

Contrastive Explanations of Plans Through Model Restrictions

Benjamin Krarup

Senka Krivic

Daniele Magazzeni

Derek Long

King's College London, Bush House, WC2B 4BG, London, UK

BENJAMIN.KRARUP@KCL.AC.UK

SENKA.KRIVIC@KCL.AC.UK

DANIELE.MAGAZZENI@KCL.AC.UK

DEREK.LONG@KCL.AC.UK

Michael Cashmore

University of Strathclyde, Livingstone Tower, G1 1XH, Glasgow, UK

MICHAEL.CASHMORE@STRATH.AC.UK

David E. Smith

PS Research, 25960 Quail Ln, Los Altos Hills, CA 94022, USA

DAVID.SMITH@PSRESEARCH.XYZ

Abstract

In automated planning, the need for explanations arises when there is a mismatch between a proposed plan and the user's expectation. We frame Explainable AI Planning as an iterative plan exploration process, in which the user asks a succession of contrastive questions that lead to the generation and solution of hypothetical planning problems that are restrictions of the original problem. The object of the exploration is for the user to understand the constraints that govern the original plan and, ultimately, to arrive at a satisfactory plan. We present the results of a user study that demonstrates that when users ask questions about plans, those questions are usually contrastive, i.e. "why A rather than B?". We use the data from this study to construct a taxonomy of user questions that often arise during plan exploration. Our approach to iterative plan exploration is a process of successive model restriction. Each contrastive user question imposes a set of constraints on the planning problem, leading to the construction of a new hypothetical planning problem as a restriction of the original. Solving this restricted problem results in a plan that can be compared with the original plan, admitting a contrastive explanation. We formally define model-based compilations in PDDL2.1 for each type of constraint derived from a contrastive user question in the taxonomy, and empirically evaluate the compilations in terms of computational complexity. The compilations were implemented as part of an explanation framework supporting iterative model restriction. We demonstrate its benefits in a second user study.

1. Introduction

Automated planning is being used in increasingly complex applications, and explanation plays an important role in building trust, both in planners and in the plans they produce. A plan is a form of communication, either as a set of instructions to be enacted by autonomous or human agents, or as a proposal of intention communicated to a user. In either case, the plan conveys the means by which a goal is to be achieved, but not the reasons for the choices it embodies. When the audience for a plan includes humans then it is natural to suppose that some users might wish to question the reasoning, intention and underlying assumptions that lead to those choices.

The need for explanations arises when there is a mismatch between a proposed plan and the user's expectation. This might be because the user had not managed to construct a viable plan, or constructed one that did not match the system's proposed plan. Explanations attempt to bridge the gap between these mismatched positions and might be *local*, focusing on the specific proposed plan and its properties, or *global*, focusing on the assumptions on which the plan rests, or the process by which it was constructed.

1.1 An Outline of the Approach

To help guide the reader through this paper, we now outline our overall approach and introduce some of the key concepts.

Following ideas of Smith (2012) we consider explanation to be an iterative process; the user is presented with explanations through a process of iterative plan exploration, according to a five-stage mixed-initiative process illustrated in Figure 1. In this figure, (i) the user asks a question; (ii) a constraint is derived from the user question (forming the *formal question*); (iii) a hypothetical planning model (HModel) is generated which encapsulates this constraint; (iv) the planner is run on the HModel producing a hypothetical plan (HPlan); (v) the HPlan is compared to the original plan resulting in a contrastive explanation that shows the consequences of the user suggestion.

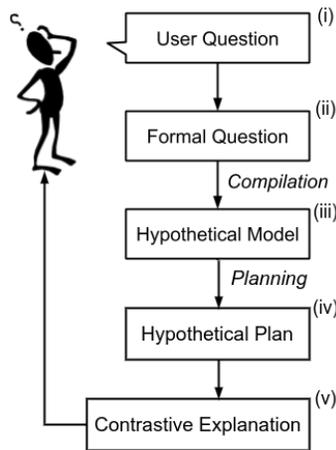


Figure 1: The five-stage iterative process for generating a contrastive explanation from a user question. The hypothetical model is created by compiling the formal question into the planning model (in PDDL 2.1), a planner is used to solve the model and produce a hypothetical plan.

The user can compare plans and iterate the process by changing or refining the question, resulting in a new HModel, and a new HPlan. This allows the user to ask a sequence of questions that progressively *restrict* the original system model. The process ends when the user is either satisfied with the explanation and accepts the plan generated for one of the HModels at some stage in this process, or otherwise decides that there is no value in further exploration. In the latter case, the user might decide that a solution of their own is better than any of the solutions the planner has generated, which could be a consequence

of discrepancies between the user’s model and the model available to the planner. We present this iterative plan exploration process in detail in Section 3.1.

We have adopted this iterative approach because of the following two assumptions and their consequences:

- The user’s domain model and problem, and the limits of their ability (or willingness) to construct detailed plans are all unknown to us.

The iterative process does not require that the planning models used by the planner and the user be the same. However, the formulation of questions does require that the user and the planner share vocabulary, including the names and parameter types of actions and predicates, the names of objects appearing in the problem, and the goal.

- The correctness of the domain model available to the planner is also unknown to us. Furthermore, the planner is necessarily incomplete, since even classical planning is P-Space complete, and planners are expected to produce answers in a small amount of time and memory.

The first assumption leads us to conclude that we cannot truly know the reason for a user’s question about a plan: whether it is puzzlement about unexpected use of actions, confusion at a perceived misapplication of an action, or simply curiosity about how a plan was found. The second assumption implies that we cannot be sure that a failure of the planner to construct a plan means that there is no plan, or that a plan produced by the planner is either correct (with respect to the user’s model) or good. We also do not assume that the user has necessarily formulated an explicit alternative plan. In some cases, the user might not have such a plan in mind and, in that case, the iterative process might simply reflect the user exploring the space of plans around the initial plan in order to gain some insight into the alternatives that exist.

As we will argue in Section 2, supported by a user study, many queries made by a user in interaction with a planner or plan-based system are contrastive (Miller, 2018) — of the form “Why *A* rather than *B*?”. Fox et al. (2017) highlight the *why* query as an important one for XAI, and discuss possible responses. As we structure these questions into a formal query (Figure 1 (ii)), we refer to *B* as a *foil*, and use it to construct a constraint on the original plan. To answer these kinds of questions, one must reason about the hypothetical alternative (Figure 1 (iii)) resulting from the foil *B*, which means constructing an alternative plan for which *B* is satisfied, rather than *A* (Figure 1 (iv)). The definition of the alternative plan, which we refer to as the *Hypothetical Plan (HPlan)* (Definition 4), is detailed and formalised in Section 3.

We take the view that the purpose of an explanation to a contrastive question is to, assist the user in understanding how the original plan differs from a plan for the foil. These differences between plans will include:

- how the actions differ;
- how the actions and plan structure contribute to satisfying the plan metric or preferences and

- how the causal structure differs.

Our existing interface, described in Section 5, highlights the action differences between the plans, addressing the first of these three, but it is left to the user to draw conclusions about the second and third, by inspection of the two plans. This step is shown in Figure 1 as stage (v).

Explanation of automatically generated plans can be seen as a special case of explanation of the output of AI programs in general. Even though the area of Explainable AI Planning (XAIP) is relatively young, there has been considerable work in the field in recent years. Chakraborti et al. (2020) outline the different approaches to XAIP that have emerged in the last couple of years, and contrast them with earlier efforts in the field. They group the approaches for XAIP into two main categories: algorithm-based explanations and model-based explanations. Psychological studies of explanation (see, for example, Mueller et al., 2019) make a distinction between global and local explanations. Algorithm-based explanations typically attempt to explain the underlying planning algorithm so that a user can better understand the workings of the planning system and are therefore global in nature. For example, Magnaguagno et al. (2017) provide an interactive visualisation of the search tree for a given problem. Model-based explanations are algorithm-agnostic methods for generating explanations for the solutions to a planning problem. These can be considered to be global or local explanations depending on whether the user is interested in the model itself, or in explaining particular decisions resulting from the model for a particular problem. The iterative plan exploration process primarily supports local model-based explanations, by focusing on alternative plans for the same problem but, through active exploration of the plan space, the process may also yield insights into properties of the domain model and of the planner itself (Lipton, 1990, 2016; Ribeiro, Singh, & Guestrin, 2016).

Despite the broad interest in explainable AI, the general question of what constitutes an explanation can be surprisingly contentious and difficult to answer. In order to provide a meaningful answer one must be clear about what it is that one is trying to explain. The term XAIP would suggest that we might be trying to explain a plan. However, the problem of trying to explain how or why a plan works is different from the problem of explaining why a plan is a good plan. In the former case, one is trying to help the user understand the correctness of the plan, whereas in the latter case, one is trying to help the user understand why the plan is better than others. In the case of contrastive questions, the shape of the response is more narrowly defined, because the user has suggested a foil. This leads us to consider several cases:

- If the foil is satisfiable, and the HPlan is worse than the original plan (according to the planning metric in our model) then “explanation” would consist of trying to help the user understand how and why the original plan is better than the HPlan. In the absence of explicit information about the user’s model, a reasonable approach is to identify the differences between the original plan and the HPlan and show how those differences influence the plan metric.
- If the HPlan is better than the original plan, then the best thing is to acknowledge that the HPlan appears to be better, and to explain why. In this case the ‘why’ consists

of indicating how the metric has improved, and which differences contribute to the difference in the metric.

- If there are multiple competing objectives, such as competing preferences, then there is the possibility that neither the original or HPlan is superior. In this case, again it seems that an explanation consists of identifying the differences between the two plans and how they influence the metric.
- If the foil cannot be satisfied then we would be interested in explaining why we think the foil is unsatisfiable (according to the model used by the planner). In this paper, we do not consider how best to handle unsolvable planning problems. We simply report the failure to find a plan and leave interpretation to the user. Refinements of this approach are obviously possible (see, for example, Sreedharan et al., 2019).

Automatically distinguishing between the last three cases is, of course, subject to the model and metric being used by the planner. Our assumption is that we do not have the user's model, or know whether the model and metric being used by the planner are correct. As a result, we are not explicitly doing *model reconciliation* (see, for example, Chakraborti et al., 2017 and Sreedharan et al., 2018, discussed in Section 7). Instead, we focus on presenting the HPlan to the user and highlighting the differences between the HPlan and the original plan.

The results of the user study show that users ask questions not only about the choice of actions and objects, but also about the timing of activities. We therefore consider temporal/numeric domains. The constraints that are implied by users' questions can require us to limit activities within particular time windows within the structure of the HPlan, either to prevent or to enforce certain actions appearing within those windows. Of course, not all of the questions a user asks will be focused on the temporal structure of the plan, so the classical plan properties, such as ordering and causal structure, are also relevant to the process.

In Section 6 we present evaluation of the approach we propose, including a second user study designed to observe the value of the iterative plan exploration process we implemented as a tool for supporting users in understanding plans.

The paper ends with a broader discussion of related work (Section 7) and, finally, we draw our conclusions (Section 8).

2. Elicitation of Questions Through a User Study

In this section, we describe a study that we conducted to gain insight into the types of questions that users pose about planning systems, when the model is well known to the user. As we indicated in the introduction, several researchers have observed (e.g: Mueller et al., 2019) that it is useful to draw a distinction between local and global questions which call for local and global explanations, respectively. In the context of a plan, a global question might be asked because the inquirer does not fully understand the model used by the planner, or the process by which it was constructed. In contrast, a local question is one focused on specific decisions made in the plan.

When plan-based control was used to automate drilling (Long, 2018), the process involved a series of stages during which the nature of required explanations evolved. During

initial development users primarily asked global questions to validate their understanding of the model, ensuring its correctness, and building trust in the system. As this trust developed and the model used by the planner became better understood, users were more likely to ask local questions seeking to understand the intention behind specific actions in a plan, or to better understand alternatives to particular choices.

Local questions are asked in a variety of contexts. Domain experts may wish to challenge a decision made by the planner when they possess insight into the domain that they believe can improve upon a sub-optimal plan. Alternatively, users may simply be interested in exploring the space of plans and ask questions to suggest alternative decisions, and better understand their impact. In the former case, a sceptical expert might seek to demonstrate weakness in the way that the system made a decision, while in the latter case the role of the system is promoted to an advisor or aide, with the user relying on the system to support exploration of the space of alternative solutions.

The interrogative word used when asking local questions is *why*, whereas for global questions it is usually *how* or *what*. We hypothesise that when the model is well-known, users ask more local, contrastive *why* questions than global *how* or *what* questions. The null hypothesis and alternate hypothesis, H_0 and H_a for our study are therefore as follows:

H_0 : Users ask an equal distribution of *why*, *how*, and *what* questions about planning scenarios, when the model is well known.

H_a : Users ask more *why* questions than *how* or *what* questions about planning scenarios, when the model is well known.

2.1 Methodology

We designed a study to elicit questions from users about plans. We recruited participants, through a website (<https://www.prolific.co>) that specialises in sourcing eligible subjects, each of which were compensated £10 for their time. We selected a sample size of 15, which is a typical number for this type of study (Chakraborti et al., 2018; Kulkarni et al., 2019) and has been shown to cause data saturation in qualitative studies (Nielsen, 2000; Faulkner, 2003). The participants were from different, non-planning related, backgrounds and professions between the ages of 21 and 39 years old. In the study, participants were presented with three planning scenarios which were described to them in detail. They were first asked to watch a short video (Chen et al., 2019) showing an animation of a planning problem being performed. Once the participants were familiar with the content of the animation they were asked to re-watch the video and write down any questions they had, and (if applicable) a reason for why they asked the question. For each question the participants were told to note down the time during the video that caused it to be asked. The participants were asked to do this twice more with two videos of new planning problems. However, this time they were told to only ask questions that were specific to decisions, or actions that were made in the plan. Participants were asked to re-watch each video until they could not produce any new questions. On average participants asked 11 questions and took 44 minutes to complete the study. From this data, we performed a content analysis to extract a taxonomy of the types of questions that people answered in our scenarios.

Question Type	Video 1	Video 2	Video 3
What?	2	1	3
How?	0	3	2
Why?	65	50	42

Table 1: Frequency of questions asked by participants for each video characterised by Miller’s taxonomy of questions (Miller, 2019).

We used three different scenarios in the study: (Video 1) a family of five must sail to the other side of a river, with some constraints placed on sailing the boat;¹ (Video 2) a logistics problem where six packages have to be delivered to specific locations using trucks and aeroplanes;² and (Video 3) a robot must place different objects into positions on a grid.³ We chose these domains because they are simple to understand and reason about without much participant training, and are varied in what they model.

2.2 Results and Analysis

The results of this study are shown in Table 1. The questions were categorised into Miller’s taxonomy by the interrogative word used in the question, either *what*, *how*, or *why*. Table 1 shows the number of questions in each category in the taxonomy and compares the questions asked in video 1, where users were asked to propose any questions, and videos 2 and 3, where the questions had to be related to the plan.

These results show that users generally want to understand why certain decisions were made by the planning system rather than how the planning system works, or what a specific component of the system’s purpose is. There were 157 instances of contrastive *why* questions, 151 of which were in reference to specific decisions made in the plan. There were only 11 *what* and *how* questions which asked for a deeper understanding of the planner behaviour. Of the questions posed by users, 89.9% were contrastive *why* questions.

Performing a chi-square test, $\chi^2(2, 168) = 273.25$, these results are therefore significant at $p < 0.001$. We can therefore reject our null hypothesis, H_0 , and accept our alternate hypothesis, H_a .

The results in Table 1 support this further when comparing the results of video 1 with videos 2 and 3. This shows that when there are no constraints on the questions users can ask, or when they are explicitly asked to question the plan, in both cases they want to understand why certain decisions were made in the plan.

2.3 Taxonomy of Questions

Following our accepted hypothesis, we focus on providing explanations for *why* questions. Research from the social sciences (Miller, 2018) argues that *why* questions are typically *contrastive*; that is, they are of the form “Why A rather than some hypothetical foil B?”.

1. <https://youtu.be/MSCakpJUcpc>

2. https://youtu.be/tuvgky0f_Bs

3. <https://youtu.be/YX08EIj3oDM>

Question Type	#
FQ1 Why is action A not used in the plan, rather than being used?	17
FQ2 Why is action A used in the plan, rather than not being used?	75
FQ3 Why is action A used in state S, rather than action B?	35
FQ4 Why is action A used outside of time window W, rather than only being allowed within W?	6
FQ5 Why is action A not performed before (after) action B, rather than A being performed after (before) B?	10
FQ6 Why is action A not used in time window W, rather than being used within W?	2
FQ7 Why is action A used at time T, rather than at least some time T' after/before T?	6
FQ8 Non-contrastive or out of scope	17

Table 2: Frequency of questions categorised into the Contrastive Taxonomy. We provide explanations for questions FQ1 - 7.

Contrastive questions capture the context of the question; they provide an insight into what the questioner needs in an explanation (Lewis, 1986). Garfinkel (1982) illustrates this with a story about a famous bank robber, Willie Sutton, who, when asked why he robbed banks, replied “That’s where the money is.” Sutton answered the question “Why do you rob banks rather than other things?”, instead of the question “Why do you rob banks rather than not robbing them?”. The foil was not explicitly stated in the question and so was left ambiguous. Garfinkel argues that explanations are relative to these contrastive contexts, and that they can be made unambiguous by explicitly stating the contrast case.

A contrastive question asked about a plan can be answered with a *contrastive explanation* which will highlight the differences between the original plan and a contrastive plan that accounts for the user suggested foil. Providing contrastive explanations is not only effective in improving understanding, but is simpler than providing a full causal analysis (Miller, 2019). They are also naturally good for comparisons, as we can directly compare the original plan with a plan containing the user foil.

We categorised each *why* question from the three different domains in the user study above into a taxonomy of *Formal Questions* (FQ) which we call the *Contrastive Taxonomy*. The Contrastive Taxonomy is shown in Table 2, and shows the frequency of questions asked by users about the plans produced for the three different domains. This represents a set of questions that are important for a plan-based system to answer. The questions in categories FQ1 to FQ7 are of the form “Why A rather than B?” and are clearly contrastive. They are also local questions because they query decisions made in the plan in terms of actions that were or were not chosen to be performed and when those actions were performed.

We categorised the questions into the Contrastive Taxonomy by splitting the question into the fact and the, sometimes implied, contrast case. For example, take the question, posed by a participant about the first planning situation:

“Why did Son swap with Fisherman?... Fisherman should pick up Daughter”

The fact is that the Son swapped places on the boat with the Fisherman, and the contrast case is that the Fisherman should have picked up the Daughter. If, like in this example, the contrast case was an explicit action which the user expected in place of some other action in the plan, the question was categorised as type FQ3. If the contrast case was implicit or explicitly negating the fact, the question was categorised as either type FQ1 or FQ2, depending on the fact. If the user questioned an ordering between two actions the question was categorised as type FQ5. If the question was in these types FQ1 or FQ2 but referred to a specific time, it was categorised as type FQ4, FQ6 or FQ7. *What* and *how* questions were categorised as type FQ8, as they are not regarded as contrastive questions. We also categorised any question about the video itself, i.e discrepancies in the animation as FQ8. Our proposed compilation techniques do not extend to these types of questions.

Table 2 shows that the most commonly asked questions (type FQ2) are about actions that were performed, rather than absent actions they expected to have been in the plan. However, when users do question why an action they expected did not happen, they are more likely to ask it as an explicit contrastive question with respect to some other action that did happen (type FQ3). Users do not question the times in which actions are performed (types FQ4, FQ6, FQ7), or the ordering of actions (type FQ5) as much as why an action was performed or not. The results show that the majority of user questions are contrastive. The contrast case is more likely to be the negation of the fact (types FQ1, FQ2, FQ4 - 7). However, a significant proportion of the questions specify a specific action as the contrast case that the user expected to have been performed instead of the factual action (type FQ3). This shows that users likely have an idea (mental model) of a plan which they use to question the factual plan. They might question why an action was performed, when it was not part of their ideal plan. Or they might question why an action, that was present in their ideal plan, did not appear in the factual plan.

The small number of questions that were not contrastive or local, *how* (5) and *what* (6) questions, were classed in the final category FQ8. There were also a small number (6) of *why* questions that were classified as out of the scope of this paper. A question was classed as out of the scope of the paper if it was not related to the planning system or the plan produced. For example, a participant questioned the animation system used to visualise the plan execution, "Why did the pink square change to green?". This is not a question about a decision but the inner workings of the animation software used. Notice that these questions were still local and contrastive in nature, just not questions relevant to planning systems and therefore not ones we are concerned with answering.

In Section 4 we present a novel approach to compiling constraints derived from these questions into planning models to demonstrate the users query. Using the Contrastive Taxonomy, we can assert the percentage of user questions that we can address with this approach, as well as gain insights into the different types of questions users ask in real world examples. Our approach directly addresses formal question types FQ1-7 which cover all of the contrastive questions asked by the users about plans in the above study. We can provide compilations of 89.8% of the 168 questions that users asked.

3. Plans: Queries and Explanations

In this section we provide the formal definitions that support our approach to explanation. We define the planning model and give a reference example, and then focus on the process of plan exploration as iterative model restriction.

3.1 Formal Definition of a Planning Problem

Our definition of a planning model follows the definition of PDDL2.1 given by Fox & Long (2003), extended by a set of time windows and explicit record of the plan metric. The formal description of such a planning model is as follows.

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 $Prob = \langle Os, I, G, M, W \rangle$ 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 W is a set of time windows.

A set of atomic *propositions* P is formed by applying the predicate symbols Ps to the objects Os (respecting arities). One proposition p is formed by applying an ordered set of objects $o \subseteq O$ to one predicate ps , respecting its arity. For example, applying the predicate (*robot_at ?v - robot ?wp - waypoint*) with arity 2 to the ordered set of objects $\{Jerry, sh3\}$ forms the proposition (*robot_at Jerry sh3*). This process is called “grounding” and is denoted with:

$$ground(ps, \chi) = p$$

where $\chi \subseteq O$ is an ordered set of objects. Similarly the set of *primitive numeric expressions* (PNEs) 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 PNE’s at that state. The initial state I is the state at time $t = 0$.

The goal $G = g_1, \dots, g_n$ is a set of propositions, including a subset of the logical state variables, P and a set of numeric propositions over the numeric variables, V , that must hold at the end of an action sequence for a plan to be valid. Similarly to propositions and PNEs, 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 $Dur(a)$ which constrains the length of time that must pass between the start and end of a ; a start (end) condition $Pre_+(a)$ ($Pre_-(a)$) which must hold at the state that a starts (ends); an invariant condition $Pre_{\leftrightarrow}(a)$ which must hold throughout the entire execution of a ; add effects $Eff(a)_+^+, Eff(a)_-^+ \subseteq P$ that are made true at the start and end of the action respectively; delete effects $Eff(a)_-^-, Eff(a)_+^- \subseteq P$ that are made false at the start and end of the action respectively; and numeric effects $Eff(a)_+^n, Eff(a)_{\leftrightarrow}^n, Eff(a)_-^n$ that act upon some $n \in V$.

A plan is a sequence of grounded actions, $\pi = \langle a_1, a_2, \dots, a_n \rangle$ each with a respective time denoted by $Dispatch(a_i)$. We use the definition of plan validity from Fox & Long (2003) (Definition 15 “Validity of a Simple Plan”). A simple plan is derived from a plan, π , by taking the end points of the durative actions of π as separate simple (non-durative) actions

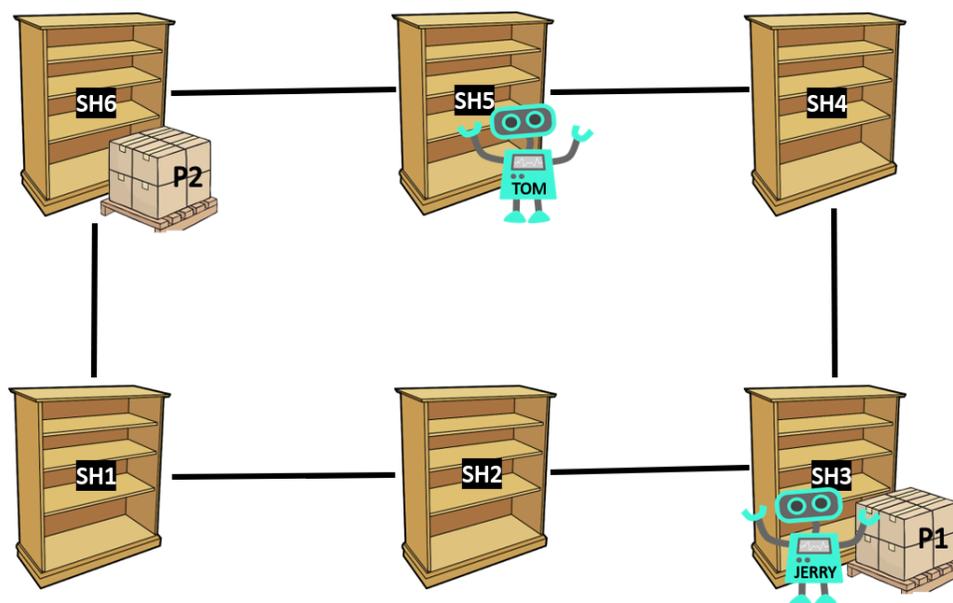


Figure 2: Diagram of the warehouse delivery system domain. SH_i are shelves, P_i are packages, and Tom and Jerry are robots responsible for delivering the packages to their correct shelves. The black lines indicate the paths the robots can traverse the shelves.

(called *happenings*) and generating a sequence of times, $t_{i=0\dots k}$ and a sequence of states, $s_{i=0\dots k+1}$ such that $s_0 = I$ and for each $i = 0 \dots k$, s_{i+1} is the result of executing the happening at time t_i from state s_i . The simple plan π is valid if $s_{k+1} \models G$ (that is, the propositions in G are satisfied in s_{k+1} with the standard interpretation of arithmetic operators).

The plan-metric function is, by default, the makespan of the plan to which it is applied. 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.

Each time window $w \in W$ is a tuple $w = \langle w_{lb}, w_{ub}, w_v \rangle$ where w_v is a proposition which becomes true or a numeric effect which acts upon some $n \in V$. $w_{lb} \in \mathbb{R}$ is the time at which the proposition becomes true, or the numeric effect is applied. $w_{ub} \in \mathbb{R}$ is the time at which the proposition becomes false. The constraint $w_{lb} < w_{ub}$ must hold. Note that the numeric effect is not applied or reverted at w_{ub} , so w_{ub} is superfluous for numeric effects.

3.2 Running Example

As a reference example, we use a simplified version of a model of a warehouse delivery system. There are multiple robots that work to move pallets from their delivery location to the correct storage shelf. Before the pallets can be stored, the shelf must be set up.

Figure 3 defines the domain for this model. There are four durative actions, *goto_waypoint*, *set_shelf*, *load_pallet*, and *unload_pallet*. The *goto_waypoint* action is used for the robots to navigate the factory, robots can only navigate along set paths and two or more robots cannot be in the same location at the same time. The *set_shelf* action ensures that the shelf is ready to store a package (the robot cannot perform this action while holding a pallet). The *load_pallet* action loads the pallet from a shelf on to the robot. Finally, the *unload_pallet* action unloads the pallet onto a previously set shelf.

For illustration purposes, we use a very simple problem with two robots, two pallets, and six waypoints. An example problem is shown in Figure 4, and an example plan for this planning problem is shown in Figure 5. Figure 5 consists of a sequence of actions each with two attached values denoting the time they are executed and for how long. A diagram illustrating this domain is shown in Figure 2. For simplicity, we assume the cost of this plan is its duration (20.003) which in this case is optimal.⁴ The domain and problem show predicates that are used to model the negated form of a predicate, for example *not_occupied*. These are used because many flavours of PDDL do not allow negative preconditions, however, the techniques we use in this paper are not dependant on this constraint.

Tying the reference example back to the definitions in Section 3.1, the first action present in Figure 5 is the operator *goto_waypoint* in Figure 3 grounded with the objects $\{Tom, sh5, sh6\}$. Each operator parameter is substituted with the corresponding object to give a ground action, this is represented in Figure 6 which shows the duration, conditions, and effects.

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 , $Pre_{\pm}(a)$ denotes all start and end preconditions of a but not invariant conditions, $Eff(a)$ denotes all effects of a including numeric effects.

3.3 Iterative Plan Exploration

Fundamentally, the need for plan explanation is driven by the fact that a human and a planning agent may have different models of the planning problem and different computational capabilities. In Definition 1 a planning model Π was defined in terms of a domain $D = \langle Ps, Vs, As, arity \rangle$ and problem $Prob = \langle Os, I, G, M, W \rangle$. For the purposes of this paper we assume that the human’s planning model Π^H , and planning agent’s model Π^P share the same vocabulary, namely the same predicate symbols Ps , function symbols Vs , and actions As from the domain D , and objects Os from the problem. However, the action durations, conditions, and effects may be different, and the initial states I , goals G , and plan metric M may be different. We do not assume that the human knows the planning agent’s model Π^P , or vice versa. This assumption differs from previous work on model reconciliation (Chakraborti et al., 2017) in that we do not assume that the planner knows (or learns) the planning model of the human.

Even when a human and a planning agent have the same planning models $\Pi^H = \Pi^P$, there are typically multiple plans satisfying this planning model. Although a planner is intended to optimise the plan with respect to the plan metric, it is common to produce only

4. Optimal under PDDL 2.1 epsilon semantics with epsilon equal to .001. The plan is obtained using the planner POPF (Coles, Coles, Fox, & Long, 2010). However, our framework theoretically works with any PDDL2.1 planner.

```

(:types
  waypoint robot - locatable
  pallet)
(:predicates
  (robot_at ?v - robot ?wp - waypoint)
  (connected ?from ?to - waypoint)
  (visited ?wp - waypoint)
  (not_occupied ?wp - waypoint)
  (set_shelf ?shelf - waypoint)
  (pallet_at ?p - pallet ?l - locatable)
  (not_holding_pallet ?v - robot))
(:functions
  (travel_time ?wp1 ?wp2 - waypoint))
(:durative-action goto_waypoint
 :parameters (?v - robot ?from ?to - waypoint)
 :duration(= ?duration (travel_time ?from ?to))
 :condition (and
  (at start (robot_at ?v ?from))
  (at start (not_occupied ?to))
  (over all (connected ?from ?to)))
 :effect (and
  (at start (not (not_occupied ?to)))
  (at end (not_occupied ?from))
  (at start (not (robot_at ?v ?from)))
  (at end (robot_at ?v ?to)))
 )
(:durative-action set_shelf
 :parameters (?v - robot
              ?shelf - waypoint)
 ...)
(:durative-action load_pallet
 :parameters (?v - robot ?p - pallet
              ?shelf - waypoint)
 ...)
(:durative-action unload_pallet ...)

(define (problem task)
  (:domain warehouse_domain)
  (:objects
    sh1 sh2 sh3 sh4 sh5 sh6 - waypoint
    p1 p2 - pallet
    Jerry Tom - robot)
  (:init
    (robot_at Jerry sh3) (robot_at Tom sh5)
    (not_holding_pallet Jerry)
    (not_holding_pallet Tom)
    (not_occupied sh1) (not_occupied sh2)
    (not_occupied sh4) (not_occupied sh6)
    (pallet_at p1 sh3) (pallet_at p2 sh6)
    (connected sh1 sh2) (connected sh2 sh1)
    (connected sh2 sh3) (connected sh3 sh2)
    (connected sh3 sh4) (connected sh4 sh3)
    (connected sh4 sh5) (connected sh5 sh4)
    (connected sh5 sh6) (connected sh6 sh5)
    (connected sh6 sh1) (connected sh1 sh6)
    (= (travel_time sh1 sh2) 4)
    (= (travel_time sh2 sh1) 4)
    (= (travel_time sh2 sh3) 8)
    (= (travel_time sh3 sh2) 8)
    (= (travel_time sh3 sh4) 5)
    (= (travel_time sh4 sh3) 5)
    (= (travel_time sh4 sh5) 1)
    (= (travel_time sh5 sh4) 1)
    (= (travel_time sh5 sh6) 3)
    (= (travel_time sh6 sh5) 3)
    (= (travel_time sh6 sh1) 4)
    (= (travel_time sh1 sh6) 4)
  )
  (:goal (and
    (pallet_at p1 sh6)
    (pallet_at p2 sh1))))

```

Figure 3: A fragment of a robotics warehouse delivery domain used as a running example. Some of the operator bodies have been omitted for space. The full description of the goto_waypoint action is shown.

Figure 4: A fragment of the problem instance used in the running example. The full initial state and goal state are shown. The robot’s Tom and Jerry start at shelves sh3 and sh5 respectively. The pallets p1 and p2 are at shelves sh3 and sh6 and must be delivered to sh6 and sh1. The connections and travel times between the shelves are shown.

```

0.000: (goto_waypoint Tom sh5 sh6) [3.000]
0.000: (load_pallet Jerry p1 sh3) [2.000]
2.000: (goto_waypoint Jerry sh3 sh4) [5.000]
3.001: (set_shelf Tom sh6) [1.000]
4.001: (goto_waypoint Tom sh6 sh1) [4.000]
7.001: (goto_waypoint Jerry sh4 sh5) [1.000]
8.001: (set_shelf Tom sh1) [1.000]
8.002: (goto_waypoint Jerry sh5 sh6) [3.000]
9.001: (goto_waypoint Tom sh1 sh2) [4.000]
11.002: (unload_pallet Jerry p1 sh6) [1.500]
12.503: (load_pallet Jerry p2 sh6) [2.000]
14.503: (goto_waypoint Jerry sh6 sh1) [4.000]
18.503: (unload_pallet Jerry p2 sh1) [1.500]

```

Figure 5: The solution plan from the domain and problem shown in Figures 3 and 4, with a cost of 20.003.

```

(:ground-action goto_waypoint Tom sh5 sh6
 :duration (= 3.000)
 :condition (and
  (at start (robot_at Tom sh5))
  (at start (not_occupied sh6))
  (over all (connected sh5 sh6)))
 :effect (and
  (at start (not (not_occupied sh6)))
  (at end (not_occupied sh5))
  (at start (not (robot_at Tom sh5)))
  (at end (robot_at Tom sh6)))
 )

```

Figure 6: The operator `goto_waypoint` from the domain in Figure 3 grounded with the objects `Tom`, `sh5`, and `sh6` to form the ground action (`goto_waypoint Tom sh5 sh6`).

one of the valid plans, rather than an optimal plan for a model. A planner might even fail to produce a plan at all, for some problems. In part, this is an inevitable consequence of the undecidability of planning problems with numeric variables and functions (Helmert, 2002), but it is also a consequence of the practical limits on the computational resources available to a planner (time and memory). These observations are equally valid for automated and human planners. In order to discuss the process of developing plan explanations, it is helpful to define the planning abilities of both the planner and the user. We model the planning capability of an agent as a partial function from planning models to plans:

Definition 3 *The **planning capability** of an agent A (human or machine), is a partial function, \mathbb{C}^A , from planning models to plans. Given the agent's planning model, Π^A , if $\mathbb{C}^A(\Pi^A)$ is defined, then it is a candidate plan π^A for the agent.*

The planning capability \mathbb{C}^A , can be affected by a multitude of factors. The part of the function domain on which \mathbb{C}^A is defined determines the planning competency of the agent – domain-problem pairs for which the agent cannot find a plan lie outside this

competency. Note that the planning competency of an agent can be restricted by a bound on the computational resources the agent is allowed to devote to the problem, as well as by the capabilities of the agent in constructing and adequately searching the search space that the problem defines. When A is an automated AI planner P , the computational ability is determined by the search strategy implemented in the planner, its heuristic (if there is one), and the resources allocated to the task. For sound planners, when $\mathbb{C}^P(\Pi^P)$ is defined it is a valid plan for Π^P .

When A is a human planner H , the planning capability is determined by the understanding that the human has of the planning model and the patience and problem-solving effort they are willing to devote to solving the problem. It cannot be assumed that, if $\mathbb{C}^H(\Pi^H)$ is defined, that the human's model Π^H accurately reflects the world, or that the reasoning \mathbb{C}^H is sound. This means that the plan may not be valid. One aspect of the process of planning and explanation is that the user can revise their model Π^H as the process unfolds. However, it is also possible that the user can change their planning capability \mathbb{C}^H , by coming to a greater understanding of the model, by engaging in more reasoning, or by simply concluding that the solution provided by an automated system is satisfactory. It is also possible that the planner responses lead to the user changing their view of what might be a good plan to solve a problem, while still not adopting the solution offered by the planner. Thus, the user's planning model and capability might be extended or modified by consideration of the planner output or question responses. This revision might include correcting flawed plans produced by the original planning model and capability of the user.

In this paper, we do not explicitly attempt to model any learning process on the part of the human, although we allow that this may happen. Furthermore, we do not consider any learning by the planning agent. Instead, we adopt the approach that the human user asks contrastive questions that impose additional restrictions ϕ on the agent's planning problem Π^P to generate a succession of hypothetical planning problems. The object of these questions and the resulting hypothetical plans is for the user to understand and ultimately arrive at a satisfactory plan. Model learning and reconciliation by the human and planning agent can be seen as complementary techniques that could make this process more effective and more efficient.

Given the planning models Π^H and Π^P , and planning capabilities \mathbb{C}^H and \mathbb{C}^P of a human and planning agent, the two agents disagree when $\mathbb{C}^H(\Pi^H) \neq \mathbb{C}^P(\Pi^P)$, which can arise in the case that either of these terms is undefined, or if both terms are defined and yield different plans. We assume that, in this case, the user is capable of inspecting the planner output and determining a question that will expose some part of the explanation for this difference. By questioning why certain decisions were made in the plan and receiving contrastive explanations the user can gain an initial understanding. As their understanding of the plan develops they can ask more educated questions to gain a deeper understanding or try to arrive at an alternative plan that they consider more satisfactory. Ultimately, this process concludes when the user is satisfied with some plan. In an ideal case, this will be when the user and the planner have converged on the same plan, but this need not happen. For example, suppose $\mathbb{C}^P(\Pi^P) = \pi$ and $\mathbb{C}^H(\Pi^H) = \pi'$ and $\pi \neq \pi'$. The user might inspect π and, after seeking explanation for the differences between it and π' , conclude that there is some deficiency in the planner's model Π^P or planner's reasoning \mathbb{C}^P and therefore

decide that π' is the plan they want. Thus, the sequence, in this case, might conclude with the user rejecting the plan offered by the planner and not changing their own model or computational ability at all.

We formalise the iterative process of questioning and explanation as one of successive *model restriction*, in which the user asks contrastive questions in an attempt to understand the planning agent's plan and potentially steer the planning agent towards a satisfactory solution. We suppose that, when $\mathbb{C}^H(\Pi^H) \neq \mathbb{C}^P(\Pi^P)$, the user can construct some *foil*, ϕ , in the form of a constraint that $\mathbb{C}^P(\Pi^P)$ does not satisfy, so that seeking an explanation for the plan, $\mathbb{C}^P(\Pi^P)$, can be seen as seeking a plan for Π^P that also satisfies ϕ . This requirement acts as a restriction on Π^P and is captured as follows.

Definition 4 *A constraint property is a predicate, ϕ , over plans.*

*A constraint operator, \times is defined so that, for a planning model Π and any constraint property ϕ , $\Pi \times \phi$ is a model (an HModel), Π' , called a **model restriction** of Π , satisfying the condition that any plan for Π' is a plan for Π that also satisfies ϕ . A plan for an HModel is referred to as an HPlan.*

The process in which the user interacts with a planner is an iterative one – the user successively views plans and seeks explanations by generating foils that impose additional restrictions on the planning problem. The collection of model restrictions forms a tree, rooted at the original model and extended by the incremental addition of new constraint properties, as shown in Figure 7. The user can visit the nodes of this tree in any order. As the user inspects the result of applying \mathbb{C}^P to a node in this tree, their own planning model and capability, Π^H and \mathbb{C}^H , may change, reflecting accumulating understanding of the plans that can be constructed for the model. As a result, the order in which the user visits the nodes matters and can lead to different outcomes. One possible path, showing the evolving capability and model for the user, is shown in Figure 8. This figure should not be interpreted as implying that the user must explore the tree in a systematic way. It is also worth emphasising that any constraint, ϕ , may be added to any model, so that the user is not forced to develop a tree of models in any particular way to arrive at the consequence of adding any specific constraint to a model.

We call what we have described above the *Iterative Plan Exploration Process*. In this process the user explores possible plans through a sequence of questions that impose model restrictions on the planning system. This problem can arise for many reasons. In the case where a user has an expectation of what the plan should look like that differs from the proposed plan, the user may not accept the proposed plan without understanding why it was produced, or exploring other plan options. A user might be unsure of the quality of the plan but not have the reasoning capabilities to properly evaluate the plan quality. The user can explore how alternate actions and decisions that could have been made in the plan affect the plan quality. This will either refute or support their concerns, that either the plan they were presented was of good quality or that there is a plan of better quality. If a better plan cannot be found under the added constraint, this might allay their concerns, while, if a better plan is found, it will confirm the user's suspicions. In either case, the user might go on to explore additional constraints, in search of a better plan, or to better understand the space of possible plans. A user might have hidden preferences that are not modelled, and through the addition of constraints can make sure that the plan behaves in such a way

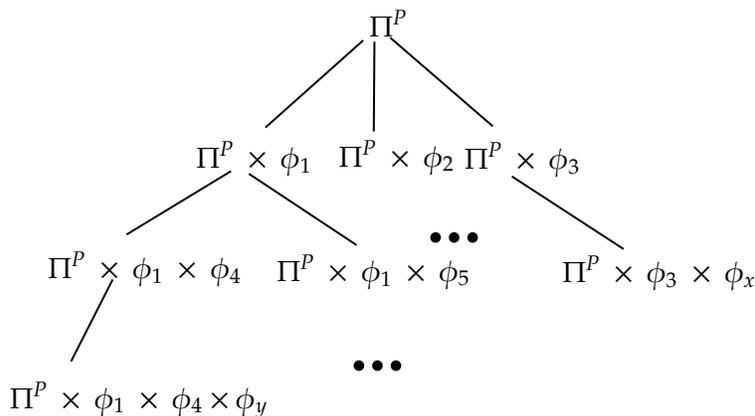


Figure 7: A fragment of a tree of model restrictions for a planner P . Each node n_i in the tree is a model restriction of the model of its parent node n_{i-1} , and a constraint ϕ_i .

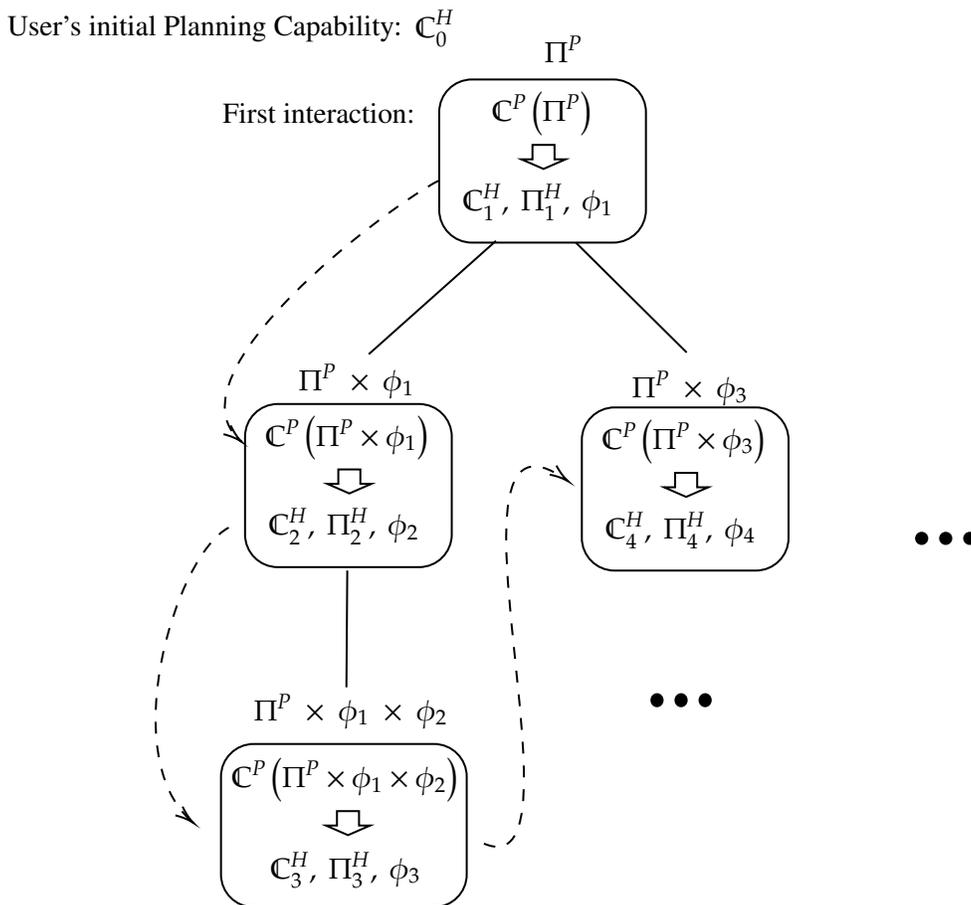


Figure 8: An example of a sequence of interactions between a user and a planner. At each interaction the user updates their planning model and capability, identifies a new constraint, which may or may not incorporate previous constraints. The order in which nodes are explored, as indicated by the dotted line, is entirely under the control of the user.

that their preferences are fulfilled. Alternatively, the user might simply want to increase their understanding of the model by questioning why certain decisions were made in the plan before they are willing to accept it.

In each of these cases, reasoning about what did not happen in the plan can give one a deeper understanding of the decisions made in the plan and simultaneously allow one to explore potentially more suitable plan candidates. The user is offered the opportunity to consider what did not happen in a plan (in particular, why a plan does not satisfy some constraint), by asking the contrastive question “why is the plan for this model as it is and not one that also satisfies the constraint ϕ ?”. As indicated earlier, we require that the user and the system share the same vocabulary. This qualification ensures that any user restriction ϕ can actually be understood by the planner – i.e. that the model Π^P makes sense. As a result, the user can restrict a model in a way that prevents the planner from using an action in states where some condition is not satisfied, effectively adding a precondition to that action. Similarly, the model can be restricted to prevent the planner from exploiting an effect of an action, by constraining the actions that can be applied after the particular action. Although this process will not allow the user to add arbitrary preconditions or eliminate arbitrary effects (since the states that are generated remain faithful to the model the planner is actually using), this observation makes the point that the model restrictions can include close approximations to model revisions that act directly on the actions themselves.

An example that illustrates a fragment of the iterative plan exploration process is as follows. Using the model Π shown in Figures 3 and 4 and the plan π shown in Figure 5, the user might think that the action (*goto_waypoint Jerry sh4 sh5*) should not be present in the plan (ϕ), so $\Pi \times \phi = \Pi'$, a model constrained to ensure that (*goto_waypoint Jerry sh4 sh5*) is not used. Π can be considered to be the root of a tree such as that shown in Figure 7 and Π' , derived by the addition of the constraint posed above, can be seen as the first branch to the left in that figure. A plan π' for Π' then represents an alternative way to solve Π , without using the action (*goto_waypoint Jerry sh4 sh5*). Whether such a plan can be found by the planner depends on \mathbb{C}^P (as seen in Figure 8) and whether the user has such a plan in mind will depend on \mathbb{C}_0^H . The evolution of the interaction between the user and system now depends on whether the new plan, π' , if it is successfully generated, prompts further questions and corresponding constraints, or satisfies the user.

The new plan π' might not entirely satisfy the user’s curiosity. It might trigger new questions or still not satisfy the user’s expectations. The user can explore the space of plans by iteratively extending and specifying the foil ϕ , until they are satisfied with the result or cease to find the interaction rewarding. Although the user’s perspective is opaque to the system, so that it is impossible to know whether the user accepts one of the plans generated in this exchange, or is persuaded that the system or the user has a flawed model. Nevertheless, our user study (reported in Section 6.2) suggests that the explanations have the intended effect of supporting a user in better understanding of the way that the first proposed plan represents a good solution to the original problem.

It should be noted that, depending on the planning models and capabilities of the two participants, there might not exist any constraint achieving a common solution. For example, in the degenerate case in which \mathbb{C}^P produces no plan at all, for any value of ϕ , then there can be no mutually satisfactory plan. Typically, the greater the differences between

the planning models and capabilities of the two agents, the more likely it will be that there is no common satisfactory plan.

We formally capture the iterative process of model restriction and planning as:

Definition 5 Iterative Model Restriction For a planner P , and a user H : Let \mathbb{C}^P and Π^P be the planner's underlying capability and planning model and \mathbb{C}_0^H and Π_0^H be the initial capability and planning model of H . Let ϕ_i be the set of user imposed constraints, which is initially empty, i.e. $\phi_0 = \emptyset$. Each stage, i (initially zero), of this process starts with the planner producing a plan $\pi_i^P = \mathbb{C}^P(\Pi_i^P)$ for the model $\Pi_i^P = \Pi^P \times \phi_i$.

The user responds to this plan π_i^P by potentially updating their capability and model to \mathbb{C}_{i+1}^H and Π_{i+1}^H and then either terminating the interaction, or asking a question that imposes a new constraint ϕ_{i+1} on the problem. This results in the planner solving a new constrained problem $\Pi_{i+1}^P = \Pi^P \times \phi_{i+1}$ at the next step.

Although it is possible that the planner will fail to produce a plan at some stage, i , we do not address the problem of explaining the unsolvability of plans in this paper (Göbelbecker et al., 2010; Sreedharan et al., 2019). Nevertheless, the failure will be observed by the user and it can trigger a decision to either select a previous plan π_j^P for some $j < i$, or explore a new constraint ϕ_{i+1} for the next iteration.

We have assumed here that the planner's underlying capability and planning model \mathbb{C}^P and Π^P do not evolve during the process. While this is not strictly necessary, possible evolution or improvement of the planner capabilities and model based on the sequence of user questions and the resulting ϕ_i is an issue we do not consider here. In contrast, the user's capability and planning model \mathbb{C}^H and Π^H are assumed to evolve, but in unknown ways. Again, we do not attempt to model the user's learning process.

3.4 Ending Exploration

The process we have described is one in which a user explores a tree of model restrictions, rooted at the original model. At each node in the tree the planner will produce some output (although possibly no plan) and the user will revise their personal planning model and capability. This revision might be trivial, in that the user might simply retain the model and capability they held at the previous iteration. The model revisions need not converge in any sense, but at some point the exploration will end. We now briefly consider the status of the exploration at the conclusion of an interaction.

One way that the exploration can end is that the plan produced for the final model yields a plan that is acceptable to the user, so that the user adopts this plan for the original model. This is a case where the system and the user converge on a plan that is mutually agreed to be a solution to the original model, meeting conditions that might or might not have been part of the original model and that the user might or might not have envisaged at the outset of the exploration.

Another way the exploration can end is with the user having explored the plans for several models and, finally, having been persuaded in this process that the first plan produced by the planner for the original model is actually the desired plan, the user modifies their planning model and capability so that this is a plan for the original model of the planner and revised model of the user. Again, this is a mutually agreed plan, but in this

case it is not the last plan produced, but the first; the exploration process in this case acts to help the user to arrive at a point where they are persuaded that it is the plan that they want. In contrast to the first case, where the user might not ever modify their planning model or capability, in this second case the user must modify their planning model and capability to accept the plan for the original planner model. This process is the idealised form of plan explanation we anticipate: the user explores the plans for restricted models in order to understand why the original plan is the correct plan for the problem and they adapt their own planning model and capability to reflect this conclusion.

The exploration can also result in the selection of a plan somewhere between these two extremes, with the user deciding to adopt a plan produced for some previous model in the exploration, modifying their planning model and capability to include this mutually agreed plan.

A final outcome is one in which the user explores the space and then rejects all of the plans the planner offers. In this case, the user might modify their planning model and capability as a consequence of what they observe and they might or might not conclude the process with a satisfactory plan for the original model. In this case, there is no mutually agreed plan and the exploration might not even have helped the user arrive at any useful conclusions about the problem.

Despite the fact that all of these outcomes are possible, it is impossible to determine, from the perspective of the system, which of them has been achieved at the end of the exploration. The system has no access to the planning model or capability of the user and does not construct queries to probe it. The hypothesis we explore, in the user study we describe in Section 6.2, is that the user will usually find value in the exploration and conclude in one of the three cases in which a mutually agreed plan is identified. As can be seen, it remains impossible to be sure which plan is the mutually agreed plan at the conclusion of the exploration.

4. Model-Based Compilations

Armed with a formal description of the interactive process of model refinement that underpins the construction of our explanations, we now consider how the system can generate plans for the series of models generated in the process. In particular, given a planning model Π and a constraint ϕ , we aim to construct a plan for $\Pi \times \phi$. The approach we adopt is to *compile* the constraint ϕ into the model Π , so that $\Pi \times \phi$ can be presented to a generic planner as another model to be solved. This approach avoids embedding the iterative process inside a planner, instead using a planner as a service inside the process of iterative model refinement.

Although the point was not explicitly addressed in Definitions 4 and 5, it is not necessarily possible to combine an arbitrary constraint, ϕ with a model Π to yield a model $\Pi \times \phi$ that is expressible in the language we use to describe our planning models (Definition 1). The compilation strategy exploits the case in which $\Pi \times \phi$ can be expressed in our modelling language and, in this section, we demonstrate how this is achieved for a collection of different forms for ϕ . In the case where the user wishes to capture some constraint that cannot be captured in this way, it is often possible to incrementally converge on a model restriction that approximates the constraint, by the addition of constraints that can be expressed and

that steadily remove parts of the plan space that violate the intended constraint. This process is discussed further in Section 4.9, and is analogous to the addition of cuts to a linear program in order to find a solution to an integer program. The constraints in this section, for which we present compilations, were chosen in response to the user study presented in Section 2 and are examples of real questions for which users sought explanations.

The addition of a constraint to a model never increases the collection of feasible solutions, and so might make the search for a solution harder. There are two reasons that this intuition might not match observations. First, let us consider the construction of feasible solutions by an incremental series of choices to variables (such as actions added to the head of a developing plan, as in forward search planning). The addition of constraints will prune the collection of feasible solutions in this space, but it can also prune states in the search space which previously could lead to goal states, but now lead to a dead end. That is, the constraints can act to prune partial solutions that previously appeared promising, leading to a reduction in search in that part of the space. Secondly, where solutions are constructed by search, the addition of features to the model can lead to choices being explored in a different order, possibly for entirely implementation-dependent reasons (such as reordering of action choices inside an internal data-structure, based on order of grounding). These changes can lead to unpredictable effects on the performance of a planner, possibly leading to a lucky reduction of search or an unlucky increase in search. These effects will be observed in all search-based solvers and different families of constraints might interact with the solution strategy of specific planners in different ways. For example, adding timed-effects to the initial conditions of a problem for POPF (Coles et al., 2010) can create additional choice branches at every step in the construction of a plan. In Section 6 we explore the effects of the compilations on performance for a range of representative examples.

4.1 Explanation Problem

Definition 6 *An explanation problem is a tuple $E = \langle \Pi, \pi, Q \rangle$, in which Π is a planning model (Definition 1), π is the plan generated by the planner, and Q is the specific question posed by the user.*

Motivated by the outcome of the user study described in Section 2, we are interested when the user question Q is a *contrastive question* of the form “Why A rather than B?”, where A occurred in the plan and B is the hypothetical alternative expected by the user. This question can be captured as a constraint that enforces the foil. A foil is normally partial – i.e. a set of additional constraints on the form of the solution rather than being a complete alternative. This fits with the framing of this entire process as being one of iterative model restriction. As discussed in Section 2 explicitly stating the foil makes the question unambiguous, so we ensure that questions are asked in this form.

As in our user study, we assume that the user knows the model Π and the plan π , so responses such as stating the goal of the problem will not increase their understanding. Based on the outcome of the user study, we provide a formal description for compilations of the questions in the Contrastive Taxonomy (Table 2), reiterated here:

- **FQ1** - Why is action a not used in the plan, rather than being used? (Section 4.2)
- **FQ2** - Why is action a used in the plan, rather than not being used? (Section 4.3)

- **FQ3** - Why is action a used in state s , rather than action b ? (Section 4.4)
- **FQ4** - Why is action a not performed before (after) action b , rather than a being performed after (before) b ? (Section 4.5)
- **FQ5** - Why is action a used outside of time window w , rather than only being allowed within w ? (Section 4.6)
- **FQ6** - Why is action a not used in time window w , rather than being used within w ? (Section 4.7)
- **FQ7** - Why is action a used at time t , rather than at least some time t' after/before t ? (Section 4.8)

This section formalises the compilations of the questions in the Contrastive Taxonomy to produce an HModel $\Pi' = \Pi \times \phi$, where ϕ is a constraint derived from Q and Π is a PDDL2.1 model (Fox & Long, 2003). The HModel Π' is:

$$\Pi' = \langle \langle Ps', Vs, As', arity' \rangle, \langle Os, I', G', W' \rangle \rangle$$

After the HModel is formed, it is solved to give the HPlan. Any new operators that are used in the compilation to enforce some constraint are trivially renamed to the original operators they represent. For each iteration of compilation the HPlan is validated against the original model Π .

4.2 Add an Action to the Plan

Given a plan π , a formal question Q is asked of the form:

Why is the operator o with parameters χ not used, rather than being used?

For example, given the example plan in Figure 5 the user might ask:

“Why is (load_pallet Tom p2 sh6) not used, rather than being used?”

They might ask this because a goal of the problem is to load and move the pallet $p2$ to shelf $sh1$. As the robot Tom moves to shelf $sh6$ where the pallet $p2$ is located early in the plan, and the pallet $p2$ is located at $sh6$ and the shelves $sh6$ and $sh1$ are connected, it might make sense to the user for the robot Tom to deliver this pallet.

To generate the HPlan, a compilation is formed such that the action $a = \text{ground}(o, \chi)$ must be applied for the plan to be valid. The compilation introduces a new predicate $\text{has_done_}a$, which represents which actions have been applied. Using this, the goal is extended to include that the user suggested action has been applied. The HModel Π' is:

$$\Pi' = \langle \langle Ps', Vs, As', arity' \rangle, \langle Os, I, G', W \rangle \rangle$$

where

- $Ps' = Ps \cup \{\text{has_done_}a\}$
- $As' = \{o_a\} \cup As \setminus \{o\}$

- $arity'(x) = arity(x), \forall x \in arity$
- $arity'(has_done_a) = arity'(o_a) = arity(o)$
- $G' = G \cup \{ground(has_done_a, \chi)\}$

where the new operator o_a extends o with the add effect has_done_a with corresponding parameters, i.e.

$$Eff_+^+(o_a) = Eff_+^+(o) \cup \{has_done_a\}$$

For example given the user question above where $a = ground(load_pallet, \{Tom, p2, sh6\})$, the operator $load_pallet$ from the running example is extended to $load_pallet_prime$ with the additional add effect $has_done_load_pallet$. The new operator is shown in the PDDL2.1 syntax in Figure 9.

```
(:durative-action load_pallet_prime
 :parameters (?v - robot ?p - pallet ?shelf - waypoint)
 :duration(= ?duration 2)
 :condition (and
  (over all (robot_at ?v ?shelf))
  (at start (pallet_at ?p ?shelf))
  (at start (not_holding_pallet ?v)))
 :effect (and
  (at start (not (pallet_at ?p ?shelf)))
  (at start (not (not_holding_pallet ?v)))
  (at end (pallet_at ?p ?v))
  (at end (has_done_load_pallet ?v ?p ?shelf)))
)
```

Figure 9: The operator $load_pallet_prime$ which extends the original operator $load_pallet$ with the new add effect $has_done_load_pallet$.

The goal is then extended to include the proposition: $(has_done_load_pallet Tom p2 sh6)$. The HPlan produced from solving the HModel described is shown in Figure 10.

```
0.000: (goto_waypoint Tom sh5 sh6) [3.000]
0.000: (load_pallet Jerry p1 sh3) [2.000]
2.000: (goto_waypoint Jerry sh3 sh4) [5.000]
3.001: (set_shelf Tom sh6) [1.000]
4.001: (goto_waypoint Tom sh6 sh1) [4.000]
7.001: (goto_waypoint Jerry sh4 sh5) [1.000]
8.001: (set_shelf Tom sh1) [1.000]
9.001: (goto_waypoint Tom sh1 sh6) [4.000]
13.001: (load_pallet Tom p2 sh6) [2.000]
15.001: (goto_waypoint Tom sh6 sh1) [4.000]
19.001: (unload_pallet Tom p2 sh1) [1.500]
19.002: (goto_waypoint Jerry sh5 sh6) [3.000]
22.002: (unload_pallet Jerry p1 sh6) [1.500]
```

Figure 10: The HPlan containing the user suggested action $load_pallet Tom p2 sh6$ with a duration of 23.502

4.2.1 JUSTIFIED ACTIONS AND EXPECTED PLANS

Usually, a user asks a contrastive question about a plan when they expected a different outcome or some sub-goal to be achieved in a certain way. In the example shown in 4.2, the user expected the robot *Tom* to load the pallet *p2* onto the shelf *sh6*, which their question reflects. It is clear why the user asked this question as it fully describes the goal they wish to achieve and how to achieve it. The constraint derived from this question causes an immediate impact in the plan. The package is delivered using a different robot than previously. However, the objective of some questions are not as clear. For example, if a user questioned “Why is (*set_shelf Tom sh4*) not used, rather than being used?”, it is not clear what they intend to achieve with this action. The HPlan produced from the HModel containing the constraint for this question is shown in Figure 11. The plan starts with some preliminary movement actions that allow the robot *Tom* to set up the required shelf *sh4*. *Tom* then traverses to the shelf *sh6*, the plan then continues the same as the original plan in Figure 5. The action (*set_shelf Tom sh4*) does not affect the plan and it would still be valid if the action were removed. The reason for this could be due to the plan that utilises the action being more expensive, or it could be due to it not being possible for the action (*set_shelf Tom sh4*) to achieve anything useful. However, it could also be because the planner could not find a plan where the action is used in such a way that it contributes to the goal. For this reason, a user may not be satisfied with an HPlan where the action is not used in a way that is necessary for achieving a goal, we discuss what this means in more detail in Section 4.10. Although the compilations formalised in this section do not guarantee that any actions that a user suggests are necessary for achieving a goal, the rest of this subsection provides a step towards this with the description and formalisation of a compilation.

```

0.000: (load_pallet jerry p1 sh3) [2.000]
0.000: (goto_waypoint tom sh5 sh4) [1.000]
1.001: (set_shelf tom sh4) [1.000]
2.001: (goto_waypoint tom sh4 sh5) [1.000]
3.002: (goto_waypoint jerry sh3 sh4) [5.000]
3.002: (goto_waypoint tom sh5 sh6) [3.000]
6.002: (set_shelf tom sh6) [1.000]
7.002: (goto_waypoint tom sh6 sh1) [4.000]
8.003: (goto_waypoint jerry sh4 sh5) [1.000]
11.002: (set_shelf tom sh1) [1.000]
11.003: (goto_waypoint jerry sh5 sh6) [3.000]
12.002: (goto_waypoint tom sh1 sh2) [4.000]
14.003: (unload_pallet jerry p1 sh6) [1.500]
15.504: (load_pallet jerry p2 sh6) [2.000]
17.504: (goto_waypoint jerry sh6 sh1) [4.000]
21.504: (unload_pallet jerry p2 sh1) [1.500]

```

Figure 11: The HPlan with a cost of 23.004 generated to satisfy the constraint derived from the question “Why is (*set_shelf Tom sh4*) not used, rather than being used?”.

The compilation works by tracking the facts that have been produced through effects of actions that the user suggested action τ has causally supported. One of these facts then has to be a goal fact. Therefore, there is a causal chain from τ to a goal and the action τ

is necessary for achieving the goal in any plan produced by a model with this constraint applied. For example this compilation ensures that, in the HPlan π' , there will be a causal chain, $\mu \subseteq \pi' = \langle \tau, a_1, a_2, \dots, a_n \rangle$ where for the state s_{n+1} after a_n is finished executing and some $g \in G$ then $s_{n+1} \models g$, and for all actions $a_i \in \mu$ if a_i was removed then $\pi' \not\models G$, assuming g is not already satisfied in the initial state.

To generate an HPlan that adheres to these properties and satisfies the user question “Why is $a = (\text{set_shelf Tom sh4})$ not used, rather than being used?”, the model is compiled in the following way. A new operator o_a is created which has the same preconditions and effects as a , but for each positive effect, has a new effect which adds a copy of the fact, we call this the prime-fact. A new operator is then created for each precondition p for each operator o in the domain. The precondition to this new operator is the same as o with a new precondition prime_p . The effects are the same as o but for each positive effect the corresponding prime-fact is also made true. These new actions behave the same as the existing actions in the domain, but they propagate the causal chains originating from a through the prime-facts. A final set of operators is added for each goal which can be applied if both a goal and it’s corresponding prime-fact are true, and at least one of these actions must appear in the plan for it to be valid. This is a work around used because the majority of PDDL2.1 planners do not accommodate disjunctive goals, however, this can be simplified by changing the goal to $G \wedge (\bigvee_{i=0} (g_i \wedge \text{prime}_{g_i}))$. If a goal has already been achieved by another action in the plan that is not part of the causal chain from a then this action can no longer be applied. The causality of the actions is tracked through these prime-facts and for any valid plan there will exist a goal that can have it’s origin traced through prime-facts back to the user suggested action a .

The HModel Π' is:

$$\Pi' = \langle \langle Ps', Vs, As', \text{arity}' \rangle, \langle Os, I', G', W \rangle \rangle$$

where:

- $Ps_p = \{\text{prime}_p, \forall p \in Ps\}$
- $G_p = \{\text{goal_prime}_p, \forall p \in Ps \text{ where } \text{ground}(p, \chi) = g \in G \text{ for some } \chi\}$
- $Ps' = Ps \cup \{\text{has_done_a}, \text{can_do_a}, \text{true}, Ps_p, G_p\}$
- $As' = As \cup \{o_a\} \cup \{\text{conjunct}_{xp}, \forall x \in As : \forall p \in \text{Pre}\neg(x)\} \cup \{\text{check_conjunct}_g, \forall g \in G_p\}$
- $\text{arity}'(x) = \text{arity}(x), \forall x \in Ps$
- $\text{arity}'(\text{goal_prime}_p) = \text{arity}(p), \forall p \in Ps \text{ where } \text{ground}(p, \chi) = g \in G \text{ for some } \chi\}$
- $\text{arity}'(\text{prime}_p) = \text{arity}(p), \forall p \in Ps$
- $\text{arity}'(\text{has_done_a}) = \text{arity}'(\text{can_do_a}) = \text{arity}'(o_a) = \text{arity}(o)$
- $\text{arity}'(\text{true}) = \emptyset$
- $I' = I \cup \{\text{ground}(\text{can_do_a}, \chi) \cup \{\text{ground}(\text{goal_prime}_p, \chi'), \forall p \in G_p \text{ where } \text{ground}(p, \chi') = g, \forall g \in G\}$

- $G' = G \cup \{ground(has_done_a, \chi)\} \cup true$

and the actions are defined such that the preconditions and effects are:

$$\begin{aligned}
 Pre_+(o_a) &= Pre_+(o) \cup \{can_do_a\} \\
 Eff_{-}^+(o_a) &= Eff_{-}^+(o) \cup \{has_done_a\} \\
 &\quad \cup \{prime_y \in Ps_p, \forall y \in Eff_{-}^+(o)\} \\
 Pre_{-}(conjunct_{xp}) &= Pre_{-}(x) \cup prime_p, \forall x \in As : \forall p \in Pre_{-}(x) \\
 Eff_{-}^+(conjunct_{xp}) &= Eff_{-}^+(x) \cup \{prime_y \in Ps_p, \forall y \in Eff_{-}^+(x)\}, \forall x \in As : \forall p \in Pre_{-}(x) \\
 Dur(check_conjunct_g) &= \epsilon \text{ where } \epsilon \text{ is a very small number, } \forall goal_prime_g \in G_p \\
 Pre_+(check_conjunct_g) &= prime_g \cup goal_prime_g, \forall g \in Ps' \text{ where } prime_g \in Ps_p \\
 &\quad \wedge goal_prime_g \in G_p \\
 Eff_{-}^+(check_conjunct_g) &= true \\
 Eff_{-}^-(o) &= Eff_{-}^-(o) \cup \{goal_prime_g, \forall g \in Eff_{-}^-(o) \text{ where } goal_prime_g \in G_p\}, \forall o \in As
 \end{aligned}$$

The plan for this is shown in Figure 12 where the action (*set_shelf tom sh4*) is necessary for performing the action (*unload_pallet tom p1 sh6*) which achieves the goal (*pallet_at p1 sh6*). However, this compilation does not guarantee that the action *a* will be perfectly justified in the plan π , that is that there is no set of actions *A* where $a \in A$ and $A \subseteq \pi$, such that if you removed the set of actions *A* then $\pi \models G$ (Fink & Yang, 1992). This means that there are no groups of actions that together are redundant in the plan. This is not the case for the HPlan in Figure 12, if the set of actions $\{(done_set_shelf tom sh4), (unload_pallet-2-conjunct tom$

```

0.000: (goto_waypoint jerry sh3 sh2) [8.000]
0.000: (goto_waypoint tom sh5 sh4) [1.000]
1.001: (done-set_shelf tom sh4) [1.000]
8.001: (goto_waypoint jerry sh2 sh1) [4.000]
8.001: (goto_waypoint tom sh4 sh3) [5.000]
12.002: (set_shelf jerry sh1) [1.000]
13.001: (load_pallet tom p1 sh3) [2.000]
13.002: (goto_waypoint jerry sh1 sh6) [4.000]
15.001: (goto_waypoint tom sh3 sh4) [5.000]
17.002: (set_shelf jerry sh6) [1.000]
18.002: (load_pallet jerry p2 sh6) [2.000]
20.001: (unload_pallet-2-conjunct tom p1 sh4) [1.500]
20.002: (goto_waypoint jerry sh6 sh1) [4.000]
21.502: (load_pallet-0-conjunct tom p1 sh4) [2.000]
23.502: (goto_waypoint tom sh4 sh5) [1.000]
24.002: (unload_pallet jerry p2 sh1) [1.500]
24.503: (goto_waypoint tom sh5 sh6) [3.000]
27.503: (unload_pallet-0-conjunct tom p1 sh6) [1.500]
29.003: (check-conjunct-pallet_at p1 sh6 true) [0.100]
    
```

Figure 12: The HPlan with a cost of 29.003 generated to satisfy the constraint derived from the question “Why is (*set_shelf Tom sh4*) not used, rather than being used?”, such that the action is necessary in the plan for achieving a goal. The action names are trivially renamed back to their corresponding actions, and the action (*check-conjunct-pallet_at p1 sh6 true*) is removed.

p1 sh4), (load_pallet-0-conjunct tom p1 sh4)) is removed, the plan is still valid. To attempt to determine whether there is a plan where a is perfectly justified would likely require an extended search over these redundancy sets. This search would be the repeated process of disallowing an action in the redundancy set to be applied in the plan, re-planning, and generating the new redundancy set. The search would end when a plan is found where the action is used in a perfectly justified way, or all the redundancy sets have been searched over and no plan was found, meaning the action cannot be used in a perfectly justified way.

This approach also works if the goal contains primitive numeric expressions in the same way. Any effects that alter the values of PNEs, will duplicate the behaviour with a prime-effect. The goal is checked in the same way as with a simple proposition. For example, if an action τ decreases the value of a PNE n , and there is a goal such that $5 < n < 10$ is true at the end of the plan. Then τ affects $prime_n$ in the same way as it does n and both $5 < n < 10$ and $5 < prime_n < 10$ must be true at the end of the plan for it to be valid.

This approach can be adapted for use in the compilations for all formal questions apart from FQ2 where it would have no use as an action is removed rather than added.

4.3 Remove a Specific Grounded Action

Given a plan π , a formal question Q is asked of the form:

Why is the operator o with parameters χ used, rather than not being used?

For example, given the example plan in Figure 5 the user might ask:

“Why is (goto_waypoint Tom sh1 sh2) used, rather than not being used?”

A user might ask this because *Tom* has already set up all of the shelves that are required. The user might question why *Tom* is doing this extra action.

The specifics of the compilation is similar to the compilation in Section 4.2. The HModel is extended to introduce a new predicate *not_done_action* which represents actions that have not yet been performed. The operator o is extended with the new predicate as an additional delete effect. The initial state and goal are then extended to include the user selected grounding of *not_done_action*. Now, when the user selected action is performed it deletes the new goal and so invalidates the plan. This ensures the user suggested action is not performed.

For example, given the user question above, an HPlan is generated that does not include the action (goto_waypoint Tom sh1 sh2), and is shown in Figure 13. This shows a plan with a longer duration than the original plan shown in Figure 5. In this HPlan *Tom* has to deliver pallet $p2$ because he is occupying shelf $sh1$ and cannot vacate it by going to shelf $sh2$. This means *Jerry* cannot pass by him to deliver the pallet more efficiently.

4.4 Replacing an Action in a State

Given a plan π , a formal question Q is asked of the form:

Why is the operator o with parameters χ used in state s , rather than the operator n with parameters χ' ? where $o \neq n$ or $\chi \neq \chi'$

```

0.000: (goto_waypoint Tom sh5 sh6) [3.000]
0.000: (load_pallet Jerry p1 sh3) [2.000]
2.000: (goto_waypoint Jerry sh3 sh4) [5.000]
3.001: (set_shelf Tom sh6) [1.000]
4.001: (goto_waypoint Tom sh6 sh1) [4.000]
7.001: (goto_waypoint Jerry sh4 sh5) [1.000]
8.001: (set_shelf Tom sh1) [1.000]
9.001: (goto_waypoint Tom sh1 sh6) [4.000]
13.001: (load_pallet Tom p2 sh6) [2.000]
15.001: (goto_waypoint Tom sh6 sh1) [4.000]
19.001: (unload_pallet Tom p2 sh1) [1.500]
19.002: (goto_waypoint Jerry sh5 sh6) [3.000]
22.002: (unload_pallet Jerry p1 sh6) [1.500]

```

Figure 13: The HPlan without the action (*goto_waypoint Tom sh1 sh2*) with a duration of 23.502

For example, given the example plan in Figure 5 the user might ask:

“Why is (*set_shelf Tom sh6*) used, rather than (*load_pallet Tom p2 sh6*)?”

The user might ask this because a goal of the problem is to deliver the pallet *p2* to the shelf *sh1*. As *Tom* is by the pallet, the user might question why *Tom* does not load the pallet in order to deliver it instead of setting up the shelf *sh6*.

To generate the HPlan, a compilation is formed such that the ground action $b = \text{ground}(n, \chi')$ appears in the plan in place of the action $a_i = \text{ground}(o, \chi)$. Given the example above $b = \text{ground}(\text{load_pallet}, \{\text{Tom}, p2, sh6\})$, and $a_i = \text{ground}(\text{set_shelf}, \{\text{Tom}, sh6\})$. Given a plan:

$$\pi = \langle a_1, a_2, \dots, a_n \rangle$$

The ground action a_i at state s is replaced with b , which is executed, resulting in state I' , which becomes the new initial state in the HModel. A time window is created for each durative action that is still executing in state s . These model the end effects of the concurrent actions. A plan is then generated from this new state with these new time windows for the original goal, which gives us the plan:

$$\pi' = \langle a'_1, a'_2, \dots, a'_n \rangle$$

The HPlan is then the initial actions of the original plan π concatenated with b and the new plan π' :

$$\langle a_1, a_2, \dots, a_{i-1}, b, a'_1, a'_2, \dots, a'_n \rangle$$

Specifically, the HModel Π' is:

$$\Pi' = \langle \langle Ps, Vs, As, arity \rangle, \langle Os, I', G, W \cup C \rangle \rangle$$

where:

- I' is the final state obtained by executing⁵ $\langle a_1, a_2, \dots, a_{i-1}, b \rangle$ from state I .

5. We use VAL to validate this execution. We use the add and delete effects of each action, at each happening (provided by VAL), up to the replacement action to compute I' .

- C is a set of time windows w_x , for each durative action a_j that is still executing in the state I' . For each such action, w_x specifies that the end effects of that action will become true at the time point at which the action is scheduled to complete. Specifically: $w_x = \langle (Dispatch(a_j) + Dur(a_j)) - (Dispatch(b) + Dur(b)), inf, u \rangle$ where $u = Eff(a_j)^- \cup Eff(a_j)^+ \cup Eff(a_j)^n$.

In the case in which an action a_j that is executing in state I' has an overall condition that is violated, this is detected when the plan is validated against the original model. As an example, given the user question above, the new initial state I' from the running example is shown in Figure 14.

```
(:init
  (not_occupied sh1) (not_occupied sh2) (not_occupied sh5)
  (connected sh1 sh2) (connected sh2 sh1) (connected sh2 sh3)
  (connected sh3 sh2) (connected sh3 sh4) (connected sh4 sh3)
  (connected sh4 sh5) (connected sh5 sh4) (connected sh5 sh6)
  (connected sh6 sh5) (connected sh6 sh1) (connected sh1 sh6)
  (pallet_at p1 jerry) (pallet_at p2 tom) (robot_at tom sh6)
  (at 3 (robot_at Jerry sh4)) (at 3 (not_occupied sh3))
  (= (travel_time sh1 sh2) 4) (= (travel_time sh1 sh6) 4)
  (= (travel_time sh2 sh1) 4) (= (travel_time sh2 sh3) 8)
  (= (travel_time sh3 sh2) 8) (= (travel_time sh3 sh4) 5)
  (= (travel_time sh4 sh3) 5) (= (travel_time sh4 sh5) 1)
  (= (travel_time sh5 sh4) 1) (= (travel_time sh5 sh6) 3)
  (= (travel_time sh6 sh5) 3) (= (travel_time sh6 sh1) 4))
(:goal (and (pallet_at p1 sh6) (pallet_at p2 sh1)))
```

Figure 14: The initial state I' which captures the state directly after executing the alternate action $b = (load_pallet\ Tom\ p2\ sh6)$ suggested by the user.

This captures the state I' , resulting from executing the actions a_1, a_2, a_3 , and b :

```
0.000: (goto_waypoint Tom sh5 sh6) [3.000]
0.000: (load_pallet Jerry p1 sh3) [2.000]
2.000: (goto_waypoint Jerry sh3 sh4) [5.000]
3.001: (load_pallet Tom p2 sh6) [2.000]
```

In this state *Tom* is at shelf *sh6* and has loaded the pallet *p2*. *Jerry* has loaded the pallet *p1* and is currently moving from shelf *sh3* to *sh4*. This new initial state is then used to plan for the original goals to get the plan π' , which, along with b and π , gives the HPlan. However, the problem is unsolvable from this state as a robot cannot set up a shelf whilst it is transporting a pallet, a shelf must be set up to unload a pallet, *Tom* and *Jerry* are both holding pallets, and there are no shelves set up. Therefore, neither *Tom* nor *Jerry* can unload a pallet at any of the shelves and so can not achieve the goal. By applying the user's constraint, and showing there are no more applicable actions, it answers the above question: "because by doing b rather than a , there is no way to complete the goals of the problem".

This compilation keeps the position of the replaced action in the plan, however, it may not be optimal. This is because we are only re-planning after the inserted action has been performed. The first half of the plan, because it was originally planned to support

a different set of actions, may now be inefficient, as shown by Borgo, Cashmore, and Magazzeni (2018).

If the user instead wishes to replace the action without necessarily retaining its position in the plan, then the add and remove compilations shown in Sections 4.2 and 4.3 can be applied iteratively. This is an example of how the compilations can be combined into something greater than the sum of its parts, that answers an entirely new question.

4.5 Reordering Actions

Given a plan π , a formal question Q is asked of the form:

Why is the operator o with parameters χ used before (after) the operator n with parameters χ' , rather than after (before)? where $o \neq n$ or $\chi \neq \chi'$

For example, given the example plan in Figure 5 the user might ask:

“Why is (unload_pallet Jerry p1 sh6) used before (unload_pallet Jerry p2 sh1), rather than after?”

A user might wonder what would be the outcome if *Jerry* delivered the pallets the other way around. There are the same amount of shelves to traverse between each of the delivery points so the user might wonder if there is a reason it was done in this order. They can therefore ask the question posed above and see what happens if *Jerry* delivered pallet $p2$ before $p1$.

The compilation to the HModel is performed in the following way. First, a directed-acyclic-graph (DAG) $\langle N, E \rangle$ is built to represent each ordering between actions suggested by the user. For example the ordering of Q is $a < b$ where $a = \text{ground}(o, \chi)$ and $b = \text{ground}(n, \chi')$.

This DAG is then encoded into the model Π to create Π' . For each edge $(a, b) \in E$ two new predicates are added: ordered_{ab} representing that an edge exists between a and b in the DAG, and traversed_{ab} representing that the edge between actions a and b has been traversed. The predicate traversed_{ab} is added as an effect for a and a precondition for b to ensure that a comes before b . The predicate ordered_{ab} is used to ensure that this ordering only applies to the ground actions a and b , and can speed up search with a wide suite of planners.

For each node representing a ground action $a \in N$, the action is disallowed using the compilation from Section 4.3. Then, for each such action a new operator o_a is added to the domain, with the same functionality of the original operator o . The arity of the new operator, $\text{arity}(o_a)$ is the combined arity of the original operator plus the arity of all of a 's sink nodes. Specifically, the HModel Π' is:

$$\Pi' = \langle \langle Ps', Vs, As', \text{arity}' \rangle, \langle Os, I', G, W \rangle \rangle$$

where:

- $Ps' = Ps \cup \{\text{ordered}_{ab}\} \cup \{\text{traversed}_{ab}\}, \forall (a, b) \in E$
- $As' = As \cup \{o_a\}, \forall a \in N$
- $\text{arity}'(x) = \text{arity}(x), \forall x \in \text{arity}$

- $arity'(o_a) = arity(o) + \sum_{(a,b) \in E} arity(b), \forall a \in N$
- $arity'(ordered_{ab}) = arity(a) + arity(b), \forall (a, b) \in E$
- $arity'(traversed_{ab}) = arity(b), \forall (a, b) \in E$
- $I' = I \cup ground(ordered_{ab}, \chi + \chi'), \forall (a, b) \in E$, where χ and χ' are the parameters of a and b , respectively.

In the above, we abuse the *arity* notation to specify the arity of an action to mean the arity of the operator from which it was ground; e.g. $arity(a) = arity(o)$ where $a = ground(o, \chi)$.

Each new operator o_a extends o with the precondition that all incoming edges must have been traversed, i.e. the source node has been performed. The effects are extended to add that its outgoing edges have been traversed. That is:

$$\begin{aligned} Pre_{\vdash}(o_a) &= Pre_{\vdash}(o) \cup \{ordered_{ab} \in Ps', \forall b\} \\ &\quad \cup \{traversed_{ca} \in Ps', \forall c\} \\ Eff_{\dashv}^+(o_a) &= Eff_{\dashv}^+(o) \cup \{traversed_{ab} \in Ps', \forall b\} \end{aligned}$$

This ensures that the ordering the user has selected is maintained within the HPlan.

As the operator o_a has a combined arity of the original operator plus the arity of all of a 's sink nodes, there exists a large set of possible ground actions. However, for all $b \in N$, $ordered_{ab}$ is a precondition of o_a ; and for each edge $(a, b) \in E$ the ground proposition $ground(ordered_{ab}, \chi + \chi')$ is added to the initial state to represent that the edge exists in the DAG. This significantly prunes the possible, valid, groundings of o_a .

Given the user question above, two new operators *node_unload_pallet_Jerry_p2_sh1* (shown in Figure 15) and *node_unload_pallet_Jerry_p1_sh6* will be added to the domain. These extend operator *unload_pallet* from Figure 3 as described above. The HPlan generated is shown in Figure 16. In this case the plan does not contain the action *unload_pallet Jerry p1 sh6* and instead uses *Tom* to deliver the pallet *p1*. If the user wants both the before and after actions to be performed in the plan they can successively apply the add compilation shown in Section 4.2.

4.6 Forbid an Action Outside a Time Window

Given a plan π , a formal question Q is asked of the form:

Why is the operator o with parameters χ used outside of time $lb < t < ub$, rather than only being allowed within this time window?

For example, given the example plan in Figure 5 the user might ask:

*“Why is (*unload_pallet Jerry p2 sh1*) used outside the interval 11 to 13, rather than being restricted to that time window?”*

From the HPlan provided as a result of the question asked in Section 4.5, the user might wonder why changing the original order of the actions $a = unload_pallet\ Jerry\ p2\ sh6$ and $b = unload_pallet\ Jerry\ p2\ sh1$, caused b to be performed at the time 23.002 rather than at 11.002, which was the time that action a was originally performed. The user might then ask

```

(:durative-action node_unload_pallet_Jerry_p2_sh1
:parameters (?v - robot ?p - pallet ?shelf - waypoint
?v0 - robot ?p0 - pallet ?shelf0 - waypoint)
:duration (= ?duration 1.5)
:condition (and (at start (pallet_at ?p ?v))
(over all (robot_at ?v ?shelf)) (over all (scanned_shelf ?shelf)))
(at start (ordered-node-unload_pallet-Jerry-p2-sh1-unload_pallet
-Jerry-p1-sh6 ?v ?p ?shelf ?v0 ?p0 ?shelf0))
:effect (and (at end (not_holding_pallet ?v))
(at end (pallet_at ?p ?shelf)) (at start (not (pallet_at ?p ?v)))
(at end (traversed-node-unload_pallet-Jerry-p2-sh1-unload_pallet
-Jerry-p1-sh6 ?v0 ?p0 ?shelf0)))
)

```

Figure 15: An operator added to the original domain to capture an ordering constraint between actions. The operator extends the original *unload_pallet* operator. Note that the long generated predicate names have overrun the lines in the pre- and post-conditions.

```

0.000: (goto_waypoint jerry sh3 sh2) [8.000]
0.000: (goto_waypoint tom sh5 sh6) [3.000]
3.001: (set_shelf tom sh6) [1.000]
4.001: (goto_waypoint tom sh6 sh5) [3.000]
7.002: (goto_waypoint tom sh5 sh4) [1.000]
8.001: (goto_waypoint jerry sh2 sh1) [4.000]
8.003: (goto_waypoint tom sh4 sh3) [5.000]
12.002: (set_shelf jerry sh1) [1.000]
13.002: (goto_waypoint jerry sh1 sh6) [4.000]
13.003: (load_pallet tom p1 sh3) [2.000]
15.003: (goto_waypoint tom sh3 sh4) [5.000]
17.002: (load_pallet jerry p2 sh6) [2.000]
19.002: (goto_waypoint jerry sh6 sh1) [4.000]
20.004: (goto_waypoint tom sh4 sh5) [1.000]
23.002: (node-unload_pallet-jerry-p2-sh1 jerry p2 sh1 jerry p1 sh6) [1.500]
23.003: (goto_waypoint tom sh5 sh6) [3.000]
26.003: (node-unload_pallet-tom-p1-sh6 tom p1 sh6) [1.500]

```

Figure 16: The HPlan with the action (*unload_pallet Jerry p2 sh1*) before (*unload_pallet Jerry p1 sh6*) with a duration of 27.503

the question above about the original plan, to receive an explanation for why the action b cannot be performed at the same time as when a was performed.

To generate the HPlan, the planning model is compiled so that the ground action $a = \text{ground}(o, \chi)$ can only be used between times lb and ub . To do this, the original operator o is replaced with two operators o_a and o_{-a} , which extend o with extra constraints.

Operator o_{-a} replaces the original operator o for all other actions $\text{ground}(o, \chi')$, where $\chi' \neq \chi$. The action $\text{ground}(o_{-a}, \chi)$ cannot be used (this is enforced using the compilation for forbidding an action described in Section 4.3). Operator o_a acts as the operator o specifically for the action $a = \text{ground}(o, \chi)$, which has an added constraint that it can only be performed

between lb and ub . Specifically, the HModel Π' is:

$$\Pi' = \langle \langle Ps', Vs, As', arity' \rangle, \langle Os, I', G', W' \rangle \rangle$$

where:

- $Ps' = Ps \cup \{can_do_a, not_done_a\}$
- $As' = \{o_a, o_{-a}\} \cup As \setminus \{o\}$
- $arity'(x) = arity(x), \forall x \in arity$
- $arity'(can_do_a) = arity'(not_done_a) = arity'(o_a) = arity'(o_{-a}) = arity(o)$
- $I' = I \cup \{ground(not_done_a, \chi)\}$
- $G' = G \cup \{ground(not_done_a, \chi)\}$
- $W' = W \cup \{\langle lb, ub, ground(can_do_a, \chi) \rangle\}$

where the new operators o_{-a} and o_a extend o with the delete effect not_done_a and the precondition can_do_a , respectively. i.e:

$$\begin{aligned} Eff_{+}^{-}(o_{-a}) &= Eff_{+}^{-}(o) \cup \{not_done_a\} \\ Pre_{+}(o_a) &= Pre_{+}(o) \cup \{can_do_a\} \end{aligned}$$

As the proposition $ground(can_do_a, \chi)$ must be true for $ground(o_a, \chi)$ to be performed, this ensures that the action a can only be performed within the times lb and ub . Other actions from the same operator can still be applied at any time using the new operator o_{-a} . As in Section 4.3 we make sure the ground action $ground(o_{-a}, \chi)$ can never appear in the plan.

For example, given the user question above, the operator *unload_pallet* from Figure 3 is extended to o_{-a} and o_a as shown in Figure 17.

The initial state is extended to include the proposition (*not_done_unload_pallet Jerry p2 sh1*) and the time window $\langle 11, 13, (can_do_load_pallet\ Jerry\ p2\ sh1) \rangle$, which enforces that the proposition is true only between the times 11 and 13. The resulting HPlan is shown in Figure 18, in this case the action (*unload_pallet Jerry p2 sh1*) is no longer present in the plan as *Tom* delivers the pallet *p2* instead.

4.7 Add an Action Within a Time Window

Given a plan π , a formal question Q is asked of the form:

Why is the operator o with parameters χ not used at time $lb < t < ub$, rather than being used in this time window?

For example, given the example plan in Figure 5 the user might ask:

“Why is (*unload_pallet Jerry p2 sh1*) not used between times 11 and 13, rather than being used in this time window?”

```

(:durative-action unload_pallet_nota
:parameters (?v - robot ?p - pallet ?shelf - waypoint)
:duration (= ?duration 1.5)
:condition (and (at start (pallet_at ?p ?v))
(over all (robot_at ?v ?shelf)) (over all (scanned_shelf ?shelf)))
:effect (and (at end (not_holding_pallet ?v))
(at end (pallet_at ?p ?shelf))
(at start (not (pallet_at ?p ?v)))
(at start (not (not-done-unload_pallet ?v ?p ?shelf)))))

(:durative-action unload_pallet_a
:parameters (?v - robot ?p - pallet ?shelf - waypoint)
:duration (= ?duration 1.5)
:condition (and (at start (pallet_at ?p ?v))
(over all (robot_at ?v ?shelf))
(over all (scanned_shelf ?shelf))
(over all (applicable-unload_pallet ?v ?p ?shelf)))
:effect (and (at end (not_holding_pallet ?v))
(at end (pallet_at ?p ?shelf))
(at start (not (pallet_at ?p ?v)))))

```

Figure 17: The PDDL2.1 representation of the operators o_{-a} and o_a .

```

0.000: (goto_waypoint tom sh5 sh6) [3.000]
0.000: (load_pallet jerry p1 sh3) [2.000]
2.000: (goto_waypoint jerry sh3 sh4) [5.000]
3.001: (set_shelf tom sh6) [1.000]
4.001: (goto_waypoint tom sh6 sh1) [4.000]
7.001: (goto_waypoint jerry sh4 sh5) [1.000]
8.001: (set_shelf tom sh1) [1.000]
9.001: (goto_waypoint tom sh1 sh6) [4.000]
13.001: (load_pallet tom p2 sh6) [2.000]
15.001: (goto_waypoint tom sh6 sh1) [4.000]
19.001: (unload_pallet_nota tom p2 sh1) [1.500]
19.002: (goto_waypoint jerry sh5 sh6) [3.000]
22.002: (unload_pallet_nota jerry p1 sh6) [1.500]

```

Figure 18: The HPlan produced from solving the HModel that allows the action (*unload_pallet Jerry p2 sh1*) to only be performed between the times 11 and 13. *Tom* was, therefore, chosen to deliver the package instead.

The HPlan in Figure 18 shows the user that there is a better plan which does not have the action in this time window. However, the user may only be satisfied once they have seen a plan where the action is performed in their given time window. To allow this the action may have to appear in other parts of the plan as well.

This constraint differs from Section 4.6 in two ways: first the action is now forced to be applied in the time window, and second the action can be applied at other times in the plan. This constraint is useful in cases such as a robot that has a fuel level. As fuel is depleted when travelling between waypoints, the robot must refuel, possibly more than once. The

user might ask “why does the robot not refuel between the times x and y (as well as the other times it refuels)?”.

To generate the HPlan, the planning model is compiled into a form that forces the ground action, $a = \text{ground}(o, \chi)$, to be used between times lb and ub , but can also appear at any other time. This is done using a combination of the compilation in Section 4.2 and a variation of the compilation in Section 4.6. The former ensures that new action $\text{ground}(o_a, \chi)$ must appear in the plan, and the latter ensures that the action can only be applied within the time window. The variation of the latter compilation is that the operator $o_{\neg a}$ is not included, and instead the original operator is kept in the domain. This allows the original action $a = \text{ground}(o, \chi)$ to be applied at other times in the plan. Given this, the HModel Π' is:

$$\Pi' = \langle \langle Ps', Vs, As', arity' \rangle, \langle Os, I, G', W' \rangle \rangle$$

where:

- $Ps' = Ps \cup \{\text{can_do_}a, \text{has_done_}a\}$
- $As' = As \cup \{o_a\}$
- $arity'(x) = arity(x), \forall x \in arity$
- $arity'(\text{can_do_}a) = arity'(\text{has_done_}a)$
 $= arity'(o_a) = arity(o)$
- $G' = G \cup \{\text{ground}(\text{has_done_}a, \chi)\}$
- $W' = W \cup \{\langle lb, ub, \text{ground}(\text{can_do_}a, \chi) \rangle\}$

Jerry cannot deliver the pallet $p2$ in the time period required by the user and so the plan is unsolvable.

4.8 Delay/Advance an Action

Given a plan π , a formal question Q is asked of the form:

Why is the operator o with parameters χ used at time t , rather than at least some duration t' earlier/later t ?

For example, given the example plan in Figure 5 the user might ask:

“Why is *set_shelf Tom sh1* used at time 8.001, rather than at least 8 minutes later?”

A user would ask this type of question when they expected an action to appear earlier or later in a plan. This could happen for a variety of reasons. In domains with resources that are depleted by specific actions, and are replenished by others, such as fuel for vehicles, these questions may arise often. A user might want an explanation of why a vehicle was refueled earlier or later than was expected. In this case the refuel action can be delayed or advanced to answer this question.

A user might ask the question posed above about our running example because they think that *Tom* is rushing to set up the shelf. *Tom* sets up the shelf *sh1* in preparation for

the delivery of the pallet $p2$ eight minutes into the plan. However, *Jerry* is not ready to deliver the pallet until the very end of the plan. *Tom* might be able to complete other goals before he is required to set up the shelf for the delivery. The reasoning behind the early preparation can be explained by delaying setting up the shelf until it is completely necessary and comparing the HPlan produced with the original solution.

To generate the HPlan, the planning model is compiled such that the ground action $a = \text{ground}(o, \chi)$ is forced to be used in time window w which is at least t' before/after t . This compilation is an example of a combination of two other compilations: adding an action (in Section 4.2) and forbidding the action outside of a time window (in Section 4.6). The latter enforces that the action can only be applied within the user specified time window, while the former enforces that the action must be applied. The HModel Π' is:

$$\Pi' = \langle \langle Ps', Vs, As', arity' \rangle, \langle Os, I', G', W' \rangle \rangle$$

where:

- $Ps' = Ps \cup \{can_do_a, not_done_a, has_done_a\}$
- $As' = \{o_a, o_{-a}\} \cup As \setminus \{o\}$
- $arity'(x) = arity(x), \forall x \in arity$
- $arity'(can_do_a) = arity'(not_done_a) =$
 $arity'(has_done_a) = arity'(o_a) =$
 $arity'(o_{-a}) = arity(o)$
- $I' = I \cup \{\text{ground}(not_done_a, \chi)\}$
- $G' = G \cup \left\{ \begin{array}{l} \text{ground}(not_done_a, \chi), \\ \text{ground}(has_done_a, \chi) \end{array} \right\}$
- $W' = W \cup \left\{ \begin{array}{l} \text{before} : \langle 0, tReal, \text{ground}(can_do_a, \chi) \rangle \\ \text{after} : \langle tReal, inf, \text{ground}(can_do_a, \chi) \rangle \end{array} \right\}$

where $tReal$ is $t \pm t'$ and the new operators o_a and o_{-a} both extend o . The latter with the delete effect not_done_a , while o_a extends o with the precondition can_do_a and the add effect has_done_a ; i.e.:

$$\begin{aligned} Eff_{-}^{-}(o_{-a}) &= Eff_{-}^{-}(o) \cup \{not_done_a\} \\ Pre_{\leftrightarrow}(o_a) &= Pre_{\leftrightarrow}(o) \cup \{can_do_a\} \\ Eff_{+}^{+}(o_a) &= Eff_{+}^{+}(o) \cup \{has_done_a\} \end{aligned}$$

This ensures that the ground action $a = \text{ground}(o_a, \chi)$ must be present in the plan between the times 0 and $tReal$, or $tReal$ and inf , depending on the user question, and between those times only. In addition, the user selected action is forced to be performed using the same approach as in Section 4.2.

The HPlan produced for the users question is shown in Figure 19. The delayed action ($set_shelf\ tom\ sh1$) is now performed at time 17 which, as the action takes one minute, would allow *Jerry* to unload the pallet. However, *Tom* is blocking *Jerry* from getting to shelf $sh1$. Consequently, *Jerry* has to wait for *Tom* to evacuate the shelf which delays the completion of the delivery by 7.5 minutes. Additionally, it can be seen from the plan that *Tom* does not contribute to the completion of any other goals in the time before setting up shelf $sh1$.

```

0.000: (goto_waypoint tom sh5 sh6) [3.000]
0.000: (set_shelf_nota jerry sh3) [1.000]
1.000: (load_pallet jerry p1 sh3) [2.000]
3.000: (goto_waypoint jerry sh3 sh4) [5.000]
3.001: (set_shelf_nota tom sh6) [1.000]
4.001: (goto_waypoint tom sh6 sh1) [4.000]
8.001: (goto_waypoint jerry sh4 sh5) [1.000]
9.002: (goto_waypoint jerry sh5 sh6) [3.000]
12.002: (unload_pallet jerry p1 sh6) [1.500]
13.503: (load_pallet jerry p2 sh6) [2.000]
17.000: (set_shelf_a tom sh1) [1.000]
18.000: (goto_waypoint tom sh1 sh2) [4.000]
22.001: (goto_waypoint jerry sh6 sh1) [4.000]
26.001: (unload_pallet jerry p2 sh1) [1.500]
    
```

Figure 19: The HPlan with the action (*set_shelf Tom sh1*) performed at least 8 minutes later than it was originally performed.

4.9 Composition of Compilations

Each compilation defined in this section can be used to answer one of the formal questions from the contrastive taxonomy that was identified in our user study. However, through the iterative approach described in Definition 5 the set of questions that can be answered is not restricted to the formal questions found in the Contrastive Taxonomy. Instead a composition of these compilations can be used to produce more complex constraints that answer a much wider set of questions. More complex questions that are not easy to specify without refinement can be posed through the iterative process of query and feedback. Moreover, humans themselves have trouble understanding a decision from a “one shot” justification, they are more likely to comprehend a decision through a conversational process resulting in a more complete explanation (Hilton, 1990).

For example, consider the multiple questions asked in sequence q_1, q_2, \dots, q_n that have constraints $\phi_1, \phi_2, \dots, \phi_n$. The user could instead have asked a single complex question q_x that has the corresponding constraint ϕ_x :

$$\Pi \times \phi_x = ((\Pi \times \phi_1) \times \phi_2) \dots \times \phi_n$$

This compilation $\Pi \times \phi_x$ would have produced the same HPlan as the final HPlan resulting from the iterative process. However, this assumes that the user knows the question q_x in advance. In practice, each question might have been prompted by the result of the previous iteration, allowing the user to refine their question during the process.

This refinement also has the consequence that the user is able to pose questions about artefacts and processes of the plan that are not obviously representable in the model. As an example a user might want to know why the pallet *p1* took too long to be transported from shelf *sh3* to *sh6*. This question refers to the time between two ground actions in the plan, and the vocabulary of the model does not allow a constraint on this time to be expressed. However, through the iterative process it is possible to incrementally converge to a set of constraints that force these two events to happen closer together in time. Moreover, it is possible to follow this process without explicitly and immediately defining the duration

that the user considers to be “too long”, instead allowing the user to refine their question as their understanding grows.

That these compilations can be used to produce more complex constraints that answer a much wider set of questions can be stated more strongly as: for every valid plan π for a model Π , there exists a sequence of constraints, ϕ_1, \dots, ϕ_n , such that π is the only valid plan for $((\Pi \times \phi_1), \dots, \phi_n)$. Trivially, we can achieve any expected HPlan by iteratively applying the replace compilation shown in 4.4. In practice our user study in Section 6.2 showed that by using a variety of questions, the users converged quickly on their desired plans.

4.10 Justified User Suggestions

For a planning model Π with goals G there can be many valid plans that satisfy G , which we call the space of plans for a planning model. Generating the plan that will best satisfy the user at each stage of the exploration process is not guaranteed. Firstly, temporal planning tasks are intractable and in fact in the general case belongs to the complexity class EXPSPACE-complete (Rintanen et al., 2007) and the introduction of numeric variables makes the problem undecidable (Helmert, 2002). Our approach is limited by these impediments, just as a human might try to explain a decision they have made. Secondly, even should an optimal plan be returned, it might not be the plan that most increases the user’s understanding of the problem, or provides the fastest route to concluding the plan exploration process.

However, while it might not be possible to completely specify the metric of user satisfaction in a plan, it is possible to make some assumptions. One reasonable assumption is that the user wants to see their suggestion have an impact in the plan. When a user questions why an action was not used in the plan, a hypothetical plan containing that action would not be satisfactory if its effects are immediately undone, or it does not contribute towards a goal. Fink and Yang (1992) use “justified actions” to refer to actions that are necessary for achieving a goal. That is to say an action B is justified in a plan π if there is a sequence of actions in π where $a_1, \dots, B, \dots, a_n \models G$ and if we remove the action B then, $a_1, \dots, a_n \not\models G$. Similarly, a valid plan is *perfectly justified* if it does not have any legal proper subplan that also achieves the goal.

Our compilations alone do not guarantee that the action suggested by the user is justified in the resultant plan. The resultant plan should show, if possible, the user’s suggestion make a purposeful contribution to the satisfying the goals of the problem. In future work we aim to build on the compilations strategy in Section 4.2.1 to develop compilations that ensure that user suggestions are used purposefully within the plan. That is, to enforce that the resultant plan is perfectly justified, or that at least the user suggestion appears in every valid subplan.

A second open question is whether the assumption is indeed reasonable. While it might seem clear that the user should be interested in their suggestion contributing towards the goal, it should also be considered that the goal G does not necessarily capture all of the user’s preferences and interests in the problem. As an example, the user might be interested in investigating the space of plans to determine if there remains enough flexibility to add additional exploratory actions, or achieve goals that they do not yet know how to concretely specify.

5. Explainable Planning as an Iterative Process

In this section we present a framework within which we have implemented the iterative model restriction process described in Section 3.3 and instantiated through the compilations described in Section 4. We use an approach we call Explainable Planning as a Service (XAIP-as-a-service). This paradigm is motivated by Definition 5 and consists of an iterative conversational process between the user and the planning system. The user asks contrastive questions about a presented plan and receives explanations until the user terminates the process. Explainable Planning *as a service* means implementing the approach as a wrapper around an existing planning system that takes as input the current planning problem and domain model, the current plan, and the user’s question. It has the ability to invoke the existing planning system on hypothetical problems in order to address contrastive questions. In Section 6 we present the results of the user study conducted with this XAIP-as-a-service framework, alongside an evaluation of the computational costs and effectiveness of the compilations.

The XAIP-as-a-service paradigm has the benefit that the known and trusted planner and model can be used to provide explanations. At each step a new hypothetical plan is created using the planner chosen by the user, and is validated against the user’s original trusted model. The user will not accept an explanation generated using a model that differs from the original one that is potentially verified and trusted. Hence the explanation generated using the model revised with constraints has to be validated against the original model. In other words, the contrastive explanation should contain an executable plan which leads to the goal state that the original planner could have created using the original model. This satisfies the requirement of Definition 4, that a model restriction satisfies the condition that any plan for the restricted model is also a plan for the original model. Updates to the model serve to force decisions from the planner and so explore the consequences of those decisions. Figure 20 summarises the implementation described in Definition 5 and user interaction illustrated in Figure 8, following these steps:

- Step 1:** The *XAIP Service* takes as input the planning problem and the domain, the plan, and a question from the user.
- Step 2:** The contrastive question implies a *hypothetical model* characterised as an additional set of constraints on the actions and timing of the original problem. These constraints can then be compiled into a revised domain model (HModel) suitable for use by the original planner.
- Step 3:** The *original planner* uses the HModel as input to produce the *hypothetical plan* (HPlan) which contains the user suggestion.
- Step 4:** The XAIP Service validates the HPlan according to the original model.
- Step 5:** The original plan and HPlan are shown to the user, with differences highlighted.
- Step 6:** The user may choose to repeat the process from Step 1, selecting the original model or any HModel and a new question.

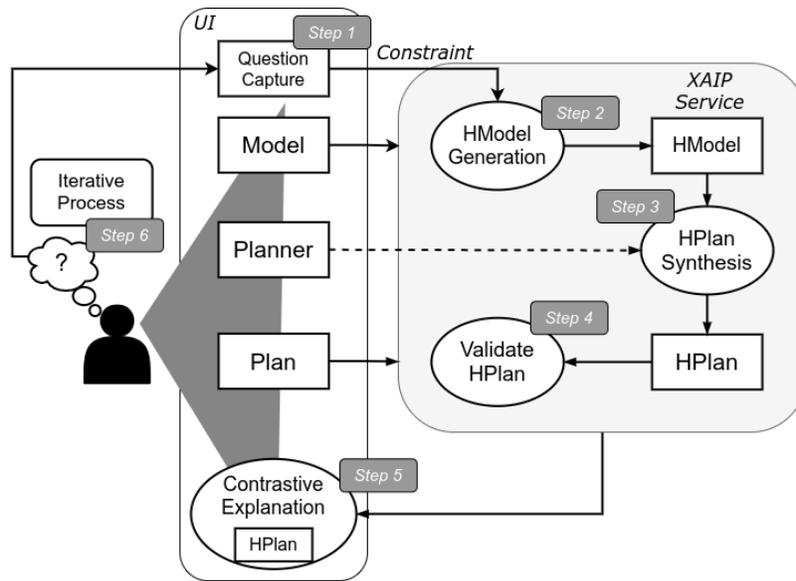


Figure 20: Proposed approach for Explainable Planning as a service. Starting from step 1, a user asks a contrastive question about a plan for a model produced from their planner. The XAIP Service first generates the HModel from the user question and uses the same planner to produce an HPlan. The HPlan is then validated and used to generate a contrastive explanation. The process can then be repeated from step 1.

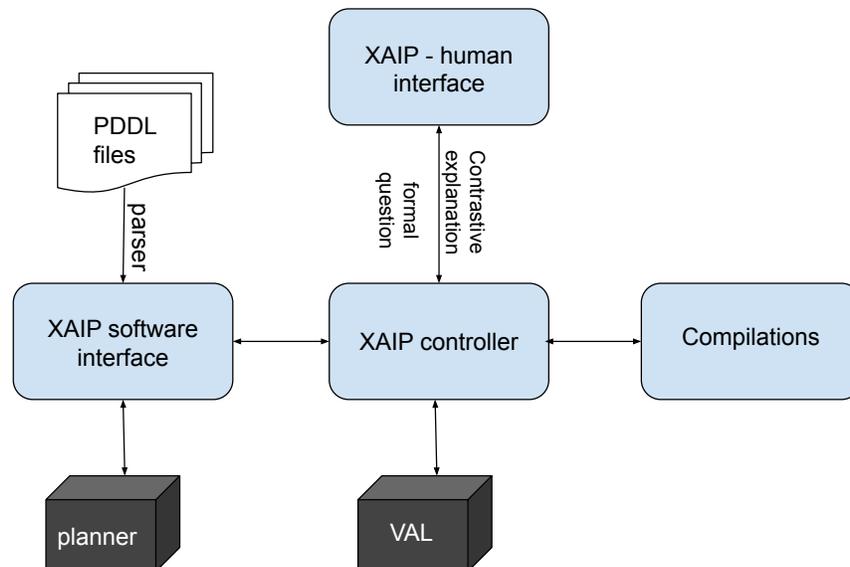
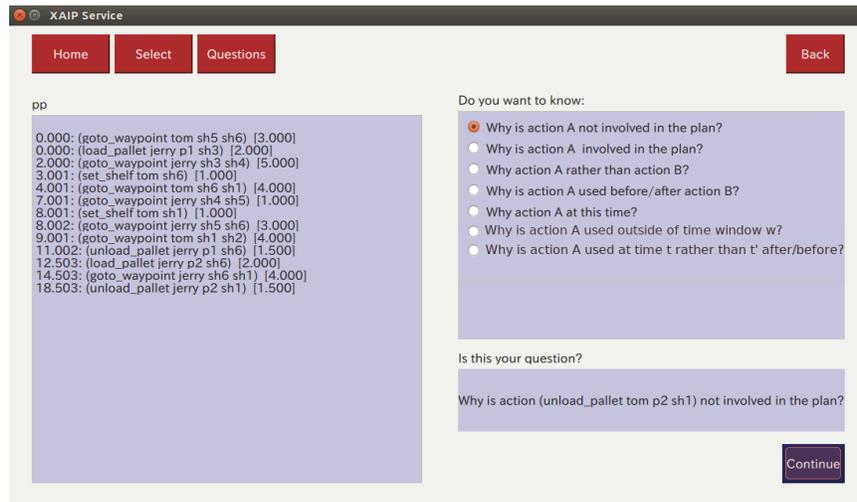


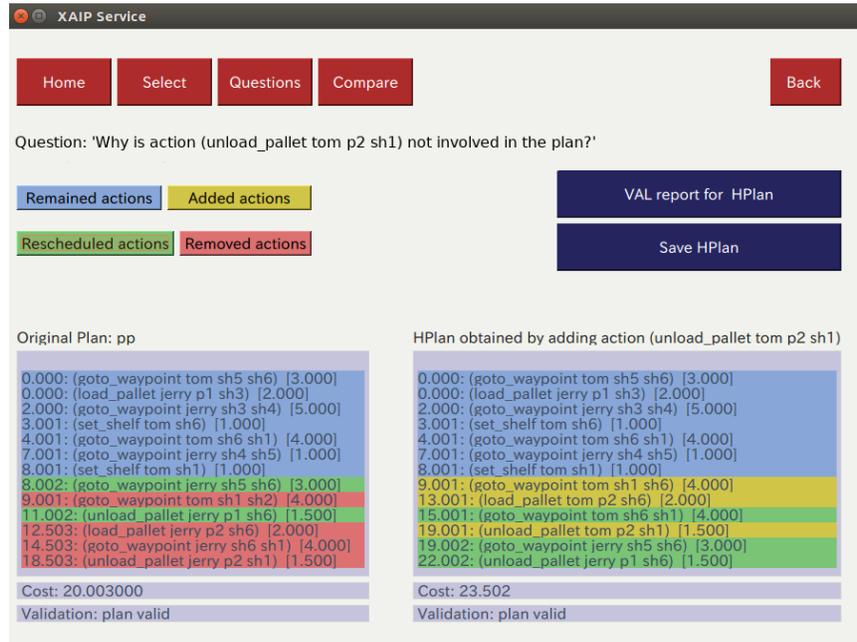
Figure 21: Architecture of the framework for Explainable Planning as a service.

5.1 Implementation Details

We implemented the XAIP-as-a-Service as modular framework for domains and problems written in PDDL2.1. (Fox & Long, 2003). This framework interfaces with any planner



(a) Plan visualisation and question selection.



(b) Explanation visualisation.

Figure 22: Screenshots of the graphical user interface of the *XAIP Service* framework. The first image displays the original plan, a user can formulate a question about the plan using the dialogue. The second image displays the side by side comparison of the original plan and the HPlan produced from the user's question. The differences in the plans are highlighted. Actions that are unchanged are coloured blue, those that are new in the HPlan are coloured yellow (only appear in the HPlan), actions are coloured green if they appear in both the plan and the HPlan but have different dispatch times, and actions are coloured red if they are removed from the plan (do not appear in the HPlan).

capable of reasoning with PDDL2.1, such as POPF (Coles et al., 2010), Metric-FF (Hoffmann, 2003), OPTIC (Benton, Coles, & Coles, 2012), etc. The architecture of the framework is illustrated in Figure 21. Interaction with a user is enabled through a graphical user interface, implemented with Qt-Designer. The modularity of the framework decouples the interfaces for providing user questions (*Step 1*), synthesising the HModel (*Step 2*), interfacing with the planner (*Step 3*), and returning HPlans to the user (*Step 5*).⁶

The process is controlled by the *XAIP controller* module of Figure 21. This module uses the interfaces of the other modules of the framework described below. The controller is also responsible for validating hypothetical plans against the original domain (*Step 4*), using the plan validation system *VAL* (Howey, Long, & Fox, 2004).

5.1.1 XAIP-HUMAN INTERFACE

The XAIP-human interface module of Figure 21 implements *Step 1*, and *Step 5* of the XAIP-as-a-service process. The module consists of a Qt interface through which the user is able to select an existing model (either the original model or a previous HModel), construct a question, and view the resulting HPlan.

The questions that can be constructed by the interface consist of those that are defined in the Contrastive Taxonomy in Table 5 in Section 2. A screenshot of the interface is shown in Figure 22a. In this screenshot the user has already selected a model and plan for which to ask a question, and selected the question option that corresponds to FQ1. The user populates the details of the question from an interactive panel so that the final question reads:

“Why is the action (*unload_pallet tom p2 sh1*) not involved in the plan?”

In the process of question selection, the interface adapts to the current domain and offers choices for the details of the action *A*, the domain operators and their valid groundings. The interface presents the HPlan to the user, as shown in Figure 22b. In this plan comparison both plans are shown side-by-side with differences highlighted. These differences include added actions which were not present in the original plan, actions which have been rescheduled/reordered, and actions which have been removed. The user is also able to compare the costs of each plan, and view the validation report produced by *VAL*. In Figure 22b the action that was suggested by the user, (*unload_pallet top p2 sh1*), appears in the HPlan at time 19.001. In case there is no plan generated, a message is displayed with the text “there is no plan obtained and compilation of the question failed”, as stated in Section 1 we do not attempt to give any further explanation than this.

5.1.2 XAIP SOFTWARE INTERFACE AND COMPILATIONS

The *XAIP software interface* module implements *Step 3* of the XAIP-as-a-service process, interacting with the planner to produce hypothetical plans. This is done by parsing the original domain and problem and storing the resultant model in an internal knowledge base. This knowledge base contains a collection of models that can be queried or passed to the planner.

6. All source code and example domain and problem files are open source and available online: <https://github.com/KCL-Planning/XAIPFramework>.

The *Compilation* module implements *Step 2* by providing an interface that given a formal question and model, applies the model restriction to produce the HModel. The *Compilations* module implements the model restriction in Section 4. When triggered by the event of the user selecting a model and formal question through the XAIP Human Interface, the Controller will fetch the model from the XAIP Software Interface, pass the model and question to the *Compilations* module, and store the resulting model in the XAIP Software Interface Knowledge Base.

6. Evaluation

Our evaluation falls into two parts: we evaluate the performance of the compilation of constraints by examining the planning time and plan quality produced for a large sample of problems, and we also present the user study that explores the value of the iterative process of plan explanation. The latter evaluation is based on observed interactions with an implemented system and is, therefore, more qualitative in style than the former evaluation. Nevertheless, both evaluations together serve to support our claims that the approach we have described provides a paradigm that allows users to usefully explore explanations of plans, by asking contrastive questions and being supplied plans in response to the constraints implied by those questions.

6.1 Performance Evaluation

Compilations can increase the difficulty of solving a problem so that it can no longer be solved in a reasonable time. For example, LTL constraints represented as Büchi-Automata and compiled into PDDL can scale very poorly and would not be appropriate for a real-time iterative dialogue with our system (Edelkamp, 2006). In order to evaluate the impact of the compilations listed in Section 4 we perform two experiments. The first is to evaluate the effect of single compilations on planning time, and the second is to determine the impact of multiple iterative compilations on planning time.

Explanation is a form of social interaction and takes the form of a conversation (Hilton, 1990). If it takes substantially longer to answer the explanatory question a user poses than to generate the original solution, it might be unreasonable to expect a user to want to wait for the explanation (depending on the context). In this case, the explanation process would be impractical in real world settings and that would undermine the value of the paradigm we have created of explanation as an iterative, conversational process. Moreover, it must not get exponentially harder to answer multiple iterations of questions.

The time to apply the compilations and generate the HModel is negligible for all the cases we consider so we do not take this into account in our evaluation.

We used four temporal domains from the recent ICAPS international planning competitions (IPC) (Long & Fox, 2003) in our experiments. The IPC produces a new set of benchmark domains each year to test the capabilities and progress made by AI planners for different types of problems. We selected domains to be varied in what they modelled and the most interesting in terms of explainability. These are the **ZenoTravel**, **Depots** (IPC3), **Crew Planning** and **Elevators** (IPC8) domains. In both experiments we used the Crew Planning and Elevators domains, in the first experiment we used the Depots domain

and used the ZenoTravel domain in the second. We explain the reason for the difference in the domains in the design of the second experiment in Section 6.1.2.

ZenoTravel is a logistics domain which models a scenario in which a number of pilots have to deliver a number of packages by plane. The planes can travel at different speeds which consumes fuel at different rates. The pilots must fly their planes at the correct speeds to minimise the time whilst maintaining the fuel to successfully deliver all of the packages. The Depots domain combines the transportation style problem of Logistics with the well-known Blocksworld domain. In this domain crates must be stacked in a certain order at their destinations. Trucks are used to move the crates between locations and hoists are used to stack the crates. The Crew Planning domain is designed to plan the itinerary of a crew on the International Space Station over a period of days. The crew have to complete tasks critical to maintenance of the station such as configuring thermals and facilitating the delivery of payloads, whilst also performing the tasks necessary for survival such as eating and sleeping. In the Elevators domain there are multiple elevators, with different speeds, that service portions of different building blocks. Each of the blocks share at least one mutual floor. The goal is to get a set of people to their desired floors using the elevators.

6.1.1 COMPILATION IMPACT BY QUESTION

Purpose We first designed an experiment to evaluate the impact each type of compilation in Section 4 has on the time taken to find a solution and the quality of the resultant solution. We designed this experiment to show that explanations can be produced in a reasonable time. We also wanted to see what effect compilations have on the quality of the solution. An explanation generated from an inefficient HPlan would not be satisfactory to the user. Although we cannot evaluate the quality of any given solution in the context of it's optimal solution, the large set of results for any problem will allow us to draw conclusions about possible inefficiencies. We also looked to determine whether there were any questions, or question types, for which it is harder to produce HPlans and so took longer to find solutions.

Design For each of the domains (Crew Planning, Depots, and Elevators) we selected four problems of varying complexity provided by the same IPC benchmark. We first used the planner to find the solutions to these as the control. Then, for each question type categorised in the Contrastive Taxonomy, we randomly generated the formal question and generated and solved the HModel, we repeated this ten times. All tests used a Core i7 1.9GHZ machine, and 16GB of memory. We used the POPF (Coles et al., 2010) planner and recorded all solutions found in the time allocated to test the effect compilations have on optimisation and solution quality. However, for the purpose of this experiment it is sufficient to evaluate if there are any obvious inefficiencies in our compilation approach, not to try to find the optimal plans for each constrained problem.

We conducted preliminary tests to determine the amount of planning time to allocate to each instance. We found that for each of the problems 3 minutes planning time was sufficient, other than problem 10 for the Depots domain which required 6 minutes. To illustrate the efficiency of our compilations, the experiment required many tests of each type of compilation, so we chose the minimum sufficient planning time.

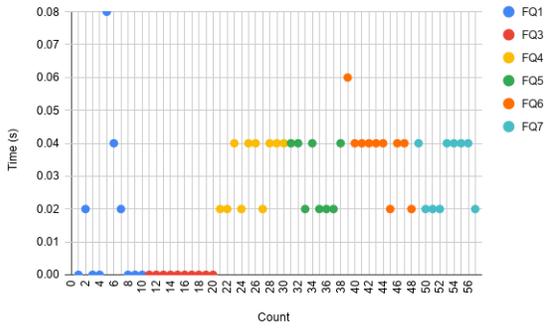
It was not practical to evaluate our approach with questions composed by humans. Therefore we randomly generated the questions used in our experiments. To ensure that the questions made sense, we had to take slightly different approaches to generating each question type. For each formal question other than FQ1 and FQ3, the actions were randomly selected from the original plan found from the appropriate model. This was all that was needed to generate FQ2. We took the extra precaution, with FQ5, to ensure that the order of the selected actions in the original plan was the opposite of the new order enforced by the question. For the formal questions FQ4, FQ6, and FQ7, time windows were also generated. The lower bound was generated using a pseudo-random number generator, constrained to within the original plan time. The upper bound was formed by first generating a number between 1.5 and 4 and then multiplying the number by the duration of the selected action. This produced a spectrum of time windows from those that are very tight to those that are quite relaxed, which mimicked how a user might ask these types of questions. To generate the formal questions FQ1 and FQ3 we had to create questions with actions that were not already present in the original plan. To do this we created a list of the possible grounded actions in the model and then randomly selected one of these grounded actions, that was not present in the original plan, to form the question. For FQ3 we also randomly selected an action from the original plan to replace. We then verified that the randomly selected (replacement) action was applicable in the state directly before the action chosen to be replaced. If the action was not applicable, a new action was generated and the process repeated until an applicable action was found. The rest of the compilation process then continued as normal.

The questions generated in this way might not be ones users would ask, being artificially constructed. However, evaluating how users interact with our framework was not the purpose of these experiments (that we consider in Section 6.2), but the broad coverage of generated questions gives a reasonable assessment of the performance of the planner on compiled HModels.

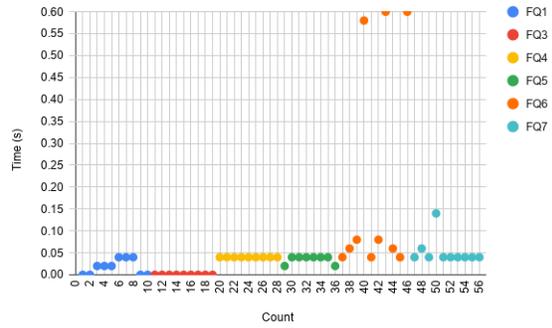
Results A subset of the results of this experiment is shown in Figures 23, 24, 25, 26, and 27. The results in these figures are a representative sample of the entire population of results and illustrate the performance characteristics we evaluated with this experiment. The full results of this experiment are available at <https://tinyurl.com/xairesults>.

Figures 23 and 24 demonstrate that our compilation approach does not significantly impact planning time. These graphs include results from every domain we evaluated as well as multiple problems and show that on average the planning time is not critically affected over multiple domains and problem variants. We show the easiest and hardest domains and problems, to reveal how the compilations affected planning time at the extremes of the range of difficulty of the problems.

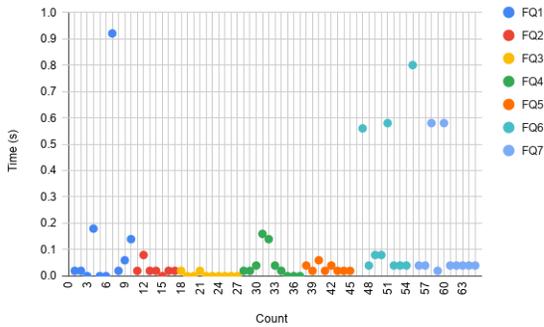
Figures 23 and 24 each contain four scatter graphs. The former containing the results of what we consider to be the easiest problems to solve in our test set, and the latter the most difficult. We classified the degree of difficulty for a problem as its size (number of literals in the problem) and the time taken to find any solution for the problem. However, the difficulty of a problem is only comparable for different problems in the same domain. Some domains are easier to solve than others, regardless of the problem size. Therefore, to keep the results representative we selected the easiest (hardest) problems to solve for



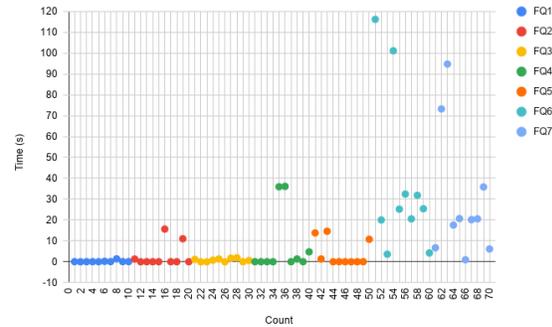
(a) Crew Planning Problem 1, Literals 30, Planning Time 0



(b) Crew Planning Problem 2, Literals 38, Planning Time 0

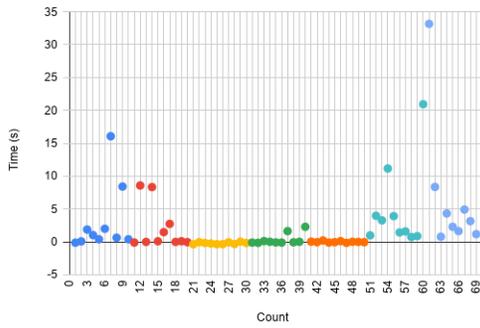


(c) Depots Problem 1, Literals 44, Planning Time 0

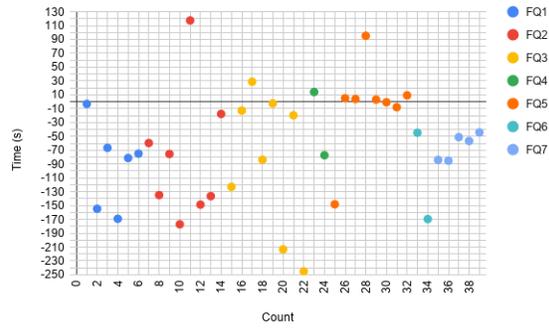


(d) Elevators Problem 1, Literals 86, Planning Time 0.1

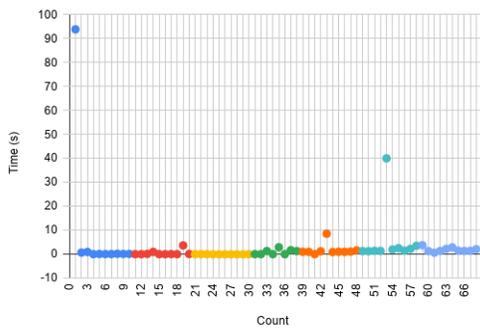
Figure 23: Scatter graph comparing the planning times of each compilation type over the simplest problems in each of the tested domains. Each sub-caption communicates the domain and problem from the IPC, the number of literals in the original problem and the time taken to solve the original problem. Each data point corresponds to a compilation of a question, colour coded by question type. The position of the data point on the y-axis corresponds to the increase (decrease) in planning time compared to the original problem. There is an arbitrary count to help distinguish between compiled problems.



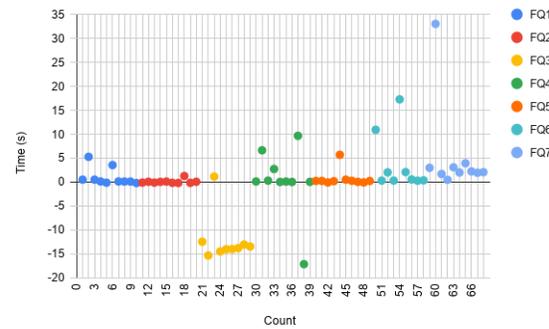
(a) Elevators Problem 5, Literals 138, Planning Time 0.38



(b) Depots Problem 10, Literals 192, Planning Time 245.06

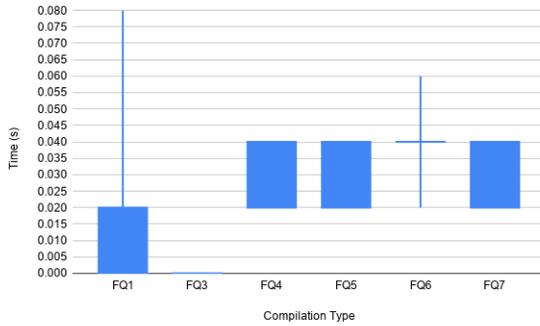


(c) Depots Problem 13, Literals 224, Planning Time 0.12

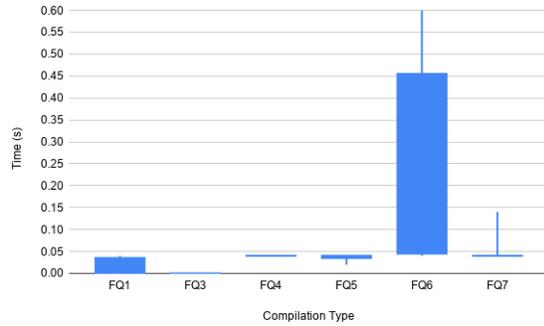


(d) Crew Planning Problem 20, Literals 270, Planning Time 17.45

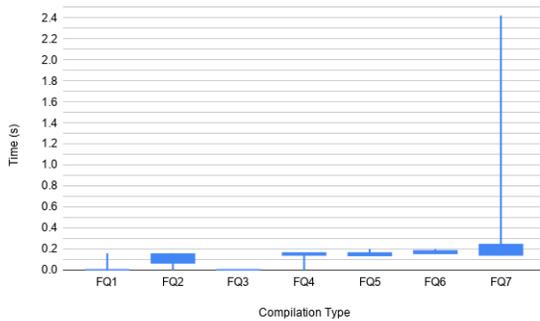
Figure 24: Scatter graph comparing the planning times of each compilation type over the hardest problems in each of the tested domains. Each sub-caption communicates the domain and problem from the IPC, the number of literals in the original problem and the time taken to solve the original problem. Each data point corresponds to a compilation of a question, colour coded by question type. The position of the data point on the y-axis corresponds to the increase (decrease) in planning time compared to the original problem. There is an arbitrary count to help distinguish between compiled problems.



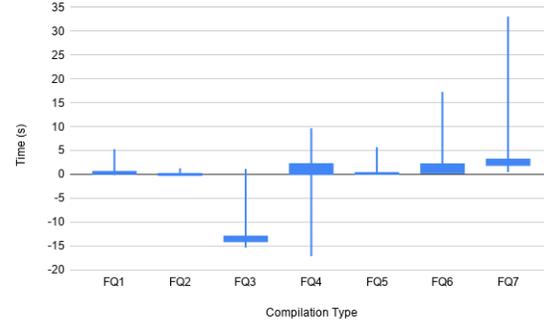
(a) Problem 1, Literals 30, Planning Time 0



(b) Problem 2, Literals 38, Planning Time 0



(c) Problem 5, Literals 62, Planning Time 0



(d) Problem 20, Literals 270, Planning Time 17.45

Figure 25: Box and whisker plots comparing the planning times of each compilation type in the Crew Planning domain over four problems. Each sub-caption communicates the problem from the IPC, the number of literals in the original problem and the time taken to solve the original problem. The y-axis shows the change in planning time for the HModel compared to the original model.

each domain, and the next easiest (hardest) domain-problem pairs from any of the three domains. This gives us the results from the four easiest (hardest) problems ordered from easiest to hardest, whilst making sure we have at least one set of results from each domain.

Each data point on the graphs in Figures 23 and 24 corresponds to a compilation, the colour corresponds to compilation type categorised by the Contrastive Taxonomy, the key is displayed on each graph. The horizontal axis of each graph displays an arbitrary count to distinguish between each compilation. Figures 23a and 23b do not have formal question FQ2, removing an action, because no plans could be found in the allocated time, we discuss why this is the case later. The vertical axis measures the difference between the time taken to find a plan for a compiled HModel and the time to find the original plan for the original model, in seconds. This means the zero on the vertical axis represents there being no difference in the time to find solution plans between the compiled model and the original model, a positive value means there was an increase in the time taken to find a solution for the compiled model, and the opposite holds for a negative value. The time taken to solve the original problem is shown in the sub-caption of each graph.

As these plots are used to demonstrate that there is no significant impact on the planning time for constrained problems, we have not shown any results using further optimising search after the discovery of the first solution. As any optimisations will only increase the planning time, it is unfair to compare the planning time of a heavily optimised plan to one with no optimisations. For example, for a model Π a planner might find the plan π in 10 seconds with a metric of $M(\pi) = 30$ and no further plans within its 3 minutes of allotted time. For a constrained model $\Pi \times \phi = \Pi'$ a planner might take 9, 10, and 120 seconds to find plans with metrics of 60, 40, and 30 respectively, and nothing further in its 3 minutes. How then should the planning times of the two models Π and Π' be compared? They both have 3 minutes to find solutions, however the quality of the solutions compared to the optimal solution is not known. It might seem sensible to select the two plans with the closest metrics for comparison. However, one of the discovered solutions could be optimal and the other very sub-optimal. For example the optimal solution for Π might be 5, whereas the optimal solution for Π' is 30. Then we have found an optimal solution in a large amount of time and a sub-optimal one quickly. It is clear that this is not a fair comparison. To use a comparison that is well-defined, only the results for the first plan found in the graphs is included in Figures 23, 24, and 25. However, the full table of results, including optimisations, can be found at <https://tinyurl.com/xairesults>.

The majority of points lie close to the horizontal axis showing that the compiled HModels in general are similar to the original models in terms of planning time. The median average increase in planning times for each of the domain-problem pairs are all below 4 seconds with one negative result which indicates an improvement in planning time. This substantiates our claim that the compilation of the constraints have an insignificant impact on the planning time. On average, a user will have to wait less than 4 seconds longer than the time taken to solve the original problem to see the outcome of their question and receive an explanation. The highest and lowest increase in planning time both occur in the Depots domain problem 10, shown in Figure 24b. The highest came from a compilation of the formal question FQ2, removing an action, which increased the planning time by 117.4 seconds. The lowest increase in planning time, and in fact improvement in planning time, came from a compilation of the formal question FQ3, replacing an action, which improved

the planning time by 245.02 seconds. The question types FQ6 and FQ7, the compilation of which is shown in Sections 4.7 and 4.8 respectively, tended to negatively impact the planning time the most, with a median increase of 1.10 and 1.18 seconds. Whereas the compilations for the question types FQ2 and FQ3 had the least effect on the planning time with an average of 0.02 and 0.00 seconds, respectively. We report the median averages to ensure extreme values do not skew the results.

The median effect on planning time across all problems ranged from -59.64 to 3.74 seconds. The domain-problem pairs that we considered to be easy had a range of 0.02 to 1.2 seconds, and the domain-problem pairs that we considered to be hard had a range of -59.64 to 0.925 seconds. Although this data suggests that compilations applied to harder problems have a much higher chance of improving the planning time than easier problems, actually the difficulty of the original problem did not have a significant effect on the planning time of the corresponding constrained problem. The results of a Mann-Whitney U test show that the sets of planning times from the easy and hard problems are statistically equal with $p < 0.05$. This shows that the impact compilations have on the planning time does not grow with the difficulty of the original problem.

Figure 25 contains four box-and-whisker plots, comparing the planning times of each compilation type in the Crew Planning domain. Each sub-figure displays the results for each of the problems we tested. This data shows that there is minimal difference between the types of compilations in their impact on the planning time. These graphs show results from each of the problems for the Crew Planning domain, this exemplifies a typical use case of our approach where a user may have a domain for which they have multiple problems, requiring explanations for each. We chose to display the results for this domain as it has the most varied results displaying relatively large and small ranges. This gives a good indication of how the compilations perform at their best and worst cases, and is largely representative of the other domains tested.

Each box and whisker plot corresponds to a data set of 10 compilations of a specific type and problem. The horizontal axis displays each of the compilation types labelled by their corresponding formal question. Figures 25a and 25b do not have formal question FQ2, because no plans could be found in the allocated time. The vertical axis represents the same as the graphs in Figures 23 and 24.

Each box in the plot represents the interquartile range (IQR) of the difference in planning times; that is, the middle 50 percent of planning times for HModels generated from one compilation type. The whiskers represent the largest and smallest difference in the planning times. The results suggest that the impact the compilations have on planning time is quite inconsequential, and that there are not any compilation types that are substantially more difficult.

The planning times for HModels generated from each compilation type are consistent across their problems. This can be seen with the overlapping interquartile ranges on most data sets. This shows that there is little variation in planning time between the types of compilations and seems to suggest that the difficulty of the original problems impacts the planning time more significantly than the type of compilation.

The interquartile range of the data sets is generally small, showing there to be little variation in the planning time for each compilation type. The IQRs of the data sets are also grouped around the horizontal axis showing that there is not a large increase or

decrease in planning time for the majority of the compilations across the problems. A compilation for a question type FQ7 in Figure 25d gave the greatest increase in planning time of 33.01 seconds. However, this is an extreme value for this data set as can be seen from the IQR of 2 – 3.08. There were a few other significant changes on planning time from compilations. For example, for a question type FQ1 in Figure 25a there was an increase of 0.08 seconds. This is quite substantial considering that the original planning time was essentially 0 seconds. However, in practice the increase in planning time is negligible. For each of these significant changes in planning time, the IQR of the data set shows that it is an extreme value.

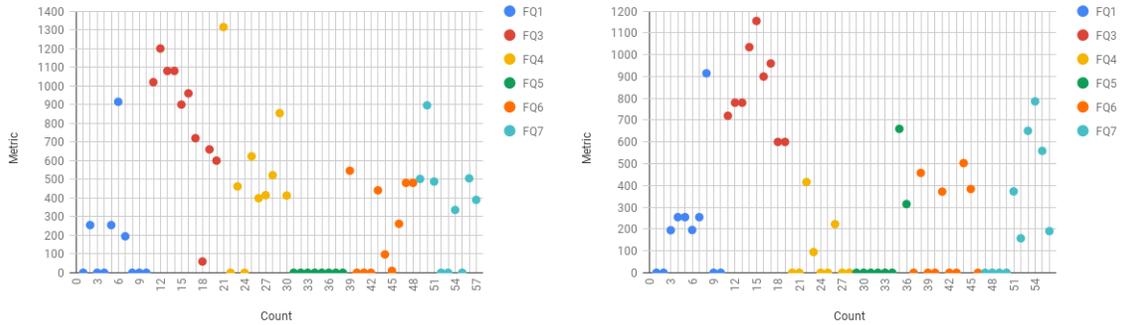
The largest IQR is for FQ4 in Figure 25d of 0.055 – 2.145. This is expected, because problem 20 is the hardest to solve for this domain. The other ranges in this problem are similar and also show little negative impact in the planning time. The data set with the largest interquartile range compared to the other compilations performed in the problem is FQ6, which corresponds to the compilation shown in Section 4.7, in Figure 25b with an IQR of 0.045 – 0.455. This stands out compared to the other results in the plot where the ranges are very small, and the values show close to zero, however, in practice an increase in planning time of 0.045 to 0.455 seconds is still negligible.

The results for problem 20, shown in Figure 25d show that for some compilations there was an improvement in planning time. For FQ4 this seems to be an extreme case where only the lowest planning time was an improvement of 17.09 seconds. Whereas, for compilations of the question type FQ3, the majority improved the planning time. In fact, across all problems the compilations for FQ3 had the least negative impact on planning time.

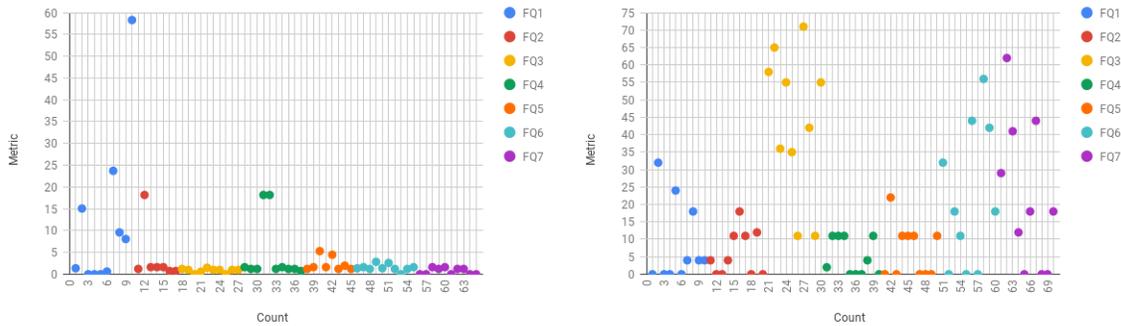
Figures 26 and 27 show the impact of the compilations on the solution quality for the easy and hard problems we defined earlier. In each of the three domains used in our experiments, the metric for quality is defined to be the total duration of the plan, keeping in mind that actions can be performed in parallel. In this case, the higher the metric, the worse the quality of the plan. The horizontal axis is the same as in Figures 23 and 24 whereas the vertical axis measures the difference between the metric for the plan for a compiled HModel and the metric for the original plan for the original model. Zero on the vertical axis represents no difference in metric for the plans from the original and compiled models, a positive value indicates the metric for a compiled model was higher, and vice versa for a negative value. Formal question FQ2 is not shown in Figures 26a and 26b, because no plans could be found in the allocated time.

As opposed to the results comparing the impact of the compilations on planning time, these results do contain the most optimised plan. This is because each problem had the same amount of time within which to find a solution, including the original problem. Although the ultimate planning time for two problems may have differed, they both had the same opportunity to improve. Therefore, we consider the quality of two plans found in the same allotted time comparable.

Nonetheless, as we observed earlier, the constraint added in response to a question could increase the metric for the solution significantly, but still be optimal under the new constraint. However, another constraint added to the same problem could marginally increase the metric, but be far from optimal. From the spread of the values in the results, potentially inefficient solutions are recognisable. Data points that lie in the same metric

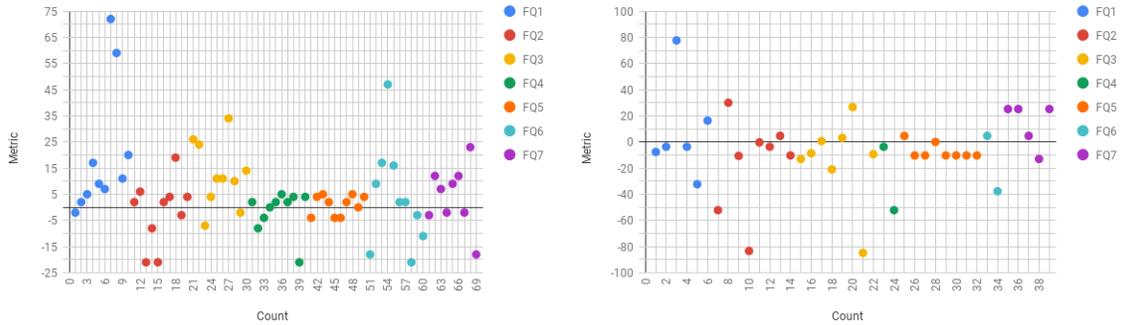


(a) Crew Planning Problem 1, Literals 30, Metric 1440 (b) Crew Planning Problem 2, Literals 38, Metric 1440

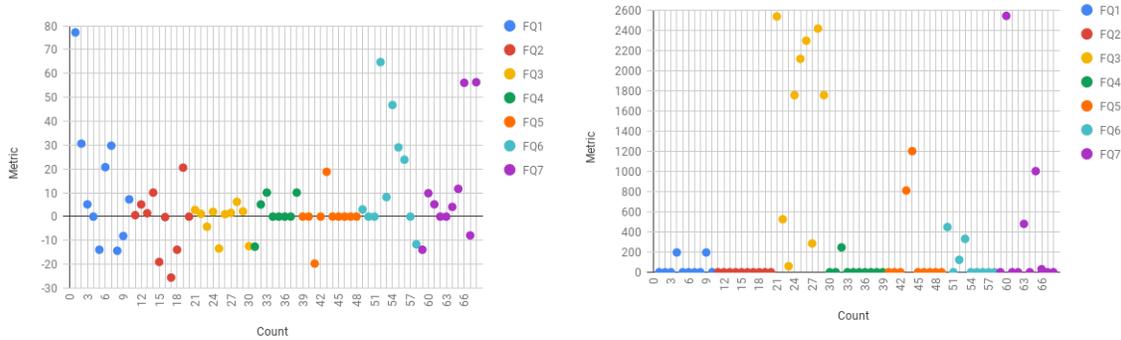


(c) Depots Problem 1, Literals 44, Metric 53.182 (d) Elevators Problem 1, Literals 86, Metric 80.001

Figure 26: Scatter graph comparing the metrics of each compilation type over the simplest problems in each of the tested domains. Each sub-caption communicates the domain and problem from the IPC, the number of literals in the original problem and the metric for the best plan found in the allotted time. Each data point corresponds to a compilation of a question, colour coded by question type. The position of the data point on the y-axis corresponds to the increase (decrease) in the plan metric compared to the original problem. There is an arbitrary count to help distinguish between compiled problems.



(a) Elevators Problem 5, Literals 138, Metric 90.002 (b) Depots Problem 10, Literals 192, Metric 256.171



(c) Depots Problem 13, Literals 224, Metric 91.601 (d) Crew Planning Problem 20, Literals 270, Metric 2880.001

Figure 27: Scatter graph comparing the metrics of each compilation type over the hardest problems in each of the tested domains. Each sub-caption communicates the domain and problem from the IPC, the number of literals in the original problem and the metric for the best plan found in the allotted time. Each data point corresponds to a compilation of a question, colour coded by question type. The position of the data point on the y-axis corresponds to the increase (decrease) in the plan metric compared to the original problem. There is an arbitrary count to help distinguish between compiled problems.

	Crew Planning				Depots				Elevators			
	1	2	5	20	1	3	10	13	1	2	3	5
FQ1	0	0	0	0	0	1	4	0	0	0	0	0
FQ2	10	10	3	0	3	2	2	0	0	0	0	0
FQ3	0	0	0	0	0	0	0	0	0	0	0	0
FQ4	0	1	0	0	0	4	8	2	0	2	1	0
FQ5	2	2	1	0	2	0	2	0	0	0	0	0
FQ6	0	0	0	1	0	2	8	0	0	2	0	0
FQ7	1	0	0	0	0	0	5	0	0	4	0	1

Table 3: Table showing the number of problems for which a solution could not be found grouped by question type. The table is divided into the domain type and then sub-divided by the problem number.

range are likely to have been caused by constraints that limit the search space of the problem, causing better quality solutions to be pruned away. Whereas lone data points such as for FQ1 in Figure 26c may indicate inefficient solutions.

The results show that the majority of the compilations do not negatively impact the metric significantly. Six of the seven compilation types have a median average increase in metric of less than 8.5, whilst the seventh has an average increase of 21.502. A compilation of the question type FQ3 for problem 20 of the Crew Planning domain had the largest impact on the metric, with an increase of 2541, shown in Figure 27d. A compilation of the question type FQ3 also lead to the biggest decrease in metric, an improvement of 84.83 to the original solution of problem 10 of the Depots domain, shown in 27b. The compilation for FQ3 on average lead to the largest increase in the metric, whereas the compilation for FQ2 had the lowest.

The compilations applied to the easier problems had no improvements on the metric. This could be because the solutions to the original problems were optimal. The compilations when applied to the easier problems had a worse effect on the metric than the harder problems, with a median increase of 3.681 compared to 2.002.

Although any constrained problems that were provably unsolvable were discounted and repeated, we did not repeat the tests for problems where the planner failed to find a solution. However, because there was no data for these problems the results were not displayed in the graphs. Table 3 displays the number of problems for which a solution could not be found within the allotted planning time. Overall, 86 of the 840 constrained problems failed to be solved within the required time. 29 of the 86 were derived from compilations applied to problem 10 for the Depots domain. This problem took the longest time to solve at 245.06 seconds, this being closer to the maximum planning time could be the reason for the failures in finding solutions. However as stated above there were many compilations that improved the planning time for this problem, so it was possible to solve (efficiently) with the correct constraints. 30 of the 120 HModels produced to answer questions of the type FQ2 were not solvable within the allotted time. This question is unlike the others as the constraint it produces enforces that an action cannot be present in a solution, rather than forcing it to. Therefore, the number of these questions that can

be asked about a plan is limited by the number of actions that appear in the plan. The limited choice on questions may have impacted the ability to find a solution, for example the plan for problem 1 for the Crew Planning domain had only 12 actions. If any of the actions were landmarks or crucial for achieving a goal in the plan, then removing these actions would have a large consequence to the resultant solution, potentially even making the problem unsolvable. Although none of the constrained problems were found to be provably unsolvable, removing an action from the larger problems with larger plans had less of an impact. The process for proving if a problem is unsolvable is difficult, therefore it is infeasible to definitively determine if these problems are provably unsolvable or not.

Analysis Many of the compilations when applied to the harder problems lead to better solution plans than the originals (see Figure 27). This is an important feature of the plan exploration process, where a user can suggest a counterfactual that leads to a better plan. This observation is very important: it demonstrates that the original plan is not optimal and that the addition of constraints in these cases actually narrows the search space in a way that reduces the work required to find a good quality plan in the remaining space. It is perhaps surprising that automatically generated questions that do not target observed weaknesses in an original solution should so often lead to improved plans. These results highlight the difficulty in finding optimal plans, the necessity to be able question unconvincing plans, and the effectiveness of our compilation approach in finding suspected improvements in plans.

The spread of results on metrics is noticeably larger than for the results on planning time. We believe that this is because the performance of the planner in finding a first solution to a problem is most significantly affected by the domain and the size of the problem being solved, neither of which is significantly altered by the compilation of constraints. In contrast, the quality of the best plan that can be found in a given time can be very much affected by the constraints in the problem: high quality solutions can be excluded by the addition of constraints, and poor quality solution branches can be pruned by the addition of constraints. This was discussed in further detail in Section 4, we see evidence for both patterns of behaviour in these results, which substantiates the discussion.

The compilations applied to problem 10 for the Depots domain, the results of which are shown in Figure 24b, produced unusual results. The planning times are considerably more varied than the results found for any other domain-problem pairs. The compilations also improved the planning time more than any other domain-problem pair. A possible reason for this is due to the planner having difficulties finding a solution to the original problem because of failing to select the choice branch leading to a simpler solution. For example in the search space there could be a more complex path which the planner is biased to go towards through a misleading heuristic. The constrained HModels produced for this problem might not have this issue. The heuristic for the new model could more accurately lead the search to a goal, or the complex parts of the search tree could be pruned away in the new model entirely.

The compilation for the formal question FQ3, shown in Section 4.4, had the lowest impact on the planning time out of any of the question types. This is likely due to the nature of the question requiring a part of the plan be fixed. Therefore unlike the other compilations, the compiled problem is smaller than the original. Although this does not guarantee that

the problem will always be easier to solve, as the specifics of the replacement performed by the compilation could have a substantial effect on the difficulty of the constrained problem.

6.1.2 PERFORMANCE OF ITERATED COMPILATIONS

We now present experiments exploring the effects of iteratively applying multiple compilations of constraints to problems. The iterated model restrictions that underpin the interaction we describe in Section 3.3 depends on the planner meeting the demands of planning for models in which multiple constraints have been compiled (using the approach described in Section 4). As we have already observed, the addition of constraints to a model can, in general, be expected to make the problem harder to solve. A well-known phenomenon affecting combinatorial optimisation problems is the *phase transition* (Hogg, Huberman, & Williams, 1996): members of a family of combinatorial problem include instances that are very easy to solve and other instances that are so over-constrained that it is trivial to determine that they are unsolvable. As constraints are added to the former, or removed from the latter, instances are created that are progressively more difficult to solve or more difficult to show unsolvable, respectively. Between these two advancing problem sets lies a transition from solvable to unsolvable and the problems at this boundary are typically the most difficult to tackle (which ever way the resolution lies). Thus, as we iteratively add constraints to a problem, we are pushing towards the phase transition where the problems are likely to become harder to solve. In these experiments we seek to determine the extent to which that expectation affects the performance in practice.

Purpose The second experiment was designed to evaluate the competence of the compilations when used within the iterative approach to explanations. For a user to engage in a conversational process with the explanation system they must receive efficient responses to their questions. As shown in the first experiment above, each type of compilation generally scales well with the complexity of the original problem. However, the results of the first experiment do not give any insight into how the compilations interact or interfere with one another and whether it is reasonable to expect a planner to produce solutions for more precisely constrained models. Therefore, a second experiment was designed to evaluate the impact of iterative compilations on the planning time and the quality of solutions.

Design In this experiment we used the same domains as the first experiment but instead of the Depots domain, the ZenoTravel domain was selected from IPC-3. This is because there was not enough feasibly solvable problems provided by the benchmarks for the Depots domain for the breadth of this experiment. We chose the ZenoTravel domain because it belongs to the same set of benchmarks as the Depots domain and there are clear justifications for the need of explanations in the domain. For each domain we selected ten problems of varying complexity provided by the IPC benchmarks. We selected problems with the same range of complexities across each of the domains. For the Crew Planning domain this was problems 1 to 10, for Elevators 1 to 9 and 14, and problems 3 to 12 were selected for the ZenoTravel domain. We did not select problems 1 or 2 in the ZenoTravel domain because they were too easy to solve, and we did not select problems 10 to 13 in the Elevators domain because they could not be solved within the designated time.

Problem Number	Crew Planning	Elevators	ZenoTravel
1, 1, 3	12	12	7
2, 2, 4	8	12	12
3, 3, 5	9	12	12
4, 4, 6	12	4	12
5, 5, 7	12	12	12
6, 6, 8	12	9	12
7, 7, 9	12	12	10
8, 8, 10	12	12	9
9, 9, 11	12	12	6
10, 14, 12	12	2	4

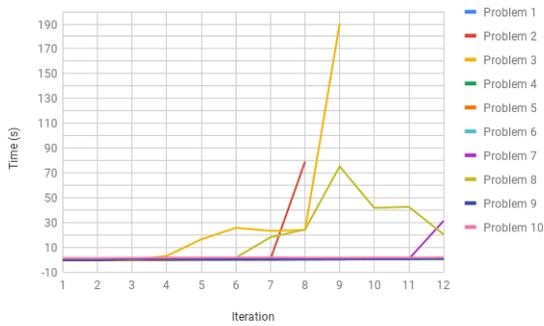
Table 4: Table showing the largest number of cumulative compilations applied to each domain-problem pair that was still solvable within the five minutes of allocated planning time. The Problem Number column denotes the problem for each domain in order, for example the bottom row shows the results from problem 10 for the Crew Planning domain, problem 14 for the Elevators domain, and problem 12 for the ZenoTravel domain.

We first solved each of these domain-problem pairs to get the original plans that are used as the control and to generate the first set of questions. We then selected a question type from the Contrastive Taxonomy at random and generated a random question of this type, using the same approach as described in Section 6.1.1. We then compiled this question into the original model to generate the HModel and used a planner to find the solution HPlan. We repeated this step for a total of twelve times, but each time generated a question from the last HPlan and compiled the question into the last HModel. The results from the user study in Section 6.2 suggest that users only ask five questions on average, however for the sake of robustness we simulated twelve for each problem. We generated questions using the same approach as the first experiment and disregarded questions that lead to provably unsolvable models.

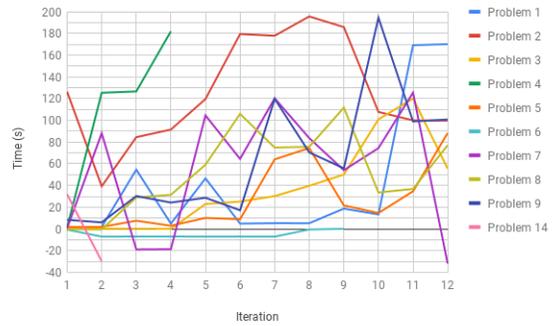
All tests used a Core i7 1.9GHZ machine, limited to five minutes and 16GB of memory. We increased the planning time from the first experiment by two minutes to offer a larger window through which to view any growth trends in the planning time for models with iterated constraints. We used the POPF (Coles et al., 2010) planner and recorded all solutions found in the time allocated to test the effect compilations have on solution quality with optimisation.

The compilation for the formal question FQ3, was not used in this experiment. This is because, by the nature of the question, the part of the plan up until the action that is being replaced is fixed. Unlike any other compilation, for all subsequent questions, the compilation is applied to an HModel that has a partially solved problem expressed as its initial state. Therefore, the FQ3 compilation is the only one that reduces the size of the problem to be solved, having a distorting effect on the impact of other compilations.

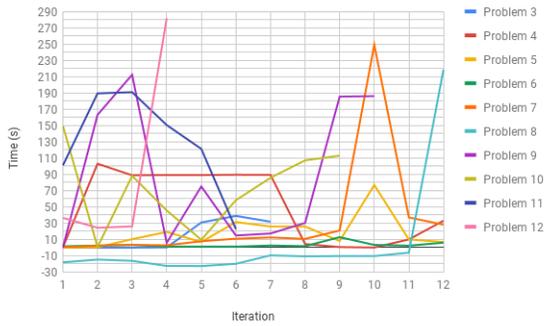
Results The results of this experiment are shown in Table 4 and Figures 28 and 29.



(a) Crew Planning Domain Problems 1 - 10

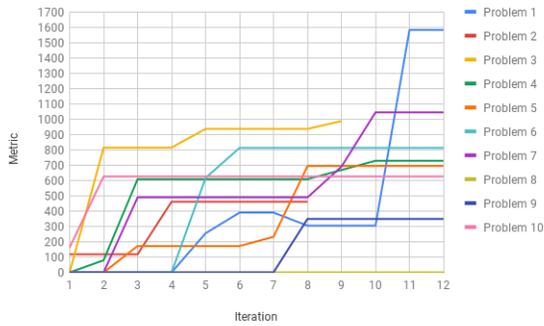


(b) Elevators Domain Problems 1 - 9, 14

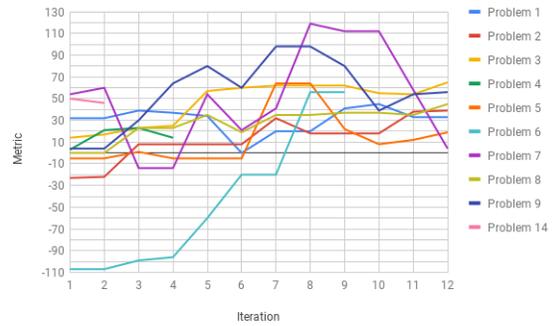


(c) ZenoTravel Domain Problems 3 - 9, 10 - 12

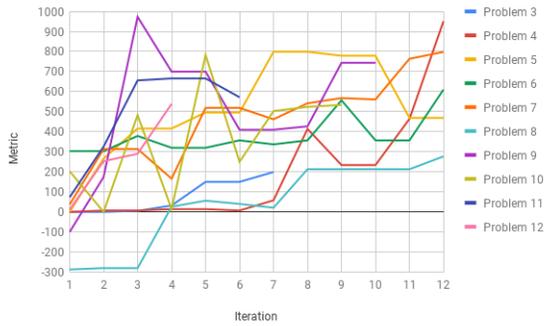
Figure 28: Line chart displaying the change in planning time over multiple iterations of compilations applied to 10 problems for 3 different planning domains. Each sub-caption communicates the domain and range of problems from the IPC. The y-axis represents the change in planning time compared to the original problem.



(a) Crew Planning Domain Problems 1 - 10



(b) Elevators Domain Problems 1 - 9, 14



(c) ZenoTravel Domain Problems 3 - 9, 10 - 12

Figure 29: Line chart displaying the change in planning quality (metric) over multiple iterations of compilations applied to 10 problems for 3 planning domains. Each sub-caption communicates the domain and range of problems from the IPC. The y-axis represents the change in the metric compared to the original problem.

Table 4 shows the number of iterations of compilations successfully applied to a problem. For example, the HModel formed from the constraints derived from 8 questions compiled into problem 2 for the Crew Planning domain was able to be solved within 5 minutes, the 9th question and compilation produced an HModel where no plan could be found within 5 minutes. The majority of the problems were still solvable after the full 12 iterations of compilations were applied, with 20 of the 30 producing plans for the most constrained problems. Of the problems that failed to find plans before the full number of iterations, only three of these were below the average number of questions users asked about plans, as found in our user study.

Figure 28 displays the change in planning time as the iterations of compilations applied to problems increases. This Figure contains three line charts, one for each domain used in the experiment. Each line corresponds to a planning problem, the key is shown on the right of each chart. The horizontal axis displays the iteration of the compilation applied, for each subsequent iteration, the compilation is applied to the previous HModel so each iteration is more constrained than the last. The vertical axis displays the difference in planning time in seconds. For each of the plots the difference is relative to the planning time for the original problem, so the zero on the vertical axis represents there being no difference in planning time between the compiled model (at any iteration) and its corresponding original problem, a positive value means there was an increase in the time taken to find a solution for the compiled model, and the opposite holds for a negative value.

The results displayed in Figure 28 show various trends on the impact iterative compilations have on the planning time. There are problems where the impact is negligible from start to finish, such as in problems 1, 4, 5, 6, 9, and 10 for the Crew Planning domain, and problems 5 and 6 for the ZenoTravel Domain. There are problems where the planning time drastically increases for a single iteration, this can lead to no solution being found within the allocated time for the HModel produced from the next iteration, such as in problem 3 in the Crew Planning domain, and problem 12 for the ZenoTravel domain. However, sometimes the planning time decreases again in subsequent iterations, for example in problem 7 in the ZenoTravel domain. This also happens multiple times in problems 7, 8, and 9 in the Elevators domain where the planning times fluctuate between drastic increases and decreases in planning time from one iteration to the next. In other cases the planning time can increase, stay level for some iterations, and then decrease again such as in problem 2 in the Elevators domain and problem 4 in the ZenoTravel domain. In some cases the compilations can decrease the planning time such as in problems 6 and 8 in the Elevators and ZenoTravel domains respectively which both have improvements in the planning time over several iterations. Although, for most of these plots there tends to be no correlations or a slightly positive correlation between the number of iterations and the planning time. This is quite easy to see in Figures 28a and 28c where the increase in planning times lies below 30 seconds for the final iteration for the majority of problems. The Elevators domain has a more positive correlation between the number of iterations and the planning time than the other domains, where the majority of problems finish with an increase in planning time of between 50 and 100 seconds. Although it does also have an example of a significant improvement in planning time for even the most constrained HModel in problem 7.

Figure 29 displays the change in the quality of the solutions as the iterations of compilations applied to the problem increases. This figure contains three line charts for each

domain in the experiment. The charts are set up the same as those in Figure 28, but the vertical axis displays the difference in the quality of the solutions as measured by the metric defined in the domains.

The results displayed in Figure 29 show that the quality of the solutions vary over the number of compilations applied to a problem. The quality of the solutions tends to get worse over the full number of compilations applied in the Crew Planning and ZenoTravel domains. At a more granular level, the metric increases and decreases from one iteration to the next quite drastically for some problems, for example in problems 9 and 10 in the ZenoTravel domain. However, some others have a more steady increase, such as in problem 7 in the ZenoTravel domain. For all problems, but problem 8, in the Crew Planning domain there is one compilation that causes a drastic increase in the metric, the subsequent compilations then have much less of an impact. Problem 8 in the Crew Planning domain is the only problem where each iterative compilation had no impact on the metric. The compilations have a much lower impact on the metric for the problems in the Elevators domain. Again, there are small fluctuations in the metric from one iteration to the next, but the overall change in the metric for the majority of the problems is minimal. This can be observed by noticing that the majority of the problems end with an insignificant increase in metric of between 4 and 65.

Analysis The results show that the majority of the constrained problems produced from a large number of iterations of compilations are still solvable and within a reasonable time. There were a few instances where a problem was constrained in such a way that it became unsolvable within the 5 minutes of allotted planning time. The results indicate that this is not because of incremental increases in the difficulty over the number of compilations applied to a model, but because of a single compilation that makes the subsequent HModel significantly more difficult to solve. This is likely due to the same reasons that have been discussed in the introduction to Section 4.

The results show that problems do not get significantly harder to solve as the number of constraints applied to the problem increases. This can be shown by running a Wilcoxon Signed-Rank test on the differences between the planning times for consecutive iterations of HModels. For each problem two populations were created from the data, μ_1 and μ_2 where $\mu_1 = \{0, p_1, \dots, p_{n-1}\}$ and $\mu_2 = \{p_1, \dots, p_n\}$ and p_i is the planning time for a model at iteration i and n is the number of iterations. We performed a Wilcoxon Signed-Rank test with the null hypothesis $H_0 : \mu_1 = \mu_2$ and alternate hypothesis $H_1 : \mu_1 \neq \mu_2$. The results were not significant for 26 of the 30 problems with $p = 0.05$. This shows that for these 26 problems the compilations did not significantly worsen the planning time as the iterations progressed. Four of the problems were statistically different and so we could reject H_0 and accept H_1 with $p = 0.05$. More specifically a one tailed test showed that for these four problems we could accept the alternate hypothesis $H_2 : \mu_1 < \mu_2$. Even though for these four problems the compilations worsened the planning time as the iterations progressed, this does not mean that they got exponentially harder.

The results on the impact in the metric, as discussed in the results, mostly show that the greatest increase in the metric comes from a single compilation. This is likely due to the constraint compiled into the model constraining the search space in such a way that the better solution plans are no longer valid, or are much harder to find. This also makes sense

when noticing that subsequent compilations rarely improve the metric once it has been drastically increased. Any constraint compiled into an HModel produces a new HModel that also has the constraints that were applied previously. If one of these constraints limits the space of valid plans by removing a plan with a better quality, then any subsequent constrained HModels will also be limited in this way. This is also supported by, more severely, the failure to find solutions for some sufficiently constrained problems within the time allotted. Unrelated to the number of constraints that have been applied, a problem can go from solvable to unsolvable with a single additional constraint. This leads us to believe that the number of constraints is of less importance than the ultimate constrained problem, whether this be the result of one compilation or of multiple.

6.2 User Study

We designed a comparative user study to evaluate the effectiveness of our XAIP-as-a-Service framework and the iterative model restriction approach that it utilises. We designed the study based on recommendations for metrics on Explanation Satisfaction by Hoffman et al. (2018b). Explanation Satisfaction is a measure of the degree to which users feel that they understand the AI system or process being explained to them and is a contextualised, a posteriori judgment of explanations. The rest of this section describes the design of our user study and discusses the results it produced.

EQ1:	“How well did the XAIP system help you to understand the plan?”
EQ2:	“What do you expect from a good explanation?”
EQ3:	“What would make an explanation satisfactory?”
EQ4:	“In what ways does it help if explanations are contrastive?”

Table 5: Open-ended questions for users who interacted with the framework.

Design A comparative study, also called a true experiment, is a method of data collection designed to test hypotheses under controlled conditions in behavioural research. True experimental designs involve manipulation of the independent variable by exposing participants to different conditions by varying this variable. In the experiment that we

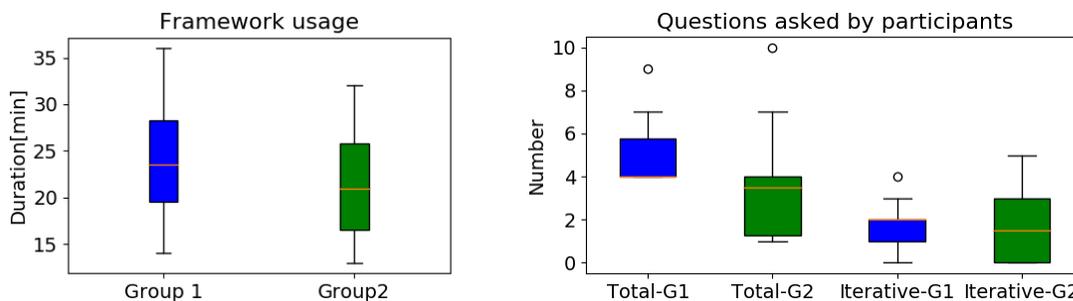


Figure 30: Standard deviation of the time participants spent understanding the plan and using the framework (left). Standard deviations of total number of questions that participants asked and the number of iterative questions that were asked (right).

designed, participants were randomly divided into two groups - the experimental group who interacted with the original framework, and the control group who interacted with a simplified framework.

Participants were using the XAIP as a Service framework in order to understand the plan in Figure 5. They were asked to imagine themselves as employees of the warehouse that we use as our running example described in Section 3.2. We provided the full description of the domain to the participants in both groups. The chosen domain was simple enough so participants could identify themselves in the roles, and complex enough to judge the XAIP system.

20 participants were divided into two groups in order to compare user experience. The experimental group **Group 1** (G1) had the opportunity to use the framework’s full capabilities and get explanations in the form of highlighted comparisons between plans, as discussed in Section 5.1.1 and illustrated in Figure 22b. The control group **Group 2** (G2) used the framework where the *comparison* of plans (with coloured differences and highlighted costs) was disabled. As the explanation, they only got the HPlan next to the original plan. Both groups were asked to use the framework until they were satisfied with their understanding of the system and the plan or until they found interaction with the framework useful. We did not attempt to track how the model of the users evolved whilst using the system, or what triggered their termination with the system, but to determine the overall experience the users had in using the framework.

To evaluate understanding of plans that participants gained using the framework, they rated the system using the Explanation Satisfaction Scale, which is a 5-point Likert scale for Explainable AI systems designed by Hoffman, Klein, and Mueller (2018b). The scale measures an attitude on a continuum from ‘strongly agree’ to ‘strongly disagree’ corresponding linearly to values from 1.0 to 5.0. Additionally, we collected qualitative information about the experiences and expectations Group 1 had whilst partaking in the study by asking them four open-ended questions as defined in Table 5.

Data We conducted a study with 20 volunteers (5 students, 4 engineers, 4 software developers, 3 researchers, 2 assistant professors, a chemist and a copywriter) divided in

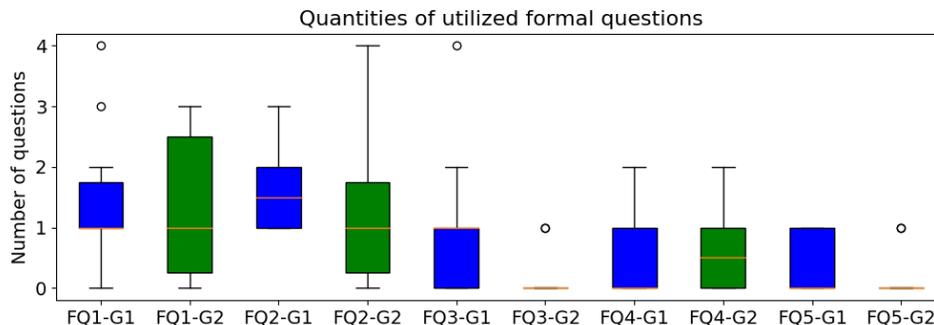


Figure 31: Standard deviation of the number of formal questions (defined in Section 2) utilised by study participants. G1 label and blue colour denote results for Group 1, while green and G2 for Group 2.

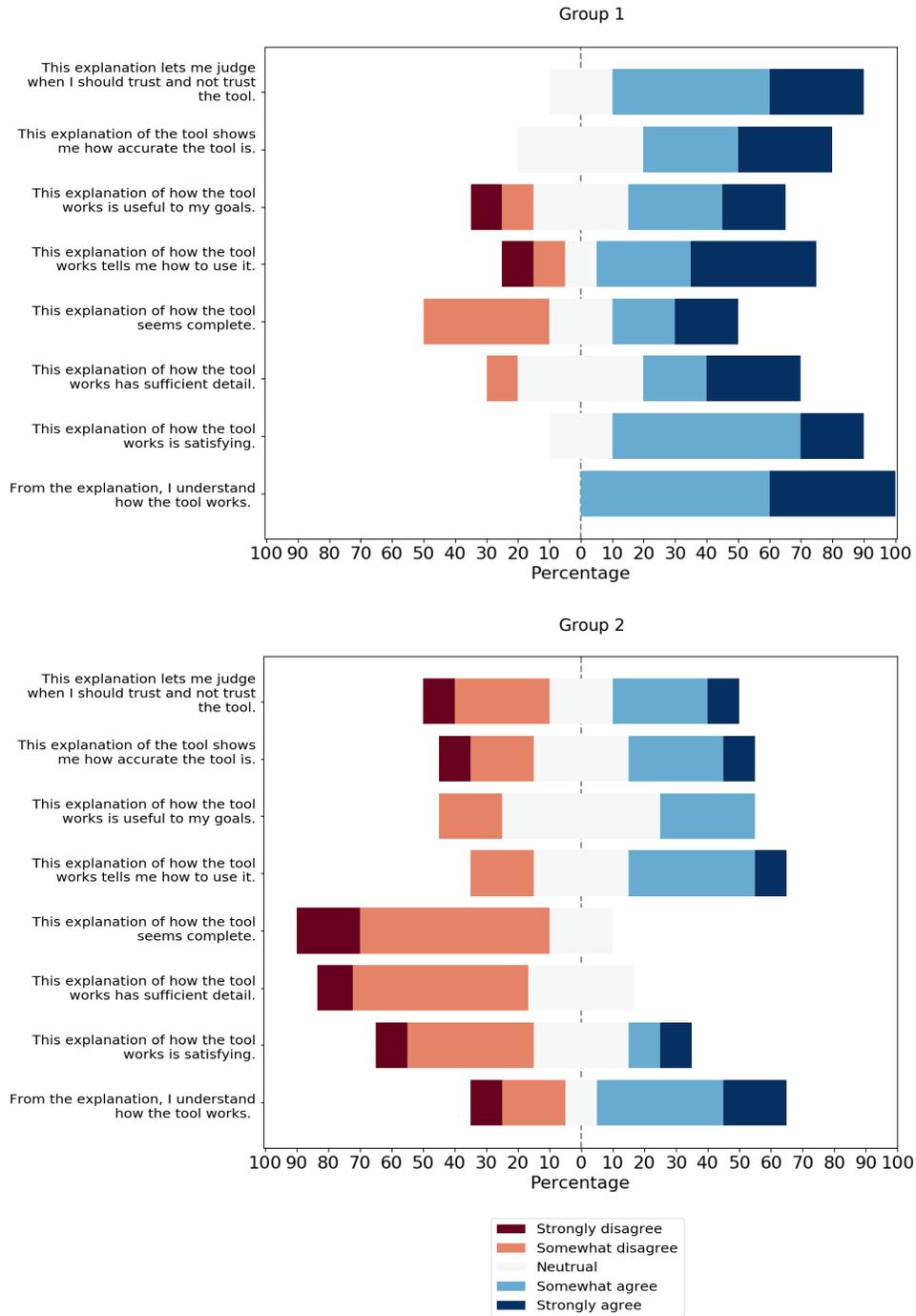


Figure 32: Explanation Satisfaction Scale results collected in the user study for both experimental (G1) and the control group (G2). Likert bar plot visualises distributions of users' answers on their attitudes towards statements about explanations.

two groups with 10 persons each. Participants' ages ranged from 23 to 43 years, with 35% identifying as female and 65% identifying as male. The average time G1 spent with the plan and the framework is 24.2 minutes, and on average, they asked 5.1 questions. The average time G2 spent with the plan and the framework is 21.1 minutes, and on average, they asked 3.7 questions as can be seen in Figure 30. The maximum number of question asked was 10, while the minimum number was 1. The most commonly utilised formal questions were types FQ1 and FQ2 which require removing actions from the plan or adding new actions to the plan. Quantity distributions of each formal question asked are given in Figure 31.

Quantitative Results The results of the evaluation with the Explanation Satisfaction Scale are shown in Figure 32. The median of all responses for Group 1 is 4.0, and the interquartile range (IQR) is 2.0, which corresponds to the overall attitude "somewhat agree" that information gained through usage of the framework is satisfying. Some of the participants did not find the plan comparisons useful to their goals, and some of them somewhat disagreed that the explanations as presented are complete. The median of all responses for Group 2 is 3.0, and IQR is 2.0, which corresponds to the overall attitude "neutral" that explanations obtained through interaction with the framework are satisfying. We also performed t-test with the results of both groups and got the values $t = 5.57$ and $p = 1.038e-7$ telling us that there is a significant difference in the explanation satisfaction of the two groups. Generally, the users found plan comparisons as in Figure 22b useful for understanding how the AI planning system works and that the process of iterative model restriction is satisfying. Also, they agreed that explanations in this way could be useful for improving judgement about whether or not to trust the system.

Qualitative Results 6 out of 10 participants of Group 1 filled out all answers for the questions in Table 5 and, overall, there was an 82.5% response rate to the questions. In the text below, we denote the study participants as P1-P10. By analysing the collected data, we were able to determine several patterns and themes amongst users' responses.

Participants agreed that the framework is helpful and that they had "no problem understanding the plan thanks to the system"(P3). Also, they found the presentation of plan comparisons useful "to understand why ... something (is) possible or not"(P2), to see "how the changes ... affect the original plan"(P3) and to aid in "allowing to reach some conclusions"(P10).

However, for a good explanation, users expect more details "explaining the rules for the plan"(P6) and "showing the logic behind the reason that a specific action has been decided"(P7). Also, they "expect that it (good explanation) would explain each step of the plan in an understandable way, to explain every change that has been made and how does that change affect the plan"(P4) and they "expect an argument for the usage of one thing over anything else"(P9). This theme also complements their attitude towards explanation completeness as revealed in the Explanation Satisfaction Scale results (Figure 32).

Additionally, participants reflected on the explanation presentation. Even though they think "it (is) great to compare plans and see what makes the difference"(P5), they also expect more user-friendly presentation of final explanation such as "in the format of a sentence"(P8) or "a visual representation of what the robots are doing"(P9).

7. Related Work

As discussed earlier, several researchers (e.g: Mueller et al. (2019)) have drawn a distinction between local and global questions and the corresponding explanations. The study in Section 2 showed that local questions are significant for XAIP, and that many of these questions are contrastive in nature. In this section we present an overview of additional related research on explanation with a focus on local questions and explanations. We start with a brief description of relevant background ideas from Philosophy, Psychology and Cognitive Science, which have contributed to current views on explanation. We then focus in on more recent related work on local explanation from XAI, and finally on work in XAIP, which is most closely related to the work described here.

7.1 Philosophy, Psychology and Cognitive Science

There is a vast body of literature in the fields of philosophy, psychology, and cognitive science on the topic of explanation. Much of the early work in this area focused on *causal explanation*, namely the idea that questions could be answered by identifying and elucidating the causes for a particular event or result being questioned. Hume (1748) noted that if there is a cause between two events, the first is always followed by the second. Lewis (1974) expanded on this, arguing that we should understand Hume's definition wherein if an event *C* causes the event *E* and, if under some hypothetical case, *C* did not happen then neither would *E*. Lewis (1986) then argued that to explain an event one must provide some information about its causal history. Although, this may be enough to explain why an event happened, it does not answer all questions about an event. One could not use the causal history of an event alone to answer if it was the best outcome, what would happen if the event did not occur, or if another event occurred in its place. As a consequence, more recent work on explanation has introduced and considered notions of contrastive questions and explanations. In particular, philosophers, such as van Fraassen (1980), have argued that "why"-questions can be implicitly or explicitly understood as: "why is *A* better than some alternatives?". This is what we describe as a local contrastive question, where the questioner wants to understand why a plan is good, or why a particular decision was made. These questions can be answered with contrastive explanations.

Van Bouwel and Weber (2002) defined four types of explanatory questions, three of the four are explicit local contrastive why questions that call for explaining the differences between either real or hypothetical alternatives. They argue that the final explanatory question is not contrastive, and is asked when the user wants a global understanding of the properties of objects. However, most other researchers think all "why" questions ask for contrastive explanations, whether the contrast case is implicitly or explicitly stated (Hilton, 1990; Lipton, 1990; Lombrozo, 2012). Hilton (1990) recognised that one does not explain an event, but instead explains why the event occurred in one case but not in a counterfactual (hypothetical) contrast case. This is the basis of contrastive explanations. Lipton (1990) argues that the cognitive burden of complete explanations is too great. He goes further by demonstrating that explaining a contrastive question is less demanding because it is enough to show what is different between the two cases instead of a full causal analysis. Miller (2018) extended structural causal models (Halpern & Pearl, 2005a, 2005b) to contrastive explanations based on Lipton's Difference Condition. Lipton's difference condition

states that to explain why A rather than B, one must cite a causal difference between A and not-B, consisting of a cause of A and the absence of a corresponding event in the history of not-B (Lipton, 1990). Hilton (1990) argues that explanation is a conversation. We have adopted this strategy by treating explanation as an iterative process where the human continues to ask contrastive questions as a means of understanding plans, and imposing additional constraints on the planning problem.

7.2 Explainable AI

Meuller et al. (2019) provide an overview of the landscape of research into Explainable AI (XAI). This work spans several decades, and includes work carried out with intelligent tutoring systems, XAI hypotheses and models, and explanation in expert systems. The early work on explanation in expert systems provided causal explanation for conclusions, often in the form of chains of rules contributing to the conclusion (Van Melle, 1978). Another earlier work on XAI, which also exploits compilation to planning models, is the exploration by Sohrabi et al. (2011) of planning explanations of behaviours in dynamical systems.

Recently, there has been a resurgence of interest in explanation in XAI, both when the model is and is not interpretable. This is, in large part, due to the difficulty of understanding the results of deep learning systems (Rudin, 2019). Madumal et al. (2019) use structural causal models to answer “*why?*” and “*why not?*” questions in Reinforcement Learning systems. They learn approximate causal models for counterfactuals to explain the local contrastive question “*why not action B (rather than A)?*”, by simulating what would happen if B was performed, and providing a contrastive explanation showing the difference between the causal chain of A and B. They provide minimally complete explanations to answer questions of the form “*why action A?*”. However, the explanation is more of a justification that the action A contributes to the goal in some way based on the global model, rather than explaining why the decision was made to perform the action as opposed to some other action or not at all. We argue that users need explanations to a wider set of questions to understand the reasoning behind a particular decision.

As a user may not have any knowledge of the inner workings of a black box system, researchers have focused on providing global explanations for how a system came to a particular outcome (Krishnan, Sivakumar, & Bhattacharya, 1999; Johansson & Niklasson, 2009; Augasta & Kathirvalavakumar, 2012). However, researchers have recently tried to tackle the problem of providing local explanations for why a black box system arrived at an outcome. For example an image classifier might recognise an image as a bird; Ribeiro et al. (2016) provide a way to highlight features of the image to help justify its decision, for example highlighting the birds beak and wings. However, Hoffman et al. (2018a) have argued that neither of these approaches is enough and that users must be able to ask contrastive why questions through a process they call explanation as exploration to best understand black box models. Our approach of explanation as an iterative process is similar to what they propose.

Hoffman et al. (2018b) give a thorough overview on evaluating explanations. They focus mainly on techniques that can be used to measure XAI constructs. These include measuring explanation goodness and satisfaction, mental models, curiosity, and trust. We

adopted the proposed metrics for evaluating explanation goodness and satisfaction in our experimental evaluation in Section 6.2.

There have also been recent efforts within XAI on *human-centered* explainability (Miller, Howe, & Sonenberg, 2017; Miller, 2019; Lim, Dey, & Avrahami, 2009). Until recently, there has been only limited investigation to determine what users actually want explained. Haynes, Cohen, and Ritter (2009) proposed a taxonomy, based on that of Graesser et al. (1992) and Lehnert (1978), that they empirically tested using a pilot simulation tool. In this case, they found that *definition* questions were the most common, as the users were tasked with operating an unfamiliar tool. Their end-goal was to understand how to use the tool rather than a plan.

Lim, Dey, and Avrahami (2009) looked at the following questions, “why”, “why-not”, “how”, and “what if”. They found that “why” and “why-not” questions led to the best improvement in understanding amongst users. Penney et al. (2018) used this taxonomy to study how users foraged for information in the game StarCraft. For assessing an agent, they found that “what” questions were asked most frequently (over 70%), followed by “why” questions. They reasoned that, in their domain, the most common questions related to uncovering hidden information about the current and past states.

In contrast, the taxonomy presented in Section 2 shows that, when presented with a plan, user questions are more commonly contrastive “why” questions that relate to actions.

7.3 Explainable AI Planning

While there is a long history of work on explanation in AI, most work on explanation of plans (XAIP) is relatively recent. Some of the earliest work in this area is that of Gobelbecker et al. (2010), which focused on providing explanations when a plan could not be found. This work is discussed in more detail in Section 7.3.3. Sohrabi et al. (2011) treat explanation as a planning problem, but the focus of the explanations is not on plans themselves. In a challenge paper, Smith (2012) argues for the importance of plan explanation in mission planning, and suggests that questioning and explanation is part of an iterative process that helps elucidate and refine the preferences for a planning problem. Fox et al. (2017) highlight contrastive ‘why’ questions as being important for plan explanation, and describe a number of different types of these questions and possible responses. Chakraborti et al. (2020) survey recent work in XAIP and categorise the different approaches that have emerged in the last several years. Below, we focus on related XAIP work in three specific areas: model reconciliation, contrastive explanations, and explanation of unsolvability for planning problems.

7.3.1 MODEL RECONCILIATION

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. The planning system therefore “suggests changes to the human’s model, so as to make its plan be optimal with respect to that changed human model”. In that work, Chakraborti et al. describe an approach for generating minimally complete and monotonic explanations that update the users model, so that it will accept a plan as correct. The approach assumes that the user model is known. In contrast,

Sreedharan (2018) generates conformant explanations that are applicable to a set of possible user models in cases where the user’s model is not precisely known. Both approaches consider only *optimal* solutions for classical planning problems. In general, the assumption that plans must be optimal in order to support explanation is troublesome. Optimal planning for temporal and numeric problems (which we consider here) is undecidable. In addition, the metric or preferences by which plans should be assessed might differ between the human and agent, as Smith (2012) suggests. Of course, the preferences and optimisation criteria could be considered as part of the model, in which case the *model reconciliation* perspective would still be appropriate. However, to date, these aspects of the model have not received much attention.

As in the model reconciliation work, we suppose that the user and agent may have different models. We also suppose that they may have different preferences, and different computational abilities. However, we don’t assume that either the agent or the human have direct knowledge about the other’s model or capabilities, or that they do optimal planning. Instead, we have considered the problem of plan explanation to be one of *model restriction* where the human can impose additional constraints on the planning problem (through contrastive questions). As we noted in Section 3.3, model restriction could be considered as a special case of model reconciliation, in which the planning agent’s model is temporarily revised by imposing the constraints implied by a contrastive question. However, these constraints do not result in permanent changes to the agent’s model of the world, or model of the planning domain – they are temporary restrictions on the plan trajectory. It’s also true that the human’s interaction and questioning might result in evolution of their own model of the planning domain or problem. However, we do not assume that the agent has any knowledge of the human’s model, and we do not attempt to model the human’s learning process.

A complementary line of work is that of generating plans that are more *explicable* within the framework of a human user’s model. Zhang et al. (2017) proposed generating explicable plans by postulating that humans understand plans by associating the actions in the plan with abstract tasks. They learn what abstract tasks humans associate to actions and use this to produce new more explicable plans. Kulkarni et al. (2016) model explicability by first computing the distances between the plans generated by a planning agent and plans expected by the user. Human subjects are then asked to label the actions in the agent plans as explicable. These results are used along with the plan distances to form a regression model called explicability distance. Explicability distance is then used as the heuristic to search for explicable plans. It is not always possible to produce explicable plans if the agent model does not allow for what the user expected and explanations are needed.

Work on generating explicable plans can be seen as complementary to work on explanation; explicable plans should reduce the need for explanation, but they do not eliminate it. To date, this work has assumed different models for the planner and the human. As a result, an explicable plan is one which is reasonable in the human’s model, so that a planner can use that model to generate explicable plans. This work has not seriously addressed situations in which the models are the same, but the plan is simply too complicated for the user to understand easily or quickly. In this case, a more explicable plan would presumably be one which is smaller and simpler for the human to reason about. This work has also not seriously addressed situations where the human and agent have the same models of

actions and of the world state, but different preferences or optimisation criteria. In this case, an explicable plan would be one which is good according to the user's preferences and optimisation criteria.

7.3.2 CONTRASTIVE EXPLANATIONS OF PLANS

In addition to the work previously cited (Fox et al., 2017), highlighting the role of contrastive questions in XAIP, Eifler et al. (2020) approach answering local contrastive questions by explaining the reason that a contrast case B was not in the plan, or a feature of the plan, by using the properties that would hold if B were the case. This is in contrast to our approach in which we give a specific plan trace containing B . Kim et al. (2019) detail a general approach for generating differences between plan traces using Bayesian inference with search for inferring contrastive explanations as linear temporal logic specifications. These resulting differences can then be used to generate contrastive explanations. However, they do not consider how to generate the different plan traces, or how they can be used to answer specific questions. Kasenberg et al. (2019) focus on justifying an agent's behaviour based on deterministic Markov decision problems. They construct explanations for the behaviour of an agent governed by temporal logic rules and answer questions including contrastive "why" queries. However, they only recognise implicit contrastive questions of the form "why A ?" (or "why $\neg A$?") which they explain by citing rules and goals dictating that they had to (or could not) perform A . Whereas we argue that users must be able to ask explicit contrastive questions of the form "why A rather than B ?" which can usually only be answered by generating a hypothetical plan where B is included and A is absent. Bercher et al. (2014) also provide explanations for implicit contrastive *why* questions in a system that helps users to assemble a home theatre. The explanations consist of the set of causal reasons that an action is present in the plan. Chakraborti et al. (2019) show how these kinds of explanations can be minimised by selecting the most relevant explanatory content. Like the previous work by Kasenberg et al., these approaches do not utilise more comprehensive techniques for counterfactual reasoning in order to construct alternative plans, and compare them with the original.

7.3.3 UNSOLVABILITY

A special kind of "why" question is: "why didn't you find a solution to this problem?" While there has been recent work on the generation of unsolvability certificates for planning problems (Eriksson, Röger, & Helmert, 2017; Eriksson & Helmert, 2020), these are not very satisfying as explanations. Gobelbecker et al. (2010) argue that excuses must be made for why a plan cannot be found. These are counterfactual alterations to the planning task such that the new planning task will be solvable. They 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. (2019) address the problem of explaining unsolvability 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. The unsatisfiability of these landmarks provides a more succinct description of critical propositions that cannot be satisfied. Eifler et al. (2020) has taken a somewhat different approach by deriving properties that must be

obeyed by all possible plans. Although this is not the focus of this work, these too could serve as explanations in cases of unsolvability.

We have not attempted to provide explanations for unsolvability of planning problems in this paper. However, since we allow for the user's and agent's models and computational abilities to differ, it is certainly possible that the model restriction imposed by a contrastive question may result in a planning problem that is unsolvable by the planning system. Addressing this issue is left to future work.

7.4 Plan Repair

One way to interpret the process of constructing a plan for a newly constrained planning problem is as a plan repair task, starting from the plan generated for the original problem and attempting to modify it to achieve the further constraints. Work on plan repair (van der Krogt & de Weerd, 2005b, 2005a; Fox, Gerevini, Long, & Serina, 2006) has generally focused on a desire to retain features of the original plan as much as possible, to benefit from the consistency of the plan structure and the strategic intentions. This is not a primary concern for us and often the explanation process benefits from highlighting a contrasting plan, rather than from compromising plan quality in order to preserve structure from the original plan. Nevertheless, approaches to plan repair might offer alternatives to compilation in the search for a suitable HPlan to match an HModel.

Plan adaptation is a form of plan repair, explored by researchers as a means to modify plans to satisfy new goals, starting with plans that solve similar goals to those being newly faced. The approaches adopted in this area (Nguyen et al., 2012; Gerevini & Serina, 2010; Hanks & Weld, 1995) might be adapted to the generation of HPlans. While identifying suitable target plans and using local search to adapt them is the approach adopted by Nguyen et al. (2012) and Gerevini and Serina (2010), Hanks and Weld (1995) concentrate on the structural properties of the original plan, modifying causal links in a systematic search.

8. Conclusion

In this paper we considered the problem of plan explanation to be an iterative process, in which the user repeatedly asks questions that are contrastive in nature. To motivate our focus on contrastive questions, in Section 2 we presented a user study examining the kinds of questions users asked about problems in three small planning domains. This study demonstrated that the vast majority of user questions were indeed contrastive. We categorised these questions into a taxonomy of 7 different types, which served as the focus for the remainder of the paper.

Each contrastive question, such as "*why did you do A rather than B?*" leads to a "constrained" or *restricted* planning problem – in this case, one in which "B" must be in the plan instead of "A". This restricted planning problem must then be solved by the planning system in order to compare and contrast the user's proposed alternative solution (foil) with the original plan generated by the planning system.

Through this iterative process, the user is able to explore the space of possible solutions. This may result in the user increasing their understanding of the problem and the solutions being proposed by the planning system, but may also lead to improvement of those

solutions, as a result of the constraints imposed by the user’s questions. Ultimately, it is up to the user to decide when they are satisfied with the resulting plan. This process increases transparency for the user, and ultimately leads to greater trust and understanding of the planning problem and the possible solutions.

8.1 Contributions

There are several contributions in this paper:

- In Section 2 we hypothesised that users are more likely to ask contrastive why questions about plans. We supported this hypothesis with a user study, and formalised the results into a contrastive taxonomy of questions.
- In Section 3 we formalised the iterative process as being one of *model restriction*, where each contrastive question leads to a hypothetical problem characterised by restrictions imposed on the model of the planning problem.
- In Section 4 we showed how these restrictions can be compiled into a PDDL2.1 model for the contrastive questions in our taxonomy.
- In Section 5 we presented the implementation of this framework as a service – namely as a wrapper around an existing temporal/numeric planning system, with a simple user interface for comparing plans.
- In Section 6 we evaluated the computational consequences of model restriction, and solution quality of the plans generated from those restricted models, and efficacy of the explanation framework as a whole.

In Section 6.1 we evaluated the impact of adding constraints on the efficiency of the plan generation process for the constraints imposed by the different types of contrastive questions. The results show that for the majority of problems and questions, the planning time for restricted planning problems is quite similar to that for the original unconstrained problem. As we noted, adding constraints to a planning problem reduces the number of possible solutions and could make it more difficult to find a plan solution. However, the constraints can also rule out significant portions of the plan search space, making it easier for the planner to find a solution. Predicting the performance impact for a particular question and problem is therefore not easy or obvious, but in general, performance does not appear to be a significant issue. The results in Section 6.1.2 shows that the number of constraints is of less importance on the planning time than the ultimate constrained problem, whether this be the result of one compilation or of multiple. This demonstrates that the difficulty in answering questions does not necessarily increase with the number of questions asked, and therefore supports our iterative model restriction approach.

We also evaluated the impact of constraints on plan quality. For easier problems the compilations had little impact on the quality of plans produced by the planner, but for harder problems, the addition of constraints sometimes resulted in significant improvement in plan quality. This is an important feature of the iterative plan exploration process – a user can impose restrictions that lead to a better plan. This demonstrates that the original

plan was not optimal and that the addition of constraints can actually narrow the search space in a way that guides the planner toward better quality solutions.

Through a comparative user study, described in Section 6.2, we evaluated the effectiveness of the framework in improving the user’s understanding of the proposed plan. The results indicate that the iterative framework and plan comparison helps users to understand plans better. In particular, it helps them to understand why actions are in the plan, why actions are in a particular order, and how changes affect the plan. However, the study also pointed out some of the weaknesses in the current system, namely in the quality of the explanations. User’s expected a more in-depth explanation of the differences between plans, rather than just a simple highlighting of the differences. We discuss this further in the next section.

8.2 Future Work

All of the below issues present interesting technical challenges that warrant further investigation.

COMPILING CONSTRAINTS

Current PDDL languages do not have the ability to express constraints on action inclusion, exclusion, or ordering, and do not allow us to place more complex constraints on *how* something is achieved or on plan structure. For example, the question “Why did you use action A rather than action B for achieving P?” requires planning with the HModel where B is required to be in the causal support for achieving P, but A is not in that causal support. This is substantially more difficult than just excluding A from the plan and forcing B into the plan. This has been touched upon in the paper with a first step towards a solution in Section 4.2.1 and a further discussion of the nuance of this issue in Section 4.10. However, we believe for a robust solution, LTL will likely play a key role in defining the semantics of any new language which enables the expression of such constraints.

As we discussed in Section 4.9, there are possible questions that cannot obviously be represented in the vocabulary of the planning model. For example, the user might ask “why did it take so long to accomplish A?” Such a constraint requires a richer language that allows one to impose trajectory constraints on the plan itself, e.g. requiring that A be achieved before some time limit. Such trajectory constraints might be expressible in PDDL 3.0, but this issue requires further investigation. These types of questions might be unexplicitly expressible through a series of questions which give the same effect, however this currently requires a user to infer which questions should be asked and in which order to achieve this. In the future we would like to automatically infer these more robust questions from natural language. We believe that both issues highlighted in this section are needed for automatic inference of complex questions, and that a contrastive language that is both representable, expressible, and actionable will be a crucial part of any solution.

EXPLANATION

Currently, our explanations for contrastive questions consist of comparing two plans side-by-side, highlighting the action differences between them. However, there is clearly room

to do much more. We do not take advantage of the causal structure of the plans within our explanations.

As we indicated above, in the user study, the users wanted deeper explanations of the differences between plans. Considering the causal structure of the two plans and how they differ appears to be part of the solution. For example, a plausible explanation might be “because you asked me not to use action A, I had to use another action to achieve P, and action B appeared to be the best choice.”

However, this approach is likely only part of the solution – abstraction may also play a key role. For example, abstracting away some of the predicates, variables, and actions that are the same in both plans would allow the explanation to focus on the key differences between the two plans. This could also help to provide better explanations for understanding very long plans. We believe that very long plans cannot be usefully explained as single structures - there is need to abstract them to arrive at views that are digestible. These abstractions can then be the subject of questions and explanations in the form we discuss. This is related to the problem of explaining unsolvability discussed below.

UNSOLVABILITY

By adding constraints to a planning problem, it's possible that the problem will become unsolvable. Sreedharan et al. (2019) address the problem of explaining unsolvability 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. The unsatisfiability of these landmarks provides a more succinct description of critical propositions that cannot be satisfied. This approach is appealing, but for temporal/numeric planning problems, it becomes more challenging. This is because the number and character of possible relaxations increases dramatically. For example, in addition to abstracting away certain predicates, one could consider abstracting away some of the numeric variables, or relaxing certain action preconditions, or allowing actions to have arbitrary durations. A more targeted approach would be to consider relaxing the constraints imposed by the user's question until the problem becomes solvable again. The removed constraints could then be analysed to determine how they prevent milestones from being achieved. A satisfying explanation for the question “Why did you not do B?”, might be something like “Action B takes too long and makes it impossible for me to achieve (milestone) M”. The advantage of this more targeted approach is that it reduces the number of abstractions that need to be considered – only the actions involved in the user imposed constraints, and the variables and predicates they reference, need to be considered for abstraction.

MODULARITY OF FRAMEWORK

The design of the framework's architecture allows the components to be independently adapted to better fit different XAIP scenarios. In the future we would like to evaluate the ability of our framework to adapt to different use cases. For example, the XAIP-Human Interface could be adapted to explanations for a robotic agent or an augmented reality setting (Chakraborti et al., 2018). The explanation visualisation could be adapted to provide different information to the user, for example to illustrate discrepancies in the model

highlighting what prevents the planner from solving the restricted model (Sreedharan et al., 2019). The Compilations module could be replaced or extended to add new compilations that accommodate niche questions that are specific to the scenario.

An actualised example of a modification to the explanation framework is demonstrated by comparing the ethics of plans (Krarup, Krivic, Lindner, & Long, 2020). In this work Krarup et al. modified the architecture of the framework with an additional module called the Ethical Explanation Generator. With extra information about the ethical attributes of the model (the extrinsic value of actions, utilities of facts, etc.) and the moral principle, the Ethical Explanation Generator shows whether a plan is permissible under the principle and provides a causal explanation. The resulting extended framework allows users to compare the ethics of plans within the iterative process.

The instantiation of the XAIP-human interface module we used was designed with the purpose of evaluating our explanations and was constructed to be as simple as possible such that unfamiliar users could interact with the system without training. However the design of the user interface will likely have an affect on the effectiveness of the framework and the explanations it presents. Evaluating the framework with multiple user interfaces employing different design principles would not only help improve the framework, but may also give insight into the best practices when creating an XAI system (Amershi et al., 2019). The user interface could also be adapted based on expertise of the user. Less experienced users may prefer more abstract higher level explanations whereas planning or domain experts may want the most detailed explanations available. Again user studies would be needed to evaluate this claim.

UTILISATION OF MENTAL MODELS

As we have stated an advantage to our approach is that we do not attempt to learn the user's mental model. Instead the user uses the system in an iterative, conversational process to further constrain the planning model until they reach their desired HPlan and receive their desired explanation. However, the process might be made more efficient by taking in to account the user's mental model. There has been work related to mental modelling techniques (Tabrez, Luebbers, & Hayes, 2020), using this in conjunction with our system may help to reduce the number of question-answer iterations needed by exploiting the known differences between the users model of the system and the true model.

Ignoring the user's mental model also has the disadvantage that we do not know how the explanations affect the user's global understanding of the planning model. This could cause the user to be happy to accept an explanation because it is misleading or they have misunderstood it. This can lead to the user unknowingly having an incorrect understanding of the planning model. This could have some unseen consequences, including ethical concerns. Being able to evaluate how the users mental model evolves over the lifespan of their use with the system could give insight into the potential impacts of this problem.

Acknowledgements

This work was partially supported by EPSRC grant EP/R033722/1 for the project Trust in Human-Machine Partnership (THuMP) and AirForce Office of Scientific Research award number FA9550-18-1-0245.

References

- Amershi, S., Weld, D., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., Suh, J., Iqbal, S., Bennett, P. N., Inkpen, K., et al. (2019). Guidelines for human-ai interaction. In *Proceedings of the 2019 chi conference on human factors in computing systems*, pp. 1–13.
- Augasta, M. G., & Kathirvalavakumar, T. (2012). Reverse engineering the neural networks for rule extraction in classification problems. *Neural processing letters*, 35(2), 131–150.
- Benton, J., Coles, A., & Coles, A. (2012). Temporal planning with preferences and time-dependent continuous costs. In *International Conference on Automated Planning and Scheduling*.
- Bercher, P., Biundo, S., Geier, T., Hoernle, T., Nothdurft, F., Richter, F., & Schattenberg, B. (2014). Plan, Repair, Execute, Explain-How Planning Helps to Assemble your Home Theater. In *Proc. International Conf. on Automated Planning and Scheduling (ICAPS)*.
- Borgo, R., Cashmore, M., & Magazzeni, D. (2018). Towards providing explanations for AI planner decisions. In *Proc. International Joint Conf. on AI-18 Workshop on Explainable AI*.
- Chakraborti, T., Sreedharan, S., Zhang, Y., & Kambhampati, S. (2017). Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *Proc. International Joint Conf. on AI*.
- Chakraborti, T., Fadnis, K. P., Talamadupula, K., Dholakia, M., Srivastava, B., Kephart, J. O., & Bellamy, R. K. (2019). Planning and visualization for a smart meeting room assistant. *AI Communications*, 32(1), 91–99.
- Chakraborti, T., Sreedharan, S., Grover, S., & Kambhampati, S. (2018). Plan Explanations as Model Reconciliation—An Empirical Study. In *arXiv preprint arXiv:1802.01013*.
- Chakraborti, T., Sreedharan, S., & Kambhampati, S. (2020). The Emerging Landscape of Explainable AI Planning and Decision Making. In *Proc. International Joint Conf. on AI*, pp. 4803–4811.
- Chakraborti, T., Sreedharan, S., Kulkarni, A., & Kambhampati, S. (2018). Projection-aware task planning and execution for human-in-the-loop operation of robots in a mixed-reality workspace. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4476–4482. IEEE.
- Chen, G., Ding, Y., Edwards, H., Chau, C. H., Hou, S., Johnson, G., Syed, M. S., Tang, H., Wu, Y., Yan, Y., Tidhar, G., & Lipovetzky, N. (2019). Planimation. In *Demonstration Program of 29th Conference on Automated Planning and Scheduling*.
- Coles, A., Coles, A., Fox, M., & Long, D. (2010). Forward-Chaining Partial-Order Planning. In *Proc. International Conf. on Automated Planning and Scheduling*.
- Edelkamp, S. (2006). On the compilation of plan constraints and preferences. In *ICAPS*, pp. 374–377.
- Eifler, R., Cashmore, M., Hoffmann, J., Magazzeni, D., & Steinmetz, M. (2020). A new approach to plan-space explanation: Analyzing plan-property dependencies in over-subscription planning. In *Proc. Association for Advancement of AI*, pp. 9818–9826.

- Eriksson, S., & Helmert, M. (2020). Certified unsolvability for sat planning with property directed reachability. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 30, pp. 90–100.
- Eriksson, S., Röger, G., & Helmert, M. (2017). Unsolvability certificates for classical planning. In *AAAI Press*.
- Faulkner, L. (2003). Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers*, 35(3), 379–383.
- Fink, E., & Yang, Q. (1992). Formalizing plan justifications. In *Proc. Canadian Artificial Intelligence Conference*, pp. 9–14.
- Fox, M., Gerevini, A., Long, D., & Serina, I. (2006). Plan Stability: Replanning versus Plan Repair. In Long, D., Smith, S. F., Borrajo, D., & McCluskey, L. (Eds.), *Proc. International Conf. on Automated Planning and Scheduling, (ICAPS)*, pp. 212–221. AAAI.
- Fox, M., & Long, D. (2003). PDDL2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20, 61–124.
- Fox, M., Long, D., & Magazzeni, D. (2017). Explainable planning. *Proc. International Joint Conf. on AI-17 workshop on Explainable AI*, abs/1709.10256.
- Garfinkel, A. (1982). *Forms of explanation: rethinking the questions in social theory*. Yale University.
- Gerevini, A., & Serina, I. (2010). Efficient plan adaptation through replanning windows and heuristic goals. *Fundam. Informaticae*, 102(3-4), 287–323.
- Göbelbecker, M., Keller, T., Eyerich, P., Brenner, M., & Nebel, B. (2010). Coming up with good excuses: What to do when no plan can be found. In *Twentieth International Conference on Automated Planning and Scheduling*.
- Göbelbecker, M., Keller, T., Eyerich, P., Brenner, M., & 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.
- Graesser, A. C., Person, N., & Huber, J. (1992). Mechanisms that generate questions. *Questions and information systems*, 1, 167–187.
- Halpern, J. Y., & Pearl, J. (2005a). Causes and explanations: A structural-model approach. part i: Causes. *The British journal for the philosophy of science*, 56(4), 843–887.
- Halpern, J. Y., & Pearl, J. (2005b). Causes and explanations: A structural-model approach. part ii: Explanations. *The British journal for the philosophy of science*, 56(4), 889–911.
- Hanks, S., & Weld, D. S. (1995). A domain-independent algorithm for plan adaptation. *J. Artif. Intell. Res.*, 2, 319–360.
- Haynes, S. R., Cohen, M. A., & Ritter, F. E. (2009). Designs for explaining intelligent agents. *International Journal of Human-Computer Studies*, 67(1), 90 – 110.
- 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*, p. 44–53. AAAI Press.

- Hilton, D. J. (1990). Conversational processes and causal explanation.. *Psychological Bulletin*, 107(1), 65.
- Hoffman, R., Miller, T., Mueller, S. T., Klein, G., & Clancey, W. J. (2018a). Explaining explanation, part 4: a deep dive on deep nets. *IEEE Intelligent Systems*, 33(3), 87–95.
- Hoffman, R. R., Mueller, S. T., Klein, G., & Litman, J. (2018b). Metrics for Explainable AI: Challenges and Prospects. *CoRR*, [abs/1812.04608](https://arxiv.org/abs/1812.04608).
- Hoffmann, J. (2003). The Metric-FF Planning System: Translating “Ignoring Delete Lists” to Numeric State Variables. *Journal of Artificial Intelligence Research*, 20, 291–341.
- Hogg, T., Huberman, B. A., & Williams, C. P. (1996). Phase transitions and the search problem. *Artificial Intelligence*, 81(1), 1–15. *Frontiers in Problem Solving: Phase Transitions and Complexity*.
- Howey, R., Long, D., & Fox, M. (2004). Val: automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pp. 294–301.
- Hume, D. (1748). *An enquiry concerning human understanding*. Harvard Classics Volume 37, 1910 P.F. Collier and Son.
- Johansson, U., & Niklasson, L. (2009). Evolving decision trees using oracle guides. In *2009 IEEE Symposium on Computational Intelligence and Data Mining*, pp. 238–244. IEEE.
- Kasenberg, D., Roque, A., Thielstrom, R., Chita-Tegmark, M., & Scheutz, M. (2019). Generating justifications for norm-related agent decisions. In *arXiv preprint arXiv:1911.00226*.
- Kim, J., Muise, C., Shah, A., Agarwal, S., & Shah, J. (2019). Bayesian inference of linear temporal logic specifications for contrastive explanations.. In *Proc. International Joint Conf. on AI*, pp. 5591–5598.
- Krarpup, B., Krivic, S., Lindner, F., & Long, D. (2020). Towards Contrastive Explanations for Comparing the Ethics of Plans. In *ICRA-20 Workshop on Against Robot Dystopias*.
- Krishnan, R., Sivakumar, G., & Bhattacharya, P. (1999). Extracting decision trees from trained neural networks. *Pattern recognition*, 32(12).
- Kulkarni, A., Zha, Y., Chakraborti, T., Vadlamudi, S. G., Zhang, Y., & Kambhampati, S. (2016). Explicability as minimizing distance from expected behavior. In *arXiv preprint arXiv:1611.05497*.
- Kulkarni, A., Zha, Y., Chakraborti, T., Vadlamudi, S. G., Zhang, Y., & Kambhampati, S. (2019). Explicable planning as minimizing distance from expected behavior. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2075–2077. International Foundation for Autonomous Agents and Multiagent Systems.
- Lehnert, W. G. (1978). *The process of question answering: A computer simulation of cognition*. Lawrence Erlbaum Associates.
- Lewis, D. (1974). Causation. *The journal of philosophy*, 70(17), 556–567.
- Lewis, D. (1986). Causal explanation. In Lewis, D. (Ed.), *Philosophical Papers Vol. Ii*, pp. 214–240. Oxford University Press.

- Lim, B. Y., Dey, A. K., & Avrahami, D. (2009). Why and why not explanations improve the intelligibility of context-aware intelligent systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '09*, pp. 2119–2128.
- Lipton, P. (1990). Contrastive explanation. *Royal Institute of Philosophy Supplement*, 27, 247–266.
- Lipton, Z. C. (2016). The mythos of model interpretability. *CoRR*, *abs/1606.03490*.
- Lombrozo, T. (2012). Explanation and abductive inference. In *Oxford handbook of thinking and reasoning*, pp. 260–276.
- Long, D. (2018). Planning a Way Into a Deep Hole. In *Invited talk at ICAPS Workshop on Planning and Scheduling Applications (SPARK)*.
- Long, D., & Fox, M. (2003). The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20, 1–59.
- Madumal, P., Miller, T., Sonenberg, L., & Vetere, F. (2019). Explainable reinforcement learning through a causal lens. In *arXiv preprint arXiv:1905.10958*.
- Magnaguagno, M. C., FRAGA PEREIRA, R., Móre, M. D., & Meneguzzi, F. R. (2017). Web planner: A tool to develop classical planning domains and visualize heuristic state-space search. In *2017 Workshop on User Interfaces and Scheduling and Planning (UISP@ ICAPS), 2017, Estados Unidos*.
- Miller, T. (2018). Contrastive explanation: A structural-model approach. In *arXiv preprint arXiv:1811.03163*.
- Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267, 1–38.
- Miller, T., Howe, P., & Sonenberg, L. (2017). Explainable AI: Beware of inmates running the asylum. In *Proc. International Joint Conf. on AI 2017 Workshop on Explainable Artificial Intelligence (XAI)*.
- Mueller, S. T., Hoffman, R. R., Clancey, W. J., Emrey, A., & Klein, G. (2019). Explanation in Human-AI Systems: A Literature Meta-Review, Synopsis of Key Ideas and Publications, and Bibliography for Explainable AI. *CoRR*, *abs/1902.01876*.
- Nguyen, T. A., Do, M. B., Gerevini, A., Serina, I., Srivastava, B., & Kambhampati, S. (2012). Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence*, 190, 1–31.
- Nielsen, J. (2000). Why you only need to test with 5 users. In *Nielsen Norman Group, Nielsen*.
- Penney, S., Dodge, J., Hilderbrand, C., Anderson, A., Simpson, L., & Burnett, M. (2018). Toward foraging for understanding of starcraft agents: An empirical study. In *23rd International Conference on Intelligent User Interfaces*, pp. 225–237. ACM.
- Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, *abs/1602.04938*.
- Rintanen, J., et al. (2007). Complexity of concurrent temporal planning.. In *ICAPS, Vol. 7*, pp. 280–287.

- Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5), 206–215.
- Smith, D. (2012). Planning as an iterative process. In *Proc. Association for Advancement of AI*.
- Sohrabi, S., Baier, J. A., & McIlraith, S. A. (2011). Preferred explanations: Theory and generation via planning. In *Proc. Association for Advancement of AI*.
- Sreedharan, S., Chakraborti, T., & Kambhampati, S. (2018). Handling model uncertainty and multiplicity in explanations via model reconciliation. In *Proc. International Conf. on Automated Planning and Scheduling*.
- Sreedharan, S., Srivastava, S., Smith, D., & Kambhampati, S. (2019). Why couldn't you do that? explaining unsolvability of classical planning problems in the presence of plan advice. In *arXiv preprint arXiv:1903.08218*.
- Tabrez, A., Luebbers, M. B., & Hayes, B. (2020). A survey of mental modeling techniques in human–robot teaming. *Current Robotics Reports*, 1, 259–267.
- Van Bouwel, J., & Weber, E. (2002). Remote causes, bad explanations?. *Journal for the Theory of Social Behaviour*, 32(4), 437–449.
- van der Krogt, R., & de Weerdt, M. (2005a). Plan Repair as an Extension of Planning. In Biundo, S., Myers, K. L., & Rajan, K. (Eds.), *Proc. International Conf. on Automated Planning and Scheduling (ICAPS)*, pp. 161–170. AAAI.
- van der Krogt, R., & de Weerdt, M. (2005b). Plan Repair using a Plan Library. In Verbeeck, K., Tuyls, K., Nowé, A., Manderick, B., & Kuijpers, B. (Eds.), *Proc. Belgium-Netherlands Conf. on AI (BNAIC)*, pp. 284–259. Koninklijke Vlaamse Academie van Belie voor Wetenschappen en Kunsten.
- van Fraassen, C. B. (1980). *The Scientific Image*. Oxford University Press.
- Van Melle, W. (1978). Mycin: a knowledge-based consultation program for infectious disease diagnosis. *International journal of man-machine studies*, 10(3), 313–322.
- Zhang, Y., Sreedharan, S., Kulkarni, A., Chakraborti, T., Zhuo, H., & Kambhampati, S. (2017). Plan explicability and predictability for robot task planning. In *Proc. IEEE Conference on Robotics and Automation*.