Proceedings of the Thirty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2026)

Finding Human-Aligned Abstractions Efficiently for Explaining Plan Quality Differences

Benjamin Krarup[1], **Amanda Coles**[1], **Derek Long**[1], **David E. Smith**[2]
[1]King's College London, UK
[2]Independent Researcher
benjamin.krarup@kcl.ac.uk, amanda.coles@kcl.ac.uk, derek.long@kcl.ac.uk, david.smith@psresearch.xyz

Abstract

Explaining differences in the quality of plans is an effective method for helping users understand the decisions made by planners. When a user suggests an alternative plan that is lower in quality, these explanations can clarify why the original plan was preferable, or reveal hidden preferences and trade-offs in the model. Recent advances in plan explanation have used abstraction to generate explanations for differences in plan quality. Details of planning problems are abstracted until the quality of plans they produce are equal in cost. These abstractions are then used as causal factors in explanations. We build on this work in two ways. Firstly, we introduce a number of pruning techniques based on properties of abstractions, extracted from the human-subject study, to improve the efficiency of finding human-aligned abstractions. Secondly, we formalise and implement a number of heuristics, derived from a human-subject study capturing how people explain differences in plan quality.

1 Introduction

Planning is ubiquitous both in everyday life and in complex operations such as oil well drilling, underwater pipe inspection, and shipping (Long 2018; Buksz et al. 2018; Tierney et al. 2012). The field of AI planning is concerned with automatically generating plans, as governed by a model of the world, to achieve certain goals. The world model is represented by users through a modelling language. However, these models are often approximate, and accurately representing complex goals, preferences, and constraints can be challenging. Therefore, it is reasonable to assume that a generated plan may not be completely satisfactory to a user.

Krarup et al. (2021) investigated what types of questions users have about plans. They found through a user study that users ask contrastive questions where users query why certain decisions, d_1, were made in plans rather than others, d_2. They provide answers to these contrastive questions by first simulating what would happen had d_2 taken place. They then generate contrastive explanations that highlight the differences between the original plan containing d_1 and the imagined alternative containing d_2.

Krarup et al. (2021) frame this as a mixed-initiative planning setting in which humans and automated planners interact and collaborate to produce satisfactory plans. An automated planner is used to produce plans quickly and a human can then add constraints and preferences to the model until they are satisfied with the resulting plan produced by the automated planner. After each addition of a constraint the newly generated plan can be of better or worse quality than the version generated before the constraint was added, or the problem could even become unsolvable. In these cases it is useful to have accompanying explanations for why additional constraints lead to a reduction in the quality of plans.

The setting described above can be more formally expressed as one of iterative model restriction. More precisely, we are given a planning problem, Π, a solution plan, π, for Π, a constraint, c, which π does not satisfy, and a solution plan, π^c, for $\Pi^c = \Pi + c$, where there is a difference in the quality of π and π^c. A special case of this is where π^c does not exist, that is the problem Π^c is unsolvable. In this case, the quality of π^c is infinitely worse.

Krarup et al. (2024) propose providing explanations for the differences in the quality of plans by abstracting away details of the planning problem until the two plans become *equi-quality* – that is, of equal quality. They find these abstractions through a blind search procedure. They then explain why one plan is better or worse than the other in terms of the abstracted features that impacted the difference in plan quality between the two. For example, if a planning system generates a plan, π, that uses truck1 to deliver a parcel, but the user proposes to instead use truck2 (the constraint), the explanation as to why the user's alternative plan (π^c) is not as good may be that truck2 exceeds the bridge weight limit so truck2 has to take a more circuitous route. If we abstract away the weight precondition for the bridge, we can now produce a plan that uses truck2 but is of equal cost to the plan that uses truck1. Thus the abstracted feature can be seen as responsible for the cost difference.

We build on the work by Krarup *et al.* (2024) in two ways. First we improve their abstraction search by noting a number of properties of abstraction that we can use to prune abstractions that will not lead to helpful explanations based on the notions of degrees of convergence and perturbation. Secondly, we propose three heuristics to help guide search to abstractions that produce human-aligned explanations. We determine which abstractions will produce explanations that are human-aligned based on the results of a user study by

Krarup et al. (2026) that investigated the nature of explanations given by users. Our new heuristics are motivated by this study, designed to result in explanations of the type humans give. When we refer to a study or user study in this paper we are referring to this study by Krarup et al (2026).

2 Related Work

The adoption of AI planning systems requires the ability to explain their behaviour. Fox et al. (2017) highlight the importance of contrastive *'why'* questions in plan explanation, describing variants of these questions and possible responses. Chakraborti et al. (2017) approach explanation as model reconciliation, that is, explanation depends on demonstrating differences between the agent's and the human's models of the planning problem. Krarup et al. (2021) automatically generate answers to contrastive questions humans might pose about plans. There has also been interest in providing explanations in path/motion planning (Almagor and Lahijanian 2020; Pozanco et al. 2022).

Abstraction has an established role in problem solving, for example using heuristics based on abstracted (relaxed) problems to guide search. The use of abstraction in generating plan explanations is relatively recent. Gobeldecker et al. (2010) focus on finding changes to the initial state that would make a planning problem solvable, and provide an algorithm to produce these 'excuses' in reasonable time. Our approach works instead by relaxing constraints in the planning problem. Eifler et al. (2022) explain why some set of soft goals cannot be achieved in through constraint relaxations. We, on the other hand, explain why some arbitrarily constrained planning problem becomes unsolvable or causes only plans of worse quality to be valid. Brandao et al (2021) explain why some path is optimal rather than another, in a path planning context, by abstracting the navigation graph of the path. We are concerned with task planning. Sreedharan et al. (2019) use abstractions of predicates to simplify an unsolvable problem to a minimallly unsolvable problem, they then find unreachable milestones in that minimal unsolvable problem to explain the unsolvability. Sreedharan et al. (2025) use a similar approach of model simplification to explain when there is a inferential capability gap between the explainer and explainee. Sreedharan et al. (2018) employ a heuristic search for finding abstractions to explain a discrepancy between models. Sreedharan et al (2021) and Vasileiou and Yeoh (2023) use abstraction for generating personalised explanations whose level is based on a human's knowledge or expertise of the task. Each of these works by Sreedharan et al. differs to ours in several major ways. Firstly, we are solving different problems: explaining different things. Fundamentally, they are explaining differences between models; we are explaining differences between plans. They assume a discrepancy between the human's model and the model used by the planning system and give explanations that indicate these differences between models. We are explaining the differences between plans that result from an added constraint. Furthermore, we give explanations that not only show these differences but provide the *causes* for these differences, which is what our abstractions are used for. They use abstraction in a different way: to simplify the model to focus on the areas where there are discrepancies, essentially for efficiency.

None of the approaches above take into account the properties of abstractions that make them useful in explanation to humans. This is, to our knowledge, the first work that seeks to utilise the results of a user study to directly influence abstraction search to find human-aligned explanations.

3 Setting and Abstraction

We use the standard definition of temporal-numeric planning problems described in detail in Fox and Long (2003). We briefly define a planning problem here in Definition 1.

Definition 1. *A **planning problem** is the tuple* $\Pi = \langle Ps, Vs, As, I, G, M, T\rangle$ *where* Ps *is a finite set of predicate symbols,* Vs *is a finite set of function symbols,* As *is a set of actions,* I *is the initial state,* G *is the goal condition,* M *is a plan-metric function from plans to real values (plan costs) and* T *is a set of timed initial literals. An action,* $a \in As$*, has a duration* $Dur(a) = \langle lb, ub\rangle$ *that dictates the lower,* lb*, and upper bound,* ub*, on the execution time for* a*; conditions that must be true at the start/end/during the execution of* a*; and effects that update the state at the start/end/during the execution of* a*.*

We use the definition of abstraction in Krarup *et al.* (2024), which is based on the underlying labelled state transition system (LST) and simple temporal network (STN) induced from a temporal-numeric planning problem.

Definition 2. *A labelled state transition system (LST),* τ*, is a triplet* $\langle S, L, T\rangle$*, where* S *is a set of states,* L *is a set of labels and* $T \subseteq S \times L \times S$ *is a set of labelled transitions from one state to another. A path,* π*, in* τ *is a pair in* $S \times L^*$*, consisting of a state,* s*, and a finite sequence of labels* $l_0, ..., l_{n-1}$*, such that there is a sequence of states,* $s_0, ..., s_n$ *such that* s=s_0 *and, for each* i=0, ..., n-1, $(s_i, l_i, s_{i+1}) \in T$*.*

An LST can be derived from a planning problem:

Definition 3. *Let* Π *be a planning problem; the LST derived from* Π *is* $\tau = \langle S, L, T\rangle$ *where* S *is the set of all valuations of valid groundings of* Ps *and* Vs*, and* L *is the set of labels corresponding to the ground actions* A*.* $T = \{(s, l, s')|s \in S$*, the ground action* l *is applicable in* s*,* s' *is the state after application of* l *to* $s\}$*. We define the function* $\sigma(\Pi) = \tau$ *to denote the derivation of an LST from a planning problem* Π*.*

In a planning problem Π, only states that can be traversed to from the initial state of the planning problem (I) are *reachable* and only states that support paths to a goal state (satisfying G) are *relevant* to the solution:

Definition 4. *Given an LST,* $\tau = (S, L, T)$*, an* LST problem *is a pair,* (I, G)*, such that* $I \in S$ *and* $G \subseteq S$*, these are referred to as the initial state and goal set of the LST problem.*

Definition 5. *Given an LST,* $\tau = (S, L, T)$*, and an* LST problem*,* (I, G)*, a state,* $s \in S$*, is* reachable *if there is a path from* I *to* s *in* τ*. A state,* $s \in S$*, is* relevant *if there is a path from* s *to some state* $g \in S$ *where* $G \subseteq g$ *in* τ*.*

A valid path for an LST problem (I, G), in the LST (note, we use an STN as well as the LST to determine validity of a plan π for a planning problem Π), is the pair $\{I, \{l_0, \ldots, l_{n-1}\}\}$, such that there is a sequence of states $I, \ldots, g$ such that for each $i = 0, \ldots, n-1$, $(s_i, li, s_{i+1}) \in T$ and $g \in G$. Such a path corresponds to a valid solution plan for the underlying planning problem.

Given an LST τ and an LST problem (I, G), we define an abstraction, τ', as follows:

Definition 6. *An LST, $\tau' = (S', L, T')$, is an* abstraction *of LST, $\tau = (S, L, T)$, with respect to initial state, $I \in S$, and goal set, $G \subseteq S$, if there is a mapping, $f : S \to S'$, such that for every transition, $\langle s, l, s' \rangle \in T$, from a reachable and relevant state, $s \in S$, there is $\langle f(s), l, f(s') \rangle \in T'$.*

Instead of thinking about abstractions of boundless LSTs, for planning problems we can think of abstractions of LSTs with relevant LST problems. Definition 6 can be generalised to define an abstraction that operates over all LST problems in some family of LST problems, $\mathcal{P}$, for a given LST, τ.

Definition 7. *An LST, $\tau' = (S', L, T')$, is a* general abstraction *of LST, $\tau = (S, L, T)$, with respect to a set of LST problems, $\mathcal{P}$, if τ' is an abstraction of τ for every LST problem, $(I, G) \in \mathcal{P}$.*

To model plans in which action durations, temporal happenings, and temporal constraints matter; we can create a Simple Temporal Network (STN) (Dechter, Meiri, and Pearl 1991) on top of the LST from Definition 3 that dictates the timings of the state transitions in the LST. An STN is a graph whose vertices represent time points and weighted edges represent the maximum/minimum separation between these.

Definition 8. *A simple temporal network (STN), G, is a directed graph denoted by the triplet $\langle V, E, \Lambda \rangle$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, and $\Lambda \subseteq E \times \mathbb{R}$ is a set of labels applied to edges to represent temporal separation between vertices.*

An STN is constructed over an LST, τ, and a potential path through the LST, a finite sequence of labels in τ $\{l'_1, l'_2, ..., l'_n\}$, as follows: the corresponding STN has $n + 2$ vertices, labelled I, l'_i (for $i = 1...n$) and G. The edges of the STN comprise an edge from l'_1 to I and from l'_{i+1} to l'_i, each weighted $-\epsilon$ (where ϵ separates interfering actions), an edge from G to l'_n weighted 0, and, for each pair of labels, l'_i and l'_{i+j}, that represent the start and the end of the same durative action instance, edges from l'_{i+j} to l'_i weighted with $-Dur(l'_i)[0]$ and from l'_i to l'_{i+j} weighted $Dur(l'_i)[1]$, i.e. for the edge e_1 from l'_i to l'_{i+j}, $\lambda(e_1) = Dur(l'_i)[1]$, and for the edge e_2 from l'_{i+j} to l'_i, $\lambda(e_2) = -Dur(l'_i)[0]$. If any action starts or ends are not paired off, the label sequence is not a valid plan; the sequence of labels is otherwise a valid plan if and only if it is both a valid path in the LST and also a consistent STN. We denote the set of all valid plans for a planning problem, Π, as $\mathbf{P}(\Pi)$, we also denote these sets of valid plans as $\mathbf{P}(\tau)$ where τ is derived from Π. A valid plan in the LST corresponds to a valid plan for the planning problem it is derived from and therefore $\mathbf{P}(\Pi) = \mathbf{P}(\tau)$. We also make use of a function, $D(\pi_1, \pi_2) = |M(\pi_1) - M(\pi_2)|$, which returns the difference in quality of any two plans, π_1 and π_2 for Π_1 and Π_2. If, for any two planning problems, Π_1 and Π_2, Π_2 is unsolvable and so π_2 is undefined, then $D(\pi_1, \pi_2) = M(\pi_1) + \gamma$ where γ is a suitably large overestimate for the for the worst case quality of a solution for Π_2. If $D(\pi_1, \pi_2) = 0$, we call π_1 and π_2 equi-cost.

TILs can be captured within the framework of the LST and STN described above as follows. Given a TIL, $t = \langle t_{time}, t_v \rangle$, specifying effect t_v occurs at time t_{time}, to be added to a planning problem, Π, a new proposition, t, is created and added to the initial state, an action, TIL_t is created with precondition t, that deletes t and with add effects t_v and $doneT$; $doneT$ is added to the goal. An LST is then created in the usual way. The STN created for this temporal domain is then adjusted by adding edges of weight t_{time} from I to TIL_t and $-t_{time}$ in the opposite direction. For this problem, a valid plan will be forced to contain exactly one copy of the action TIL_t, in order to satisfy the goal. No additional copies can appear because of the deleted precondition. The temporal constraints can only be satisfied if TIL_t occurs at exactly time t_{time}, along with all the other constraints of the temporal structure of the plan.

We can now define an abstraction of an STN:

Definition 9. *An STN $G' = \langle V', E', \Lambda' \rangle$ is an abstraction of STN $G = \langle V, E, \Lambda \rangle$, where $V' \subseteq V$ and $E' \subseteq E$, if for any two vertices $v, v' \in V$ that are connected by an edge $e \in E$ where there is a label λ for e in Λ, we have $\lambda' \in \Lambda'$ where $\lambda'(e) \geq \lambda(e)$, if $v, v' \in V'$ and $e \in E'$.*

Finally, as we are working with temporal-numeric planning domains, we can define what a valid abstraction of a planning problem is based on their derived LST and STN:

Definition 10. *For any two planning problems, Π and Π', Π' is an abstraction of Π under the mapping f if the LST τ' derived from Π' is an abstraction of the LST τ derived from Π and all plans in τ that have a consistent STN are plans in τ', under the mapping f, that have a consistent STN.*

In this paper we make use of the abstractions provided by Krarup et al (2024) which they show to be valid under the definitions above. Rather than providing a new suite of abstractions, this paper focuses on improving the way in which we search over these abstractions to find those that are suitable for explanation, per Definition 11.

Definition 11. *Given a planning problem Π and a constrained problem Π^c, we consider an abstraction set, $\mathbb{A}$, to be suitable for explanation if the solution to the abstracted constrained problem $\Pi^c_{\mathbb{A}}$, $\pi^c_{\mathbb{A}}$, and the abstracted original problem $\Pi_{\mathbb{A}}$, $\pi_{\mathbb{A}}$, have costs, $M(\pi^c_{\mathbb{A}})$ and $M(\pi_{\mathbb{A}})$ and $M(\pi^c_{\mathbb{A}}) = M(\pi_{\mathbb{A}})$.*

We consider explanations in the setting described by Krarup et al. (2024), as follows: given a planning problem, Π; a plan, π, for Π; a constraint, c, which π does not satisfy, and a solution plan, π^c, for $\Pi^c = \Pi + c$ (Π restricted to admit only solutions that obey c). We assume that there is a difference in the quality of π and π^c. A special case is where the problem Π^c is unsolvable and π^c does not exist. We seek to explain why there is a difference in the quality of π and π^c. We assume that an explanation of the form "the difference is

because of the constraint c" is not helpful. An explanation of a discrepancy in plan quality should consist of elements of the planning problem (apart from c) that cause the discrepancy. A planning problem, Π_α, is an abstraction of Π if every solution of Π is a solution of Π_α. If the plans π and π^c are not of the same quality, but under the abstraction α, the plans π_α and π^c_α are the same quality, then we can say that α is a *cause* of the difference in quality of the plans for Π and Π^c These causes can be found by abstracting away elements, α, of both planning problems, Π and Π^c, until $D(\pi_\alpha, \pi^c_\alpha) = 0$. We call these abstractions **suitable** for explanation.

A user may question decisions made in the plan, π, for Π and explanations may uncover un-modelled preferences or goals, but we assume that the desire is to do minimal damage to the the original planning problem, Π. With this assumption in mind, we consider two quality measures for abstractions, degree of convergence and degree of perturbation. The degree of convergence measures the degree to which an abstraction causes the plans produced by the abstracted problems (π_α and π^c_α) to converge. The degree of perturbation measures the degree to which an abstraction causes the plan π_α produced by the abstracted original problem to change with respect to the plan π for the original problem. We use plan quality as a proxy for these measurements. More specifically, the degree of convergence is defined by:

$$Conv(\Pi, \Pi^c, \alpha) = 1 - \frac{D(\pi_\alpha, \pi^c_\alpha)}{D(\pi, \pi^c)}$$

The degree of perturbation is defined by:

$$Pert(\Pi, \alpha) = \frac{D(\pi, \pi_\alpha)}{M(\pi)}$$

We assume that Π is always solvable, so we always have a plan π: given that our intention is to explain the difference in quality between a plan and a reference plan, π, if we do not have π we do not have a question to answer. Finally we note that the denominator is larger than the numerator for $Pert$ and $Conv$. For $Conv$ this holds because if π^c does not exist, i.e. Π^c is unsolvable, then we let $D(\pi, \pi^c) = M(\pi) + \gamma$, where γ is sufficiently large so that $M(\pi^c_\alpha) < \gamma$. For $Pert$ and if both plans exist for $Conv$, this holds due to the nature of abstraction – any plan, π, for a planning problem, Π, and an abstraction, α, is also a plan for Π_α, so a plan for Π_α, π_α, is at least as good as π, $M(\pi_\alpha) \leq M(\pi)$. Also note that $Pert$ does not take the constrained problem as an argument, this is because the degree of perturbation for an abstraction is strictly a measure of how much the abstraction damages the solution to the original problem, Π.

We are interested in four cases for Π, Π^c, and α:

1. $Conv(\Pi, \Pi^c, \alpha) = 1$ and $Pert(\Pi, \alpha) = 0$. These abstractions seem most suitable for explantation since they fully explain the difference in the quality of π and π^c and they do not perturb the original problem, Π. We call these abstractions **fully convergent** with respect to Π and Π^c and **non-perturbing** with respect to Π.
2. $Conv(\Pi, \Pi^c, \alpha) = 1$ and $Pert(\Pi, \alpha) > 0$. These abstractions still fully explain the difference in the quality of π and π^c but perturb the the original problem, Π. These abstractions are **fully convergent** and **perturbing**.
3. $0 < Conv(\Pi, \Pi^c, \alpha) < 1$ and $Pert(\Pi, \alpha) \geq 0$. These abstractions partially explain the difference in the quality of plans and do not perturb the original problem, Π. These abstractions are **partially convergent** and may or may not be **perturbing**.
4. $Conv(\Pi, \Pi^c, \alpha) = 0$ and $Pert(\Pi, \alpha) \geq 0$. These abstractions do not reduce the difference in quality of plans at all and therefore cannot be used to even partially explain the difference, we call these **non-convergent**.

There are further cases e.g. **fully perturbing** abstractions, which abstract problem so that an empty plan is valid (e.g. by removing all goals), but this is not useful for explanation.

4 Summary of User Study

The study conducted by Krarup et al. (2026) was conducted to investigate how humans explain the difference in the quality of plans produced by model restriction. The authors presented participants with a number of planning problems. These problems were then constrained in two ways: either by adding a constraint that led to production of worse quality solutions or one that made the problem unsolvable. Participants were asked to explain why the problems produced worse quality solutions or became unsolvable when the constraint was added. Explanations produced by participants were recorded and a number of hypotheses were explored.

The authors found that humans do indeed utilise abstractions in these explanation tasks (H1). They also found that the abstractions completely explained the differences in the quality of the solutions (H2), that is the abstractions were fully convergent. The abstractions generated by Krarup et al. (2024) also satisfy these two hypotheses. Where we differ is in the following hypotheses.

The authors found that these fully convergent abstractions are also non-perturbing (H3). When a human questions a candidate plan, they prefer an explanation that does not deviate from the original solution unnecessarily. This supports the hypothesis that an abstraction that makes minimal changes to the original planning problem is preferable. This is based on the premise that the original planning problem is the most accurate description of the environment that is being planned in that we have. Therefore, abstractions that cause minimal changes to the ground truth problem produce more satisfying explanations.

For example, in a problem where there are two trucks, one electric and one powered by petrol, a user might question why the electric truck was used instead of the petrol one. The plan that uses the petrol truck may be of worse quality due to emitting more emissions. An abstraction that aligns the emissions cost with that of the electric truck can provide an explanation such as, "the plan is worse quality due to the amount of emissions produced, if the petrol truck produced the same amount of emissions as the electric truck then we could use the petrol truck". However, an abstraction that instead removed the need to drive between locations, and therefore produced zero emissions, provides an explanation that more perturbing and therefore less satisfying. It completely removes the need to drive between locations that is present in the original problem. Finally, the authors found

that humans tend to use fewer abstractions to produce their explanations (H4).

5 Improvements to Search

The previous work on utilising abstractions for explaining the difference in the quality of plans (Krarup et al. 2024) used a breadth first search algorithm that considers all permutations of abstractions. Based on the study and what it informs us about the features of useful abstractions we have devised a number of pruning techniques and improvements to the search algorithm: 1) the realisation that the ordering of abstractions does not matter, 2) that abstractions monotonically improve the quality of the solutions of planning problems they are applied to, 3) that minimal suitable abstractions are always preferable, and 4) that there exist a class of abstractions that we call *detail abstractions* that avoid introducing unnecessary changes to the original problem/plan.

Order of Abstractions Does Not Matter Abstractions are commutative, that is, for any two abstractions, $\alpha_1, \alpha_2 \subseteq \mathbb{A}$, applied to a planning problem, Π, $\Pi_{\alpha_1 \circ \alpha_2} = \Pi_{\alpha_2 \circ \alpha_1}$. Per Definition 6 an abstraction manifests as the addition of edges to the LST that the planning problem induces. This is clearly commutative. We define abstractions not by the operations performed on a planning problem but by the result of the operation on the underlying LST and STN. Therefore a conditional abstraction operation may not be commutative, for example "for each abstraction already applied to the problem, Π, apply an abstraction from set $\mathbb{A}$". However, we would consider this abstraction as a suite of separate abstractions and assume they do not come in this form. We assume the abstraction mechanisms are set operations applied to the planning problem, $\Pi = \langle D, Prob \rangle$, this is true from the abstractions we use (taken from Krarup et al (2024)). From this observation we can prune the search space significantly as we now only have to consider combinations of abstractions, not permutations.

Quality Assumption For the next two properties an assumption, based on the hypothesis H3 from the study, and a proposition about the monotonic improvement of the quality of solutions to abstracted planning problems is necessary.

Assumption 1. *Based on Hypothesis H3, we assume that abstractions of planning problems with the same convergence that are less perturbing are preferable. That is for any two suitable abstractions, α_1, α_2, for a planning problem, Π, and a constrained problem, Π^c, if $Conv(\Pi, \Pi^c, \alpha_1) = Conv(\Pi, \Pi^c, \alpha_2)$ and $Pert(\Pi, \alpha_1) < Pert(\Pi, \alpha_2)$ then α_1 is preferable to α_2.*

Proposition 1. *For any planning model, Π, with a solution, π, and any abstraction, α, applied to Π to give the abstracted model, Π_α, with a solution, π_α, $M(\pi_\alpha) \leq M(\pi)$.*

The proof for Proposition 1 is trivial given Definition 10, as any plan π for Π is also a plan for Π_α and so $M(\pi_\alpha) \leq M(\pi)$. In practice, even if we are not doing optimal planning, we can guarantee a solution Π_α that respects this property by imposing a cost bound on the planner's search for Π_α: $cost \leq M(\pi)$, if search fails to find a solution we can simply return π, as by definition it is a solution to Π_α.

Stopping Conditions for Search Under Proposition 1 any solution to a planning problem, Π, is improving with respect to any applied valid abstraction, α. In other words, iteratively applying abstractions to Π monotonically improves the quality of its solutions with respect to the metric $M \in \Pi$.

From this proposition we can determine certain stopping conditions based on the quality of abstracted solutions for a planning model, Π, a constrained model Π^c, and an abstraction, α, that we are interested in:

1. The original, original abstracted, and constrained abstracted solutions are equi-cost: i.e. $M(\pi) = M(\pi_\alpha) = M(\pi^c_\alpha)$.
2. The original abstracted and constrained abstracted solutions are equi-cost and are better quality than the original solution: i.e. $M(\pi_\alpha) = M(\pi^c_\alpha) < M(\pi)$.
3. The original abstracted and constrained abstracted solutions are not equi-cost and are better quality than the original solution: i.e. $M(\pi_\alpha) < M(\pi^c_\alpha) < M(\pi)$.
4. The constrained abstracted solution is worse quality than the original solution: i.e. $M(\pi^c_\alpha) > M(\pi)$.

Note these are not termination criteria for search, rather pruning rules that mean we will no longer continue expanding a given node. In category 1, $Conv(\Pi, \Pi^c, \alpha) = 1$ and $Pert(\Pi, \alpha) = 0$, these abstractions are optimal for explanation so we should not continue to abstract these models and do not have to search over them further. In category 2, $Conv(\Pi, \Pi^c, \alpha) = 1, Pert(\Pi, \alpha) > 0$, these abstractions are suitable for explanation but are not optimal. Furthermore, because $M(\pi_\alpha) = M(\pi^c_\alpha) < M(\pi)$, $Pert(\Pi, \alpha) \leq Pert(\Pi, \alpha \circ \alpha')$ for any abstraction, α'. Additional abstractions can only cause an improvement in quality therefore the distance in quality to the un-abstracted plan for Π (perturbation) can only increase. Under Assumption 1 we always prefer abstracted solutions that are less perturbing. Therefore, we should not continue to abstract the models in category 2. This similarly holds for category 3, but these are not suitable abstractions for explanation, i.e. $Conv(\Pi, \Pi^c, \alpha) < 1$. The only models that we should continue to abstract are those in category 4 because $Conv(\Pi, \Pi^c, \alpha) < 1$ and $M(\pi^c_\alpha) > M(\pi)$ which means, trivially, $Pert(\Pi, \alpha) = 0$, so further abstraction can potentially result in $Conv(\Pi, \Pi^c, \alpha) = 1$ and $Pert(\Pi, \alpha) = 0$.

Minimal Suitable Abstractions are Always Preferable Given Definition 10 it can be concluded that for any two suitable abstractions, α_1, α_2, for a planning problem, Π, then for the abstracted problems, Π_{α_1}, Π_{α_2}, and $\Pi_{\alpha_1 \circ \alpha_2}$, the space of plans $\mathbf{P}(\Pi_{\alpha_1}) \subseteq \mathbf{P}(\Pi_{\alpha_1 \circ \alpha_2})$ and $\mathbf{P}(\Pi_{\alpha_2}) \subseteq \mathbf{P}(\Pi_{\alpha_1 \circ \alpha_2})$. Given Definition 11 and Proposition 1, for any suitable plan $\pi_{\alpha_1 \circ \alpha_2} \in \mathbf{P}(\Pi_{\alpha_1 \circ \alpha_2})$ there exists a suitable plan $\pi_{\alpha_1} \in \mathbf{P}(\Pi_{\alpha_1})$ and a suitable plan in $\pi_{\alpha_2} \in \mathbf{P}(\Pi_{\alpha_2})$ where $M(\pi_{\alpha_1}) \geq M(\pi_{\alpha_1 \circ \alpha_2})$ and $M(\pi_{\alpha_2}) \geq M(\pi_{\alpha_1 \circ \alpha_2})$. Therefore, $Pert(\Pi, \alpha_1) \leq Pert(\Pi, \alpha_1 \circ \alpha_2)$ and $Pert(\Pi, \alpha_2) \leq Pert(\Pi, \alpha_1 \circ \alpha_2)$, and given Assumption 1, we can conclude that minimal suitable abstractions are always preferable. We can use this to prune our search. We should never check a

set of abstractions, $\mathbb{A}$, if there exists a subset of abstractions $\mathbb{A}_s \subset \mathbb{A}$ that are suitable.

Detail Abstractions Based on Hypothesis H3 we propose a novel type of abstraction called a detail abstraction. These are abstractions that only remove details of the planning problem that result in simplification of the original plan. These abstractions are perturbing: they alter the original model such that the plan produced using the abstracted model is different. However, they are not *damaging* to the physics of the original model. An abstraction that does not lead to addition of any actions to the plan is a detail abstraction. Detail abstractions are defined in Definitions 12 and 13.

Definition 12. *Given a planning problem* Π *with an LST,* $\tau = \langle S, L, T \rangle$*, derived from* Π *with valid plans,* $\mathbf{P}(\tau)$*, that have consistent STNs. The planning problem's,* Π*'s, shortest path,* $\pi^* \in \mathbf{P}(\tau)$*, is the shortest valid plan as constrained by each plan's STN. We call* π^* *the optimal plan of* τ*.*

Definition 13. *Given a planning problem* Π *with an optimal plan* π^* *and an abstraction,* α*, with some mapping,* f*, which applied to* Π *gives the abstracted model,* Π_α*, if there exists an optimal plan,* π^*_α*, that is a subset of* π^* *with the mapping* f*, then* α *is a detail abstraction.*

Detail abstractions have two aims. The first is the aim we have been discussing in this paper so far, to remove the difference between similar planning solutions in order to determine the causes for those differences. The second is to remove unnecessary details from the planning problem that are not relevant to the thing we are trying to explain. A planning problem, Π, can often be thought of as a set of simpler problems, $\Pi_1, \ldots, \Pi_n$, that when solved in some order are a solution to Π. If the cause for the difference in the solutions of the original and constrained models is only contained within a sub problem, Π_i, then abstracting the unnecessary details contained in $\Pi_1, \ldots, \Pi_{i-1}, \Pi_{i+1}, \ldots, \Pi_n$ until there is only Π_i left can help to simplify the planning problem to help find the cause [1]. Returning to the delivery example in the Introduction, imagine that a pre-check procedure must be performed on any truck that was selected to make the deliveries to ensure it is safe to operate, and that this check takes x units of time. Then abstracting away this pre-check procedure from both the original and constrained models, Π and Π^c, is a detail abstraction, the lack of pre-check procedure does not cause the delivery part of the plan to change, so the abstracted plan is a subset of the original plan. Furthermore, this abstraction does not reduce the difference in quality of the original and constrained solutions. These types of detail abstractions, *simplifying abstractions*, are defined in Definition 14.

Definition 14. *Given a planning model,* Π*, and* Π *constrained with c,* Π^c*, if there exists a detail abstraction,* α*, such that* $Conv(\Pi, \Pi^c, \alpha) = 0$ *then* α *is a* ***simplifying abstraction****.*

Search Procedure We have devised a search procedure, shown in Algorithm 1, that utilises the pruning techniques and improvements discussed in this section. This algorithm takes a planning model, Π, a constrained version of the model, Π^c, the original plan π for Π, a set of abstractions, $\mathbb{A}$, and a heuristic function, g, and gives back four sets of models and their abstractions categorised into the four categories discussed previously.

Algorithm 1: A* search procedure over a set of abstractions $\mathbb{A}$ applied to the planning models Π and Π^c to find suitable abstractions for explaining the difference in the solutions of Π and Π^c.

Data: $\{\Pi, \Pi^c, \pi, \mathbb{A}, g\}$
Result: $Categories$
$Category1, Category2, Category3, Category4 \leftarrow \emptyset$;
$abstractionsApplied \leftarrow \emptyset$;
$\mathbb{A} \leftarrow getDetailAbstractions(\mathbb{A})$;
$nodeQueue \leftarrow createNodes(\Pi, \Pi^c, \mathbb{A})$;
while $\neg empty(nodeQueue)$ **do**
  $curr_\Pi, curr'_\Pi, curr_\alpha \leftarrow dequeue(nodeQueue, g)$;
  if $\neg in(a(curr_\Pi) \cup curr_\alpha, abstractionsApplied)$, & $\neg minimalSuitable(a(curr_\Pi) \cup curr_\alpha)$ **then**
    $curr_{\Pi_\alpha} \leftarrow abstract(curr_\Pi, curr_\alpha)$;
    $curr_{\Pi^c_\alpha} \leftarrow abstract(curr_{\Pi^c}, curr_\alpha)$;
    $\pi_\alpha \leftarrow solve(curr_{\Pi_\alpha})$;
    $\pi^c_\alpha \leftarrow solve(curr_{\Pi^c_\alpha})$;
    if $detailAbstraction(\pi_\alpha, \pi^c_\alpha, \pi)$ **then**
      if $M(\pi) = M(\pi_\alpha) = M(\pi^c_\alpha)$ **then**
        $Category1 \leftarrow \{curr_{\Pi_\alpha}, curr_{\Pi^c_\alpha}, curr_\alpha\}$;
      else if $M(\pi_\alpha) = M(\pi^c_\alpha) < M(\pi)$ **then**
        $Category2 \leftarrow \{curr_{\Pi_\alpha}, curr_{\Pi^c_\alpha}, curr_\alpha\}$;
        $enqueue(nodeQueue, createNodes(curr_\Pi, curr'_\Pi, \mathbb{A}))$;
      else if $M(\pi^c_\alpha) > M(\pi)$ **then**
        $Category3 \leftarrow \{curr_{\Pi_\alpha}, curr_{\Pi^c_\alpha}, curr_\alpha\}$;
      else if $M(\pi_\alpha) \neq M(\pi^c_\alpha) < M(\pi)$ **then**
        $Category4 \leftarrow \{curr_{\Pi_\alpha}, curr_{\Pi^c_\alpha}, curr_\alpha\}$;
return $Categories \leftarrow Category1, Category2, Category3, Category4$;

The algorithm starts by creating a queue of nodes, $nodeQueue$, these nodes are made up of the original and constrained planning models and the abstractions applied to them as well as an abstraction that is yet to be applied. The function $createNodes$ takes two planning models and a set of abstractions and returns a node for each abstraction in the set paired with the two models. In each iteration a node, $curr_\Pi$, $curr'_\Pi$, $curr_\alpha$, is dequeued from the $nodeQueue$ based on the heuristic function g. We will discuss these heuristic functions further in the next section. The function, $a : \Pi \rightarrow \mathbb{A}$, takes a planning model and returns

[1] Similar to the approach in Sreedharan et al. (2019)

any abstractions that have already been applied to it. If the abstraction $a(curr_\Pi) \cup curr_{\mathbb{a}}$ has not been checked before and there does not exist $\mathbb{a}_s \subset a(curr_\Pi) \cup curr_{\mathbb{a}}$ such that $\mathbb{a}_s$ is suitable then we apply that abstraction to both $curr_\Pi$ and $curr'_\Pi$. The abstracted models $curr_{\Pi_{\mathbb{a}}}$ and $curr_{\Pi^c_{\mathbb{a}}}$ are then solved giving the abstracted plans $\pi_{\mathbb{a}}$ and $\pi^c_{\mathbb{a}}$. If this is a detail abstraction we then categorise the abstracted models and the abstraction applied based on the abstracted and original plan qualities. Note that we only add models to the queue if they are in category 2 as stated previously in this section.

If an abstraction is a simplifying abstraction then we know that this abstraction does not reduce difference in the quality of the solutions alone. However, it may be useful in combination with another abstraction. For example, if there is an additional constraint stopping truck drivers driving above a certain speed limit, and if the pre-check procedure prohibits this at the truck level, then abstracting both of these together may allow for the heavier truck2 to take an alternative route at a higher speed limit, giving equi-cost plans. Therefore, we can apply this simplifying abstraction to all nodes checked.

6 Heuristics

The development of heuristics aimed at both finding abstractions that are suitable or better for producing explanations, along with the techniques described in the previous section, have helped to feasibly explore larger abstraction spaces. These heuristics are motivated by finding suitable abstractions for explanations and finding better abstractions for explanations through Hypothesis H3.

The first heuristic is aimed at maximising convergence. That is finding abstractions that, when applied to the original and constrained planning problems, produce solutions that are equi-cost. For each heuristic we assume we have a set of abstractions $\mathbb{A}$, a planning problem Π, and Π constrained with c, Π^c. The heuristic is then defined as follows:

$$g(\mathbb{a}) = |M(\pi_{\mathbb{a}}) - M(\pi^c_{\mathbb{a}})|, \forall \mathbb{a} \in \mathbb{A}$$

where $\pi_{\mathbb{a}}$ is the solution to the planning problem Π abstracted with $\mathbb{a}$ and similarly $\pi^c_{\mathbb{a}}$ for the problem Π^c.

The second two heuristics, based on Hypothesis H3, aim to guide the search into finding abstractions that minimise that minimise perturbation. The first is based on the quality of the solutions and is defined as follows:

$$g(\mathbb{a}) = |M(\pi) - M(\pi_{\mathbb{a}})|, \forall \mathbb{a} \in \mathbb{A}$$

where π is the solution to the planning problem Π and $\pi_{\mathbb{a}}$ is the solution to the planning problem Π abstracted with $\mathbb{a}$.

The final heuristic is based on a notion of plan similarity defined by Fox et al. 2006. This measures the distance between two plans as the number of actions that are in both plans, ignoring the action ordering of the plans. The heuristic is defined as follows:

$$g(\mathbb{a}) = |\pi \setminus \pi_{\mathbb{a}}| + |\pi_{\mathbb{a}} \setminus \pi|$$

where π is the solution to the planning problem Π and $\pi_{\mathbb{a}}$ is the solution to Π abstracted with $\mathbb{a}$ and the operator $\setminus$ is defined so that for any two planning problems π_1 and π_2, $\pi_1 \setminus \pi_2$ contains n instances of action a iff π_1 contains x instances of a, π_2 contains y instance of a and $x - y = n > 0$, and $n = 0$ otherwise.

7 Evaluation

In this section we discuss an empirical study we conducted to evaluate the performance of eight variations of Algorithm 1. The first variation is a breadth first search, with no heuristic guidance (BFS). This baseline is the algorithm of Krarup et al. 2024 updated with the enhancements listed in Section 5, so this is already an advance beyond the current state-of-the-art. The next three variations are Algorithm 1 with the heuristic, g, as each of the three heuristics we define in Section 6, the suitability heuristic (S), the quality heuristic (Q), and the plan distance heuristic (PD). We ran each of these heuristics (and breadth first search) using the planners POPF (Coles et al. 2010) and OPTIC (Benton, Coles, and Coles 2012), creating a total of 8 variants.

We tested each of these eight algorithms on a Intel(R) Xeon(R) CPU E5-2660 v4 @ 2.00GHz machine with 188GB of memory. We tested each algorithm on five problems for Crew planning, Elevators, and Rover domains and three from the Depots domain from the temporal and numeric track of the 3rd and 8th international planning competition (Long and Fox 2003)[2]. Based on the abstractions given by Krarup et al. 2024, there were 51, 27, 66, and 22 possible abstractions for each domain in the listed order. We selected these domains and problem instances based on their varied complexity. We only selected three problem instances for Rovers as the others took too long so solve. We ran each experiment with both POPF (Coles et al. 2010) and OPTIC (Benton, Coles, and Coles 2012).

We restricted each problem instance with the addition of a constraint through the method proposed by Krarup et al. (2019; 2021) until the constrained problem produced a plan that was worse quality than the original problem's solution or it was unsolvable. The constraints were based on action inclusion/exclusion, the total ordering of actions, and the temporal ordering of actions. We repeated this to get three constrained problems for each problem instance. As it is difficult to prove optimality for temporal/numeric problems due to their complexity (Helmert 2002) we solved each problem with a 30 minute timeout to give a reasonable assumption of optimality. We extended the time taken to solve each problem instance and each constrained problem by three. Whichever of these was the longest became the planner timeout for solving each abstracted problem in the experiment run. This ensured the planner was afforded a reasonable time to find an optimal solution (on the same order as the time to solve the original problem) whilst not spending a disproportionate amount of time solving any one abstracted problem. Each run was given a maximum of two hours or 2000 nodes expanded without finding a suitable abstraction to run.

For problem instance 1 of Rovers and instance 10 of Elevators we did not use POPF as it could not find a solution to the original problem in 30 minutes, for instance 3 of Depots and instance 7 of Rovers we did not use OPTIC for the same reason. In total we evaluated eight variations of Algorithm 1 on 42 different planning instances using two planners.

[2]In the listed order the problem instances were: [1, 2, 3, 4, 5], [3, 5, 8, 9, 10], [1, 2, 3, 4, 7], [1, 2, 3]

Algorithm	First S/s	First O/s	# Suitable	# Optimal
BFS	323	429	6	6
S	232	339	8	8
Q	252	356	8	7
PD	218	327	6	6

Table 1: Average time (in seconds) to find the first suitable and first optimal abstraction and average number of suitable and optimal abstractions found, over all problem instances.

Table 1 shows the average time to find the first suitable abstraction, average time to find the first optimal abstraction, over the problem instances where we find suitable and optimal abstractions. Reporting results for mutually solved problems ensures that no configuration is penalised for finding a solution that another didn't, but taking a long time to do so. To indicate success in finding solutions we also show the average number of suitable and optimal abstractions found, over all problem instances. All times are in seconds. As a reminder, an abstraction is suitable if it is fully convergent. An abstraction is optimal for explanation if it is fully convergent and non-perturbing.

The plan distance heuristic on average found the first suitable and optimal abstraction the quickest out of each heuristic, taking 218 and 327 seconds respectively. The simple breadth first search algorithm with no heuristic took the longest, on average, to find the first suitable and optimal abstractions, taking 323 and 429 seconds respectively. The suitability and quality heuristics found the most suitable abstractions, averaging 8 suitable abstractions per problem. The suitability heuristic found the most optimal abstractions, 8 optimal per problem.

The breadth first search algorithm performed the worst taking on average 105 seconds longer to find a suitable abstraction than the plan difference heuristic and 102 seconds longer to find the first optimal abstraction. The plan distance heuristic was the fastest at finding both suitable and optimal abstractions. The plan distance heuristic seems to capture both the need for finding suitable and optimal abstractions. This may be because plan structure represents something that plan quality alone does not capture. However, the plan distance heuristic seems to perform worse at the number of suitable and optimal abstractions it finds. It was surprising that the quality heuristic seemed to perform relatively poorly at finding optimal abstractions. However, it is not surprising that the suitability heuristic was adept at finding suitable solutions, being the second fastest to find a suitable abstraction and finding the joint most.

8 Discussion and Future Work

The main limitation of this explanation approach is that most of the computational cost in each run is dominated by solving the planning problems. As a result, the choice and expansion of nodes can have more limited impact on total runtime. However, selecting the correct abstract problem early can still provide a substantial benefit, since it avoids spending time solving inappropriate problems before reaching the correct one. We evaluated our approach on IPC domains which were designed to test the performance of planners, not with explanation tasks in mind. Furthermore, we added random constraints to the problems, not informed by real human questions. These constraints often made the problem unsolvable such that a number of abstractions were needed to make the problem solvable, for example a constraint enforcing that hoists are used from their incorrect locations in the Depots domain.

We also evaluated the four variations of Algorithm 1 with the addition of pruning with simplifying abstractions, which manifests as adding simplifying abstractions to all nodes in the search tree. However, the results were not promising. We believe this is due to the additive nature of abstractions, the nature of planning problems, and the experiment design.

Given a simplifying abstraction, α_1, which is applied to each node in the search tree, if we add another abstraction, α_2, so that the abstraction $\alpha_1 \circ \alpha_2$ is suitable, it is not clear if just α_2 is suitable or $\alpha_1 \circ \alpha_2$. It is necessary to remove the simplifying abstraction α_1 and re-solve to determine the suitable abstraction for explanation. This could be worth the extra check if the simplifying abstractions significantly improve the time to solve the problem, because the planning time is often the bottleneck in this approach. However, in our experience most IPC benchmarks are not set up this way; it is rare that there is a significant sub-planning problem that one could simplify to significantly decrease solving times. We do, however, believe these types of problems exist in the real world, for example, rigorous safety checks or validation steps before starting/finishing some activity. Finally, in our experimental set up we used a static timeout for planners. We believe that simplifying abstractions help search by making problems easier to solve, but in our experiments the planner will keep optimising up to the timeout, unless it is proved optimal, which is rare in temporal/numeric planning.

Definition 13 does not make any claims about the degree of perturbation of the abstraction α. This is because a simplifying abstraction may perturb the planning problem. It may be better to instead separate the degree of perturbation of an abstraction with respect to a planning problem into degrees of damage and simplification. An abstraction, α, damages a problem, Π, if it perturbs the problem such that $Pert(\Pi, \alpha) > 0$ and α is not a simplifying abstraction. The plan distance metric may be more effective at capturing this nuance than plan quality, and this may be the reason it outperformed others.

9 Conclusion

In this paper a number of new insights into efficiently finding abstractions for producing human-aligned explanations are presented. These insights are supported by a user study determining how humans explain the difference in the quality of similarly constrained plans. We present a number different pruning techniques and heuristics that make searching over these abstractions more efficient. This abstraction search can be used for producing human-aligned explanations; i.e. explanations that are like those given by humans. However, this does not mean that humans will find them satisfactory. In the future we plan to evaluate the explanations produced from these abstractions through user studies.

References

Almagor, S.; and Lahijanian, M. 2020. Explainable multi agent path finding. In *AAMAS*.

Benton, J.; Coles, A.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *International Conference on Automated Planning and Scheduling*.

Brandao, M.; Coles, A.; and Magazzeni, D. 2021. Explaining path plan optimality: Fast explanation methods for navigation meshes using full and incremental inverse optimization. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, 56–64.

Buksz, D.; Cashmore, M.; Krarup, B.; Magazzeni, D.; and Ridder, B. 2018. Strategic-Tactical Planning for Autonomous Underwater Vehicles over Long Horizons. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3565–3572.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *Proc. International Joint Conf. on AI*.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proc. International Conf. on Automated Planning and Scheduling*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial intelligence*, 49(1-3): 61–95.

Eifler, R.; Hoffmann, J.; and Frank, J. 2022. Explaining softgoal conflicts through constraint relaxations. In *31st International Joint Conference on Artificial Intelligence*.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan Stability: Replanning versus Plan Repair. In Long, D.; Smith, S. F.; Borrajo, D.; and McCluskey, L., eds., *Proc. International Conf. on Automated Planning and Scheduling, (ICAPS)*, 212–221. AAAI.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelliigence Research*, 20: 61–124.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable Planning. *Proc. International Joint Conf. on AI-17 workshop on Explainable AI*, abs/1709.10256.

Göbeldecker, 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 *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*, AIPS'02, 44–53. AAAI Press. ISBN 1577351428.

Krarup, B.; Cashmore, M.; Magazzeni, D.; and Miller, T. 2019. Model-Based Contrastive Explanations for Explainable Planning. *ICAPS-19 Workshop on Explainable Planning*.

Krarup, B.; Coles, A.; Gao, D.; Long, D.; and Smith, D. E. 2026. How Humans Explain the Difference in the Quality of Plans – A User Study. In *Proc. International Conf. on Automated Planning and Scheduling*.

Krarup, B.; Coles, A.; Long, D.; and Smith, D. E. 2024. Explaining Plan Quality Differences. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 324–332.

Krarup, B.; Krivic, S.; Magazzeni, D.; Long, D.; Cashmore, M.; and Smith, D. E. 2021. Contrastive Explanations of Plans through Model Restrictions. *JAIR*, 533–612.

Long, D. 2018. Planning a Way Into a Deep Hole. In *Invited talk at ICAPS Workshop on Planning and Scheduling Applications (SPARK)*.

Long, D.; and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20: 1–59.

Pozanco, A.; Mosca, F.; Zehtabi, P.; Magazzeni, D.; and Kraus, S. 2022. Explaining preference-driven schedules: the expres framework. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 710–718.

Sreedharan, S.; Madhusoodanan, M. P.; Srivastava, S.; and Kambhampati, S. 2018. Plan explanation through search in an abstract model space. In *International Conference on Automated Planning and Scheduling (ICAPS) Workshop on Explainable Planning*, 67–75.

Sreedharan, S.; Srivastava, S.; and Kambhampati, S. 2021. Using state abstractions to compute personalized contrastive explanations for AI agent behavior. *Artificial Intelligence*, 301: 103570.

Sreedharan, S.; Srivastava, S.; and Kambhampati, S. 2025. Explain it as simple as possible, but no simpler–Explanation via model simplification for addressing inferential gap. *Artificial Intelligence*, 340: 104279.

Sreedharan, S.; Srivastava, S.; Smith, D.; and Kambhampati, S. 2019. Why can't you do that hal? explaining unsolvability of planning tasks. In *International Joint Conference on Artificial Intelligence*.

Tierney, K.; Coles, A.; Coles, A.; Kroer, C.; Britt, A.; and Jensen, R. 2012. Automated planning for liner shipping fleet repositioning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 22, 279–287.

Vasileiou, S. L.; and Yeoh, W. 2023. PLEASE: Generating Personalized Explanations in Human-Aware Planning. In *ECAI 2023*, 2411–2418. IOS Press.