Explaining Plan Quality Differences

Benjamin Krarup[1], Amanda Coles[1], Derek Long[1], David E. Smith[2]

[1]Department of Informatics, King's College London, London, UK
[2]PS Research, Los Altos Hills, CA, USA
benjamin.krarup@kcl.ac.uk, amanda.coles@kcl.ac.uk, derek.long@kcl.ac.uk, david.smith@psresearch.xyz

Abstract

We describe a method for explaining the differences between the quality of plans produced for similar planning problems. The method exploits a process of abstracting away details of the planning problems until the difference in solution quality they support has been minimised. We give a general definition of a valid abstraction of a planning problem. We then give the details of the implementation of a number of useful abstractions. Finally, we present a breadth-first search algorithm for finding suitable abstractions for explanations; and detail the results of an evaluation of the approach.

1 Introduction

In mixed-initiative planning settings, human 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 compared with the version generated without the constraint, or the problem could even become unsolvable. In each of these cases it would be useful to have accompanying explanations for why additional constraints lead to a difference in plan quality.

The setting in which these types of explanations are useful does not have to be a mixed-initiative setting. We assume that the setting is as follows: there is a planning model, Π, a solution plan, π, for Π, a constraint, c, which π does not satisfy, and a solution plan, π', for $\Pi + c$, where there is a difference in the quality of π and π'. A special case of this is where π' does not exist, that is the model $\Pi+c$ is unsolvable. As in Krarup *et al.* (2021), we refer to this process as *model restriction*. We restrict planning models such that they admit only solutions that obey a certain constraint. The problem is then to explain why there is a difference in the quality of π and π'. We assume that an explanation of the form "the difference is because of the constraint c'' is not helpful.

While there is longstanding interest in explanation in AI, most work on explanation of plans (XAIP) is relatively recent. Fox et al. (2017) highlight contrastive 'why' questions as being important for plan explanation, and describe different types of these questions and possible responses. Chakraborti et al. (2017) adopt the position that explanation is a model reconciliation problem – namely, that the need for explanation is due to differences between the agent's and the human's model of the planning problem. Eifler et al. (2020) approach answering contrastive questions by explaining the reason that a contrast case *B* was not in, or a feature of, the plan by using the properties that would hold if *B were* the case. Kasenberg et al. (2019) focus on justifying an agent's behaviour in deterministic Markov decision problems.

We are not aware of any work that focusses on explaining the difference in the quality of plans. However, there is work on explaining why planning models are unsolvable. Gobeldecker et al. 2010 focus on finding changes to the initial state that would make the planning problem solvable, and provide an algorithm to produce these excuses in a reasonable time. Sreedharan *et al.* use abstractions to explain the unsolvability of planning problems, as opposed to explaining the difference in quality of problems. They do this by considering relaxations of the planning problem until a solution can be found, and then looking for landmarks of this relaxed problem that cannot be satisfied in less relaxed versions of the problem. They assume that the existence of certain predicates cause a planning problem to be unsolvable. Thus the only abstraction they use is the removal of predicates from the model. We recognise a larger set of useful abstractions for AI planning systems, especially those that have languages to specify temporal characteristics of plans.

An explanation for why two similar problems produce different quality plans should focus on the essential characteristics of the problems. For example, in a delivery problem, a user might ask why a particular truck was used rather than an alternative. The answer might be that the selected truck is better because of a weight limit on a bridge, or refrigeration properties of the truck. Inspired by Lipton's difference condition (1990) we want to find and explain more accurately the causal differences between the original and hypothetical plans. Finding this kind of explanation requires abstracting away unimportant details like drivers, cargo, and perhaps route details. We show how these explanations can be generated by abstracting features of the planning problem until the two plans become *equi-quality* – that is, of equal or similar quality. We can 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.

The primary contribution of this paper is to define a framework for the use of abstractions to explain plan quality differences. In this paper, we first define a valid abstraction for a planning model, these can be used to extend this work for new abstractions. We then introduce a running example we will use to motivate some useful abstractions. We formalise the implementation of a number of useful abstractions for explaining quality differences of temporal plans. We show how these abstractions fit our definition of an abstraction. We detail a proof-of-concept breadth-first search algorithm for finding suitable abstractions for explanation, and present the results of an experiment evaluating it.

2 Planning Formalism

Definition 1. *A **planning model** is a pair $\Pi = \langle D, Prob\rangle$. The domain $D = \langle Ps, Vs, As, arity\rangle$ is a tuple where Ps is a finite set of predicate symbols, Vs is a finite set of function symbols, As is a set of action schemas, called operators, and $arity$ is a function mapping all of these symbols to their respective arity. The problem $Prob = \langle Os, I, G, M, T\rangle$ is a tuple where Os is the set of objects in the planning instance, 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.*

A set of atomic *propositions* P is formed by applying the predicate symbols Ps to the objects Os. One proposition p is formed by applying an ordered set of objects $o \subseteq Os$ to one predicate ps, respecting its arity. This process is called "grounding" and is denoted with $ground(ps, \chi) = p$, where $\chi \subseteq Os$ is an ordered set of objects. The inverse of this function $ground^{-1}(p) = \langle ps, \chi\rangle$ returns the predicate symbol and objects. Similarly the set of *ground functions* V are formed by applying the function symbols Vs to Os.

A state s consists of a time $t \in \mathbb{R}$, a logical part $s_l \subseteq P$, and a numeric part s_v that describes the values for the ground functions at that state. The initial state I is the state at time $t = 0$. We use the function $S(\Pi)$ to denote the state space for a model Π, i.e. all states reachable from the initial state in Π. The goal $G = g_1, ..., g_n$ is a set of propositions, including a subset of the logical state variables, P, and a set of constraints over numeric state variables in V, that must hold at the end of an action sequence for a plan to be valid.

Similarly to propositions and functions, the set of ground actions A is generated by the substitution of objects for operator parameters. Each ground action is defined as follows:

Definition 2. *A **ground action** $a \in A$ has a duration $Dur(a) = \langle lb, ub\rangle$ which constrains the length of time that must pass between the start and end of a; a start (end) condition $Pre_\vdash(a)$ ($Pre_\dashv(a)$) which must hold in the state that a starts (ends); an invariant condition $Pre_\leftrightarrow(a)$ which must hold throughout the entire execution of a; add effects $Eff(a)^+_\vdash, Eff(a)^+_\dashv \subseteq P$ that are made true at the start and end of the action respectively; delete effects $Eff(a)^-_\vdash, Eff(a)^-_\dashv \subseteq P$ that are made false at the start and end of the action respectively; and numeric effects $Eff(a)^n_\vdash$, $Eff(a)^n_\leftrightarrow$, $Eff(a)^n_\dashv$ that act upon some $v \in V$, at the start of an action, continuously over the entire execution of an action, and the end of an action.*

For ease of notation we allow access to multiple types of effects or preconditions through the ground action functions at once. For example for some ground action a, Eff^+ denotes all add effects of a, $Pre_{\vdash\dashv}(a)$ denotes all start and end preconditions of a but not invariant conditions, $Eff(a)$ denotes all effects of a including numeric effects, etc.

A plan is a sequence of grounded actions, $\pi = \langle a_1, a_2, \ldots, a_n\rangle$ each with a respective time denoted by $Dispatch(a_i)$ and duration $Dur(a_i)$. A valid plan is a plan that transforms the initial state from I into a state s such that $s \models G$; such that the start and end preconditions of all actions are satisifed at the time they start/end; all invariant conditions hold throughout the durations of each action, and all actions execute for their respective durations.

The plan-metric function is, by default, the makespan of the plan. Therefore usually, and throughout this paper, the higher the metric of the plan the worse the quality of the plan. More generally, the metric assesses plan quality by taking into account both the extent to which a plan respects user preferences and also the costs associated with the choices of action or combinations of actions within a plan. It is often the case that plans fail to meet expectations because of a mismatch in the way that plans are evaluated.

Timed initial literals (TILs) were introduced as part of PDDL2.2 (Hoffmann and Edelkamp 2005), allowing an initial state to include simple effects triggered at specified times, regardless of the plan. Each timed initial literal $t \in T$ is a tuple $t = \langle t_{time}, t_v\rangle$ where t_v is a proposition which becomes true or false, or a numeric effect which acts upon some $n \in V$ at the time t_{time}.

3 Food Delivery Problem

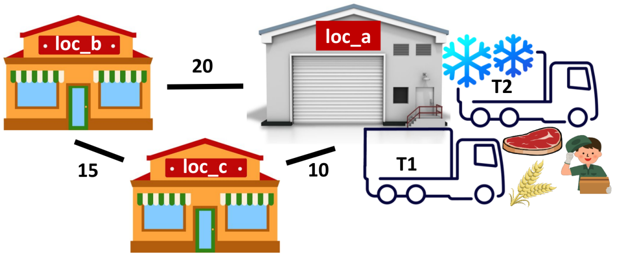

Figure 1: A diagram of the food delivery domain.

As a reference example, we use a simplified version of a food delivery domain. There are two trucks: truck T1 is a normal truck, whereas truck T2 is refrigerated. There is only one driver. The goal of the problem is to deliver the meat and cereals packages to their respective locations. The meat must be delivered to loc_b and the cereals to loc_c. Both trucks, the meat, and the cereals are initially at loc_a. It takes 20 minutes for a truck to move between the loc_a and loc_b, 15 minutes from loc_b and loc_c, and 10 minutes between loc_a and loc_c. The meat will spoil after 22 minutes, unless it is refrigerated. The cereals do not spoil.

For simplicity, we define two possible routes. Route 1 is any plan in which we load the meat and cereals packages into any truck, drive from loc_a to loc_b and deliver the meat, and finally drive from loc_b to loc_c and deliver the cereals. Route 1 takes 35 minutes. Route 2 is any plan in which we load the meat and cereals packages into any truck, drive from loc_a to loc_c and deliver the cereals, and finally drive from loc_c to loc_b and deliver the meat. Route 2 takes 25 minutes. Note, not every route is necessarily possible with every truck. The optimal plan is to perform Route 2 with the refrigerated truck T2.

4 Abstraction

Using model restriction as described in Section 1, we want to explain the difference in the quality of plans π and π' based on the addition of a constraint c to a planning model Π, yielding $\Pi + c$. An explanation of the discrepancy in plan quality should consist of elements of the planning problem (apart from c) that cause the discrepancy. An abstraction of a planning model is a relaxation of the problem that the planning model describes. A planning model Π' is an abstraction of Π if every solution of Π is a solution of Π'. Therefore, if the plans π and π' are not of the same quality; and the plans, π_α and π'_α, for the same problems abstracted with α are the same quality, then we can say that what was abstracted (relaxed) in the problem is a cause for the difference in the quality of the plans that can be produced for Π and $\Pi + c$ as the abstraction is what extended the plan space to include the equi-cost plans. Therefore, to find these causes we abstract away elements of both of the planning models, Π and $\Pi + c$, until $M(\pi) = M(\pi')$.

There is a large body of work on abstraction in planning. The majority of this work focuses on the use of abstractions for computing heuristics. A planning problem can be abstracted, making it easier to solve, this abstracted solution can then be used as an estimate for the actual solution. One abstraction for computing a heuristic is the relaxed planning graph heuristic (Hoffmann and Nebel 2001), where all delete effects are removed from the planning model to make the problem easier to solve. This is then used to generate a relaxed plan that provides an estimate of distance to a goal state in the original problem. Another example of an abstraction for computing heuristics is merge-and-shrink heuristics (Helmert et al. 2014), where a planning model is separated into transition graphs on each proposition. These transition graphs can then be merged and shrunk until a good estimate for the total transition system implied by the original planning model is found that can then be used for generating a heuristic. Helmert et al. (2007) define what a general abstraction is, for providing flexible abstraction heuristics, in the context of AI planning based on a labeled state transition system that maps to the semantics of a planning system.

In this section we propose a more general approach to abstraction, where state merging is one realisation, based on the idea that abstraction is a relaxation. We define the space of legal abstractions using a labeled state transition system (LST), which represents the state space corresponding to the grounding of a planning problem, and a simple temporal network (STN), which represents the temporal constraints and orderings on possible solutions of a planning problem. We rigorously define what a valid abstraction of a planning model is as we later give the implementations of model surgeries that abstract planning models. We use the definitions in this section to prove that these model surgeries are valid abstractions. Abstractions that are not valid could make planning problems harder to solve, and so would not be useful for providing explanations.

The states of a planning model and transitions between them via applying actions can be modeled with an LST.

Definition 3. *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 transitions. A path, π, in τ is a pair in $S \times L^*$, consisting of a state 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, $(s_i, l_i, s_{i+1}) \in T$.*

Given an LST τ we define an abstraction, τ', as follows:

Definition 4. *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, $\langle x, l, y\rangle \in T$, from a reachable and relevant state, $x \in S$, there is $\langle f(x), l, f(y)\rangle \in T'$.*

An LST can be derived from a planning model:

Definition 5. *Let Π be a planning model; the LST derived from Π is $\tau = \langle S, L, T\rangle$. S is the set of all valuations of valid groundings of Ps and Vs. L is the set of labels corresponding to the ground actions A. $T = \{(s, o, s') | s \in S$, the ground action o is applicable in s, s' is the state after application of o to $s\}$. We define the function $\sigma(\Pi) = \tau$ to denote the derivation of an LST from a planning model Π.*

In planning problems, only states that can be traversed to from the initial state are *reachable* and only states that support paths to a goal state are *relevant* to the solution:

Definition 6. *Given an LST, $\tau = (S, L, T)$, a* problem *is a pair, (I, G), such that $I \in S$ and $G \subseteq S$, and these are referred to as the initial state and goal set of the problem.*

Definition 7. *Given an LST, $\tau = (S, L, T)$, and a* 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 in from s to some state $g \in G$ in τ.*

Instead of thinking about abstractions of boundless LSTs, for planning problems we can think of abstractions of LSTs with relevant problems. Definition 4 can be generalised to define an abstraction that operates over all problems in some family of problems, $\mathcal{P}$, for a given LST.

Definition 8. *An LST, $\tau' = (S', L, T')$, is a* general abstraction *of LST, $\tau = (S, L, T)$, with respect to a set of problems, $\mathcal{P}$, if τ' is an abstraction of τ for every problem, $(I, G) \in \mathcal{P}$.*

The next set of corollaries trivially follow from the set of definitions above. These are useful for our presented proofs in Section 5 and for proving some properties about abstractions that are generally useful, and expected from the definition of an abstraction.

Corollary 1 is very informative for abstractions for planning problems. Many relaxations of a planning problem will add transitions to the LST of the planning problem. Through this corollary, we can know that these suite of relaxations are all valid abstractions. Note that not all relaxations to a planning problem are trivially valid abstractions due to this abstraction. For example, if we remove negative effects from a planning problem but we allow negative preconditions, then this will remove edges in the derived LST. This is, of course, not the case if the modelling language does not allow negative preconditions where this relaxation would be a valid abstraction trivially proved by this corollary.

Corollary 1. *A second specific type of abstraction that falls trivially within those defined in Definition 4 is edge insertion abstraction: the addition of transitions to an LST, with f as the identity function, certainly contains all transitions in T.*

Corollary 2 is a more formal way of denoting this effect removal abstraction that is common in model relaxation.

Corollary 2. *For a planning model Π with an initial state I, goal state G, and ground actions A we can remove an effect $e \in \mathit{Eff}^{+-}_{\vdash\dashv}(a) \cup \mathit{Eff}^{n}_{\vdash\leftrightarrow\dashv}(a)$ for some action $a \in A$, to produce a planning model Π'. The LST τ' with the problem (I, G) derived from Π' is an abstraction of the LST τ with the problem (I, G) derived from Π, if e does not make any reachable and relevant condition $c \in Pre_{\vdash\leftrightarrow\dashv}(a')$ true for any $a' \in A$ or satisfy any goal $g \in G$.*

Corollary 3 is similarly informative. As abstractions of planning problems are relevant to a problem, (I, G), then any relaxation that affects only states that are not relevant to the solution of the problem, of which there are often many, is a valid abstraction shown by this corollary.

Corollary 3. *Removing states s and transitions $\langle s, l, y\rangle$ where s is not reachable or relevant for any problem $\mathcal{P}$ is a valid abstraction.*

Finally, Corollary 4 shows that state merging is valid according to our more general definition of an abstraction. This is useful as state merging abstractions have been long used as powerful tools in planning. However, we do not rely on this corollary for any of our proofs.

Corollary 4. *The state merging abstraction is an abstraction according to Definition 4 using the following construction: states $u, v \in S$ are merged into the new state, m, by taking $S'{=}S \setminus \{u, v\} \cup \{m\}$ and $f(s){=}s$ if $s \notin \{u, v\}$, $f(u){=}f(v){=}m$, and $T' = \{\langle f(s), l, f(s')\rangle | \langle s, l, s'\rangle \in T\}$. This is an abstraction by construction.*

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

An STN is constructed over an LST, τ, as follows: given a finite sequence of labels in τ, $\{a_1, a_2, ..., a_n\}$. the corresponding STN has $n + 2$ vertices, labelled I, a_i (for $i = 1...n$) and G. The edges of the STN comprise an edge from a_1 to I and from a_{i+1} to a_i, each weighted $-\epsilon$ (where ϵ separates interfering actions), an edge from G to a_n weighted 0, and, for each pair of labels, a_i and a_{i+j}, that represent the start and the end of the same durative action instance, edges from a_{i+j} to a_i weighted with $-Dur(a_i)[0]$ and from a_i to a_{i+j} weighted $Dur(a_i)[1]$, i.e. for the edge e_1 from a_i to a_{i+j}, $l(e_1) = Dur(a_i)[1]$, and for the edge e_2 from a_{i+j} to a_i, $l(e_2) = -Dur(a_i)[0]$. 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.

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 model 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. In this model, 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 then define an abstraction of an STN:

Definition 10. *An STN $G'{=}\langle V', E', L'\rangle$ is an abstraction of STN $G{=}\langle V, E, L\rangle$, where $V' \subseteq V$ and $E' \subseteq E$, and if for any two vertices $v, v' \in V$ that are connected by an edge $e \in E$ where there is a label l for e in L, we have $l' \in L'$ where $l'(e) \leq l(e)$, 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 model is based on their derived LST and STN:

Definition 11. *A planning model Π' is an abstraction of Π if the LST τ' and the STN G' derived from Π' are abstractions of the LST τ and STN G derived from Π.*

In a perfect world, we would be able to explain why a model is not solvable, by adding states, labels, and transitions to the LST of a planning model. The LST would have to represent the entire state space of a problem. LSTs can be large even for simple planning problems (Helmert 2009).

Not only can it become infeasible to explicitly represent the LSTs for these problems, but it can become intractable to realise what modifications to the LST we must make for precise abstractions. We instead make changes to the lifted representation of the planning problem with model surgeries. This can simplify the abstraction process. However, it leads to abstractions having potentially larger affects to the state space than expected.

5 Abstraction Implementation

In this section we describe the implementation of abstractions that we have identified as important for explanation. We motivate each abstraction's use in explanation with an example, we give the formal process of performing the abstraction on a planning model, and show how these fit our definition of abstraction. We prove proposition 1 for each abstraction α. This proposition must hold to guarantee that the model surgeries we propose do not reduce the number of valid plans for a model, and therefore can be used to search for valid solutions or solutions of better quality.

The abstractions we present here are for temporal planning problems. However the approach is general and can be used with other abstractions in classical, numeric, temporal, and temporal/numeric problems. Our approach requires only that an abstraction be valid, our LST/STN methodology for proving the validity of an abstraction can be used.

Proposition 1 (Valid Abstraction)**.** *Given an LST $\tau = \langle S, L, T\rangle$ and STN $G = \langle V, E, L\rangle$ both derived from a planning model Π and an LST $\tau' = \langle S, L', T'\rangle$ and STN $G = \langle V', E', L'\rangle$ derived from the planning model Π_α, τ' is a valid abstraction of τ and G' is a valid abstraction of G, and therefore Π_α is a valid abstraction of Π.*

Each proof of a valid abstraction in this section follows the same approach. Definition 11 defines what it means for a planning problem to be an abstraction of another. Each proof consists of showing that for τ, τ', G, and G' derived from Π and $\alpha(\Pi)$, τ' is an abstraction of τ and G' is an abstraction of G and therefore $\alpha(\Pi)$ is an abstraction of Π.

The example we use to motivate each abstraction's use in an explanation is: Given the example spoken about in Section 3 with the plan being presented to a user as one in which truck T2 performs route 2, the user might instead prefer truck T1 to be used. Through the system of model restriction we can force the planner to ensure that the truck T1 is used throughout the plan. The resultant plan will consist of truck T1 performing route 1. This is because truck T1 cannot perform route 2 as the meat will spoil at time 22 and it is not possible to extend the life of the meat in the unrefrigerated truck T1. Route 2 takes only 25 minutes. However more than 22 minutes would have passed before the meat was delivered and so it would no longer be fresh. This new plan π takes 35 minutes compared to the original 25 minutes. The example will be the same for each abstraction but the explanations from which they are derived are different.

5.1 Abstracting Predicates

Through the use of abstractions we can determine the cause of the disparity in solution quality. If we abstract away the predicates, $ps \in Ps$, that are responsible for modelling the need for refrigeration, then we can produce a plan, π', in which truck T1 can perform route 2. This is a plan of the same quality as the original plan, but we are instead using truck T1 rather than truck T2 as the user expected. We can therefore produce an explanation: {Remove predicate: (refrigerated t - truck)}, which combined with the constraint added and the plans allows the user to infer that if there were no need for refrigeration, then truck T1 could be used such as in π', otherwise our new plan will be π which is slower by 10 minutes. Abstracting a predicate consists of removing the predicate from everywhere that is appears in the model.

The formal process for abstracting a predicate from a planning model is as follows. Given a planning model, Π, and a set of predicates, $ps \subseteq Ps$, the abstracted model is $\Pi_{ps} = \langle D, Prob\rangle$, where $D = \langle Ps \setminus \{ps\}, Vs, As', arity\rangle$ and $As' = \{a' | \forall a \in As : Dur(a') = Dur(a), Pre(a') = Pre(a) \setminus \{ps\}, Eff(a') = Eff(a) \setminus \{ps\}\}$; and $Prob = \langle Os, I \setminus \{p\}, G \setminus \{p\}, M, T\rangle$, where $Prop(ps) = p$ and $Prop$ takes a set of predicates and returns each of the propositions in P that were formed from it's grounding, i.e. for a set of predicates ps, $Prop(ps) = \{p \in P | \exists ps' \in ps, \chi \subseteq Os : ground^{-1}(p) = \langle ps', \chi\rangle\}$.

Proof. Removing a predicate ps from the planning model is a valid abstraction. Removing a predicate obviously causes all preconditions involving that predicate to be removed. However, it also removes all effects (positive and negative) on that predicate, but these effects no longer matter, since no transition depends on them. This operation is equivalent to merging states that are otherwise identical except for the presence or absence of ps.

This can be seen by realising that, for the LST τ with a set of problems $\mathcal{P}$, the mapping $f(s) = s \setminus \{ps\}$ maintains that for every transition $\langle x, l, y\rangle \in T$ from a reachable and relevant state s with respect to the problems in $\mathcal{P}$, $\langle f(x), l, f(y)\rangle \in T'$. □

5.2 Abstracting Preconditions

In some cases abstracting away an entire predicate may be extreme. Instead we can abstract away certain preconditions for actions. If we abstract away the preconditions, $ps \in Ps$, that are responsible for checking that the produce is fresh before it is delivered, then we can produce a plan π' in which truck T1 can perform route 2. We can therefore produce an explanation: {Remove precondition: (fresh p - produce)}, which would allow the user to infer that if the produce did not need to be fresh for it to be delivered, then truck T1 could be used such as in π', otherwise our new plan will be π which is slower by 10 minutes. An abstraction of a precondition consists of removing a predicate from all preconditions that the predicate appears in.

The formal process of abstracting a precondition from a planning model is as follows. Given a planning model, Π, and a set of predicates, $ps \subseteq Ps$, the abstracted model is $\Pi_{pre(ps)} = \langle D, Prob\rangle$, where $D = \langle Ps, Vs, As', arity\rangle$ and $As' = \{a' | \forall a \in As : Dur(a') = Dur(a), Pre(a') = Pre(a) \setminus \{ps\}, Eff(a') = Eff(a)\}$.

Proof. Removal of preconditions from actions is a valid abstraction. This is because removing preconditions manifests as adding transitions in the LST, as can be seen from Definition 5 (*the ground action o represents is applicable in s*), so by Corollary 1, it is a valid abstraction.

By Corollary 1 this is an abstraction with $T' = T \cup \{\langle s \setminus \{ps\}, a, Eff_a(s \setminus \{ps\})\rangle | \langle s, a, Eff_a(s)\rangle \in T\}$ where $Eff_a(s)$ is the result of updating s with the effects of a. □

5.3 Abstracting Durations

In our example, if we abstracted the duration of the action for driving the truck, then we can produce a plan π' in which truck T1 can perform route 1 quicker. An abstraction of an action's duration involves editing the action's duration constraint so that the planner can select any positive duration.

Given a planning model, Π, and action schemas, $as \subseteq As$, the abstracted model is $\Pi_{as} = \langle D, Prob\rangle$, where $D = \langle Ps, Vs, As', arity\rangle$ and $As' = As \setminus \{as\} \cup \{a' | \forall a \in as : Dur(a') = \langle 0, \text{inf}\rangle, Pre(a') = Pre(a), \mathit{Eff}(a') = (a)\}$.

Proof. Our abstraction has no effect on the derived LST: $\tau == \tau'$, there is therefore no effect on the vertices and edges in the STN: $V' == V$ and $E' == E$, only $L' \neq L$.

For any two vertices $v_i, v_{i+1} \in V$ connected by an edge $e_1 \in E$ and v_{i+1}, v_i connected by $e_2 \in E$, then $v_i, v_{i+1} \in V'$ and $e_1, e_2 \in E'$ and $l(e_1)' >= l(e_1)$ and $l(e_2)' <= l(e_2)$. This is because the labels in the STN are defined by the duration's of actions, the smallest possible duration of an action is 0, and the largest is inf. Therefore the labels in G are contained within the labels of G'. □

5.4 Abstracting Timed-Initial-Literals

Rather than abstracting the time it takes to drive the truck, so that we can deliver the meat before it spoils, we could abstract the TIL that is responsible for the meat spoiling. Through this abstraction we can produce a plan π' in which truck T1 takes route 2. We can still deliver the meat when we arrive to location c in the unrefrigerated truck because it would not have spoiled. We can therefore produce an explanation: {Remove TIL: (at 22 (not (fresh meat)))}, which combined with the constraint added and the plans allows the user to infer that if the meat did not stop being fresh after 22 minutes, then truck T1 could be used such as in π', otherwise our new plan will be π which is slower by 10 minutes. An abstraction of a TIL consists of creating an action that models the TIL. This action can be performed at any time and will have the same effect as the TIL.

Given a planning model, Π, and a set of TILs, $t \subseteq T$, the abstracted model is $\Pi_t = \langle D, Prob\rangle$, where $D = \langle Ps', Vs, As', arity\rangle$ and $Ps' = Ps \cup \{p_t, d_t | \forall t\}$, and $As' = As \cup \{a_t | \forall t : Dur(a_t) = \langle 0, 0\rangle, Pre(a_t) = \{p_t\}, \mathit{Eff}^x(a_t) = \{t_v\}, \mathit{Eff}^-(a_t) = \{p_t\}\}$, x is + if the TIL is positive and - otherwise; and $Prob = \langle Os, I, G \cup \{d_t | \forall t\}, M, T \setminus t$.

Proof. The abstraction of a TIL consists of modelling it as an action, which is exactly how TILs are handled in the LST and STN. This is an exact copy of the TIL in the LST and therefore the TIL is not changed and therefore $\tau == \tau'$. This differs in the LST only in that the action can now be executed at any time, i.e. $V' == V$ and $E' == E$, only $L' \neq L$. In the same way as the proof in Section 5.3 this is a valid abstraction as per Definition 10. □

6 Searching for Suitable Abstractions

As described in Section 1 we consider the problem where we have a planning model, Π, with a solution, π, and a constrained model, $\Pi + c$, which we will call Π', with a solution, π', and there is a difference in quality between π and π'.

We consider an abstraction, α, of a model, Π', to be suitable as part of an explanation if both the solution to the abstracted constrained model Π'_α, π'_α, and the abstracted original model Π_α, π_α, have costs, $M(\pi'_\alpha)$ and $M(\pi_\alpha)$ and $|M(\pi'_\alpha) - M(\pi_\alpha)| < n$. Where n is a user-defined bound below which we consider plans to be *equi-cost*. We do not insist on a difference of zero because it might not be possible to find a plan of exactly equal cost.

This creates a suitable explanation as it abstracts details from both of the models until they produce equi-cost plans. We can therefore say that it is those details that cause the discrepancy between the quality of the solutions for the original and constrained models. However, there may be many possible abstractions that produce equi-cost plans. To distinguish between these we are guided by the principle that we want to maintain, as much as possible, the most important and relevant structures of the problem. To this end we prioritise abstractions first by their size (fewer composed abstractions is better) and then by the similarity of the solutions to the abstracted and original problems, i.e the abstraction with the smallest $|M(\pi_\alpha) - M(\pi)|$.

6.1 Search

In this section we propose a search algorithm to identify correct abstractions that produce informative explanations.

The approach we take is to search over a poset of possible abstractions. Definition 12 allows us to determine a poset given a planning model, Π, the set of possible abstractions, α, as $L(\Pi, \alpha) = (\mathcal{M}_\Pi, \leq)$. This gives an ordering on the models that can be reached through the set of abstractions α. This underpins our approach of abstraction for explanation.

Definition 12. *Given a planning model,* Π*, and a set of abstractions,* α*, the collection of* associated *models,* $\mathcal{M}_\Pi$*, is the closure of the image of* Π *under the application of the abstractions,* α*. The partial ordering on* $\mathcal{M}_\Pi$ *is defined by: for* $\Pi_1, \Pi_2 \in \mathcal{M}_\Pi$, $\Pi_1 \leq \Pi_2$ *iff* $\rho(\Pi_1) \subseteq \rho(\Pi_2)$*. This determines the poset* $L(\Pi, \alpha) = (\mathcal{M}_\Pi, \leq)$*.*

For an abstraction poset, elements higher up in the poset are more abstract than those at the bottom. The abstractions that will form $\mathbb{A}$ in our search are all the possible abstractions defined in Section 5. We do not construct this poset a priori, we generate parts of the poset during search time.

We provide a search algorithm for finding suitable abstractions for explanations in Algorithm 1. This algorithm implements a bottom-up breadth first search. We start from the constrained model and apply all possible abstractions at each level of the poset before traversing up the poset in a single step. This is repeated until we reach a model that produces an equi-cost plan to the original solution. This algorithm takes as input the constrained model, Π', the set of possible abstractions, $\mathbb{A}$, (see Sec. 5), the original plan, π, and a real value specifying the bound for equi-cost plans, n. A queue is constructed, $queue\mathbb{A}$, from the abstractions, $\mathbb{A}$, the current abstraction, $curr_{\mathbb{a}}$, is then dequeued. The first model that abstractions are applied to, $curr_\Pi$, is a copy of the model Π'. This is abstracted with $curr_{\mathbb{a}}$ to give the abstracted model, $\Pi'_{\mathbb{a}}$, the function $abstract$ takes an abstraction, $\mathbb{a}$, and a model, Π, and returns an abstracted model, $\Pi_{\mathbb{a}}$.

A queue, $queue\Pi$, is constructed from the abstracted model, $curr_{\mathbb{a}}$, that model is then solved, the function, $solve$, takes a model, Π, and returns a valid solution plan for, Π. The algorithm then checks if the abstracted plan and original plan are equi-cost, the function $bound$ is as described in the introduction to this section. The algorithm then loops, repeating this process for each model in the queue, for each abstraction in the queue until there are no models left in the queue. In each iteration there is a further check that an abstraction, $curr_{\mathbb{a}}$, has not already been applied to the current model, $curr_{\Pi}$.

Algorithm 1: Breadth first search over an abstraction poset $L(\Pi', \mathbb{A})$ to find a set of abstractions $\mathbb{a} \in \mathbb{A}$ supporting a plan π' such that $|M(\pi') - M(\pi)| < n$

```
Data: $\{\Pi, \Pi', \mathbb{A}, n\}$
Result: $\mathbb{a} \subseteq \mathbb{A} \vee FAIL$
$queue\mathbb{A} \leftarrow queue(\mathbb{A})$;
$curr_{\mathbb{a}} \leftarrow dequeue(queue\mathbb{A})$;
$\Pi_{\mathbb{a}} \leftarrow abstract(\Pi, curr_{\mathbb{a}})$;
$\Pi'_{\mathbb{a}} \leftarrow abstract(\Pi', curr_{\mathbb{a}})$;
$queue\Pi \leftarrow queue(\Pi_{\mathbb{a}})$;
$queue\Pi' \leftarrow queue(\Pi'_{\mathbb{a}})$;
$\pi \leftarrow solve(\Pi_{\mathbb{a}})$;
$\pi' \leftarrow solve(\Pi'_{\mathbb{a}})$;
if $|M(\pi') - M(\pi)| < n$ then return $\mathbb{a}$ ;
while $\neg empty(queue\Pi')$ do
    while $\neg empty(queue\mathbb{A})$ do
        $curr_{\mathbb{a}} \leftarrow dequeue(queue\mathbb{A})'$;
        if $\neg applied(\Pi'_{\mathbb{a}}, curr_{\mathbb{a}})$ then
            $\Pi_{\mathbb{a}} \leftarrow abstract(curr_{\Pi}, curr_{\mathbb{a}})$;
            $\Pi'_{\mathbb{a}} \leftarrow abstract(curr_{\Pi'}, curr_{\mathbb{a}})$;
            $enqueue(queue\Pi, \Pi_{\mathbb{a}})$;
            $enqueue(queue\Pi', \Pi'_{\mathbb{a}})$;
            $\pi \leftarrow solve(\Pi_{\mathbb{a}})$;
            $\pi' \leftarrow solve(\Pi'_{\mathbb{a}})$;
            if $|M(\pi') - M(\pi)| < n$ then return $\mathbb{a}$ ;
    $\Pi_{\mathbb{a}} \leftarrow dequeue(queue\Pi)$;
    $\Pi'_{\mathbb{a}} \leftarrow dequeue(queue\Pi')$;
    $queue\mathbb{A} \leftarrow queue(\mathbb{A})$;
return $FAIL$;
```

Algorithm 1 gives a satisficing solution. It gives the first of abstractions that produces a model whose solution is equi-cost to the original. If we assume that all abstractions are of equal cost, then this would be an optimal solution as we will apply all possible abstractions at each level of the poset before moving on. However, as noted earlier in this section, we can evaluate abstractions based on the distance between the quality of the solution of the abstracted model and the original solution. We can modify the stopping condition of Algorithm 1 to give an optimising search algorithm that evaluates the quality of certain solutions based on this metric.

6.2 The Explanation

The explanation will be composed of the abstractions that were used in order to get a satisfiable plan under the restricted model. The specific explanation will be contextualised by the constraint applied to the original model that lead to a difference in the quality of the solutions. Each abstraction $\mathbb{a}_{curr}$ is popped from the set of abstractions $\mathbb{a}$. The type of the abstraction is then checked, these are each of the abstractions outlined in Section 5. The explanation is then formed by the types of, and the abstractions that allow the problem to produce an equi-cost solution. The user does not need to understand the implementation of these abstractions, which are automatically applied. They need only understand the explanations that are generated. In the next section we give some more examples of these generated explanations.

7 Experiments and Results

In this section we present an empirical evaluation of our approach to finding explanations. Whilst we present results indicating the performance of our algorithm; our focus is on testing the ability of our approach to find explanations. Optimizing the efficiency of the algorithm is left to future work.

Evaluation uses 6 differently structured planning problems. 2 of these problems are based on our running example and the other 4 are instance 1 of Rovers and 2 of Satellite from the International Planning Competition (IPC) (Long and Fox 2003). The 2 explainable problems are the Delivery domain discussed in Section 3 and an augmented version of this domain in which there are two drivers, D1 and D2, and only one driver, D2, has the ability to drive the refrigerated truck, we call this problem **delivery+**.

In Rovers, rovers take samples of soil, rock, and image data to send back to a lander. Not all rovers are equipped for all sample types. Cameras must be calibrated to take images, rock data must be stored before it can be communicated, and soil data can be collected and communicated. The optimal plan for problem 1 is to use rover R1, calibrate the camera and take an image, traverse to and collect the rock sample, store it, and communicate the data; and finally traverse to the soil sample, collect it, and communicate the data.

In our Satellite problem there is a satellite with two instruments able to take photos in different modes. Instrument I1 can take infrared images and I2 can take both infrared and visible images. The goal of the problem is to take images of certain targets in certain modes. The optimal plan is to turn on and calibrate I2 and take the required images.

Each problem is solved, then constraints are added to make the problem unsolvable or to require a plan of worse quality. We search for suitable abstractions using Algorithm 1 set with a bound of $n = 2$. Experiments ran on a 4gb machine with an i7-12800H CPU, and a 4 hour timeout. We use the full set of abstractions detailed in Section 5.

The delivery problem is constrained so that truck T1 must be used, requiring the poorer route 1 to be taken. The delivery+ problem is similarly constrained, but also so that driver, D1, who cannot use the refrigerated truck, must make the deliveries. This is unsolvable.

The Rovers problem is constrained so that the image must be taken before the camera is calibrated. The resulting plan has the image capture in a different place in the solution. The problem is separately constrained by removing the capability of the rover to sample soil or rocks, which is unsolvable.

Domain	Size of $\mathbb{A}$	First/s	Optimal/s	# Suitable	# Optimal	Size	# Nodes	Time/s
Delivery	21	20.04	20.04	489	8	1	585	20
Delivery+	25	220.37	1382.22	307	12	2	868	20
Rovers Worse	64	520.61	1802.31	33	2	1	178	40
Rovers Unsolvable	64	10295.09	10295.09	10	4	2	350	40
Satellite Worse	22	420.31	420.31	11	2	1	52	140
Satellite Unsolvable	22	280.20	280.20	26	2	1	79	140

Table 1: Time taken to find and number of suitable and optimal abstractions for a selection of planning problems.

The Satellite problem is constrained so that instrument I1 must be used to take an image; the plan then uses I1 to take the infrared images and instrument I2 is switched on to take the visible light images. The problem is separately constrained so I2 cannot be used, which is unsolvable.

The results in Table 1 show, for each problem, the number of abstractions at each search level, (the size of $\mathbb{A}$); the time to find the first suitable abstraction (henceforth, abbreviated to *fsa*); the time to find the first optimal abstraction; the numbers of suitable and of optimal abstraction sets for explanations found; the size of the abstraction set; the number of nodes expanded in the search; and finally the time given to solve each problem. This is calculated by finding the largest time taken to solve the original problem, or any of the constrained problems, and doubling it.

The *fsa* for the delivery problem was found in 20.04 seconds; it is also optimal. 585 combinations of abstractions were searched of which 489 were suitable and 8 optimal. The first optimal solution found, of size 1, abstracts away the predicate determining whether meat or cereal is in a truck. Now, the action that prolongs the life of the meat can be used with truck T2 as the meat is no longer required to be in the truck for this action to be performed. The plan is then to use truck T1 to perform route 2 whilst using truck T2 to act as the refrigerator from location loc_a. The explanation produced is: {Remove predicate: (in ?p - produce ?t - truck)}. There are other optimal solutions for this problem. Another is to abstract the TIL determining when the meat spoils. The plan for this abstracted problem uses truck T1 to perform route 2 without worrying about the meat spoiling at all. The explanation is: {Remove TIL: (at 22 (not (fresh meat)))}.

The *fsa* for the delivery+ domain is found in 220.37 seconds and the first optimal abstraction in 1382.22 seconds. 868 combinations of abstractions were searched: 307 were suitable and 12 were optimal. The first optimal solution found, of size 2, abstracts away both the predicate for a truck to be refrigerated and the requirement for a driver to be qualified to drive certain trucks. Then truck T1 can be used in place of the refrigerated truck, and any driver can drive any truck. The plan for this abstracted problem is to use T1 with the driver D1 to perform route 2.

The *fsa* for the rovers problem constrained to produce a worse quality plan was found in 520.61 seconds. The first optimal abstraction was found in 1802.31 seconds. 178 combinations of abstractions were searched, of which 33 were suitable and 2 optimal. The first optimal solution found abstracts away the predicate responsible for ensuring the target of calibration matches is the objective image to be taken. This allows for a camera to be calibrated on any target. This removes the need for the calibration action, so the constraint is trivially satisfied. The plan continues as in the original plan after the camera is calibrated.

The *fsa* for the rovers problem constrained to be unsolvable is found in 10295.09 seconds; it is also optimal. 350 combinations of abstractions were searched: 10 were suitable and 4 optimal. The first optimal solution, of size 2, abstracts away the preconditions of the **sample_rock** and **sample_soil** actions that ensure the rover can sample rock and soil; any rover can then gather rock and soil samples and the plan is then the same as the original. The explanation is: {Remove precondition: (equipped_for_soil_analysis ?r - rover),(equipped_for_rock_analysis ?r - rover)}. This exemplifies the need to search over combinations of abstractions in more complex domains. Here, multiple abstractions were necessary to minimise the quality difference. We note that in this domain, even finding the first solution takes a length of time that would not be reasonable for a rapid-turnaround interactive process; however we note that this would remain useful in safety critical scenarios. For example, ground teams making plans for rovers/satellites usually plan over the course of several hours for a single day of operations, so here waiting longer for an explanation to ensure a robust or more optimal plan is not unreasonable.

The *fsa* for the Satellite problem constrained to produce a worse quality plan is 420.31 seconds; it is also optimal. 52 combinations of abstractions were searched, of which 11 were suitable and 2 optimal. The first optimal solution, of size 1, abstracts the precondition of the **take_image** action ensuring the instrument supports the relevant mode. The instrument I0 can then be used to take images of visible light and the plan for the abstracted model is the same as the original plan, other than using instrument I0 instead of I1.

The *fsa* for the Satellite problem constrained to become unsolvable is 280.20 seconds; it is also optimal. 52 combinations of abstractions were searched of which 26 were suitable and 2 optimal. The first optimal abstraction and resulting plan is the same as the other Satellite problem.

8 Conclusion

In this paper a general definition for abstraction in planning problems is presented. A number of surgeries on planning models are defined, implementing a useful set of abstractions for explanation. A proof-of-concept algorithm is presented and used to find suitable and optimal abstractions for explanation. Building natural language explanations from these abstractions and evaluating them remains future work.

Acknowledgements

This work was supported by the Air Force Office of Scientific Research (AOFSR) under award number FA9550-18-1-0245 and the EPSRC-funded project 'COHERENT' (EP/V062506/1).

References

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.*

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

Eifler, R.; Cashmore, M.; Hoffmann, J.; Magazzeni, D.; and Steinmetz, M. 2020. A New Approach to Plan-Space Explanation: Analyzing Plan-Property Dependencies in Oversubscription Planning. In *Proc. Association for Advancement of AI*, 9818–9826.

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. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5-6): 503–535.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. In *Proceedings of the Seventeenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'07, 176–183. AAAI Press. ISBN 9781577353447.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM (JACM)*, 61(3): 1–63.

Hoffmann, J.; and Edelkamp, S. 2005. The Deterministic Part of IPC-4: An Overview. *Journal of Artificial Intelligence Research*, 24: 519–579.

Hoffmann, J.; and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Kasenberg, D.; Roque, A.; Thielstrom, R.; Chita-Tegmark, M.; and Scheutz, M. 2019. Generating justifications for norm-related agent decisions. In *arXiv preprint arXiv:1911.00226*.

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.

Lipton, P. 1990. Contrastive Explanation. *Royal Institute of Philosophy Supplement*, 27: 247–266.

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