figure 2: Hitting set tree from Example 4

and which is again represented by marking the node n_4 with $\bot$. The same holds for the sibling node n_5 which is also marked with $\bot$. The rest of the tree is expanded and evaluated in the same way, but no further minimal unsolvable abstractions can be found. Therefore, the nodes n_6 and n_7 are also marked with $\bot$, and, no further calls to Algorithm 1 are required. By comparing the labels of the non-leaf nodes to rows (3) and (6) of Table 1, we can see that this procedure finds every set that induces a minimal unsolvable abstraction, although the set $\{a, g\}$ is found twice. By joining into a set the edge labels of the paths from each leaf node to the root, we obtain valid repairs for Π_1 (i.e. $\{a\}$, $\{a, c\}$, $\{g, c\}$, $\{a, g'\}$, and $\{g, g'\}$).

In fact, every path from a leaf node to the root node in Figure 2 denotes a valid repair, although not necessarily minimal. If calculated exhaustively, the tree will contain all minimal repairs of the problem (Lemma 7), which, by our Lemma 2, are minimal hitting-sets for the set of all sets inducing a minimal solvable abstraction. Hence, the tree data structure, created in Example 4 and depicted in Figure 2, is also denoted as a hitting set tree: all sets along each path from an arbitrary leaf node to the root represent hitting sets for the set of non-leaf node labels. We now provide a formal definition for this data structure:

Definition 7. Let $\Pi = \langle V, A, I, G\rangle$ be a planning problem. A *hitting set tree* for minimal unsolvable abstractions for Π is a labeled tree $T = \langle N, E, \mathcal{L}, \mathcal{E}\rangle$, with $N \neq \emptyset$ denoting a set of nodes, E referring to a set of edges $\langle n, m\rangle$ with $n, m \in N$ and $\mathcal{L}\colon N \to 2^V \cup \{\bot\}$ assigns to each node $n \in N$ a sub-set of facts from V or the special symbol $\bot$, and $\mathcal{E}\colon N \times N \to V$ assigns to each edge $e \in E$ a fact $v \in V$, such that:

1. each non-leaf node $n \in N$ is labeled with a set $\mathcal{L}(n) \subseteq V$, such that $\Pi|_{\mathcal{L}(n)}$ is a minimal unsolvable abstraction and, for each $v \in \mathcal{L}(n)$, n has an outgoing edge $\langle n, m\rangle \in E$ with $\mathcal{E}(n, m) = v$.
2. each leaf node $n \in N$ is labeled by $\mathcal{L}(n) = \bot$.

For each $n \in N$, let $H(n)$ be the set of labels appearing on the path from n to the root node of T. Then the following properties should additionally hold:

3. for each non-leaf node $n \in N$, we have $\mathcal{L}(n) \cap H(n) = \emptyset$, and
4. for each leaf node $\ell \in N$, we have that $H(\ell)$ denotes a valid repair for Π.

We are showing, that such a data structure indeed will contain all sets inducing minimal unsolvable abstractions in the nodes, and all minimal repairs by joining the paths from the leaf nodes to the roots:

Lemma 6. Let $T = \langle N, E, \mathcal{L}, \mathcal{E}\rangle$ be a hitting set tree for minimal unsolvable abstractions for a planning problem $\Pi = \langle V, A, I, G\rangle$. Then, for each projection P_u that induces a minimal unsolvable abstraction $\Pi|_{P_u}$, there exists a node $n \in N$ such that $\mathcal{L}(n) = P_u$.

Proof. Let $n \in N$ be a node with a maximal (w.r.t. set inclusion) set $H(n)$ (cf. Definition 7) such that $H(n) \cap P_u = \emptyset$, i.e. for every other node $m \in N$ either $H(m) \subseteq H(n)$ or $H(m) \cap P_u \neq \emptyset$. We prove that $\mathcal{L}(n) = P_u$.

Since $H(n) \cap P_u = \emptyset$ and $P_u \subseteq V$, we have $P_u \subseteq V \setminus H(n)$. Since $\Pi|_{P_u}$ is unsolvable, we obtain, by contrapositive form of Lemma 1, $\Pi|_{V\setminus H(n)}$ is unsolvable. Therefore, by Condition 4 of Definition 7, n cannot be a leaf node. Hence, $\mathcal{L}(n) = {P_u}'$ for some set ${P_u}'$ inducing a minimal unsolvable abstraction $\Pi|_{{P_u}'}$. If $P_u = {P_u}'$ we have proved what is required. Otherwise, since P_u is already subset minimal w.r.t. inducing a minimal unsolvable abstraction and ${P_u}'$ also induces an unsolvable abstraction, we have ${P_u}' \not\subseteq P_u$. Hence, there exists some $v \in {P_u}' \setminus P_u$. By Condition 1 of Definition 7, there exists $\langle n, m\rangle \in E$ with $\mathcal{E}(n, m) = v$. Furthermore, by Condition 3 of Definition 7, ${P_u}' \cap H(n) = \emptyset$. Hence, $v \notin H(n)$ since $v \in {P_u}'$. Hence, $H(m) = H(n) \cap \{v\} \not\subseteq H(n)$ and, since $v \notin P_u$ and $H(n) \cap P_u = \emptyset$, we have $H(m) \cap P_u = \emptyset$. This contradicts our assumption that $H(n)$ is a maximal set such that $H(n) \cap P_u = \emptyset$. This contradiction proves that $\mathcal{L}(n) = P_u$ is the only possible case. $\square$

Lemma 7. Let $T = \langle N, E, \mathcal{L}, \mathcal{E}\rangle$ be a hitting set tree for minimal unsolvable abstractions for an unsolvable planning problem $\Pi = \langle V, A, I, G\rangle$. Then, for each set P_s that induces a maximal solvable complement abstraction $\Pi|_{V\setminus P_s}$, there exists a leaf node $\ell \in N$ such that $H(\ell) = P_s$.

Proof Sketch 3. We can demonstrate with Theorem 1 and Definition 7 that, for a leaf node $\ell \in N$, it holds that, if $H(\ell)$ is maximal (w.r.t. set inclusion) and $H(\ell) \subseteq P_s$ then it follows that $H(\ell) = P_s$.

As already demonstrated in Example 4, some node labels may occur more than once in a hitting set tree. The following lemma addresses the question of the maximal number of nodes that can occur in a hitting set tree w.r.t. to the size of the initial planning problem:

Lemma 8. Every hitting set tree for minimal unsolvable abstractions for an unsolvable planning problem $\Pi = \langle V, A, I, G\rangle$ has at most $\sum_{0\leq k\leq n} n^k$ nodes, where n is the number of facts in V.

Proof Sketch 4. By analyzing Conditions 1 and 3 of Definition 7, we can show that both the depth and the branching factor of T is bounded by $|V|$, which gives the desired bound.

After introducing the idea of hitting set trees for our problem at hand, we can now provide an appropriate algorithm that constructs such a data structure.

Algorithm 2: Finding all minimal unsolvable abstractions

MinUnsolvAbstrHST compute $\mathcal{U}$ s.t. $\Pi|_{P_u}$ is a minimal unsolvable abstraction of Π
Input: $\Pi = \langle V, A, I, G\rangle$: a planning problem (solvable or unsolvable)
Output: $\mathcal{U}$: for each $P_u \in \mathcal{U}$, $\Pi|_{P_u}$ is a minimal unsolvable abstraction of Π and $\mathcal{U}$ is maximal

```
1: $\mathcal{U} \leftarrow \emptyset$;
2: $Q \leftarrow \{\emptyset\}$;
3: while $Q \neq \emptyset$ do
4:    $H \leftarrow$ choose $H \in Q$;
5:    $Q \leftarrow Q \setminus \{H\}$;
6:    if $\Pi|_{V \setminus H}$ is unsolvable then
7:       $P_u \leftarrow$ MinUnsolvAbstr($\Pi|_{V \setminus H}$);
8:       $\mathcal{U} \leftarrow \mathcal{U} \cup \{P_u\}$;
9:       for $v \in P_u$ do
10:         $Q \leftarrow Q \cup \{H \cup \{v\}\}$;
11:      end for
12:   end if
13: end while
14: return $\mathcal{U}$;
```

Algorithm 2 starts by initializing an empty result set $\mathcal{U}$ that will hold all sets inducing a minimal unsolvable abstraction for the input $\Pi = \langle V, A, I, G\rangle$ eventually (provided such abstractions exist, i.e. the input planning problem is unsolvable). Additionally, a queue Q, that holds the unprocessed sub-paths of the tree, is initialized with a single element – the empty set.

The algorithm then starts by taking out elements of the queue as long as Q is not empty. The first element in Q, assigned to H in the subsequent lines, is now the empty set. In Line 6, it is tested if the reduced set of facts induces an unsolvable planning problem. For $H = \emptyset$, this denotes our input problem Π. If Π is solvable, the algorithm terminates and returns the empty set. If Π is unsolvable, this calculates the root node of the hitting set tree by calling Algorithm 1. Additionally, the result is added to the overall result set $\mathcal{U}$ in Line 8.

The algorithm then proceeds to put a new unprocessed path into the queue Q for each fact v of the last set P_u known to induce a minimal unsolvable abstraction for Π. The newly added path is calculated by adding v to the currently processed path H, yielding a new path deeper down the tree. The tree is traversed in a breadth-first manner if Q is processed based on first-in-first-out principle. Processing Q in a last-in-first-out principle, on the other hand, would result in a depth-first traversal of the tree.

Continuing the while-loop, the next unprocessed path is taken out from Q, which is the first fact v from the last P_u, such that the next to be tested sub-problem would be the abstraction induced by $V \setminus \{v\}$. This addition corresponds to the intuition as provided prior to Example 4, that the next result set has to differ in at least one proposition from the previously calculated set.

As new paths are added only if the sub-problem $\Pi|_{V \setminus H}$ is still unsolvable, the algorithm terminates eventually, since it will trivially stop in worst case when H equals V in each path of the tree, such that $V \setminus H$ becomes empty. This yields a trivially solvable planning problem, and, thus, stops the algorithm from putting further sup-paths into Q.

Algorithm 3: Finding a single repair

MaxSolvAbstr compute a single P_s s.t. $\Pi|_{V \setminus P_s}$ is a maximal solvable complement abstraction of Π
Input: $\Pi = \langle V, A, I, G\rangle$: a solvable planning problem
Output: P_s: a projection s.t. $\Pi|_{V \setminus P_s}$ is a maximal solvable complement abstraction of Π

```
1: $P_s \leftarrow V$;
2: for $v \leftarrow V$ do
3:    if $\Pi|_{V \setminus (P_s \setminus \{v\})}$ is solvable then
4:       $P_s \leftarrow P_s \setminus \{v\}$;
5:    end if
6: end for
7: return $P_s$;
```

Our approach is able to calculate minimal unsolvable abstractions and valid repairs at once. By examining both simultaneously, the knowledge engineer can better understand the potential shortcomings of the unsolvable model. Furthermore, the iterative procedure of Algorithm 2 enables an interactive debugging scenario. That is, the algorithm could continuously provide minimal unsolvable abstractions of the planning problem to the knowledge engineer, who then might be able to identify ill-defined parts of the planning problem. This strategy has the potential to reduce the need for exhaustively calculating the whole tree structure.

As already pointed out, the repairs denoted by the leaf nodes in our approach are not necessarily minimal, albeit all minimal repairs will be contained in some path of the tree eventually, as stated by Lemma 7. However, minimal repairs can be more intuitive in describing potentially problematic parts of the planning problem to the knowledge engineer. Thus, being able to calculate them without exhaustively calculating the whole tree is necessary. Algorithm 3, which represents a modified version of Algorithm 1, provides a way to obtain a single minimal repair. The procedure minimizes a set P_s, which is initialized with the facts V. The testing condition of Algorithm 3 is inverted, in comparison to Algorithm 1, to test for solvability of an induced complement abstraction instead. To minimize a repair P_s indicated by some currently processed leaf node, the abstraction $\Pi|_{P_s}$ could be inserted in Algorithm 3 to obtain the minimal repair.

Inverting Hitting Set Trees

The presented Algorithms 1 and 2 use a planning system on several occasions as a black-box for unsolvability testing. However, the planning community has shown more interest in optimizing automated planning systems towards finding plans, i.e. showing solvability, rather than showing unsolvability.[1] Hence, it could be beneficial to take advantage of the shown hitting set duality and invert the hitting set tree

[1]As of now, only the 2016 International Planning Competition was concerned with showing unsolvability.

$$\begin{tikzcd}[column sep=small] & & n_0 : \{a\} \arrow[d, dash, "a"'] & \\ & & n_1 : \{c, g\} \arrow[dl, dash, "c"'] \arrow[dr, dash, "g"] & \\ & n_2 : \{g, g'\} \arrow[dl, dash, "g"'] \arrow[dr, dash, "g'"] & & n_3 : \bot \\ n_4 : \bot & & n_5 : \bot & \end{tikzcd}$$

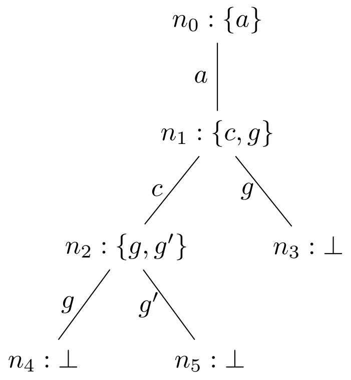

Figure 3: Inverted hitting set tree from Example 4

and the respective minimization in order to directly search for minimal repairs instead. The inverted tree, as depicted in Figure 3, contains the minimal repairs of Example 4 as node labels, whereas the paths from the leaf nodes to the root now denote sets inducing unsolvable abstractions. As already indicated, Algorithm 3 is able to calculate minimal repairs for the node labels. This procedure can then be used in conjunction with Algorithm 2 to calculate the inverted tree, by changing the sub-problem creation and testing condition in Line 6 of Algorithm 2 accordingly. Our theoretical results would then assure that such an inverted hitting set tree would provide sets inducing minimal repairs in the nodes and sets inducing solvable abstractions in the conjunction of the path labels from leaf nodes to the root. In the example depicted in Figure 3, all paths do contain minimal sets inducing unsolvable abstractions, this is, however, not necessarily the case. Further minimization of the sets indicated by the leaf nodes can be achieved with Algorithm 2.

Discussion

Besides presenting minimal reductions of the planning problem at hand, our approach also provides potential repairs. These repairs are only meant to guide the knowledge engineer in locating the source of the problem. In that sense, the presented methods are not suited for automatic repair because the intention of the knowledge engineer is unknown.

Modeling a planning problem is an incremental process (cf. Sreedharan et al. (2020)), during which the model may be rendered unsolvable after a certain change. In that sense, restricting the debugging process to the latest changes would certainly speed up the whole process. One way to account for that, would be to extend the presented approach to respect a certain signature of facts that are included in the search process. Alternatively, the knowledge engineer can additionally guide the already goal-oriented algorithm by pointing out more or less interesting propositions to cover next. If, for example, the knowledge engineer is certain about how a given action should be defined, some propositions can be ruled out in advance to the next step of the algorithm and, thus, guide the algorithm for what subsets of facts to cover next.

Exploring more properties of sets inducing minimal unsolvable abstractions and maximal solvable complement abstractions, w.r.t. their explainability potential, would certainly be beneficial. It could be interesting, for example, to rank propositions by their occurrence in those sets, or in the respective non-empty intersections.

After discussing possible extensions of the explanatory potential of our approach, we discus some technical optimizations: Lemma 8 demonstrates that the number of nodes of a hitting set tree can be quite high in general. Hence, optimizations for calculating hitting set trees more effectively have been described in the literature, such as pruning methods or reducing memory consumption of the procedure by only keeping one branch in memory at a time. Additionally, establishing some kind of memoization technique to enable the reuse of results produced by already processed input sets H is an alternative to reduce the number of expensive planner calls. Calling Algorithm 1 after a positive condition testing in Line 6 of Algorithm 2 leads, for example, in all cases to a second call of the planner with an already established result.

Another possible extension to directly reduce the costs of the necessary planner calls could include calculating the cost of the respective delete relaxed sub-problem first. This way, testing the unrelaxed sub-problem for unsolvability would only be necessary if the relaxed problem proves to be solvable.

Conclusion

In this paper, we showed how minimal unsolvable and maximal solvable abstractions can aid the debugging of an unsolvable planning problem. The projections that induce such (complement) abstractions can be used, for example, to detect ill-defined effects or over-restricted preconditions of actions or errors in the initial or goal state that potentially cause the unsolvability of a planning problem at hand. We demonstrated the diametrical nature of the hitting set properties between projections inducing minimal unsolvable abstractions and projections inducing maximal solvable complement abstractions. Subsequently, we introduced algorithms, based on hitting set trees, that utilize the demonstrated theoretical properties to enumerate projections inducing minimal unsolvable abstractions. The dual nature of our approach enables the calculation of the maximal solvable complement abstractions “en passant” as a by-product of the former enumeration. We closed by sketching how these algorithms can be used in an interactive debugging scenario, where a system can present minimal unsolvable abstractions and suggested repairs to knowledge engineers confronted with an unsolvable planning task. Finally, we discussed possible optimizations to our algorithms as well as further research to be conducted in order to explore the full potential for explainability of minimal unsolvable and maximal solvable abstractions, respectively.

References

Bäckström, C.; Jonsson, P.; and Ståhlberg, S. 2013. Fast Detection of Unsolvable Planning Instances Using Local Consistency. In Helmert, M.; and Röger, G., eds., *Proceedings of the 6th Annual Symposium on Combinatorial Search (SOCS 2013)*. AAAI Press.

Bailey, J.; Manoukian, T.; and Ramamohanarao, K. 2003. A Fast Algorithm for Computing Hypergraph Transversals and its Application in Mining Emerging Patterns. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA*, 485–488. IEEE Computer Society.

Bailey, J.; and Stuckey, P. J. 2005. Discovery of Minimal Unsatisfiable Subsets of Constraints Using Hitting Set Dualization. In Hermenegildo, M. V.; and Cabeza, D., eds., *Practical Aspects of Declarative Languages, 7th International Symposium, PADL 2005, Long Beach, CA, USA, January 10-11, 2005, Proceedings*, volume 3350 of *Lecture Notes in Computer Science*, 174–186. Springer.

Eriksson, S.; Röger, G.; and Helmert, M. 2017. Unsolvability certificates for classical planning. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

Glimm, B.; and Kazakov, Y. 2019. Classical Algorithms for Reasoning and Explanation in Description Logics. In Krötzsch, M.; and Stepanova, D., eds., *Reasoning Web. Explainable Artificial Intelligence - 15th International Summer School 2019, Bolzano, Italy, September 20-24, 2019, Tutorial Lectures*, volume 11810 of *Lecture Notes in Computer Science*, 1–64. Springer.

Göbelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming Up With Good Excuses: What to do When no Plan Can be Found. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, 81–88. AAAI.

Gunopulos, D.; Khardon, R.; Mannila, H.; Saluja, S.; Toivonen, H.; and Sharm, R. S. 2003. Discovering all most specific sentences. *ACM Trans. Database Syst.*, 28(2): 140–174.

Haslum, P.; Slaney, J. K.; and Thiébaux, S. 2012. Minimal Landmarks for Optimal Delete-Free Planning. In McCluskey, L.; Williams, B. C.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*. AAAI.

Herzig, A.; de Menezes, M. V.; De Barros, L. N.; and Wassermann, R. 2014. On the revision of planning tasks. In *ECAI*, 435–440.

Käser, L. G.; Büchner, C.; Corrêa, A. B.; Pommerening, F.; and Röger, G. 2022. Machetli: Simplifying Input Files for Debugging. ICAPS 2022 System Demonstrations.

Koehler, J. 1999. RIFO within IPP. Technical report, Tech. rep. 126, University of Freiburg.

Koehler, J.; Nebel, B.; Hoffmann, J.; and Dimopoulos, Y. 1997. Extending Planning Graphs to an ADL Subset. In Steel, S.; and Alami, R., eds., *Recent Advances in AI Planning, 4th European Conference on Planning, ECP'97, Toulouse, France, September 24-26, 1997, Proceedings*, volume 1348 of *Lecture Notes in Computer Science*, 273–285. Springer.

Long, D.; Kautz, H. A.; Selman, B.; Bonet, B.; Geffner, H.; Koehler, J.; Brenner, M.; Hoffmann, J.; Rittinger, F.; Anderson, C. R.; Weld, D. S.; Smith, D. E.; and Fox, M. 2000. The AIPS-98 Planning Competition. *AI Mag.*, 21(2): 13–33.

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

Sreedharan, S.; Chakraborti, T.; Muise, C.; Khazaeni, Y.; and Kambhampati, S. 2020. –d3wa+–a case study of XAIP in a model acquisition task for dialogue planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 488–497.

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