Proceedings · XAIP 2019

2nd International Workshop on Explainable AI Planning

Berkeley, USA
12th July 2019

Chairs
Tathagata Chakraborti (IBM Research AI, USA)
Dustin Dannenhauer (Navatek, LLC, USA)
Joerg Hoffmann (Saarland University, Germany)
Daniele Magazzeni (King's College London, UK)

Collocated with the 29th International Conference on Automated Planning and Scheduling (ICAPS) ▪ Berkeley, USA ▪ 11-15 July 2019

Mission:

Explainable Artificial Intelligence (XAI) concerns the challenge of shedding light on opaque models in contexts for which transparency is important, i.e. where these models could be used to solve analysis or synthesis tasks. In particular, as AI is increasingly being adopted into application solutions, the challenge of supporting interaction with humans is becoming more apparent. Partly this is to support integrated working styles, in which humans and intelligent systems cooperate in problem-solving, but also it is a necessary step in the process of building trust as humans migrate greater competence and responsibility to such systems. The challenge is to find effective ways to characterize, and to communicate, the foundations of AI-driven behavior, when the algorithms that drive it are far from transparent to humans. While XAI at large is primarily concerned with learning-based approaches, model-based approaches are well suited – arguably better suited – for explanation, and Explainable AI Planning (XAIP) can play an important role in addressing complex decision-making procedures.

After the success of previous workshops on XAI and XAIP, the mission of this workshop is to mature and broaden the XAIP community, fostering continued exchange on XAIP topics at ICAPS.

Topics

- Frameworks for defining meaningful explanations in planning and scheduling contexts;
- Representation, organization, and memory content used in explanation;
- The creation of such content during plan generation or understanding;
- Generation and evaluation of explanations;
- The explanation process, i.e. the way in which explanations are communicated to humans (e.g., plan summaries, answers to questions);
- The role of knowledge and learning in explainable planners;
- Human vs AI models in explanations;
- Links between explainable planning and other disciplines (e.g. social science, argumentation);
- Model differences and model reconciliation;
- Goal reasoning and plan explanations;
- Excuse generation, unsolvability, and explanations;
- Use cases and applications of explainable planning.

Page

Invited Speaker:

Robert R. Hoffman, Institute for Human and Machine Cognition

Macrocognition: Foundations for Planning and Explanation

Macrocognition is how cognition adapts to complexity. The historical roots of macrocognition reach back to the late 1800s, and the essentials of the paradigm have been fairly well specified.

The models of sensemaking, flexecution, coordination, re-learning, and mental projection help clarify differences between macrocognitive and microcognitive approaches. Microcognitive models are based on causal chains having distinct start and stop (or input-output) points. On the other hand, macrocognitive models are cyclical and closed-loop. Microcognitive models are useful in hindsight, to tell stories; macrocognitive models are transcendent and anticipatory.

The primary macrocognitive functions correspond with the "Families of Laws of Complex Cognitive Systems" developed by David Woods. The Families are based on five fundamental bounds on complex human-machine work systems. Noteworthy aspects of macrocognition are pertinent to planning systems technology:

- *Flexecution emphasizes the fact that goals morph even as they are being pursued.*
- *Re-grounding embraces the fact that planning and plan execution are team activities.*
- *Projection elaborates on how planning is anticipatory.*

These macrocognitive concepts and models have implications for Explainable AI (XAI) systems. If we present to a user an AI planning system that explains how it works, how do we know whether the explanation works and the user has made sense of the AI and is able to flexecute with it? In other words, how do we know that an XAI system is any good? Key concepts of measurement include specific methods for evaluating: (1) the goodness of explanations, (2) whether users are satisfied by explanations, (3) how well users understand the AI systems, (4) how curiosity motivates the search for explanations, (5) whether the user's trust and reliance on the AI are appropriate, and finally, (6) how well the human-XAI work system performs.

Branching-Bounded Contingent Planning via Belief Space Search

Kevin McAreavey[1] and **Kim Bauters**[1] and **Weiru Liu**[1] and **Jun Hong**[2]
[1]University of Bristol, UK
{kevin.mcareavey, kim.bauters, weiru.liu}@bristol.ac.uk
[2]University of the West of England, UK
jun.hong@uwe.ac.uk

Abstract

A contingent plan can be encoded as a rooted graph where branching occurs due to sensing. In many applications it is desirable to limit this branching; either to reduce the complexity of the plan (e.g. for subsequent execution by a human), or because sensing itself is deemed to be too expensive. This leads to an established planning problem that we refer to as branching-bounded contingent planning. In this paper, we formalise solutions to such problems in the context of history-, and belief-based policies: under noisy sensing, these policies exhibit differing notions of sensor actions. We also propose a new algorithm, called BAO*, that is able to find optimal solutions via belief space search. This work subsumes both conformant and contingent planning frameworks, and represents the first practical treatment of branching-bounded contingent planning that is valid under partial observability.

1 Introduction

In planning under uncertainty, a contingent plan is the most general solution form. A typical encoding of a contingent plan is a rooted graph (or tree) that exhibits branching. This branching occurs because a contingent plan must account for all possible feedback that might be received during plan execution in response to *sensing*. However, in many applications it is desirable to limit this branching; either to reduce the complexity of the plan, or because sensing itself is deemed to be too expensive. This leads to an established planning problem (Baral, Kreinovich, and Trejo 2000; Meuleau and Smith 2002; Bonet 2010) that we will refer to as *branching-bounded contingent planning*.[1]

Our intuition is that there exists a positive correlation between the complexity of a contingent plan (e.g. how difficult it is for a human to comprehend or execute), and the amount of branching that it contains. Conversely, studies have shown that humans are demonstrably bad at following complex plans (Dodson et al. 2013). In this sense, branching-bounded contingent planning provides a means to ensure that plans are sufficiently simple so as to be understood by humans. This idea has previously been referred to as the “cognitive simplicity” of a plan (Meuleau and Smith 2002), and is an important consideration in numerous explainable AI planning (XAIP) applications, including where humans are required to verify plans generated by automated planners (Meuleau and Smith 2002), and where humans are required to execute such plans (Green et al. 2011).

Considerations around plan complexity also extend to the field of autonomous agents. For example, if agents have limited computational resources, then it may not be feasible to maintain the agent’s belief state online, which precludes the direct use of functional plan representations such as belief-based policies (Kaelbling, Littman, and Cassandra 1998; Meuleau and Smith 2002). This is true of recent work on augmenting belief-desire-intention (BDI) agents with automated planners and reusable plans (Meneguzzi and De Silva 2015), where it is important to limit the complexity of new plans so as to maintain the agent’s reactiveness: the greater the amount of branching in the plan, the greater the increase in the size of the agent’s plan library, and the greater the computational cost associated with future plan selection.

A related approach to contingent planning is the field of conformant planning, which deals with domains that are non-observable (i.e. that have no sensing). Although such domains are sometimes dismissed as having little practical interest (Taig and Brafman 2015), a common motivation for conformant planning is in applications where sensing is deemed to be too expensive (Domshlak and Hoffmann 2006). This suggests that such applications are not truly non-observable, but rather that the use of sensing should be bounded (and potentially avoided altogether). In fact, branching-bounded contingent planning can be seen as a generalisation of both conformant and contingent planning.

The only practical treatment of branching-bounded contingent planning appears to be the work of Meuleau and Smith (2002) in the context of partially observable Markov decision processes (POMDPs). Informally, their method restricts sensing to a special *observe-and-branch* action, and then bounds the number of times that this action will be included in the solution plan. Unfortunately, the method is only valid under the assumption of full observability (Bonet 2010), and limits the generality of contingent planning by prohibiting richer forms of sensing. As far as we are aware, the only other work on bounded branching has been theoretical analyses of the complexity of various planning problems (Baral, Kreinovich, and Trejo 2000; Bonet 2010).

[1]Also known as *limited contingency planning* (Meuleau and Smith 2002). Not to be confused with other forms of planning with bounded parameters (e.g. see Section 4 for a discussion).

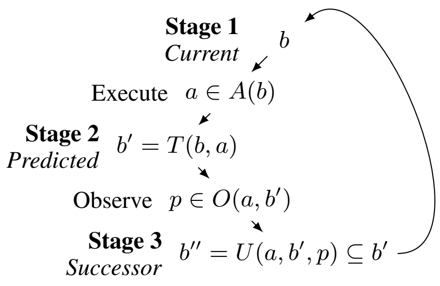

Stage 1 *Current* b
Execute $a \in A(b)$
Stage 2 *Predicted* $b' = T(b, a)$
Observe $p \in O(a, b')$
Stage 3 *Successor* $b'' = U(a, b', p) \subseteq b'$

Figure 1: Belief state update procedure.

In this paper, we propose the first practical treatment of branching-bounded contingent planning that is valid under partial observability. We account for uncertainty over the initial state, action-effects, and observations. The main contributions are as follows: (i) we propose a definition of branching-bounded contingent plans in the context of history-based policies; (ii) we explore the implications of bounded branching in the context of belief-based policies; (iii) we propose a definition of branching-bounded contingent plans as generalised belief-based policies that track sensing; and (iv) we propose a variant of the AO* search algorithm for AND/OR graphs, called BAO*, that is able to find optimal solutions via belief space search. We will rely on the partially observable non-deterministic (POND) model of contingent planning, where a belief state is a set of states, but our ideas can likely extend to other models (e.g. goal-POMDPs, where a belief state is a probability distribution over the state space). We focus on offline planning (i.e. where a complete plan is generated and then executed in full) and will not consider online planning (i.e. where planning and plan execution are interleaved).

The remainder of the paper is organised as follows: in Section 2, we recall preliminaries on contingent planning and AND/OR graphs; in Section 3, we describe our solution to branching-bounded contingent planning; in Section 4, we discuss related work; and, in Section 5, we conclude.

2 Preliminaries

In this section, we recall necessary preliminaries on contingent planning and AND/OR graphs. We rely on some standard mathematical notation: $|S|$ is the cardinality of set S, 2^S is the powerset of S, $\mathbb{R}^+$ is the set of positive real numbers ($0 \notin \mathbb{R}^+$), and $\mathbb{N}$ is the set of natural numbers ($0 \in \mathbb{N}$).

2.1 Contingent Planning

A contingent planning domain is a tuple (S, A, P, C, T, O) where S is a set of states (called the state space) with $B = 2^S \setminus \{\emptyset\}$ the set of belief states (called the belief space), A is a set of actions with $A(s) \subseteq A$ the set of applicable actions in state $s \in S$, P is a set of percepts with $\varnothing_P \in P$ the null percept, $C : A \to \mathbb{R}^+$ is a cost function, $T : S \times A \to B$ is a transition function, and $O : A \times S \to 2^P \setminus \{\emptyset\}$ is an observation function. We say that T (resp. O) is deterministic if $T(s, a)$ (resp. $O(a, s)$) is a singleton for each $s \in S$ and each $a \in A$, otherwise T (resp. O) is non-deterministic. We assume that a percept will be observed after every action-execution (see Figure 1), but if it is possible to observe *nothing* in state s, then this is encoded simply as the null percept $\varnothing_P \in O(s)$. Finally, a contingent planning task is a tuple (M, b_1, S_G) where M is a contingent planning domain, $b_1 \in B$ is an initial belief state, and $S_G \subseteq S$ is a goal. We say that a belief state $b \in B$ satisfies the goal iff $b \subseteq S_G$, i.e. when the agent is guaranteed to be in a goal state.

The notion of applicable actions is extended to a belief state $b \in B$ as $A(b) = A(s_1) \cap \cdots \cap A(s_n)$ such that $b = \{s_1, \ldots, s_n\}$, meaning that an action is applicable in b iff it is applicable in each state $s \in b$.[2] The transition function T is extended as a function $T : B \times A \to B$ defined as $T(b, a) = \{s \in T(s', a) \mid s' \in b\}$. The observation function O is extended as a function $O : A \times B \to 2^P \setminus \{\emptyset\}$ defined as $O(a, b) = \{p \in O(a, s) \mid s \in b\}$. Finally, an update function $U : A \times B \times P \to B$ is defined as:

$$U(a, b, p) = \begin{cases} \{s \in b \mid p \in O(a, s)\} & \text{if non-empty} \\ \textit{undefined} & \text{otherwise} \end{cases}$$

Evidently, $U(a, b, p) = \textit{undefined}$ iff $p \notin O(a, b)$. We say that $b' = T(b, a)$ is a predicted belief state and $U(a, b', p) \subseteq b'$ is a successor belief state (again, see Figure 1).

In practice, contingent planning problems typically require a factorised representation in order to express problems of any meaningful complexity. For example, S can be defined by a set of (independent) state variables, $A(s)$ (resp. T and O) can be defined by a set of action schema preconditions (resp. effects and observations) associated with A, and b_1 (resp. S_G) can be defined by a logical formula over the set of state variables. We refer the interested reader to a planning language capable of expressing contingent planning problems, such as NuPDDL[3] or PO-PPDDL[4].

2.2 AND/OR Graphs

A (directed) graph is a tuple (N, E) where N is a set of nodes and $E \subseteq N \times N$ is a set of (directed) edges. A multigraph is a tuple (N, I, E') where I is a set of identifiers and $E' \subseteq N \times I \times N$ is a set of multiedges such that for each $i \in I$ we have that $(N, \{(n, n') \mid (n, i, n') \in E'\})$ is a graph. Given nodes $n, n' \in N$ in a multigraph, then n is said to be a parent of n' (resp. n' is a child of n) iff $(n, i, n') \in E'$ for some $i \in I$. A node $n \in N$ is said to be a branch point if n has more than one child. Given nodes $n_1, n_{m+1} \in N$ in a multigraph, then a sequence of nodes and identifiers $(n_1, i_1, \ldots, n_m, i_m, n_{m+1})$ is a path from n_1 to n_{m+1} iff each n_j is unique and $(n_j, i_j, n_{j+1}) \in E'$ for $j = 1, \ldots, m$. A multigraph is acyclic if, for each $n \in N$, there does not exist a path from n to itself. A rooted (and connected) multigraph is a tuple (N, I, E', n) where (N, I, E') is a multigraph and $n \in N$ is a root node such that, for each $n' \in N$, there exists a path from n to n'.

[2]This is the cautious approach to applicable actions; its dual $A(s_1) \cup \cdots \cup A(s_n)$ is possible but complicates later definitions.

[3]http://mbp.fbk.eu/NuPDDL.html

[4]http://users.cecs.anu.edu.au/~ssanner/IPPC_2011

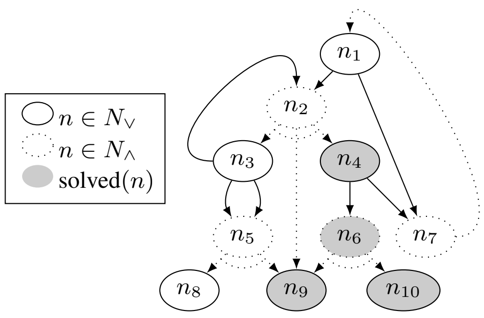


Figure 2: AND/OR graph.

An AND/OR graph $(N_\vee \cup N_\wedge, I_\vee \cup I_\wedge, E_\vee \cup E_\wedge, n)$ is a rooted multigraph where $N_\vee$ (resp. $N_\wedge$) is a set of OR-nodes (resp. AND-nodes), $I_\vee$ (resp. $I_\wedge$) is a set of OR-identifiers (resp. AND-identifiers), $E_\vee \subseteq N_\vee \times I_\vee \times N_\wedge$ (resp. $E_\wedge \subseteq N_\wedge \times I_\wedge \times N_\vee$) is a set of OR-edges (resp. AND-edges), and $n \in N_\vee$ is a root node. An AND/OR graph is typically used to reduce problems into decomposable sub-problems. Intuitively, an AND-node is a solution if each of its child nodes is a solution, while an OR-node is a solution if it is a primitive solution, or at least one of its child nodes is a solution (e.g. see Figure 2). In the next section, we will demonstrate how branching-bounded contingent planning can be cast as a search problem over AND/OR graphs.

3 Framework

In this section, we formalise branching-bounded contingent planning in the context of history-, and belief-based policies, and propose a new solution that relies on belief space search.

3.1 History-Based Policies

An execution is a possibly infinite sequence $(a_1, p_1, a_2, p_2, \dots)$ where $a_i \in A$ and $p_i \in P$. A finite execution is also called a history, with H the set of histories (called the history space). The length of history $h_{i+1} = (a_1, p_1, \dots, a_i, p_i)$ is defined as $|h_{i+1}| = i$. A history-based policy is a function $\pi_h : H' \rightarrow A$ where $H' \subseteq H$. The executions that are possible with respect to b_1 and π_h are defined inductively along with their associated belief states as follows: the empty execution h_1 is possible and b_1 is its belief state; if h_i is possible and b_i is its belief state, then $h_{i+1} = (h_i, a_i, p_i)$ is possible and $b_{i+1} = U(a_i, b'_i, p_i)$ is its belief state iff $h_i \in H'$ such that $a_i = \pi_h(h_i)$, $p_i \in O(a_i, b'_i)$, and $b'_i = T(b_i, a_i)$. An execution h that is possible with respect to b_1 and π_h is said to be complete if $h \notin H'$ or if h is infinite. Finally, π_h is said to be a *strong* solution to a contingent planning task (M, b_1, S_G) if each complete execution h_i is finite and $b_i \subseteq S_G$.

Definition 1. *A history-based contingent plan is a history-based policy π_h that is a strong solution to a contingent planning task (M, b_1, S_G).*

A history-based policy π_h can be encoded as a rooted acyclic multigraph over histories (called a history-based policy graph) where π_h is a node label function and where each multiedge identifier is a percept (Kaelbling, Littman, and Cassandra 1998).[5] More precisely, a history-based policy graph is a rooted tree. To execute the plan, an agent simply needs to execute the actions specified by node labels, while tracing a single path in line with observed percepts. In this way, branch points are those nodes where no single percept is guaranteed to occur after executing the specified action, while the actions themselves can be thought of as information gathering actions, called *sensor* actions.

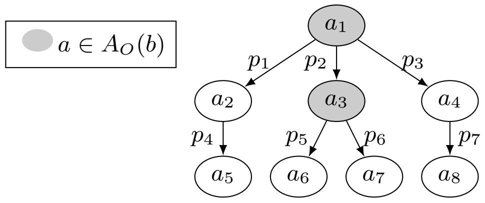


Figure 3: History-based k-branching-bounded plan, $k \geq 2$.

Definition 2. *The set of history-based sensor actions in belief state $b \in B$, denoted $A_O(b)$, is defined as:*

$$A_O(b) = \{a \in A(b) \mid b' = T(b, a), \\ \exists p, p' \in O(a, b'), p \neq p'\}$$

Definition 3. *The number of history-based sensor actions in execution h, denoted $\Psi_O(h)$, is defined as:*

$$\Psi_O(h) = |\{i = 1, 2, \cdots \mid a_i \in A_O(b_i)\}|$$

where $h = (a_1, p_1, a_2, p_2, \dots)$.

Definition 4. *A history-based contingent plan π_h is a history-based k-branching-bounded contingent plan with $k \in \mathbb{N} \cup \{\infty\}$ iff π_h satisfies:*

$$\max_{h \in H^*} \Psi_O(h) \leq k$$

where H^* is the set of complete executions of π_h.

Definition 4 says that a history-based k-branching-bounded contingent plan is a history-based contingent plan where, in the corresponding history-based policy graph, there is at most k branch points on any path from the root node to a leaf node (e.g. as in Figure 3). If $k = 0$ (resp. $k = \infty$), then a history-based k-branching-bounded contingent plan is a conformant plan (resp. contingent plan). This definition is similar to the definition of balanced k-contingency plans from (Meuleau and Smith 2002). As we will see in subsequent sections, however, this definition is too strong in the context of a special type of history-based policy known as a belief-based policy.

3.2 Belief-Based Policies

A history-based policy π_h is called a belief-based policy if $\pi_h(h_i) = \pi_h(h_j)$ for any $h_i, h_j \in H'$ such that $b_i = b_j$ and $|h_i| = |h_j|$. For this reason, a belief-based policy can

[5]An equivalent definition is a rooted acyclic graph over histories where each edge is labelled with a percept.

be defined as a function $\pi : X \to A$ where $X \subseteq B \times D$ with $D \subseteq \mathbb{N}$ the set of time steps. We say that π is stationary if $\pi(b,t) = \pi(b,t')$ for all $t,t' \in D$ such that $t \neq t'$, otherwise π is non-stationary. A stationary belief-based policy can be defined as a function $\pi : B' \to A$ where $B' \subseteq B$. Belief-based policies are typically easier to find than their history-based counterparts: the belief space is large but bounded, whereas the history space is unbounded. Analogous to history-based policy graphs, a belief-based policy can be encoded as a rooted multigraph over (time-indexed) belief states, called a belief-based policy graph. Importantly, while history-based policies lead to policy graphs that are trees, belief-based policy graphs can be more compact, since it is possible to arrive at the same node via different executions. In fact, if the policy is stationary, then a belief-based policy graph may exhibit cycles, because it is also possible to return to a previously visited node.

The notion of a strong solution for (acyclic) history-based policies is extended to (potentially cyclic) belief-based policies through the notion of a *strong-cyclic* solution (Cimatti et al. 2003). Intuitively, cycles in a belief-based policy can lead to infinite executions, but such executions are only permitted when they are *unfair*. Formally, an infinite execution h is said to be fair if, when action a is executed an infinite number of times in belief state b, then every percept $p \in O(a,b')$ with $b' = T(b,a)$ is also observed an infinite number of times, otherwise h is unfair. A belief-based policy π is said to be a strong-cyclic solution to a contingent planning task (M, b_1, S_G) if, for each complete execution h_i, either: (i) h_i is finite and $b_i \subseteq S_G$, or (ii) h_i is infinite and unfair. It follows that a strong solution is a strong-cyclic solution where every complete execution is finite.

Definition 5. *A belief-based contingent plan is a belief-based policy π that is a strong-cyclic solution to a contingent planning task (M, b_1, S_G).*

In the context of history-based policies, branching actions are those action-executions that can lead to distinct successor histories (i.e. distinct nodes in the policy graph). However, the fact that it is possible to arrive at the same (time-indexed) belief state via different executions suggests that Definition 2 is not valid in the context of belief-based policy graphs. Thus, in order to better understand branching in belief-based policies, let us now explore the relationship between possible percepts and successor belief states:

Lemma 1. $1 \leq |\{U(a,b,p) \mid p \in O(a,b)\}| \leq |O(a,b)|$.

Proof. By definition, $O(a,b) \subseteq P$ such that $O(a,b) \neq \emptyset$. Thus, $|O(a,b)| \geq 1$. Similarly, if $p \in O(a,b)$, then $U(a,b,p) \in B$, otherwise $U(a,b,p) =$ *undefined*. Thus, $|\{U(a,b,p) \mid p \in O(a,b)\}| \geq 1$ and $|\{U(a,b,p) \mid p \in O(a,b)\}| \leq |O(a,b)|$. □

Lemma 2. *It is guaranteed that $|O(a,b)| = |\{U(a,b,p) \mid p \in O(a,b)\}|$ iff O is deterministic.*

Proof. Given Lemma 1, we just need to prove (i) that there exists a bijection (i.e. a one-to-one correspondence) $f : O(a,b) \to \{U(a,b,p) \mid p \in O(a,b)\}$ when O is deterministic, and (ii) that such a bijection is not guaranteed to exist when O is non-deterministic.

(i) Suppose O is deterministic. By Definition, $O(a,s)$ is a singleton for each $s \in b$. Similarly, if $s \in b$ and $p \in O(a,s)$, then $s \in U(a,b,p)$. Conversely, if $s \in b$ and $p \notin O(a,s)$, then $s \notin U(a,b,p)$. It follows that, if $O(a,b) = \{p_1, \ldots, p_n\}$, then $\{f(p_1), \ldots, f(p_n)\}$ forms a partition[6] of b with $f(p_i) = U(a,b,p_i)$, which satisfies the definition of a bijection.

(ii) Suppose O is non-deterministic such that $O(a,s) = P$ for each $a \in A$ and each $s \in b$ with $|P| > 1$. By definition, we have that $O(a,b) = P$. Moreover, $U(a,b,p) = b$ for each $p \in P$, since $s \in U(a,b,p)$ iff $s \in b$. It follows that $|O(a,b)| > |\{U(a,b,p) \mid p \in O(a,b)\}| \Leftrightarrow |P| > |\{b\}|$, which contradicts the definition of a bijection. □

Corollary 1. $U(a,b,p) = b$ *if* $O(a,b) = \{p\}$.

Proof. This follows from proof (i) of Lemma 2 and the fact that $O(a,b) = \{p\}$ is a deterministic observation, regardless whether O is itself a deterministic function. □

Lemma 1 says, firstly, that every action-execution is guaranteed to result in at least one possible percept and one successor belief state and, secondly, that the number of possible percepts is an upper bound on the number of possible successor belief states. Lemma 2 then says that, if O is deterministic, there exists a unique successor belief state for each possible percept, but that this is not guaranteed if O is non-deterministic. Finally, Corollary 1 says that, if there is only one possible percept (whether $p = \varnothing_P$ or otherwise), then the (single) successor belief state will be the same as the predicted belief state. Given these properties, we can now propose a definition of branching actions in the context belief-based policies that remains valid for both deterministic and non-deterministic observations:

Definition 6. *The set of belief-based sensor actions in belief state $b \in B$, denoted $A_U(b)$, is defined as:*

$$A_U(b) = \{a \in A(b) \mid b' = T(b,a), \exists p, p' \in O(a,b'), U(a,b',p) \neq U(a,b',p')\}$$

Proposition 1. $A_U(b) \subseteq A_O(b)$.

Proof. By definition, $a \in A_O(b)$ iff $O(a,b')$ is not a singleton with $b' = T(b,a)$. Conversely, $a \in A_U(b)$ iff $\{U(a,b,p) \mid p \in O(a,b')\}$ is not a singleton. Thus, it follows from Lemma 1 that, if $a \in A_U(b)$, then it must also be that $a \in A_O(b)$. □

Proposition 2. *It is guaranteed that $A_U(b) = A_O(b)$ iff O is deterministic.*

Proof. This follows directly from Lemma 2 and Proposition 1, in that a bijection $f : O(a,b) \to \{U(a,b,p) \mid p \in O(a,b)\}$ is guaranteed to exist iff O is deterministic. □

[6]This observation has been made previously (Russell and Norvig 2009, Chapter 4).

Definition 7. *The number of belief-based sensor actions in execution h, denoted $\Psi_U(h)$, is defined as:*

$$\Psi_U(h) = |\{i = 1, 2, \cdots \mid a_i \in A_U(b_i)\}|$$

where $h = (a_1, p_1, a_2, p_2, \dots)$.

Definition 8. *A belief-based contingent plan π is a belief-based k-branching-bounded contingent plan with $k \in \mathbb{N} \cup \{\infty\}$ iff π satisfies:*

$$\max_{h \in H^*} \Psi_U(h) \leq k$$

where H^* is the set of complete executions of π.

The problem with Definition 8 is that, in the context of bounded branching, a (time-indexed) belief state is not a *sufficient statistic* (Striebel 1965) for a history. Specifically, we know it is possible to arrive at the same (time-indexed) belief state via different executions, but this also means that those executions may contain different numbers of sensor actions; this may have implications for the choice of action, and may even preclude further sensing. For example, suppose there are two executions h_i and h_j such that $b_i = b_j$ and $|h_i| = |h_j|$. If $\Psi(h_i) = k - 1$, then the best action (to reach the goal) in h_i might be to execute a sensor action, but if $\Psi(h_j) = k$, then executing a sensor action in h_j is not an option, although the goal may still be reachable from b_j via some other non-sensor action. This suggests that belief-based policies (whether stationary or not) are too restrictive to properly capture bounded branching. In the next section, we will solve this problem by proposing a generalisation of belief-based policies, called tracking-based policies.

3.3 Tracking-Based Policies

A history-based policy π_h is called a tracking-based policy with $k \in \mathbb{N} \cup \{\infty\}$ if π_h satisfies the following: (i) $h \notin H'$ if $\Psi_U(h) > k$; (ii) $\pi_h(h_i) \notin A_U(b_i)$ if $\Psi_U(h_i) = k$; and (iii) $\pi_h(h_i) = \pi_h(h_j)$ for any $h_i, h_j \in H'$ such that $b_i = b_j$ and $k - \Psi_U(h_i) = k - \Psi_U(h_j)$. As such, a tracking-based policy can be defined as a function $\pi_k : X \to A$ where $X \subseteq B \times D$ with $D = \{t \in \mathbb{N} \mid t < k\} \cup \{k\}$ the set of decision steps. Intuitively, a tracking-based policy generalises a belief-based policy where $\pi_k(b, t)$ denotes the action to execute in belief state b with t remaining sensor actions.[7] This is similar to belief-based policies in fault-tolerant planning, where actions may depend on the number of "failures so far" (Domshlak 2013). Conversely, while non-stationary belief-based policies are typical in finite horizon planning problems (Geffner and Bonet 2013, Chapter 6), it is worth emphasising that branching-bounded contingent planning problems do not technically have a finite horizon (e.g. if $k = \infty$, or if the domain is non-observable).

Definition 9. *A tracking-based contingent plan is a tracking-based policy π_h that is a strong-cyclic solution to a contingent planning task (M, b_1, S_G).*

As with belief-based policies, a tracking-based policy can be encoded as a rooted multigraph over (step-indexed) belief states, called a tracking-based policy graph. As such, tracking-based policy graphs are generally more compact than history-based policy graphs, and may contain cycles.

[7] Time-dependent tracking-based policies are also possible.

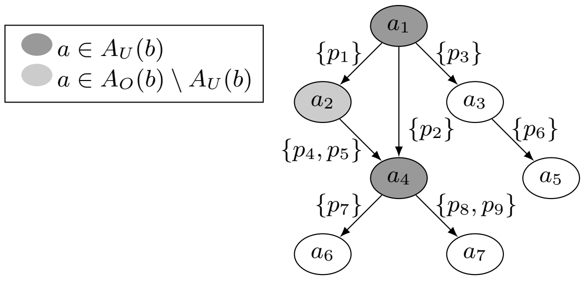


Figure 4: Tracking-based k-branching-bounded plan, $k \geq 2$.

Proposition 3. *Let π_k be a tracking-based contingent plan with H^* its set of complete executions. Then π_k satisfies:*

$$\max_{h \in H^*} \Psi_U(h) \leq k$$

Proof. By definition, if π_k is a tracking-based policy and $\Psi_U(h) > k$, then $h \notin H'$. If $h \notin H'$, then by definition (h, a, p) cannot be a possible execution of π_k for any $a \in A$ and any $p \in P$. Conversely, if $\Psi_U(h_i) = k$ and $h_i \in H'$, then by definition (h_i, a, p) cannot be a possible execution of π_k with $a = \pi_k(b_i, 0)$ for any $p \in P$ if $a \in A_U(b_i)$. Finally, if h is not a possible execution of π_k, then by definition h cannot be a complete execution of π_k. Thus, it must be that $\Psi_U(h) \leq k$ if h is a complete execution of π_k. $\square$

Definition 10. *A tracking-based contingent plan π_k is also called a tracking-based k-branching-bounded contingent plan.*

Proposition 3 says that tracking-based contingent plans directly encode the intuition of belief-based k-branching-bounded contingent plans. Once again, Definition 10 is similar to balanced k-contingency plans from (Meuleau and Smith 2002), and if $k = 0$ (resp. $k = \infty$), then a tracking-based k-branching-bounded contingent plan is a conformant plan (resp. contingent plan). Finally, Figure 4 demonstrates how tracking-based contingent plans can be less sensitive to bounded branching than their history-based counterparts (e.g. the plan is a history-based k-branching-bounded contingent plan iff $k \geq 3$).

Theorem 1. *It is guaranteed that a tracking-based k-branching-bounded contingent plan π_k is a strong solution iff $k < \infty$.*

Proof. We need to prove: (i) that a strong-cyclic solution may not be a strong solution when $k = \infty$, and (ii) that every strong-cyclic solution is also a strong solution if $k < \infty$.

(i) Suppose $k = \infty$. By definition, π_k reduces to a stationary belief-based policy π where $\pi(b) = \pi_k(b, \infty)$ for any $(b, \infty) \in X$. Thus, π_k may not be a strong solution, since this is true of any stationary belief-based policy.

(ii) Suppose $k < \infty$. By definition, if $\pi_k(b, t) \in A_U(b)$ for some $(b, t) \in X$, then belief state b can only be revisited as part of a distinct step-indexed belief state (b, t') such that $t' < t$. Conversely, if $\pi_k(b, t) \notin A_U(b)$ for some $(b, t) \in X$, then (b, t) can only be revisited as part of an infinite loop with a deterministic sequence of successor belief states (i.e. no branching) leading back to (b, t). Infinite loops correspond to fair executions. Thus, there cannot be an unfair infinite execution of π_k, which means that all complete executions of π_k must be finite. □

Theorem 1 guarantees that, if $k < \infty$, then a tracking-based k-branching-bounded contingent plan will be acyclic. Combined with our observation that (time-indexed) belief states are not a sufficient statistic for bounded branching, this theorem solves an open question about defining branching-bounded contingent plans in the context of (potentially cyclic) belief-based policies (Bonet 2010). More importantly, this theorem implies that certain types of algorithms (i.e. those for acyclic plans) may be more convenient than others for branching-bounded contingent planning.

3.4 Algorithm

In this section, we propose a variant of the AO* optimal top-down heuristic search algorithm for acyclic AND/OR graphs (Nilsson 1971, Chapter 3; Martelli and Montanari 1973). AO* itself is the search algorithm employed by numerous existing contingent planners (Bonet and Geffner 2000; Hoffmann and Brafman 2005; Bryce, Kambhampati, and Smith 2006), where it is typically used to construct *optimal* history-based policies incrementally (Geffner and Bonet 2013, Chapter 5). However, AO* can also be used to construct optimal *acyclic* belief-based policies via belief space search. In a similar way, our algorithm (called Bounded AO*, or BAO* for short) is able to find optimal history- and tracking-based k-branching-bounded contingent plans. BAO* is sound and complete when $k < \infty$, but may be incomplete when $k = \infty$ where strong-cyclic solutions are not guaranteed to be strong solutions (see Theorem 1).

We first need to formalise what we mean by optimality. Let Π be the set of history-based policies. The cost function C is extended to Π as a function $C : \Pi \to \mathbb{R}^+$ defined as:

$$C(\pi_h) = \begin{cases} \max\limits_{h \in H^*} \sum\limits_{i=1}^{|h|} C(a_i) & \text{if } H^* \neq \emptyset \\ \infty & \text{otherwise} \end{cases}$$

where $h = (a_1, p_1, a_2, p_2, \dots)$ and H^* is the set of complete executions of π_h. Note that $C(\pi_h) = \infty$ if π_h has an infinite execution. Of course, if π_h is a tracking-based k-branching-bounded contingent plan with $k < \infty$, then Theorem 1 guarantees that all executions will be finite.

Definition 11. *Let $\Pi' \subseteq \Pi$ be the set of history- (resp. tracking-based) k-branching-bounded contingent plans. The set of optimal history- (resp. tracking-based) k-branching-bounded contingent plans $\Pi^* \subseteq \Pi'$ is defined as:*

$$\Pi^* = \operatorname*{argmin}_{\pi_h \in \Pi'} C(\pi_h)$$

Definition 11 says that an optimal k-branching-bounded contingent plan minimizes cost in the worst case (that is, the maximum cost for any complete execution). Next, we can formalise the search space of BAO* as follows:

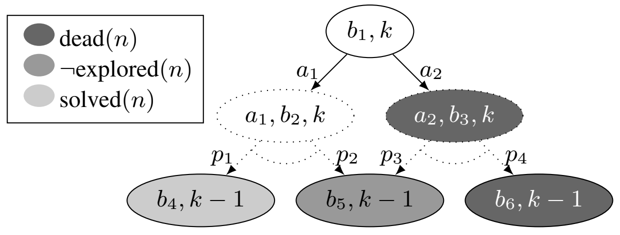


Figure 5: BAO* search graph.

Definition 12. *A belief space search graph is an acyclic AND/OR graph $(N_\vee \cup N_\wedge, A \cup P, E_\vee \cup E_\wedge, n)$ where $N_\vee \subseteq X$, $N_\wedge \subseteq A \times X$, $E_\vee \subseteq N_\vee \times A \times N_\wedge$, $E_\wedge \subseteq N_\wedge \times P \times N_\vee$, and $n \in N_\vee$ is the root node.*

Intuitively, nodes in a belief space search graph are (step-indexed) belief states such that OR-edges link belief states via actions, and AND-edges link belief states via percepts (see Figure 5). The step-index in each node provides the mechanism by which we track the number of sensor actions on a given path. Notice also that AND-nodes are further augmented with an action. The reason for this follows from the fact that the set of possible percepts for a given belief state depends on the action that lead to that belief state, and thus it is necessary to track those actions.

Before describing BAO* in detail, let us introduce the notion of a heuristic function in the context of BAO*. Let $V : N_\vee \cup N_\wedge \to \mathbb{R}^+ \cup \{\infty\}$ be a heuristic function. We say that V is admissible if it never overestimates the cost (with respect to cost function C) of reaching the goal. A binary relation over nodes, denoted $\preceq_V$, is defined for nodes $n, n' \in N_\vee \cup N_\wedge$ as follows:

$$n \preceq_V n' \Leftrightarrow V(n) \leq V(n')$$

Moreover, $n \simeq_V n'$ if $n \preceq_V n'$ and $n' \preceq_V n$. Also, $n \prec_V n'$ if $n \preceq_V n'$ and $n' \not\preceq_V n$. Finally, $\min(N, \preceq_V)$ denotes the single most preferred node in $N \subseteq N_\vee \cup N_\wedge$ with respect to V, with ties broken arbitrarily. As input, BAO* takes a contingent planning task (M, b_1, S_G), an admissible heuristic function V^*, and a bound $k \in \mathbb{N} \cup \{\infty\}$. Importantly, the admissible heuristic function V^* should satisfy the following: $V^*(b, t) = V^*(b, t')$ for all $t, t' \in D$; $V^*(a, b, t) = V^*(b, t)$ for each $a \in A$; and $V^*(a, b, t) = V^*(a', b, t')$ for all $a, a' \in A$ and all $t, t' \in D$. In other words, V^* is independent of the step-index and action.

An outline of BAO* is provided in Algorithm 1, and supplementary definitions (which are identical to AO*) are provided in Table 1. In particular, Table 1b describes how another heuristic function V is derived from the cost function C and the input heuristic function V^*. The heuristic function V represents a revised cost estimate and is computed by BAO* during the search. Therefore, V is the heuristic function that actually guides the search, and is admissible if V^* is admissible. Of course, AO* does not typically compute V at each step; instead it maintains a single heuristic value for each node, which it then updates during the search via a back-propagation procedure. We omit these details for conciseness. The main changes to AO* can be found in Algorithm 1, and relate to the tracking of sensor actions on a

Algorithm 1: BAO*

```
   Input: Contingent planning task $(M, b_1, S_G)$, admissible
          heuristic $V^*$, bound $k \in \mathbb{N} \cup \{\infty\}$
   Output: $\pi_k$ = extract(*root*)
1  *root* $\leftarrow (b_1, k)$
2  while $\neg$solved(*root*) $\wedge$ $V$(*root*) $\neq \infty$ do
3      *leaf* $\leftarrow$ choose(*root*)
4      expand(*leaf*)
5  return extract(*root*)
6  function choose($n$)                          /* OR-node */
7      if $\neg$expanded($n$) then
8          return $n$
9      $n' \leftarrow \min(\{n''' \mid (n, a, n''') \in E_\vee\}, \preceq_V)$
10     $N'_\vee \leftarrow \{n''' \mid (n', p, n''') \in E_\wedge, \neg\text{solved}(n''')\}$
11     $n'' \leftarrow \min(N'_\vee, \preceq_V)$
12     return choose($n''$)
13 procedure expand($n$)                         /* OR-node */
14     $(b, t) \leftarrow n$
15     for each $a \in A(b)$ do
16         $b' \leftarrow T(b, a)$
17         $n' \leftarrow (a, b', t)$
18         if $\neg$path($n', n$) then
19             $E'_\wedge \leftarrow$ expand($n'$)
20             if $E'_\wedge \neq \emptyset$ then
21                 $E_\vee \leftarrow E_\vee \cup \{(n, a, n')\}$
22                 $E_\wedge \leftarrow E_\wedge \cup E'_\wedge$
23 function expand($n$)                          /* AND-node */
24     $(a, b, t) \leftarrow n$
25     $X \leftarrow \emptyset$
26     for each $p \in O(a, b)$ do
27         $b' \leftarrow U(a, b, p)$
28         $X \leftarrow X \cup \{(p, b')\}$
29     if $|\{b' \mid (p, b') \in X\}| > 1$ then
30         if $t = 0$ then
31             return $\emptyset$
32         $t \leftarrow t - 1$
33     $E'_\wedge \leftarrow \emptyset$
34     for each $(p, b') \in X$ do
35         $n' \leftarrow (b', t)$
36         if path($n', n$) then
37             return $\emptyset$
38         $E'_\wedge \leftarrow E'_\wedge \cup \{(n, p, n')\}$
39     return $E'_\wedge$
```

given path, as well as the avoidance of cycles. We can summarise the algorithm as follows:

Lines 3–4 In each iteration, we select an OR-node $n = (b, t)$ for expansion in the current best partial solution.

Lines 14–16 For each applicable action $a \in A(b)$, we generate the predicted belief state $b' = T(b, a)$.

Lines 17–19, 24–28 If possible to add an OR-edge from n to AND-node $n' = (a, b', t)$ without creating a cycle, then for each possible percept $p \in O(a, b)$ we generate the successor belief state $b'' = U(a, b', p)$.

Predicate	Value
expanded(n)	*true* after execution of expand(n), otherwise *false*
path(n, n')	*true* if there is a path from n to n' in the current belief space search graph, otherwise *false*
goal(n)	*true* if $n \in N_\vee$ and $b \subseteq S_G$ with $n = (i, b)$, otherwise *false*
dead(n)	*true* if expanded(n) and n has no children, otherwise *false*
solved(n)	*true* if goal(n), or $n \in N_\vee$ and solved(n') for some child n' of n, or $n \in N_\wedge$ and solved(n') for each child n' of n, otherwise *false*

(a) Predicates in BAO*.

$V(n)$	Condition
0	If goal(n)
∞	If dead(n)
$\min\limits_{n' \in N} C(a) + V(n')$	If $n \in N_\vee$ and expanded(n) such that N is the set of children of n and $n' = (a, b)$
$\max\limits_{n' \in N} V(n')$	If $n \in N_\wedge$ and expanded(n) such that N is the set of children of n
$V^*(n)$	Otherwise

(b) Heuristic function V in BAO*.

Table 1: Supplementary details for BAO*.

Lines 29–32 We record the remaining number of sensor actions as $t' = t - 1$ if $a \in A_U(b)$, or $t' = t$ otherwise.

Lines 33–39, 20–22 If possible to add an AND-edge from n' to OR-node $n'' = (b'', t')$ without creating a cycle, then we add n' as a child of n and each n'' as a child of n'.

Line 2 The search terminates when the root node is solved, or is deemed to be unsolvable via $V(n) = \infty$.

Line 5 A tracking-based k-branching-bounded contingent plan is returned, if found, via extract(n).

Notice that a negative result at line 18 or line 36 does not mean that no solution exists from n involving n' or n'', but simply that no acyclic solution exists (Russell and Norvig 2009, Chapter 4). Specifically, the admissible heuristic function V in (B)AO* ensures that, if n' or n'' are part of the optimal solution, then they will be part of the solution at the point that they were originally expanded. Of course, while Definition 12 and Algorithm 1 focus on tracking-based policies, a simpler variant (i.e. tree-based search, omitted due to space considerations) can also be used to incrementally construct history-based k-branching-bounded contingent plans.

4 Related Work

The only other practical treatment of branching-bounded contingent planning appears to be the work of Meuleau and Smith (2002), whose method is known to be valid only under full observability (Bonet 2010). Thus, our work represents the first practical treatment of this problem that is valid in the general case (i.e. partial observability). As far as we are aware, the only other work that deals with branching-bounded contingent plans has been theoretical analyses of computational complexity in planning (Baral, Kreinovich, and Trejo 2000; Bonet 2010). Bounded branching is a subclass of the broader problem of planning with bounded parameters. In this broader class, finite-horizon planning is perhaps the best known instance, requiring that plans have some bounded execution length (e.g. Rintanen 2007). Another example is conformant probabilistic planning, where *satisficing* plans guarantee some lower bound on the probability of goal achievement under an indefinite horizon (Domshlak and Hoffmann 2006). In fault-tolerant planning, partial plans are permitted under the assumption that only a bounded number of *non-primary* effects will occur during execution (Domshlak 2013). Recent work on compact plans requires that $\pi(\cdot)$ be defined only for some bounded number of *controller states* (Geffner and Geffner 2018) or *memory states* (Chatterjee, Chmelík, and Davies 2018; Pandey and Rintanen 2018). Of course, while all these works belong to the broad class of bounded planning problems, they do not share the same characteristics as branching-bounded contingent planning: they do not reduce branching, and do not generalise conformant planning.

Contingent planners that rely on belief space search can be classified in terms of: (i) their underlying search algorithm for AND/OR graphs; (ii) their belief state representation; and (iii) their heuristics. Our method is agnostic to (ii) and (iii). The best-known algorithm for (i) is probably AO* (Nilsson 1971, Chapter 3; Martelli and Montanari 1973), but other examples include LAO* (Hansen and Zilberstein 2001), LDFS (Bonet and Geffner 2005), and A*LD (Felzenszwalb and McAllester 2007). Contingent planners based on AO* include GPT (Bonet and Geffner 2000), Contingent-FF (Hoffmann and Brafman 2005), and POND (Bryce, Kambhampati, and Smith 2006). To the best of our knowledge, belief space search remains state-of-the-art in general[8] contingent planning (e.g. Contingent-FF can be regarded as state-of-the-art). That being said, alternative techniques include plan space search (Weld, Anderson, and Smith 1998), answer set programming (Tu, Son, and Baral 2007), and compilation (Muise, Belle, and McIlraith 2014). We expect that branching-bounded contingent planning could be achieved with many of these techniques.

5 Conclusion

An implementation of this work is available online.[9] From a practical perspective, we hope to further develop this implementation into a planner that is competitive with (but subsumes) existing state-of-the-art conformant and contingent planners. This could be achieved, for example, by the use of better heuristics (Bryce, Kambhampati, and Smith 2006), or compact belief state representations (Darwiche 2011). Subsequently, we hope to experimentally validate this work using benchmarks that permit different types of branching-bounded contingent plans (e.g. conformant and non-conformant plans). Finally, it is worth mentioning that this work was motivated by an XAIP application involving the recommendation of plans for execution by humans, so it would be interesting to evaluate the effect of bounded branching on human comprehension.

From a theoretical perspective, there are many interesting directions for future work. For example, we have defined branching-bounded contingent plans in terms of a *local* bound (i.e. where branching is bound only on complete paths of the policy graph), but a *global* bound may also be desirable (i.e. where branching is bound across the entire policy graph). The latter would be comparable to the notion of a general k-contingency plans from (Meuleau and Smith 2002). Such plans might be found by maintaining an explicit partial solution during the search, then discarding when the bound is exceeded. However, this would likely complicate backtracking. Another example is that, while we bound the number of branch points, we do not bound the number of branches (i.e. child nodes). Doing so could help to further reduce the complexity of the plan. Finding complete plans would certainly require an online partitioning of $O(a, b)$ such that successor belief states could be defined for sets of percepts in that partition, e.g. where $U(a, b, \{p_1, \ldots, p_n\}) = U(a, b, p_1) \cup \ldots \cup U(a, b, p_n)$. However, optimally choosing that partition is not straightfoward, and there would be no guarantees on the optimality of the resulting plan. As an alternative, fault-tolerant planning techniques (Domshlak 2013) could be used to find partial plans that only account for a bounded number of "most significant" branches.

Acknowledgements

This work received funding from the European Union's Horizon 2020 research and innovation programme, through the DEVELOP project, under grant agreement No. 688127.

References

Baral, C.; Kreinovich, V.; and Trejo, R. 2000. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence* 122(1-2):241–267.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, 52–61.

Bonet, B., and Geffner, H. 2005. An algorithm better than AO*? In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'05)*, 1343–1348.

Bonet, B. 2010. Conformant plans and beyond: Principles and complexity. *Artificial Intelligence* 174(3-4):245–269.

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2006. Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research* 26:35–99.

Chatterjee, K.; Chmelík, M.; and Davies, J. 2018. A symbolic SAT-based algorithm for almost-sure reachability with small strategies in POMDPs. In *Proceedings of the*

[8]That is, where there range of solvable problems is not restricted a priori by the underlying planning technique.

[9]https://github.com/kevinmcareavey/bcp

30th AAAI Conference on Artificial Intelligence (AAAI'16), 3225–3232.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 147(1-2):35–84.

Darwiche, A. 2011. SDD: A new canonical representation of propositional knowledge bases. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJ-CAI'11)*, 819–826.

Dodson, T.; Mattei, N.; Guerin, J. T.; and Goldsmith, J. 2013. An English-language argumentation interface for explanation generation with Markov decision processes in the domain of academic advising. *ACM Transactions on Interactive Intelligent Systems* 3(3):18:1–30.

Domshlak, C., and Hoffmann, J. 2006. Fast probabilistic planning through weighted model counting. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS'06)*, 243–252.

Domshlak, C. 2013. Fault tolerant planning: Complexity and compilation. In *Proceedings of the 23th International Conference on Automated Planning and Scheduling (ICAPS'13)*, 64–72.

Felzenszwalb, P. F., and McAllester, D. 2007. The generalized A* architecture. *Journal of Artificial Intelligence Research* 29:153–190.

Geffner, H., and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers.

Geffner, T., and Geffner, H. 2018. Compact policies for fully-observable non-deterministic planning as SAT. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS'18)*, 88–96.

Green, D. T.; Walsh, T. J.; Cohen, P. R.; and Chang, Y.-H. 2011. Learning a skill-teaching curriculum with dynamic Bayes nets. In *Proceedings of the 23rd Conference on Innovative Applications of Artificial Intelligence (IAAI'11)*, 1648–1654.

Hansen, E. A., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129(1-2):35–62.

Hoffmann, J., and Brafman, R. 2005. Contingent planning via heuristic forward search with implicit belief states. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS'05)*, 71–88.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1-2):99–134.

Martelli, A., and Montanari, U. 1973. Additive AND/OR graphs. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence (IJCAI'73)*, 1–11.

Meneguzzi, F., and De Silva, L. 2015. Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning. *The Knowledge Engineering Review* 30(1):1–44.

Meuleau, N., and Smith, D. E. 2002. Optimal limited contingency planning. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI'02)*, 417–426.

Muise, C. J.; Belle, V.; and McIlraith, S. A. 2014. Computing contingent plans via fully observable non-deterministic planning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI'14)*, 2322–2329.

Nilsson, N. J. 1971. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill.

Pandey, B., and Rintanen, J. 2018. Planning for partial observability by SAT and graph constraints. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS'18)*, 190–198.

Rintanen, J. 2007. Asymptotically optimal encodings of conformant planning in QBF. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI'07)*, 1045–1050.

Russell, S. J., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition.

Striebel, C. 1965. Sufficient statistics in the optimum control of stochastic systems. *Journal of Mathematical Analysis and Applications* 12(3):576–592.

Taig, R., and Brafman, R. I. 2015. A compilation based approach to conformant probabilistic planning with stochastic actions. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, 220–224.

Tu, P. H.; Son, T. C.; and Baral, C. 2007. Reasoning and planning with sensing actions, incomplete information, and static causal laws using answer set programming. *Theory and Practice of Logic Programming* 7(4):377–450.

Weld, D. S.; Anderson, C. R.; and Smith, D. E. 1998. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI'98)*, 897–904.

Robust Goal Recognition with Operator-Counting Heuristics

Felipe Meneguzzi[1], André Grahl Pereira[2], and Ramon Fraga Pereira[1]
[1]Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil
[2]Federal University of Rio Grande do Sul (UFRGS), Brazil
felipe.meneguzzi@pucrs.br, ramon.pereira@edu.pucrs.br
agpereira@inf.ufrgs.br

Abstract

Goal recognition is the problem of inferring the correct goal towards which an agent executes a plan, given a set of goal hypotheses, a domain model, and a (possibly noisy) sample of the plan being executed. This is a key problem in both cooperative and competitive agent interactions and recent approaches have produced fast and accurate goal recognition algorithms. In this paper, we leverage advances in operator-counting heuristics computed using linear programs over constraints derived from classical planning problems to solve goal recognition problems. Our approach uses additional operator-counting constraints derived from the observations to efficiently infer the correct goal, and serves as basis for a number of further methods with additional constraints.

Introduction

Agents that act autonomously on behalf of a human user must choose goals independently of user input and generate plans to achieve such goals (Meneguzzi 2009). When such agents have complex sets goals and require interaction with multiple agents that are not under the user's control, the resulting plans are likely to be equally complex and non-obvious for human users to interpret (Chakraborti *et al.* 2018). In such environments, the ability to accurately and quickly identify the goals and plans of all involved agents is key to provide meaningful explanation for the observed behavior. Goal recognition is the problem of inferring one or more goals from a set of hypotheses that best account for a sequence of observations, given a fixed initial state, a goal state, and a behavior model of the agent under observation. Recent approaches to goal recognition based on classical planning domains have leveraged data-structures and heuristic information used to improve planner efficiency to develop increasingly accurate and faster goal recognition algorithms (Martín *et al.* 2015; Pereira *et al.* 2017). Specifically, Pereira *et al.* (2017) use heuristics based on planning landmarks (Hoffmann *et al.* 2004) to accurately and efficiently recognize goals in a wide range of domains with various degrees of observability and noise. This approach, however, does not deal with noise explicitly, relying on the implicit necessity of landmarks in valid plans for goal hypotheses to achieve competitive accuracy with other methods (Sohrabi *et al.* 2016; Ramírez and Geffner 2010), while increasing the number of recognized goals (spread).

Thus, goal recognition under partial observability (i.e., missing observations) in the presence of noisy observation is a difficult problem to address while achieving both reasonable recognition time (i.e., a few seconds), high accuracy and low spread. In this paper, we address these limitations by leveraging recent advances on operator-counting heuristics (Pommerening *et al.* 2014; van den Briel *et al.* 2007). Operator-counting heuristics provide a unifying framework for a variety of sources of information from planning heuristics (Hoffmann *et al.* 2004) that provide both an estimate of the total cost of a goal from any given state and and indication of the actual operators likely to be in such plans. This information proves to be effective at differentiating between goal hypotheses in goal recognition.

Our contributions are threefold. First, we develop three, increasingly more accurate goal recognition approaches using operator-counting heuristics.Second, we empirically show that these heuristics are very effective at goal recognition, overcoming existing approaches in almost all domains in terms of accuracy while diminishing the spread of recognized goals. Such approaches are substantially more effective for noisy settings. Third, we discuss a broad class of operator-counting heuristics for goal recognition that can use additional constraints to provide even finer handling of noise and missing observations.

Background

We review the key background for the approaches we develop in this paper. First, the recognition settings we assume for our approach follows the standard formalization of goal recognition as planningSecond, while there is substantial literature on linear programming heuristic unified on the operator-counting framework, we focus on the specific types of operator-counting constraints we actually use in our experimentation.

Planning and Goal Recognition

Planning aims to find a sequence of actions that transforms an initial state into a goal state. Next, we formally define each of these elements.

Definition 1 (**Predicates and State**). *A predicate is denoted by an n-ary predicate symbol* p *applied to a sequence of zero or more terms* (τ_1, τ_2, ..., τ_n) *– terms are either constants or variables. We refer to grounded predicates that represent logical values according to some interpretation as facts, which are divided into two types: positive and negated facts, as well as constants for truth* ($\top$) *and falsehood* ($\bot$)*. A state* S *is a finite set of positive facts* f *that follows the closed world assumption so that if* $f \in S$*, then* f *is true in* S*. We assume a simple inference relation* $\models$ *such that* $S \models f$ *iff* $f \in S$, $S \not\models f$ *iff* $f \notin S$*, and* $S \models f_1 \wedge ... \wedge f_n$ *iff* $\{f_1, ..., f_n\} \subseteq S$.

Definition 2 (**Operator and Action**). *An operator* a *is represented by a triple* $\langle$*name(a), pre(a), eff(a)*$\rangle$*: name(a) represents the description or signature of* a*; pre(a) describes the preconditions of* a*, a set of predicates that must exist in the current state for* a *to be executed; eff(a) represents the effects of* a*. These effects are divided into eff(a)*$^+$ *(i.e., an add-list of positive predicates) and eff(a)*$^-$ *(i.e., a delete-list of negated predicates). An action is a ground operator instantiated over its free variables.*

Definition 3 (**Planning Domain**). *A planning domain definition* Ξ *is represented by a pair* $\langle \Sigma, \mathcal{A} \rangle$*, which specifies the knowledge of the domain, and consists of a finite set of facts* Σ *(e.g., environment properties) and a finite set of actions* $\mathcal{A}$.

Definition 4 (**Planning Instance**). *A planning instance* Π *is represented by a triple* $\langle \Xi, \mathcal{I}, G \rangle$*, where* $\Xi = \langle \Sigma, \mathcal{A} \rangle$ *is the domain definition;* $\mathcal{I} \subseteq \Sigma$ *is the initial state specification, which is defined by specifying values for all facts in the initial state; and* $G \subseteq \Sigma$ *is the goal state specification, which represents a desired state to be achieved.*

Definition 5 (**Plan**). *An s-plan* π *for a planning instance* $\Pi = \langle \Xi, \mathcal{I}, G \rangle$ *is a sequence of actions* $\langle a_1, a_2, ..., a_n \rangle$ *that modifies a state* s *into a state* $S \models G$ *in which the goal state* G *holds by the successive execution of actions in* π *starting from* s*. An* $\mathcal{I}$*-plan is just called a plan. A plan* π^* *with length* $|\pi^*|$ *is optimal if there exists no other plan* π' *for* Π *such that* $|\pi'| < |\pi^*|$.

A goal recognition problem aims to select the correct goal of an agent among a set of possible goals using as evidence a sequence of observations. These observations might be actions executed by the agent or noise observation which are part of a valid plan but are not executed by the agent.

Definition 6 (**Observation Sequence**). *An observation sequence* $O = \langle o_1, o_2, ..., o_n \rangle$ *is said to be satisfied by a plan* $\pi = \langle a_1, a_2, ..., a_m \rangle$*, if there is a monotonic function* f *that maps the observation indices* $j = 1, ..., n$ *into action indices* $i = 1, ..., n$*, such that* $a_{f(j)} = o_j$.

Definition 7 (**Goal Recognition Problem**). *A goal recognition problem is a tuple* $T_{GR} = \langle \Xi, \mathcal{I}, \mathcal{G}, O \rangle$*, where:* $\Xi = \langle \Sigma, \mathcal{A} \rangle$ *is a planning domain definition;* $\mathcal{I}$ *is the initial state;* $\mathcal{G}$ *is the set of possible goals, which include a correct hidden goal* G^* *(i.e.,* $G^* \in \mathcal{G}$*); and* $O = \langle o_1, o_2, ..., o_n \rangle$ *is an observation sequence of executed actions, with each observation* $o_i \in \mathcal{A}$*, and the corresponding action being part of a valid plan* π *(from Definition 5) that transitions* $\mathcal{I}$ *into* G^* *through the sequential execution of actions in* π.

Definition 8 (**Solution to a Goal Recognition Problem**). *A solution to a goal recognition problem* $T_{GR} = \langle \Xi, \mathcal{I}, \mathcal{G}, O \rangle$ *is a nonempty subset of the set of possible goals* $\mathbf{G} \subseteq \mathcal{G}$ *such that* $\forall G \in \mathbf{G}$ *there exists a plan* π_G *generated from a planning instance* $\langle \Xi, \mathcal{I}, G \rangle$ *and* π_G *is consistent with* O.

Operator-Counting Heuristics

Recent work on linear programming (LP) based heuristics has generated a number of informative and efficient heuristics for optimal-cost planning (van den Briel *et al.* 2007; Pommerening *et al.* 2014; Bonet 2013). These heuristics rely on constraints from different sources of information that every plan π (Definition 5) must satisfy. All operator-counting constraints contain variables of the form Y_a for each operator a such that setting Y_a to the number of occurrences of a in π satisfies the constraints. In this paper we adopt the formalism and definitions of Pommerening *et al.* for LP-based heuristics[1].

Definition 9 (**Operator-Counting Constraints**). *Let* Π *be a planning instance with operator set* $\mathcal{A}$ *and let* s *be one of tis states. Let* $\mathcal{Y}$ *be a set of non-negative real-valued and integer variables, including an integer variable* Y_a *for each operator* $a \in \mathcal{A}$ *along with any number of additional variables. The variables* Y_a *are called operator-counting variables. If* π *is an s-plan, we denote the number of occurrences of operator* $a \in \mathcal{A}$ *in* π *with* Y_a^π*. A set of linear inequalities over* $\mathcal{Y}$ *is called an operator counting constraint for* s *if for every* s*-plan there exists a feasible solution with* $\mathsf{Y}_a = \mathsf{Y}_a^\pi$ *for all* $a \in \mathcal{A}$*. A constraint set for* s *is a set of operator-counting constraints for* s *where the only common variables between constraints are the operator-counting constraints.*

Definition 10 (**Operator-Counting Integer-Linear Program**). *The operator-counting integer program* IP_C *for constraint set* C *aims to minimise*

$$\sum_{a \in \mathcal{A}} \mathit{cost}(a) \mathsf{Y}_a \text{ subject to } C$$

where operator-counting linear program LP_C *is the LP-relaxation of* IP_C.

Definition 11 (**IP and LP Heuristic**). *Let* Π *be a planning instance, and let* $\mathcal{C}$ *be a function that maps states* s *of* Π *to constraint sets for* s*. The* IP *heuristic* $h_{\mathcal{C}}^{IP}(s)$ *is the objective value of the integer program* $IP_{\mathcal{C}(s)}$*. The* LP *heuristic* $h_{\mathcal{C}}^{LP}(s)$ *is the objective value of the linear program* $LP_{\mathcal{C}(s)}$*. Infeasible IPs/LPs are treated as having* ∞ *objective value.*

While the framework from Pommerening *et al.* 2013 unifies many types of constraints for operator-counting heuristics, we rely on three types of constraints for our goal recognition approaches: landmarks, state-equations, and post-hoc optimization. Planning landmarks consist of actions (alternatively state-formulas) that must be executed (alternatively

[1]The only difference between their formalism and ours is that we refer to operators/actions with an $a/\mathcal{A}$ variable name to differentiate it from the observations $o/\mathcal{O}$

made true) in any valid plan for a particular goal (Hoffmann *et al.* 2004). Thus, landmarks are necessary conditions for all valid plans towards a goal, and, as such, provide the basis for a number of admissible heuristics (Karpas and Domshlak 2009) and as conditions to strengthen existing heuristics (Bonet 2013). Importantly, planning landmarks form the basis for the current state-of-the-art goal recognition algorithms (Pereira *et al.* 2017; Pereira and Meneguzzi 2018). *Disjunctive action landmarks* (Zhu and Givan 2003) for a state s are sets of actions such that at least one action in the set must be true for any s-plan, and make for a natural operator-counting constraint.

Definition 12 (**Landmark Constraints**). *Let $L \subseteq \mathcal{A}$ be a disjunctive action landmark for state s of task Π. The landmark constraint $c^{\mathrm{lm}}_{s,L}$ for L is:*

$$\sum_{a \in L} \mathsf{Y}_a \geq 1$$

Net change constraints generalize Bonet's (2013) state equation heuristic, which itself relate the planning instance in question with Petri nets that represent the transitions of state variables induced by the actions, and such that tokens in this task represent net changes to the states of the problem.

Finally, Post-hoc optimization constraints (Pommerening *et al.* 2013) use the fact that certain heuristics can rule out the necessity of certain operators from plans (and thus from the heuristic estimate). For example, Pattern Database (PDBs) heuristics (Culberson and Schaeffer 1998) create projections of the planning task into a subset of state variables (with this subset being the *pattern*), such that the heuristic can partition operators into two sets of each pattern, one that changes variables in the pattern (i.e., contributes towards transitions) and the other than does not (i.e., is non-contributing).

Definition 13 (**Post-hoc Optimization Constraint**). *Let Π be a planning task with operator set $\mathcal{A}$, let h be an admissible heuristic for Π, and let $N \subseteq \mathcal{A}$ be a set of operators that are noncontributing in that h is still admissible in a modified planning task where the cost of all operators in N is set to* 0.

Then the post-hoc optimization constraint $c^{PH}_{s,h,N}$ for h, N, and state s of Π consists of the inequality.

$$\sum_{a \in \mathcal{A} \setminus N} cost(a) \mathsf{Y}_a \geq h(s)$$

Goal Recognition using Operator Counts

We now bring together the operator-counting constraints into three operator-counting heuristics suitable for goal recognition, ranging from the simplest way to employ operator counts to compute the overlap between counts and observed actions, to modifying the constraints used by the operator counts to enforce solutions that agree with such observations,and finally accounting for possible noise by comparing heuristic values.

Computing Observation Overlap Count

We start with a basic operator-counting heuristic $h(s)$, which we define to be the LP-heuristic of Def. 11 where C comprises the constraints generated by Landmarks (Def. 12), post-hoc optimization (Def. 13), and net change constraints as described by Pommerening *et al.* (2014). This heuristic, computed following Def. 11, yields two important bits of information for our first technique, first, it generates the actual operator counts Y_a for all $a \in \mathcal{A}$ from Def. 10, whose minimization comprises the objective function $h(s)$.

Algorithm 1 Goal Recognition using the Operator Counts.

Input: Ξ *planning domain definition*, $\mathcal{I}$ *initial state*, $\mathcal{G}$ *set of candidate goals*, and O *observations*.
Output: *Recognized goal(s).*

```
 1: function OpCountRecognize($\Xi, \mathcal{I}, \mathcal{G}, O$)
 2:     $Hits$ ← Initialize empty dictionary
 3:     for all $G \in \mathcal{G}$ do                    ▷ Compute overlap for $G$
 4:         $Hits_G \leftarrow 0$
 5:         $\mathcal{Y} \leftarrow$ GenerateConstraints($\Xi, \mathcal{I}, G$)
 6:         $\mathsf{Y} \leftarrow$ ComputeOperatorCounts($\mathcal{Y}$)
 7:         for all $o \in O$ do
 8:             if $\mathsf{Y}_o > 0$ then
 9:                 $Hits_G \leftarrow Hits_G + 1$
10:                 $\mathsf{Y}_o \leftarrow \mathsf{Y}_o - 1$
11:     return all $G$ s.t $G \in \mathcal{G} \land Hits_G = \max_G Hits_G$
```

The heuristic values $h(s)$ of each goal candidate $G \in \mathcal{G}$ tells us about the optimal distance between the initial state $\mathcal{I}$ and G, while the operator counts indicate *possible* operators in a valid plan from $\mathcal{I}$ to G. We can use these counts to account for the observations $\mathcal{O}$ by computing the *overlap* between operators with counts greater than one and operators observed for recognition. Algorithm 1 shows how we can use the operator counts directly in a goal recognition technique. In order to rank the goal hypotheses we keep a dictionary of $Hits$ (Line 2) to store the overlap, or count the times operators counts hit observed actions. The algorithm then iterates over all goal hypotheses (Lines 3-10) computing the operator counts for each hypothesis G and comparing these counts with the actual observations (Lines 7–10). We recognize goals by choosing the hypotheses whose operator counts hit the most observations (Line 11).

Enforcing Observation Constraints

The technique of Algorithm 1 is conceptually similar to the *Goal Completion* heuristic of Pereira *et al.* (2017) in that it tries to compare heuristically computed information with the observations. However, this initial approach has a number of shortcomings in relation to their technique. First, while the landmarks themselves are enforced by the LP used to compute the operator counts (and thus observations that correspond to landmarks count as hits), the overlap computation loses the ordering of the landmarks that the Goal Completion heuristic uses to account for missing observations. Second, a solution to a set of operator-constraints, i.e., a set of operators with non-negative counts may not be a feasible plan for a planning instance. Thus, these counts may not correspond to the plan that generated the observations.

While operator-counting heuristics on their own are fast and informative enough to help guide search when dealing with millions of nodes, goal recognition problems often re-

Algorithm 2 Goal Recognition using Observation-Constrained Operator Counts.

```
Input: $\Xi$ planning domain definition, $\mathcal{I}$ initial state, $\mathcal{G}$ set of candidate goals, and $O$ observations.
Output: Recognized goal(s).
1: function OpCountObsRecognize($\Xi, \mathcal{I}, \mathcal{G}, O$)
2:     for all $G \in \mathcal{G}$ do                ▷ Compute $h_c(\mathcal{I})$ for $G$
3:         $\mathcal{Y} \leftarrow$ GenerateConstraints($\Xi, \mathcal{I}, G$)
4:         for all $o \in O$ do
5:             $\mathcal{Y} \leftarrow \mathcal{Y} \cup (\mathsf{Y}_o > 1)$
6:         $\mathsf{Y} \leftarrow$ ComputeOperatorCounts($\mathcal{Y}$)
7:         $H_G \leftarrow \sum_{a \in \mathcal{A}} \mathsf{Y}_a$
8:     return all $G$ s.t $G \in \mathcal{G} \wedge H_G = \min_G H_G$
```

quire the disambiguation of a dozen or less goal hypotheses. Such goal hypotheses are often very similar so that the operator-counting heuristic value (i.e., the objective function over the operator counts) for each goal hypothesis is very similar, especially if the goals are more or less equidistant from the initial state.

Thus, we refine the technique of Observation Overlap by introducing additional constraints into the LP used to compute operator counts. Specifically, we force the operator counting heuristic to *only consider* operator counts that include every single observation $o \in \mathcal{O}$. The resulting LP heuristic (which we call h_C) then minimizes the cost of the operator counts for plans that necessarily agree with all observations. We summarize this *Observation Constraint Enforcement* approach in Algorithm 2. This technique is similar to that of Algorithm 1 in that it iterates over all goals computing a heuristic value. However, instead of computing observation hits by looking at individual counts, it generates the constraints for the operator-counting heuristic (Line 3) and adds constraints to ensure that the count of the operators corresponding to each observation is greater than one (Lines 4–5). Finally, we choose the goal hypotheses that minimize the operator count heuristic distance from the initial state (Line 8).

Enforcement Delta

Although enforcing constraints to ensure that the LP heuristic computes only plans that do contain all observations helps us overcome the limitations of computing the overlap of the operator counts, this approach has a major shortcoming: it considers all observations as valid operators generated by the observed agent. Therefore, the heuristic resulting from the minimization of the LP might overestimate the actual length of the plan for the goal hypothesis due to noise. This may happen for one of two reasons: either the noise is simply a sub-optimal operator in a valid plan, or it is an operator that is completely unrelated to the plan that generated the observations. In both cases, the resulting heuristic value may prevent the algorithm from selecting the actual goal from among the goal hypotheses. This overestimation, however, has an important property in relation to the basic operator counting heuristic, which is that h_C always dominates the operator counting heuristic h, in Proposition 1.

Algorithm 3 Goal Recognition using Heuristic Difference of Operator Counts.

```
Input: $\Xi$ planning domain definition, $\mathcal{I}$ initial state, $\mathcal{G}$ set of candidate goals, and $O$ observations.
Output: Recognized goal(s).
1: function DeltaRecognize($\Xi, \mathcal{I}, \mathcal{G}, O$)
2:     for all $G \in \mathcal{G}$ do                ▷ Compute $h_\delta(\mathcal{I})$ for $G$
3:         $\mathcal{Y} \leftarrow$ GenerateConstraints($\Xi, \mathcal{I}, G$)
4:         $\mathsf{Y} \leftarrow$ ComputeOperatorCounts($\mathcal{Y}$)
5:         $H_G \leftarrow \sum_{a \in \mathcal{A}} \mathsf{Y}_a$
6:         for all $o \in O$ do
7:             $\mathcal{Y} \leftarrow \mathcal{Y} \cup \mathsf{Y}_o > 0$
8:         $\mathsf{Y} \leftarrow$ ComputeOperatorCounts($\mathcal{Y}$)
9:         $H_{C,G} \leftarrow \sum_{a \in \mathcal{A}} \mathsf{Y}_a$
10:        $H_{\delta,G} \leftarrow H_{C,G} - H_G$
11:    return all $G$ s.t $G \in \mathcal{G} \wedge H_{\delta,G} = \min_G H_{\delta,G}$
```

Proposition 1 (h_C **dominates** h). *Let h be the operator-counting heuristic from Defs. 10-11, h_C be the over-constrained heuristic that accounts for all observations $o \in \mathcal{O}$, and s a state of* Π. *Then $h_C(s) \geq h(s)$.*

Proof. Let C^h be set of constraints used in $h(s)$, and C^{h_C} be set of constraint used to compute $h_C(s)$. Every feasible solution to C_{h_C} is a solution to C_h. This is because to generate C_{h_C} we only *add* constraints to C_h. Thus, a solution to C^{h_C} has to satisfy all constraints in C^h. Therefore, since we are solving a minimization problem the value of the solution for C^h cannot be larger than the solution to C^{h_C}. □

The intuition here is that the operator-counting heuristic h estimates the total cost of any optimal plan, regardless of the observations, while h_C estimates a plan following all observations, including noise, if any. If there is no noise, the sum of the counts must agree (even if the counts are different), whereas if there is noise and assuming the noise is evenly distributed, there will be differences in all counts. Thus, our last approach consists of computing the difference between h_C and h, and infer that the goal hypothesis for which these values are closer must be the correct goal. We call the resulting heuristic h_δ and formalize this approach in Algorithm 3. Here we compute the LP twice, once with only the basic operator-counting constraints (Line 4), and once with the constraints enforcing the observations in the operator counts (Line 8), using these two values to compute h_δ (Line 10). The algorithm then returns goal hypotheses that minimize h_δ (Line 11).

Experiments and Results

To evaluate the effectiveness of our approaches, we implemented each of the algorithms described earlier and performed the goal recognition process over the large dataset introduced by Pereira *et al.* (2017). This dataset contains thousands of problems for goal and plan recognition under varying levels of observability for a number of traditional IPC domains (Vallati *et al.* 2018), including BLOCKS-WORLD, CAMPUS, DEPOTS, DRIVERLOG, Dockworker robots (DWR), IPC-GRID, FERRY, Intrusion Detection

(INTRUSION), KITCHEN, LOGISTICS, MICONIC, ROVER, SATELLITE, SOKOBAN, and Zeno Travel (ZENO). It also contains over a thousand problems under partial observability *and* noisy observations in the CAMPUS, IPC-GRID, INTRUSION and KITCHEN domains. The baselines of our experimentation were the original deterministic approach from Ramírez and Geffner (2009) (R&G 2009) and the recent algorithms from Pereira *et al.* (2017) (h_{uniq}) and Martín *et al.* (2015) (FG2015)[2]. We implemented our approaches using PYTHON 2.7 for the main recognition algorithms with external calls to a customized version of the FAST-DOWNWARD (Helmert 2006) planning system to compute the operator counts. Our customized planner returns not only the operator counts and can also introduce additional constraints before running the CPLEX 128 optimization system. We ran experiments in a single core of a 24 core Intel® Xeon® CPU E5-2620 @2.00Ghz with 48GB of RAM, with a 2-minute time limit and a 2GB memory limit.

Table 1 shows the results for the partially observable, non-noisy fragment of the dataset, whereas Table 2 shows the noisy fragment of the dataset[3]. For the noisy experiments, each set of observations contained at least two spurious actions, which, while valid for the plan, were not actually executed by the agent being observed. These results show that, while not nearly as fast as the h_{uniq} approach from Pereira *et al.* with a $\theta = 0$ recognition threshold, the accuracy (Acc %) of our h_δ approach is either competitive or superior in virtually all domains (except for some levels of observability in IPC-GRID, DWR and KITCHEN), and, even for the domains where the accuracy is similar, or lower, the spread ($Sin\mathcal{G}$) of the resulting goals is consistently lower, i.e., the returned goals are unique for most problems. The accuracy of our approach, thus, consistently matches or surpasses that of R&G 2009, with a computational cost that is also often smaller than FG 2015. Importantly, the cost of all of our approaches is basically the same within each domain, regardless of the level of observability and noise, since our technique relies on a single call to a planner that computes the operator counts for a single state and then stops the planner. We argue that this is attributable to our inefficient implementation rather than the technique, for the h_δ approach, the overhead of the FAST-DOWNWARD pre-processing step is paid multiple times. Unlike R&G 2009, that uses a modified planning heuristic, and FG 2015, that builds a data structure and explores it at very high computational cost. We note that the results for noisy observations show the greatest impact of h_δ with an overall higher accuracy and lower spread across all domains but KITCHEN.

Finally, results for the KITCHEN domain stand out in our experiments in that our some of our approaches consistently show underwhelming performance both in noisy and non-noisy domains. Counter-intuitively, for this particular domain, the more observations we have available, the worse the performance. This seems to be a problem for all other approaches under noisy conditions, though not under incomplete observations. Moreover, since the loss of accuracy with fuller observability also occurs for the non-noisy setting, we surmise this to stem from the domain itself, rather than the algorithm's ability to handle noise, and defer investigation of this issue to future work.

Related Work

Our work follows the traditional of goal and plan recognition as planning algorithms as defined by Ramírez and Geffner (2009; 2010). The former work yields higher recognition accuracy in our settings (and hence we chose it as a baseline), whereas the latter models goal recognition as a problem of estimating the probability of a goal given the observations. Such work uses a Bayesian framework to compute the probability of goals given observations by computing the probability of generating a plan given a goal, which they accomplish by running a planner multiple times to estimate the probability of the plans that either comply or not with the observations. Recent research on goal recognition has yielded a number of approaches to deal with partial observability and noisy observations, of which we single out three key contributions. First, Martín *et al.* (2015) developed a goal recognition approach based on constructing a planning graph and propagating operator costs and the interaction among operators to provide an estimate of the probabilities of each goal hypothesis. While their approach provides probabilistic estimates for each goal, its precision in inferring the topmost goals is consistently lower than ours, often ranking multiple goals with equal probabilities (i.e., having a large spread). Second, Sohrabi *et al.* (2016) developed an approach that also provides a probabilistic interpretation and explicitly deals with noisy observations. Their approach works through a compilation of the recognition problem into a planning problem that is processed by a planner that computes a number of approximately optimal plans to compute goal probabilities under R&G's Bayesian framework. Finally, Pereira *et al.* (2017) develop heuristic goal recognition approaches using landmark information. This approach is conceptually closer to ours in that we also compute heuristics, but we aim to overcome the potential sparsity of landmarks in each domain by using operator-count information, as well as explicitly handle noise by introducing additional constraints in heuristic h_C and comparing the distance to the unconstrained h heuristic.

Conclusion and Discussion

We developed a novel class goal recognition technique based on operator-counting heuristics from classical planning (Pommerening *et al.* 2014) which, themselves rely on ILP constraints to estimate which operators occur in valid optimal plans towards a goal. The resulting approaches are competitive with the state of the art in terms of high accuracy and low false positive rate (i.e., the spread of returned goals), at a moderate computational cost. We show empirically that the overall accuracy of our best approach is sub-

[2]We excluded the results of (Sohrabi *et al.* 2016) from our comparison as it timed out for virtually all problems in all domains, even with a 20-minute timeout.

[3]*Timeout* indicates that approach exceeded the two-minute timeout we set for the experiments, whereas the † symbol indicates a runtime failure for most problems in the domain.

				h			h_C			h_δ			R&G 2009			FG 2015			h_{uniq}		
#	$\|\mathcal{G}\|$	% Obs	$\|O\|$	**Time**	**Acc %**	S **in** $\mathcal{G}$	**Time**	**Acc %**	S **in** $\mathcal{G}$	**Time**	**Acc %**	S **in** $\mathcal{G}$	**Time**	**Acc %**	S **in** $\mathcal{G}$	**Time**	**Acc %**	S **in** $\mathcal{G}$	**Time**	**Acc %**	S **in** $\mathcal{G}$
Blocks (1076)	20.0	10	1.8	8.896	23.2%	2.17	8.916	45.1%	2.6	17.812	95.1%	7.74	1.235	86.8%	7.84	36.562	65.8%	9.11	0.131	31.6%	1.03
		30	4.9	8.877	16.3%	1.54	8.897	67.9%	2.02	17.774	87.8%	2.71	1.698	87.2%	3.56	36.648	78.1%	10.53	0.144	51.4%	1.06
		50	7.6	8.874	14.2%	1.37	8.875	79.3%	1.55	17.749	91.5%	1.74	2.497	97.9%	2.63	34.290	81.3%	10.68	0.168	60.1%	1.08
		70	11.1	8.382	11.0%	1.32	8.4	93.9%	1.21	16.782	98.4%	1.4	3.704	97.5%	1.83	37.056	89.8%	8.63	0.184	79.1%	1.13
		100	14.5	8.208	10.9%	1.12	8.227	100.0%	1.03	16.435	100.0%	1.21	6.123	100%	1.46	40.405	100.0%	1.22	0.238	100%	1.09
Campus (75)	2.0	10	1.0	0.631	53.3%	1.0	0.628	60.0%	1.07	1.259	100.0%	1.27	0.084	100%	1.46	0.717	53.3%	1.0	0.027	100%	1.13
		30	2.0	0.628	53.3%	1.0	0.631	73.3%	1.2	1.259	100.0%	1.07	0.097	100%	1.33	0.696	80.0%	1.13	0.042	100%	1.13
		50	3.0	0.634	40.0%	1.0	0.63	93.3%	1.27	1.264	100.0%	1.0	0.104	100%	1.33	0.676	66.6%	1.26	0.055	93.3%	1.13
		70	4.4	0.628	53.3%	1.0	0.629	100.0%	1.07	1.257	100.0%	1.07	0.115	100%	1.26	0.668	86.6%	1.6	0.058	100%	1.0
		100	5.5	0.624	60.0%	1.0	0.626	100.0%	1.0	1.25	100.0%	1.07	0.128	100%	1.13	0.631	93.3%	1.53	0.061	100%	1.0
Depots (364)	8.5	10	3.1	5.76	15.5%	1.29	5.715	32.1%	1.54	11.475	53.6%	1.83	1.485	77.3%	3.98	†	†	†	0.331	32.1%	1.09
		30	8.6	5.767	14.3%	1.31	5.693	69.0%	1.64	11.46	64.3%	1.19	2.307	77.3%	2.39	†	†	†	0.356	47.6%	1.07
		50	14.1	5.476	14.3%	1.21	5.438	91.7%	1.33	10.914	85.7%	1.1	3.433	84.5%	1.91	†	†	†	0.415	71.4%	1.02
		70	19.7	5.304	14.3%	1.2	5.252	100.0%	1.08	10.556	94.0%	1.01	5.149	91.6%	1.67	†	†	†	0.481	84.5%	1.01
		100	24.4	5.238	14.3%	1.21	5.205	100.0%	1.0	10.443	100.0%	1.0	7.094	92.8%	1.46	†	†	†	0.575	100%	1.03
Driverlog (364)	10.5	10	2.6	3.342	32.1%	1.43	3.316	42.9%	1.74	6.658	73.8%	2.43	1.192	96.4%	4.71	79.487	42.8%	1.91	0.284	35.7%	1.10
		30	6.9	3.337	28.6%	1.23	3.351	75.0%	1.45	6.688	77.4%	1.55	1.444	92.8%	3.34	60.168	70.2%	3.19	0.284	35.7%	1.10
		50	11.1	3.338	28.6%	1.13	3.35	92.9%	1.15	6.688	91.7%	1.17	1.608	94.1%	2.88	64.427	79.7%	4.59	0.290	64.2%	1.14
		70	15.6	3.304	28.6%	1.18	3.308	97.6%	1.12	6.612	95.2%	1.11	1.925	89.2%	2.46	75.084	82.1%	4.10	0.298	90.4%	1.14
		100	21.7	3.301	28.6%	1.04	3.347	100.0%	1.0	6.648	100.0%	1.04	2.809	89.2%	2.14	96.091	96.4%	1.11	0.305	100%	1.17
DWR (364)	7.3	10	5.7	3.604	38.1%	1.6	3.601	50.0%	1.71	7.205	56.0%	2.19	1.634	83.3%	4.21	66.496	92.8%	6.38	0.491	33.3%	1.05
		30	16.0	3.63	36.9%	1.2	3.579	81.0%	1.42	7.209	76.2%	1.46	2.977	80.9%	3.34	54.461	97.6%	6.56	0.518	51.1%	1.05
		50	26.2	3.611	36.9%	1.04	3.583	98.8%	1.2	7.194	84.5%	1.15	4.485	72.6%	2.27	56.255	98.8%	6.27	0.533	61.9%	1.04
		70	36.8	3.625	36.9%	1.04	3.569	100.0%	1.02	7.194	94.0%	1.04	10.432	70.2%	2.04	65.101	98.8%	6.0	0.540	78.5%	1.03
		100	51.9	3.581	35.7%	1.0	3.58	100.0%	1.0	7.161	100.0%	1.0	25.091	67.8%	1.67	86.459	100.0%	1.0	0.559	100%	1.01
IPC-Grid (673)	9.0	10	2.9	3.811	9.8%	1.0	3.828	18.9%	1.01	7.639	90.8%	1.88	1.084	96.1%	2.45	*Timeout*	-	-	0.220	62.7%	2.34
		30	7.8	3.871	9.2%	1.0	3.867	49.7%	1.2	7.738	94.1%	1.25	1.475	97.3%	1.42	*Timeout*	-	-	0.234	83.6%	1.66
		50	12.7	3.821	9.8%	1.0	3.82	79.1%	1.03	7.641	96.7%	1.07	1.932	100%	1.15	*Timeout*	-	-	0.245	90.1%	1.18
		70	17.9	3.902	9.2%	1.0	3.878	96.1%	1.03	7.78	94.1%	1.05	2.556	100%	1.05	*Timeout*	-	-	0.253	97.3%	1.11
		100	24.8	3.62	9.8%	1.0	3.637	98.4%	1.0	7.257	96.7%	1.0	3.868	100%	1.0	*Timeout*	-	-	0.261	100%	1.0
Ferry (364)	7.5	10	2.9	2.683	39.3%	1.65	2.686	72.6%	2.05	5.369	100.0%	3.17	0.511	98.8%	3.36	6.659	91.6%	6.65	0.068	58.3%	1.17
		30	7.6	2.693	39.3%	1.31	2.686	94.0%	1.48	5.379	100.0%	1.56	0.677	100%	1.76	6.801	100.0%	7.57	0.073	83.3%	1.05
		50	12.3	2.673	39.3%	1.17	2.671	97.6%	1.2	5.344	100.0%	1.29	0.794	100%	1.41	8.296	100.0%	7.57	0.084	91.6%	1.01
		70	17.3	2.661	39.3%	1.12	2.673	100.0%	1.08	5.334	100.0%	1.1	1.202	98.8%	1.14	10.649	100.0%	7.32	0.092	100%	1.0
		100	24.2	2.695	39.3%	1.11	2.708	100.0%	1.07	5.403	100.0%	1.07	1.693	100%	1.07	13.625	100.0%	1.07	0.099	100%	1.0
Intrusion (465)	15.0	10	1.9	4.701	10.5%	1.25	4.713	27.6%	1.81	9.414	100.0%	2.52	0.724	100%	2.53	0.475	89.5%	3.18	0.077	64.7%	1.23
		30	4.5	4.511	9.5%	1.12	4.518	80.0%	1.4	9.029	100.0%	1.11	0.804	100%	1.11	0.476	90.5%	1.88	0.083	85.7%	1.02
		50	6.7	4.421	9.5%	1.09	4.424	94.3%	1.12	8.845	100.0%	1.02	0.888	100%	1.02	0.496	94.3%	1.45	0.089	94.2%	1.04
		70	9.5	4.458	10.5%	1.09	4.453	97.1%	1.13	8.911	100.0%	1.0	1.012	100%	1.0	0.637	99.1%	1.05	0.093	94.2%	1.0
		100	13.1	4.419	8.9%	1.13	4.413	100.0%	1.0	8.832	100.0%	1.0	1.257	100%	1.0	0.828	100.0%	1.04	0.098	100%	1.0
Kitchen (75)	2.0	10	1.3	0.801	53.3%	1.0	0.807	53.3%	1.0	1.608	100.0%	1.87	0.085	100%	1.86	0.373	100.0%	1.86	0.002	100%	1.33
		30	3.5	0.789	26.7%	1.0	0.785	33.3%	1.07	1.574	100.0%	1.33	0.097	100%	1.33	0.360	100.0%	1.33	0.003	100%	1.33
		50	4.0	0.792	46.7%	1.0	0.802	53.3%	1.07	1.594	93.3%	1.33	0.104	100%	1.46	0.392	100.0%	1.33	0.006	100%	1.33
		70	5.0	0.795	46.7%	1.0	0.787	66.7%	1.13	1.582	80.0%	1.0	0.115	100%	1.26	0.378	100.0%	1.20	0.006	100%	1.46
		100	7.4	0.805	46.7%	1.0	0.812	73.3%	1.27	1.617	60.0%	1.0	0.119	100%	1.26	0.483	100.0%	1.40	0.007	100%	1.0
Logistics (673)	10.5	10	2.9	3.668	28.1%	1.47	3.658	54.9%	1.66	7.326	90.2%	2.27	1.201	99.3%	2.98	†	†	†	0.563	55.5%	1.24
		30	8.2	3.416	27.5%	1.07	3.416	75.8%	1.08	6.832	90.2%	1.2	1.798	98.6%	1.39	†	†	†	0.571	76.4%	1.20
		50	13.4	3.417	28.1%	1.01	3.409	91.5%	1.05	6.826	90.8%	1.03	2.545	98.6%	1.29	†	†	†	0.599	86.2%	1.10
		70	18.9	3.799	28.1%	0.96	3.8	91.5%	0.95	7.599	92.2%	0.99	3.460	100%	1.13	†	†	†	0.608	96.7%	1.05
		100	26.5	3.786	31.1%	0.97	3.77	93.4%	0.93	7.556	93.4%	0.93	4.887	100%	1.0	†	†	†	0.615	100%	1.0
Miconic (364)	6.0	10	3.9	2.617	39.3%	1.32	2.616	69.0%	1.4	5.233	100.0%	2.12	0.838	100%	3.26	†	†	†	0.321	54.7%	1.26
		30	11.1	2.612	39.3%	1.14	2.614	95.2%	1.24	5.226	100.0%	1.19	1.196	100%	1.58	†	†	†	0.326	90.1%	1.08
		50	18.1	2.614	39.3%	1.13	2.61	100.0%	1.1	5.224	100.0%	1.1	1.722	100%	1.28	†	†	†	0.339	96.4%	1.01
		70	25.3	3.941	39.3%	1.06	3.941	100.0%	1.0	7.882	100.0%	1.01	2.504	100%	1.03	†	†	†	0.344	100%	1.0
		100	35.6	4.116	39.3%	1.07	4.048	100.0%	1.0	8.164	100.0%	1.0	5.105	100%	1.0	†	†	†	0.356	100%	1.0
Rover (364)	6.0	10	3.0	3.993	52.4%	2.26	4.023	69.0%	1.58	8.016	92.9%	2.39	0.704	98.8%	2.85	†	†	†	0.310	51.1%	1.10
		30	7.9	3.952	48.8%	1.68	3.916	90.5%	1.27	7.868	84.5%	1.14	1.029	100%	1.66	†	†	†	0.323	69.1%	1.07
		50	12.7	3.79	50.0%	1.43	3.781	100.0%	1.13	7.571	97.6%	1.11	1.355	100%	1.29	†	†	†	0.331	85.7%	1.01
		70	17.9	3.763	52.4%	1.32	3.79	100.0%	1.02	7.553	97.6%	1.0	1.796	100%	1.07	†	†	†	0.345	91.6%	1.0
		100	24.9	3.772	53.6%	1.21	3.776	100.0%	1.0	7.548	100.0%	1.0	2.292	100%	1.07	†	†	†	0.356	100%	1.0
Satellite (364)	6.5	10	2.1	3.922	30.9%	1.67	3.902	64.3%	2.21	7.824	91.7%	2.7	1.049	97.6%	3.41	14.821	89.3%	4.86	0.431	47.6%	1.21
		30	5.4	3.928	28.6%	1.51	3.892	91.7%	1.73	7.82	91.7%	1.65	1.182	97.6%	2.40	32.172	86.9%	4.21	0.442	69.1%	1.14
		50	8.7	3.956	32.1%	1.29	3.928	95.2%	1.26	7.884	95.2%	1.27	1.398	97.6%	1.69	51.567	88.1%	3.65	0.458	80.9%	1.10
		70	12.2	3.904	32.1%	1.19	3.929	100.0%	1.08	7.833	96.4%	1.07	1.884	96.4%	1.52	75.363	92.8%	2.89	0.460	94.1%	1.03
		100	16.8	3.955	32.1%	1.14	3.903	100.0%	1.04	7.858	96.4%	1.04	2.107	96.4%	1.33	113.381	100.0%	2.57	0.475	100%	1.07
Sokoban (364)	7.3	10	3.1	5.883	22.6%	1.19	5.851	64.3%	1.27	11.734	67.9%	1.27	3.025	69.1%	4.02	461.701	67.8%	2.98	0.523	51.1%	1.85
		30	8.7	5.854	19.1%	1.08	5.73	89.3%	1.02	11.584	85.7%	1.06	4.429	89.2%	4.10	370.412	83.3%	3.14	0.531	55.9%	1.21
		50	14.1	5.911	21.4%	1.04	5.71	96.4%	1.02	11.621	90.5%	1.0	7.553	89.2%	4.16	358.028	82.1%	2.27	0.540	69.1%	1.20
		70	19.8	5.897	22.6%	1.08	5.653	100.0%	1.01	11.55	96.4%	1.01	9.112	89.2%	4.17	353.721	85.7%	1.84	0.554	86.9%	1.08
		100	35.5	5.849	25.0%	1.07	5.572	100.0%	1.0	11.421	100.0%	1.0	12.008	89.2%	4.53	353.183	85.7%	1.03	0.562	100%	1.0
Zeno (364)	7.5	10	2.6	5.474	34.5%	1.33	5.45	58.3%	1.68	10.924	82.1%	2.62	1.834	96.4%	3.41	93.917	66.6%	1.63	0.491	36.9%	1.04
		30	6.7	5.424	33.3%	1.24	5.449	86.9%	1.35	10.873	89.3%	1.57	2.528	88.1%	2.11	88.285	78.6%	2.27	0.504	60.7%	1.02
		50	10.8	5.005	33.3%	1.2	5.003	95.2%	1.1	10.008	91.7%	1.1	3.071	92.8%	1.41	105.814	91.6%	2.56	0.516	76.1%	1.0
		70	15.2	4.377	35.7%	1.17	4.321	100.0%	1.0	8.698	100.0%	1.0	3.986	96.4%	1.13	125.652	94.1%	2.58	0.522	90.4%	1.0
		100	21.1	4.378	35.7%	1.18	4.297	100.0%	1.0	8.675	100.0%	1.0	4.815	100%	1.07	168.674	100.0%	1.0	0.530	100%	1.0
Average				3.927	30.6%	1.202	3.908	82.7%	1.25	11.761	92.4%	1.440	2.697	94.7%	2.110	7.834	63.0%	3.464	0.311	79.7%	1.122

Table 1: Goal recognition experiments at various levels of observability.

				h			h_C			h_δ			R&G 2009			FG 2015			h_{uniq}		
#	$\|\mathcal{G}\|$	% Obs	$\|O\|$	**Time**	**Acc %**	S **in** $\mathcal{G}$	**Time**	**Acc %**	S **in** $\mathcal{G}$	**Time**	**Acc %**	S **in** $\mathcal{G}$	**Time**	**Acc %**	S **in** $\mathcal{G}$	**Time**	**Acc %**	S **in** $\mathcal{G}$	**Time**	**Acc %**	S **in** $\mathcal{G}$
Campus (516)	2	25	3.1	0.627	53.5%	1.0	0.629	83.0%	1.24	1.256	87.6%	1.12	0.073	88.3%	1.27	0.713	79.8%	1.33	0.030	82.1%	1.13
		50	4.5	0.634	53.5%	1.0	0.634	91.5%	1.19	1.268	92.3%	1.06	0.076	89.9%	1.26	0.666	90.6%	1.67	0.031	78.2%	1.02
		75	6.4	0.625	53.5%	1.0	0.625	95.3%	1.19	1.25	94.6%	1.09	0.079	90.6%	1.27	0.655	94.6%	1.79	0.034	73.6%	1.0
		100	7.5	0.619	53.5%	1.0	0.62	95.3%	1.22	1.239	94.6%	1.04	0.084	89.1%	1.22	0.644	97.7%	1.81	0.037	72.1%	1.0
IPC-Grid (300)	8.3	25	4.1	3.421	13.3%	1.0	3.406	30.0%	1.02	6.827	86.7%	1.49	0.537	71.1%	2.65	0.494	43.3%	2.31	0.102	30.0%	1.11
		50	7.6	3.392	13.3%	1.0	3.384	68.9%	1.1	6.776	96.7%	1.14	0.649	95.5%	1.28	0.511	81.1%	1.78	0.116	64.4%	1.03
		75	11.5	3.399	13.3%	1.0	3.392	98.9%	1.01	6.791	97.8%	1.07	0.712	100%	1.01	0.654	93.3%	1.10	0.124	87.7%	1.03
		100	16.9	3.403	13.3%	1.0	3.41	100.0%	1.0	6.813	100.0%	1.0	0.805	100%	1.0	0.885	100.0%	1.06	0.136	100%	1.0
Intrusion (300)	16.6	25	3.6	4.422	10.0%	1.24	4.433	26.7%	1.71	8.855	71.1%	2.7	0.462	12.2%	7.55	*Timeout*	-	-	0.208	53.3%	1.72
		50	6.7	4.457	10.0%	1.12	4.468	75.6%	1.41	8.925	96.7%	1.33	0.469	4.4%	8.06	*Timeout*	-	-	0.212	83.3%	1.33
		75	10.2	4.396	10.0%	1.07	4.394	90.0%	1.11	8.79	100.0%	1.01	0.475	6.6%	7.88	*Timeout*	-	-	0.224	94.4%	1.08
		100	15.1	4.451	10.0%	1.03	4.44	100.0%	1.0	8.891	100.0%	1.0	0.476	10.0%	7.76	*Timeout*	-	-	0.239	100%	1.0
Kitchen (150)	2.0	25	2.5	0.678	0.0%	0.0	0.812	46.7%	1.0	1.49	73.3%	1.69	0.139	71.1%	1.57	0.381	53.3%	1.33	0.081	88.8%	2.55
		50	4.8	0.681	0.0%	0.0	0.809	51.1%	1.04	1.49	55.6%	1.33	0.135	57.7%	1.42	0.410	51.1%	1.22	0.084	64.4%	1.71
		75	7.3	0.682	0.0%	0.0	0.819	48.9%	1.02	1.501	53.3%	1.33	0.138	57.7%	1.31	0.426	53.3%	1.20	0.090	57.7%	1.66
		100	11.0	0.683	0.0%	0.0	0.811	73.3%	1.27	1.494	53.3%	1.13	0.144	60.0%	1.46	0.538	73.3%	1.26	0.093	66.6%	1.13
Average				2.286	19.2%	0.779	2.318	73.5%	1.158	4.604	84.6%	1.283	0.341	62.8%	2.998	30.436	76.0%	1.488	0.115	74.8%	1.281

Table 2: Goal recognition experiments with noisy observations at various levels of observability.

stantially superior to the state-of-the-art over a large dataset. Importantly, the values of the operator-counting constraints we compute for each of the heuristics can be used as explanations for recognized goals.

The techniques described in this paper use a set of simple additional constraints in the ILP formulation to achieve substantial performance, so we expect substantial future work towards further goal recognition approaches and heuristics that explore more refined constraints to improve accuracy and reduce spread, as well as deriving a probabilistic approach using operator-counting information. Examples of such work include using the constraints to force the LP to generate the counterfactual operator-counts (i.e., non-compliant with the observations) used by the R&G approach, or, given an estimate of the noise, relax the observation constraints to allow a number of observations to not be included in the resulting operator-counts.

Acknowledgements: This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior – Brasil (CAPES) - Finance Code 001. Felipe acknowledges support from CNPq under project numbers 407058/2018-4 and 305969/2016-1, as well as FAPERGS process number 18/2551-0000500-2.

References

[Bonet 2013] B Bonet. An Admissible Heuristic for SAS+ Planning Obtained from the State Equation. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2013.

[Chakraborti *et al.* 2018] Tathagata Chakraborti, Anagha Kulkarni, Sarath Sreedharan, David E Smith, and Subbarao Kambhampati. Explicability? Legibility? Predictability? Transparency? Privacy? Security? The Emerging Landscape of Interpretable Agent Behavior. *arXiv.org*, November 2018.

[Culberson and Schaeffer 1998] Joseph C Culberson and Jonathan Schaeffer. Pattern Databases. *Computational Intelligence*, 14(3):318–334, August 1998.

[Helmert 2006] Malte Helmert. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[Hoffmann *et al.* 2004] J Hoffmann, J Porteous, and L Sebastia. Ordered Landmarks in Planning. *Journal of Artificial Intelligence Research*, 22(1):215–278, April 2004.

[Karpas and Domshlak 2009] Erez Karpas and Carmel Domshlak. Cost-Optimal Planning with Landmarks. *IJCAI*, 2009.

[Martín *et al.* 2015] Yolanda E Martín, María D R Moreno, and David E Smith. A Fast Goal Recognition Technique Based on Interaction Estimates. *IJCAI*, 2015.

[Meneguzzi 2009] Felipe Meneguzzi. *Extending agent languages for multiagent domains*. PhD thesis, King's College London, 2009.

[Pereira and Meneguzzi 2017] Ramon Fraga Pereira and Felipe Meneguzzi. Goal and plan recognition datasets using classical planning domains, July 2017.

[Pereira and Meneguzzi 2018] Ramon Pereira and Felipe Meneguzzi. Goal Recognition in Incomplete Domain Models. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 8127–8128, 2018.

[Pereira *et al.* 2017] Ramon Fraga Pereira, Nir Oren, and Felipe Meneguzzi. Landmark-Based Heuristics for Goal Recognition. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017.

[Pommerening *et al.* 2013] Florian Pommerening, Gabriele Röger, and Malte Helmert. Getting the Most Out of Pattern Databases for Classical Planning. *IJCAI*, 2013.

[Pommerening *et al.* 2014] Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet. LP-Based Heuristics for Cost-Optimal Planning. *ICAPS*, 2014.

[Ramírez and Geffner 2009] Miquel Ramírez and Hector Geffner. Plan recognition as planning. In *International Joint Conference on Artificial Intelligence*, pages 1778–1783, 2009.

[Ramírez and Geffner 2010] Miquel Ramírez and Hector Geffner. Probabilistic Plan Recognition Using Off-the-Shelf Classical Planners. In *AAAI*, pages 1121–1126, 2010.

[Sohrabi *et al.* 2016] S Sohrabi, A V Riabov, and O Udrea. Plan Recognition as Planning Revisited. In *International Joint Conference on Artificial Intelligence*, pages 3258–3264, 2016.

[Vallati *et al.* 2018] Mauro Vallati, Lukás Chrpa, and Thomas Leo McCluskey. What you always wanted to know about the deterministic part of the International Planning Competition (IPC) 2014 (but were too afraid to ask). *Knowledge Engineering Review*, 33:383, 2018.

[van den Briel *et al.* 2007] Menkes van den Briel, J Benton, Subbarao Kambhampati, and Thomas Vossen. An LP-Based Heuristic for Optimal Planning. *CP*, 4741(Chapter 46):651–665, 2007.

[Zhu and Givan 2003] Lin Zhu and Robert Givan. Landmark Extraction via Planning Graph Propagation. In *ICAPS 2003 Doctoral Consortium*, pages 156–160, 2003.

Towards Model-Based Contrastive Explanations for Explainable Planning

Benjamin Krarup[*], **Michael Cashmore**[*], **Daniele Magazzeni**[*], **Tim Miller**[†]

Abstract

An important type of question that arises in Explainable Planning is a contrastive question, of the form "Why action A instead of action B?". These kinds of questions can be answered with a *contrastive explanation* that compares properties of the original plan containing A against the contrastive plan containing B. An effective explanation of this type serves to highlight the differences between the decisions that have been made by the planner and what the user would expect, as well as to provide further insight into the model and the planning process. Producing this kind of explanation requires the generation of the contrastive plan. This paper introduces domain-independent compilations of user questions into constraints. These constraints are added to the planning model, so that a solution to the new model represents the contrastive plan. We introduce a formal description of the compilation from user question to constraints in a temporal and numeric PDDL2.1 planning setting.

1 Introduction

Explainable AI (XAI) is an emerging and important research area within AI. Recent work has shown that AI Planning is an important tool in XAI, as its decision-making mechanisms are model-based and so in principle more transparent. This recent work includes many approaches towards providing explanations in AI planning.

Chakraborti et al. (2019) gives an in-depth overview of this work and different terms used within the XAI landscape. In particular, Zhang et al. (2017) shows that if an AI system behaves "explicably" there is less of a need for explanations. However, this is not always possible and explanation is sometimes required. Chakraborti et al. (2017) tackles explanation as a model reconciliation problem, arguing that the explanation must be a difference between the human model and AI model. Seegebarth et al. (2012) show that by representing plans as first order logic formulae generating explanations is feasible in real time. In contrast, in this paper we focus on contrastive *"why"* questions. Fox, Long, and Magazzeni (2017) highlight some important questions in XAIP and discuss possible answers, and also describe how these *"why"* questions are especially important. Smith (2012) outlines the approach to planning as an iterative process for better modelling preferences and providing explanations. We propose to follow this same approach.

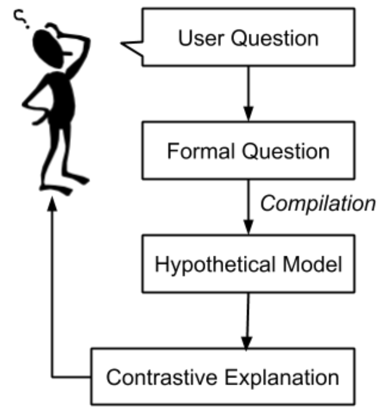


Figure 1: The four-stage process for generating a contrastive explanation from a user question. The hypothetical model is created by compiling the formal question into the planning model (in PDDL 2.1).

The aim of explanations is to improve the user's levels of understanding and trust in the system they are using. These explanations can be local (regarding a specific plan) or global (concerning how the planning system works in general). In this paper we focus on local explanations of temporal and numeric planning problems, introducing an approach for explaining why a planner has made a certain decision. Through active exploration of these specific cases, the user may also gain global insight into the way in which the planner makes decisions. (See (Lipton 1990; 2016; Ribeiro, Singh, and Guestrin 2016)).

To achieve an understanding of a decision, it is important that explanations adapt to the specific context and mental model of the user. One step towards this is to support the user iteratively asking different questions suitable for their context. Haynes et al. (2009) identify ten question types that a user might have about an intelligent system, also described by Mueller et al. (2019). Lim et al. (2009) show in a grounded study that of these, the questions *why* and *why not* provided the most benefit in terms of objective understanding and feelings of trust. In the context of planning *why not*

[*]King's College London, UK, {*firstname.lastname*}@*kcl.ac.uk*
[†]University of Melbourne, Australia, *tmiller@unimelb.edu.au*

questions are *contrastive questions*, because the user is asking why some action was selected rather than some other action that was not.

Instead, Miller argues that all such questions can be asked as contrastive questions of the form "Why action A rather than action B?" (Miller 2018). Contrastive questions capture the context of the question; they more precisely identify the gaps in the user's understanding of a plan that needs to be explained (Lewis 1986). A contrastive question about a plan can be answered by a *contrastive explanation*. Contrastive explanations will compare the original plan against a contrastive plan that accounts for the user expectation. Providing contrastive explanations is not only effective in improving understanding, but is simpler than providing a full causal analysis (Miller 2019).

Following the approach of Smith (2012) we propose an approach to contrastive explanations through a dialogue with the user. The proposed approach consists of an iterative four-stage process illustrated in Figure 1. First the user asks a contrastive question in natural language. Second, a constraint is derived from the user question, in the following we refer to this constraint as the *formal question*. Third a hypothetical model (HModel) is generated which encapsulates this constraint. A solution to this model is the hypothetical plan (HPlan) that can be compared to the original plan to show the consequence of the user suggestion. The user can compare plans and iterate the process by asking further questions, and refining the HModel. This allows the user to combine different compilations to create a more constrained HModel, producing more meaningful explanations, until the explanation is satisfactory. Each stage of this process represents a vital research challenge. This paper describes and formalises the third stage of this process: compiling the formal question into a hypothetical model for temporal and numeric planning.

We are interested in temporal and numeric planning problems, for which optimal solutions are difficult to find. Therefore, while the process described above serves for explanation, the insight of the user can also result in guiding the planning process to a more efficient solution. As noted by (Smith 2012), the explanations could also give the user the opportunity to improve the plan with respect to their own preferences. The user could have hidden preferences which have not been captured in the model. The user could ask questions which enforce constraints that favour these preferences. The new plan could be sub-optimal, but more preferable to the user.

The contribution of this paper is a formalisation of domain-independent and planner-agnostic compilations from formal contrastive questions to PDDL2.1 (Fox and Long 2003), necessary for providing contrastive explanations. The compilations shown are not exhaustive. However, they do cover an interesting set of questions which users would commonly have about both classical and temporal plans. The paper is organised as follows. The next section describes the planning definitions we will use throughout the paper. In Section 3 we describe the running example that we use to demonstrate our compilations throughout the paper. In Section 4 we list the set of formal questions that we are interested in, and formalise the compilations of each of these into constraints. Finally, we conclude the paper in Section 5 whilst touching on some interesting future work.

2 Background

Our definition of a planning model follows the definition of PDDL2.1 given by (Fox and Long 2003), extended by a set of time windows 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* $Prob = \langle Os, I, G, W \rangle$ *is a tuple where* Os *is the set of objects in the planning instance,* I *is the initial state,* G *is the goal condition, 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 $(block\_on\ ?a\ ?b)$ with arity 2 to the ordered set of objects $\{blockA, blockB\}$ forms the proposition $(block\_on\ blockA\ blockB)$. 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 $G = g_1, ..., g_n$ is a set of constraints over P and V that must hold at the end of an action sequence for a plan to be valid. More specifically, for an action sequence $\phi = \langle a_1, a_2, \ldots, a_n \rangle$ each with a respective time denoted by $Dispatch(a_i)$, we use the definition of plan validity from (Fox and Long 2003) (Definition 15 "Validity of a Simple Plan"). A simple plan is the sequence of actions ϕ which defines a happening sequence, $t_{i=0\ldots k}$ and a sequence of states, $s_{i=0\ldots k+1}$ such that $s_0 = I$ and for each $i = 0 \ldots k$, s_{i+1} is the result of executing the happening at time t_i. The simple plan ϕ is valid if $s_{k+1} \models G$.

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 effected at w_{ub}.

Similar to propositions and PNEs, the set of ground actions A is generated from the substitution of objects for operator parameters with respect to it's arity. 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*

```
(define (domain turtlebot_demo)
(:types waypoint robot)
(:predicates
  (robot_at ?v - robot ?wp - waypoint)
  (connected ?from ?to - waypoint)
  (visited ?wp - waypoint))
(: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))
    (over all (connected ?from ?to)))
  :effect (and
    (at start (not (robot_at ?v ?from)))
    (at end (visited ?to))
    (at end (robot_at ?v ?to)))))
```

Figure 2: The robotics domain used as a running example.

```
(define (problem task)
(:domain turtlebot_demo)
(:objects
    wp0 wp1 wp2 wp3 wp4 wp5 - waypoint
    kenny - robot)
(:init
    (robot_at kenny wp0) (visited wp0)
    (connected wp0 wp2) (connected wp0 wp4)
    (connected wp1 wp0) (connected wp1 wp2)
    (connected wp2 wp1) (connected wp2 wp4)
    (connected wp2 wp5) (connected wp3 wp5)
    (connected wp5 wp0) (connected wp5 wp2)
    (connected wp5 wp3)
    (= (travel_time wp0 wp2) 1.45)
    (= (travel_time wp0 wp4) 2)
    ...
(:goal (and (visited wp1) (visited wp2)
  (visited wp3)  (visited wp4) (visited wp5)
  )))
```

Figure 3: Example Problem with some travel time functions omitted for space.

pass between the start and end of a; a start (end) condition $Pre_{\vdash}(a)$ ($Pre_{\dashv}(a)$) 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 $Eff(a)^+_{\vdash}, Eff(a)^+_{\dashv} \subseteq P$ that are made true at the start and ends of the action respectively; delete effects $Eff(a)^-_{\vdash}, Eff(a)^-_{\dashv} \subseteq P$ that are made false at the start and end of the action respectively; and numeric effects $Eff(a)^n_{\vdash}$, $Eff(a)^n_{\leftrightarrow}$, $Eff(a)^n_{\dashv}$ that act upon some $n \in V$.

3 Running Example

We use as a running example the following planning model. Figure 2 shows the domain D. The domain describes a scenario in which a robot is able to move between connected waypoints and mark them as visited. The domain contains three predicate symbols ($robot\_at$, $connected$, $visited$) with arities 2, 2, and 1 respectively. The domain includes only a single function symbol $travel\_time$ with arity 2. There is a single operator $goto\_waypoint$.

Figure 3 shows the problem $Prob$. The problem specifies 7 objects: $wp0, wp1, wp2, wp3, wp4, wp5$ and $kenny$. The initial state specifies which propositions are initially true, such as the current location of the robot ($robot\_at\, kenny\, wp0$), and the initial values of the PNEs, e.g. $(= (travel\_time\, wp5\, wp3)\; 4.68)$. The goal is specified as a constraint over $P \cup V$, in this example it is that the robot has visited all of the locations.

Figure 5 shows an example plan that solves this problem. This plan might appear sub-optimal. The robot moves from waypoint $wp2$ to $wp1$ and then immediately returns to $wp2$. This second action might seem redundant to the user. However, upon closer inspection of the connectivity of waypoints (shown in Figure 4) we can see that the plan is in fact the optimal one. Visiting waypoint $wp1$ is a goal of the problem, and it is only connected to waypoints $wp0$ and $wp2$, both of which have already been visited. Waypoint $wp0$ is only connected to waypoints $wp2$ and $wp4$, $wp2$ has been visited and $wp4$ is a dead end. For these reasons combined, the only logical option is to move back to $wp2$ after completing the goal of visiting $wp1$. This type of behaviour similarly happens between waypoints $wp3$ and $wp5$.

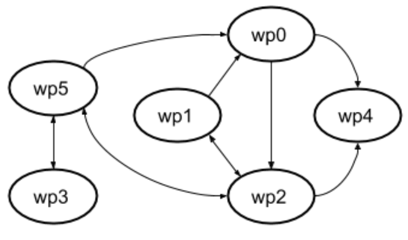


Figure 4: Waypoint connectivity in the running example. The robot is only allowed to move along the directed arrows.

```
0.00: (goto_waypoint kenny wp0 wp2)  [1.45]
1.45: (goto_waypoint kenny wp2 wp1)  [2.00]
3.45: (goto_waypoint kenny wp1 wp2)  [2.00]
5.45: (goto_waypoint kenny wp2 wp5)  [2.00]
7.45: (goto_waypoint kenny wp5 wp3)  [4.68]
12.13: (goto_waypoint kenny wp3 wp5)  [4.68]
16.81: (goto_waypoint kenny wp5 wp0)  [0.99]
17.80: (goto_waypoint kenny wp0 wp4)  [2.00]
```

Figure 5: Plan generated from the example domain and problem. The cost of the plan is its duration (19.80).

A graphical representation such as Figure 4 is not always available, and so even for this simple model and plan, deducing the reasoning behind the planned actions is not trivial. This is an example of where XAIP is useful. Using our proposed approach the user could have asked the question: "Why do we use the action ($goto\_waypoint\, kenny\, wp1\, wp2$), rather than not using

```
0.00: (goto_waypoint kenny wp0 wp2)  [1.45]
1.45: (goto_waypoint kenny wp2 wp5)  [2.00]
3.45: (goto_waypoint kenny wp5 wp3)  [4.68]
8.13: (goto_waypoint kenny wp3 wp5)  [4.68]
12.81: (goto_waypoint kenny wp5 wp2)  [2.00]
14.81: (goto_waypoint kenny wp2 wp1)  [2.00]
16.81: (goto_waypoint kenny wp1 wp0)  [2.00]
18.81: (goto_waypoint kenny wp0 wp4)  [2.00]
```

Figure 6: The hypothetical plan that accounts for the user's suggestion, avoiding the action of moving from $wp1$ to $wp2$. The cost of the plan is its duration (20.81).

it?". From this question we could generate a contrastive plan with this constraint enforced (shown in Figure 6). Comparing the actions and costs of the original and the new plan could shed light on why the action needed to be used. The user can carry on asking questions until they were satisfied.

4 Formalisation

Definition 3 *An* ***explanation problem*** *is a tuple* $E = \langle \Pi, \phi, Q \rangle$*, in which* Π *is a planning model (Definition 1),* ϕ *is the plan generated by the planner, and* Q *is the specific question posed by the user. The problem is to provide insight that helps the user to answer question* Q.

In this paper, we assume that the user knows the model Π and the plan ϕ, so answers such as stating the goal of the problem will not increase their understanding. Given this, we propose the following set of questions, and provide a formal description for compilations of this set of formal questions of temporal plans:

1. Why is action a used in state s, rather than action b? (Section 4.1)
2. Why is action a not used in the plan, rather than being used? (Section 4.2)
3. Why is action a used in the plan, rather than not being used? (Section 4.3)
4. Why is action a used outside of time window w, rather than only being allowed within w? (Section 4.4)
5. Why is action a not used in time window w, rather than being used within w? (Section 4.5)
6. Why is action a used at time t, rather than at least some time t' after/before t? (Section 4.6)
7. Why is action a not performed before (after) action b, rather than a being performed after (before) b? (Section 4.7)

These questions were derived by systematically assessing ways that counterfactual situations could occur in plans, and choosing those that would be useful over many applications. This is not an exhaustive list of possible constraints that can be enforced upon the original model, however, it does represent a list of questions that would be useful in specific contexts and applications.

Part of being able to answer these questions is the ability to reason about what would happen in the counterfactual cases. We approach this problem by generating plans for the counterfactual cases via *compilations*. A compilation of a planning instance where the model is given by Π, and a question is given by Q is shown as $Compilation(\Pi, Q) = \Pi'$ where:

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

We call Π' the *hypothetical model*, or HModel.

However, Π' can also be used as the input model so that the user can iteratively ask questions about some model, i.e:

$$Compilation(Compilation(\Pi, Q), Q')$$

This allows the user to stack questions, further increasing their understanding of the plan through combining compilations. Combining compilations this way provides a much wider set of possible constraints.

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 back to the original operators they represent. For each iteration of compilation the HPlan is validated against the original model Π.

4.1 Replacing an Action in a State

Given a plan ϕ, 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* $\chi \neq \chi'$

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

"Why is $(goto\_waypoint\, kenny\, wp2\, wp5)$ used, rather than $(goto\_waypoint\, kenny\, wp2\, wp4)$?"

They might ask this because a goal of the problem is to visit $wp4$. As the robot visits $wp5$ from $wp3$ later in the plan, it might make sense to the user for the robot to visit $wp4$ earlier, as $wp5$ will be visited at a later point.

To generate the HPlan, a compilation is formed such that the ground action $b = ground(n, \chi')$ appears in the plan in place of the action $a_i = ground(o, \chi)$. Given the example above $b = ground(goto\_waypoint, \{kenny, wp2, wp4\})$, and $a_i = ground(goto\_waypoint, \{kenny, wp2, wp5\})$. Given a plan:

$$\phi = \langle a_1, a_2, \ldots, a_n\rangle$$

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:

$$\phi' = \langle a'_1, a'_2, \ldots, a'_n\rangle$$

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

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

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 executing[1] $\langle a_1, a_2, \ldots, a_{i-1}, b\rangle$ from state 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: $w_x = \langle \mathit{Dispatch}(a_j) + \mathit{Dur}(a_j) - \mathit{Dispatch}(b), \mathit{inf}, u\rangle$ where $u = \mathit{Eff}(a_j)^-_\dashv \cup \mathit{Eff}(a_j)^+_\dashv \cup \mathit{Eff}(a_j)^n_\dashv$.

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 below:

```
(:init
(robot_at kenny wp4) (visited wp2)
(visited wp1) (visited wp4)
(connected wp0 wp2) (connected wp0 wp4)
(connected wp1 wp0) (connected wp1 wp2)
...)
(:goal (and (visited wp1)(visited wp2)
  (visited wp3)(visited wp4) (visited wp5)
  )))
```

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

```
0.00: (goto_waypoint kenny wp0 wp2)  [1.45]
1.45: (goto_waypoint kenny wp2 wp1)  [2.00]
3.45: (goto_waypoint kenny wp1 wp2)  [2.00]
5.45: (goto_waypoint kenny wp2 wp4)  [2.00]
```

In this state the robot has visited the waypoints $wp2$, $wp1$, and $wp4$, and is currently at $wp4$. This new initial state is then used to plan for the original goals to get the plan ϕ', which, along with b and ϕ, gives the HPlan. However, the problem is unsolvable from this state as there are no connections from $wp4$ to any other waypoint. By applying the user's constraint, and showing there are no more applicable actions, it answers the above question: "because by doing this 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 following constraints on adding and removing an action from the plan can be applied iteratively, as mentioned previously.

4.2 Add an Action to the Plan

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

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

[1]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'.

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

> "Why is $(\mathit{goto\_waypoint}\ \mathit{kenny}\ wp2\ wp4)$ not used, rather than being used?"

They might ask this because a goal of the problem is to visit $wp4$. As the robot is at $wp2$ early in the plan, and you can visit $wp4$ from $wp2$, it might make sense to the user for the robot to visit $wp4$ at that time.

To generate the HPlan, a compilation is formed such that the action $a = \mathit{ground}(o, \chi)$ must be applied for the plan to be valid. The compilation introduces a new predicate $\mathit{has\_done\_a}$, 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

- $Ps' = Ps \cup \{\mathit{has\_done\_a}\}$
- $As' = \{o'\} \cup As \setminus \{o\}$
- $arity'(x) = arity(x), \forall x \in arity$
- $arity'(\mathit{has\_done\_a}) = arity'(o') = arity(o)$
- $G' = G \cup \{\mathit{ground}(\mathit{has\_done\_a}, \chi)\}$

where the new operator o' extends o with the add effect $\mathit{has\_done\_a}$ with corresponding parameters, i.e.

$$\mathit{Eff}^+_\dashv(o') = \mathit{Eff}^+_\dashv(o) \cup \{\mathit{has\_done\_a}\}$$

For example, given the user question above, the operator $\mathit{goto\_waypoint}$ from the running example is extended to $\mathit{goto\_waypoint}'$ with the additional add effect $\mathit{has\_done\_a}$:

```
(:durative-action goto_waypoint'
  :parameters (?v - robot
    ?from ?to - waypoint)
  :duration( = ?duration
    (travel_time ?from ?to))
  :condition (at start (robot_at ?v ?from)
             (over all (connected ?from ?to))
  :effect (and (at end (visited ?to))
    (at start (not (robot_at ?v ?from)))
    (at end (robot_at ?v ?to))
    (at end (has_done_goto_waypoint' ?v ?from
?to))))
```

and the goal is extended to include the proposition: $(\mathit{has\_done\_goto\_waypoint}\ \mathit{kenny}\ wp2\ wp4)$.

4.3 Remove a Specific Grounded Action

Given a plan ϕ, 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 $(\mathit{goto\_waypoint}\ \mathit{kenny}\ wp1\ wp2)$ used, rather than not being used?"

A user might ask this because the robot has already satisfied the goal to visit $wp2$ before this point with the action $(goto\_waypoint\ kenny\ wp0\ wp2)$. The user might think the second action $(goto\_waypoint\ kenny\ wp1\ wp2)$ seems redundant.

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 $(goto\_waypoint\ kenny\ wp1\ wp2)$, and is shown in Figure 6.

4.4 Forbid an Action Outside a Time Window

Given a plan ϕ, 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 $(goto\_waypoint\ kenny\ wp0\ wp4)$ used outside of times 0 and 2, rather than being restricted to that time window?"

A user can ask this because the action $(goto\_waypoint\ kenny\ wp0\ wp4)$ is used at the end of the plan, the robot starts at $wp0$ and must visit $wp4$ to satisfy a goal. The user might think that satisfying this goal earlier in the plan will free up time for the robot to complete the other goals.

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

Operator $o_{\neg a}$ replaces the original operator o for all other actions $ground(o, \chi')$, where $\chi' \neq \chi$. The action $ground(o_{\neg a}, \chi)$ 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 $a = ground(o, \chi)$, 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:

- $Ps' = Ps \cup \{can\_do\_a, not\_done\_a\}$
- $As' = \{o_a, o_{\neg a}\} \cup As \setminus \{o\}$
- $arity'(x) = arity(x), \forall x \in arity$
- $arity'(can\_do\_a) = arity'(not\_done\_a) = arity'(o_a) = arity'(o_{\neg a}) = arity(o)$
- $I' = I \cup \{ground(not\_done\_a, \chi)\}$
- $G' = G \cup \{ground(not\_done\_a, \chi)\}$
- $W' = W \cup \{\langle lb, ub, ground(can\_do\_a, \chi)\rangle\}$

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

$$Eff^{-}_{\vdash}(o_{\neg a}) = Eff^{-}_{\vdash}(o) \cup \{not\_done\_a\}$$
$$Pre_{\vdash}(o_a) = Pre_{\vdash}(o) \cup \{can\_do\_a\}$$

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 $o_{\neg a}$. As in Section 4.3 we make sure the ground action $ground(o_{\neg a}, \chi)$ can never appear in the plan.

For example, given the user question above, the operator $goto\_waypoint$ from Figure 2 is extended to $o_{\neg a}$ and o_a as shown below:

```
(:durative-action goto_waypoint_nota
:parameters (?v - robot
    ?from ?to - waypoint)
  :duration(= ?duration
    (travel_time ?from ?to))
  :condition (and
    (at start (robot_at ?v ?from))
    (over all (connected ?from ?to)))
  :effect (and
    (at end (visited ?to))
    (at start (not (robot_at ?v ?from)))
    (at end (robot_at ?v ?to))
    (at start (not (not_done_goto_waypoint ?v
?from ?to)))))
```

```
(:durative-action goto_waypoint_a
:parameters (?v - robot
    ?from ?to - waypoint)
  :duration(= ?duration
    (travel_time ?from ?to))
  :condition (and (at start
(can_do_goto_waypoint ?v ?from ?to))
    (at start (robot_at ?v ?from))
    (over all (connected ?from ?to)))
  :effect (and (at end (visited ?to))
    (at start (not (robot_at ?v ?from)))
    (at end (robot_at ?v ?to))))
```

The initial state is extended to include the proposition $(not\_done\_goto\_waypoint\ kenny\ wp0\ wp4)$ and the time window $\langle 0, 2, (can\_do\_goto\_waypoint\ kenny\ wp0\ wp4)\rangle$. This time window enforces that the proposition $(can\_do\_goto\_waypoint\ kenny\ wp0\ wp4)$ is true between times 0 and 2. The resulting HPlan is:

```
0.00: (goto_waypoint kenny wp0 wp2)  [1.45]
1.45: (goto_waypoint kenny wp2 wp1)  [2.00]
3.45: (goto_waypoint kenny wp1 wp2)  [2.00]
5.45: (goto_waypoint kenny wp2 wp5)  [2.00]
7.45: (goto_waypoint kenny wp5 wp3)  [4.69]
```

```
12.14: (goto_waypoint kenny wp3 wp5)  [4.69]
16.83: (goto_waypoint kenny wp5 wp2)  [2.00]
18.84: (goto_waypoint kenny wp2 wp4)  [2.98]
```

Following the user suggestion, the action is no longer applied outside of the time window, and in fact does not appear in the plan at all.

4.5 Add an Action Within a Time Window

Given a plan ϕ, 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 $(goto\_waypoint\,kenny\,wp0\,wp4)$ not used between times 0 and 2, rather than being used in this time window?"

The HPlan given in Section 4.4 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 4.4 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 such that the ground action $a = ground(o, \chi)$ is forced 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 4.4. Simply, the former ensures that new action $ground(o_a, \chi)$ 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 $a = ground(o, \chi)$ 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:

- $Ps' = Ps \cup \{can\_do\_a, has\_done\_a\}$
- $As' = \{o_a\} \cup As$
- $arity'(x) = arity(x), \forall x \in arity$
- $arity'(can\_do\_a) = arity'(has\_done\_a)$ $= arity'(o_a) = arity(o)$
- $G' = G \cup \{ground(has\_done\_a, \chi)\}$
- $W' = W \cup \{\langle lb, ub, ground(can\_do\_a, \chi)\rangle\}$

As $wp4$ is a dead end there is no valid HPlan following this suggestion.

4.6 Delay/Advance an Action

Given a plan ϕ, 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' after/before t?*

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

> "Why is $(goto\_waypoint\,kenny\,wp2\,wp5)$ used at time 5.45, rather than at least 4 seconds earlier?"

A user might ask this question in general because 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 for why a vehicle was refueled earlier or later than what was expected. In this case the refuel action can be delayed or advanced to answer this question.

For this particular example the user might want the action $(goto\_waypoint\,kenny\,wp2\,wp5)$ to be advanced nearer the start of the plan. The user might see that in the original plan the robot goes from $wp2$ to $wp1$ at time 1.45 and then instantly goes back again. The user might think that a better action would be to go from $wp2$ to $wp5$ before this. The user might notice that $wp5$ is connected to more waypoints than $wp1$. Having these extra options might prevent redundant actions that revisit waypoints.

To generate the HPlan, the planning model is compiled such that the ground action $a = ground(o, \chi)$ 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 4.4). 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:

- $Ps' = Ps \cup \{can\_do\_a, not\_done\_a, has\_done\_a\}$
- $As' = \{o_a, o_{\neg a}\} \cup As \setminus \{o\}$
- $arity'(x) = arity(x), \forall x \in arity$
- $arity'(can\_do\_a) = arity'(not\_done\_a) =$ $arity'(has\_done\_a) = arity'(o_a) =$ $arity'(o_{\neg a}) = arity(o)$
- $I' = I \cup \{ground(not\_done\_a, \chi)\}$
- $G' = G \cup \left\{ \begin{array}{c} ground(not\_done\_a, \chi), \\ ground(has\_done\_a, \chi) \end{array} \right\}$
- $W' = W \cup \begin{cases} before : \langle 0, tReal, ground(can\_do\_a, \chi)\rangle \\ after : \langle tReal, inf, ground(can\_do\_a, \chi)\rangle \end{cases}$

where the new operators o_a and $o_{\neg a}$ both extend o. The latter with the delete effect not_done_a, while o_a extends o with the precondition can_do_a and add effect has_done_a; i.e.:

$$Eff^-_{\dashv}(o_{\neg a}) = Eff^-_{\dashv}(o) \cup \{not\_done\_a\}$$
$$Pre_{\leftrightarrow}(o_a) = Pre_{\leftrightarrow}(o) \cup \{can\_do\_a\}$$
$$Eff^+_{\dashv}(o_a) = Eff^+_{\dashv}(o) \cup \{has\_done\_a\}$$

This ensures that the ground action $a = ground(o_a, \chi)$ 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. Given the user question above, the HPlan is:

```
0.00: (goto_waypoint kenny wp0 wp2)  [1.45]
1.45: (goto_waypoint_a kenny wp2 wp5) [2.00]
3.45: (goto_waypoint kenny wp5 wp3)  [4.68]
8.13: (goto_waypoint kenny wp3 wp5)  [4.68]
12.81: (goto_waypoint kenny wp5 wp2)  [2.00]
14.81: (goto_waypoint kenny wp2 wp1)  [2.00]
16.81: (goto_waypoint kenny wp1 wp0)  [2.00]
18.81: (goto_waypoint kenny wp0 wp4)  [2.00]
```

4.7 Reordering Actions

Given a plan ϕ, 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 $(goto\_waypoint\, kenny\, wp2\, wp1)$ used before $(goto\_waypoint\, kenny\, wp2\, wp5)$, rather than after?"

A user might ask this because there are more connections from $wp5$ than $wp2$. The user might think that if the robot has more choice of where to move to, the planner could make a better choice, giving a more efficient plan.

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 $a \prec b$ where $a = ground(o, \chi)$ and $b = ground(n, \chi')$.

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.

For each node representing a ground action $a \in N$, the action is disallowed using the compilation from Section 4.3. Also, 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, $arity(o_a)$ is the combined arity of the original operator plus the arity of all of a's sink nodes. Specifically, the HModel Π' is:

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

where:

- $Ps' = Ps \cup \{ordered_{ab}\} \cup \{traversed_{ab}\}, \forall(a, b) \in E$
- $As' = \{o_a\} \cup As, \forall a \in N$
- $arity'(x) = arity(x), \forall x \in arity$
- $arity'(o_a) = arity(o) + \sum_{(a,b)\in E} arity(b), \forall a \in N$
- $arity'(ordered_{ab}) = arity(a) + arity(b), \forall(a, b) \in E$
- $arity'(traversed_{ab}) = arity(b), \forall(a, b) \in E$
- $I' = I \cup ground(ordered_{ab}, \chi + \chi'), \forall(a, b) \in E$, 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. Therefore, the only grounding of the operator that can be performed is the action with parameters $\chi + \chi'$. This drastically reduces the size of the search space.

For example given the user question above, two new operators $node\_goto\_waypoint\_kenny\_wp2\_wp5$ (shown in Figure 7) and $node\_goto\_waypoint\_kenny\_wp2\_wp1$ are added to the domain. These extend operator $goto\_waypoint$ from Figure 2 as described above. The HPlan generated is shown below:

```
(:durative-action
    node_goto_waypoint_kenny_wp2_wp5
  :parameters (?v1 ?v2 - robot
    ?from1 ?to1 ?from2 ?to2 - waypoint)
  :duration ( = ?duration
    (travel_time ?from1 ?to1))
  :condition (and (at start
    (robot_at ?v1 ?from1))
    (over all (connected ?from ?to))
    (at start (ordered_wp2_wp5_wp2_wp1 ?v1
?v2 ?from1 ?to1 ?from2 ?to2)))
  :effect (and (at end (visited ?to1))
    (at start (not (robot_at ?v1 ?from1)))
    (at end (robot_at ?v1 ?to1))
    (at end (traversed_v2_from2_to2 ?v2
?from2 ?to2)) ))
```

Figure 7: An operator added to the original domain to capture an ordering constraint between actions. The operator extends the original $goto\_waypoint$ operator.

```
0.00: (goto_waypoint kenny wp0 wp2)  [1.45]
1.45: (node_goto_waypoint_kenny_wp2_wp5 kenny
kenny wp2 wp5 wp2 wp1) [2.00]
3.45: (goto_waypoint kenny wp5 wp3)  [4.68]
8.13: (goto_waypoint kenny wp3 wp5)  [4.68]
12.81: (goto_waypoint kenny wp5 wp2)  [2.00]
14.81: (node_goto_waypoint_kenny_wp2_wp1 kenny
wp2 wp1) [2.00]
16.81: (goto_waypoint kenny wp1 wp0)  [2.00]
18.81: (goto_waypoint kenny wp0 wp4)  [2.00]
```

5 Conclusion

In this paper we have presented an approach to compiling a set of formal contrastive questions into domain independent constraints. These are then used within the XAI paradigm to provide explanations. We have described how these compilations form a part of a series of stages which start with a user question and end with an explanation. This paper formalises and provides examples of these compilations in PDDL 2.1 for temporal and numeric domains and planners.

We have defined a series of questions which we believe a user may have about a plan in a PDDL2.1 setting. These questions cover a large set of scenarios, and can be stacked to create new interesting constraints which may answer a much richer set of questions.

We acknowledge that the questions we provide compilations for do not cover the full set of contrastive questions one may have about a plan. For example the question, "Why is the operator o with parameters χ used at time $lb < t < ub$, rather than not being used in this time window?", can be answered using a variant of Section 4.5. For future work we plan to investigate which compilations will form an atomic set whose elements can be stacked to cover the full set of possible contrastive questions. We also acknowledge that the compilations we have formalised may have equivalent compilations. However, the ones we have described have proven successful for explanations.

In future work, we will look to extend this work in several ways. While we define how to calculate plans for contrastive cases, we do not take full advantage of contrastive explanations by explaining the *difference* between two plans (Miller 2018). In particular, we will look to extend the presentation beyond just plans into showing the difference between two causal chains as well.

We will explore contrastive explanations with preferences in PDDL 3 (Gerevini and Long 2005).

We will look at producing a language for expressing questions and constraints on plans. LTL will likely play a role in defining the semantics of any such language. Additional concepts concerning plan structure, such as the ability to specify that an action is part of the causal support for a goal or sub-goal, will be needed. As it stands when we add a constraint to include an action, the constraint may be satisfied in trivial ways not relevant to answering the users question. The action may be redundant, or undone in the HPlan as described in (Fox, Long, and Magazzeni 2017). In this case the explanation may not be deemed satisfactory. These additional concepts will help solve this problem, as well as allowing users to ask more expressive questions such as, "Why did you use action A rather than action B for achieving P?".

Finally, we will provide functional and human-behavioural evaluations of our explanations, to assess their effectiveness. To make sure they are both satisfactory from a user perspective, and that they provide actionable insight into the plan.

Acknowledgements This work was partially supported by Innovate UK grant 133549: *Intelligent Situational Awareness Platform*, and by EPSRC grant EP/R033722/1: *Trust in Human-Machine Partnerships*.

References

Borgo, R.; Cashmore, M.; and Magazzeni, D. 2018. Towards providing explanations for AI planner decisions. *IJCAI-18 Workshop on Explainable AI.*

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *IJCAI.*

Chakraborti, T.; Kulkarni, A.; Sreedharan, S.; Smith, D. E.; and Kambhampati, S. 2019. Explicability? legibility? predictability? transparency? privacy? security? the emerging landscape of interpretable agent behavior. In *ICAPS.*

Fox, M., and Long, D. 2003. PDDL2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelliigence Research* 20:61–124.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable planning. *IJCAI-17 workshop on Explainable AI* abs/1709.10256.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in pddl3 - the language of the fifth international planning competition. Technical report.

Haynes, S. R.; Cohen, M. A.; and Ritter, F. E. 2009. Designs for explaining intelligent agents. *International Journal of Human-Computer Studies* 67(1):90 – 110.

Lewis, D. 1986. Causal explanation. In Lewis, D., ed., *Philosophical Papers Vol. Ii.* Oxford University Press. 214–240.

Lim, B. Y.; Dey, A. K.; and 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, 2119–2128.

Lipton, P. 1990. Contrastive explanation. *Royal Institute of Philosophy Supplement* 27:247266.

Lipton, Z. C. 2016. The mythos of model interpretability. *CoRR* abs/1606.03490.

Miller, T. 2018. Contrastive explanation: A structural-model approach. *CoRR* abs/1811.03163.

Miller, T. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence* 267:1–38.

Mueller, S. T.; Hoffman, R. R.; Clancey, W. J.; Emrey, A.; and 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.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. "why should I trust you?": Explaining the predictions of any classifier. *CoRR* abs/1602.04938.

Seegebarth, B.; Müller, F.; Schattenberg, B.; and Biundo, S. 2012. Making hybrid plans more clear to human users — a formal approach for generating sound explanations. In *ICAPS.*

Smith, D. 2012. Planning as an iterative process. In *AAAI.*

Zhang, Y.; Sreedharan, S.; Kulkarni, A.; Chakraborti, T.; Zhuo, H.; and Kambhampati, S. 2017. Plan explicability and predictability for robot task planning. In *ICRA.*

Domain-independent Plan Intervention When Users Unwittingly Facilitate Attacks

Sachini Weerawardhana and **Darrell Whitley**[1] **Mark Roberts**[2]
[1]Computer Science Department, Colorado State University, Fort Collins, CO, USA | {sachini, whitley}@cs.colostate.edu
[2]The U.S. Naval Research Laboratory, Code 5514; Washington, DC, USA | mark.roberts@nrl.navy.mil

Abstract

In competitive situations, agents may take actions to achieve their goals that unwittingly facilitate an opponent's goals. We consider a domain where three agents operate: (1) a user (human), (2) an attacker (human or a software) agent and (3) an observer (a software) agent. The user and the attacker compete to achieve different goals. When there is a disparity in the domain knowledge the user and the attacker possess, the attacker may use the user's unfamiliarity with the domain to its advantage and further its own goal. In this situation, the observer, whose goal is to support the user may need to intervene, and this intervention needs to occur online, on-time and be accurate. We formalize the online plan intervention problem and propose a solution that uses a decision tree classifier to identify intervention points in situations where agents unwittingly facilitate an opponent's goal. We trained a classifier using domain-independent features extracted from the observer's decision space to evaluate the "criticality" of the current state. The trained model is then used in an online setting on IPC benchmarks to identify observations that warrant intervention. Our contributions lay a foundation for further work in the area of deciding when to intervene.

1 Introduction

When an agent is executing a plan to achieve some goal, it's progress may be challenged by unforeseen changes such as an unexpected modification to the environment or an adversary subverting the agent's goal. In these situations, a passive observer intervening to help the agent reach it's intended goal will be beneficial. Intervention is different from the typical plan recognition problem because we assume the observed agent pursues desirable goals while avoiding undesirable states. Therefore, the observer must (1) monitor actions/state unobtrusively to predict trajectories of the observed agent (keyhole recognition) and (2) assist the observed agent to safely complete the intended task or block the current step if unsafe. Consider a user checking email on a computer. An attacker who wants to steal the user's password makes several approaches: sending an email with a link to a phishing website and sending a PDF file attachment embedded with a keylogger. The user, despite being unaware of the attacker's plan, would like to complete the task of checking email safely and avoid the attacker's goal. Through learning, our observer can recognize risky actions the user may execute in the environment and ensure safety.

The decision of when to intervene must be made judicially. Intervening too early may lead to wasted effort chasing down false positives, helpful warnings being ignored as a nuisances, or leaking information for the next attack. Intervening too late may result in the undesirable state. Further, we are interested in assisting a human user with different skill levels, who would benefit more from customized intervention. To this end, we need to identify actions that warrant intervention over three different time horizons: (1) critical action, which if unchecked will definitely trigger the undesirable state, (2) mitigating action, which gives the user some time to react because the threat is not imminent and (3) preventing actions, which allows for long term planning to help the user avoid threats. Based on the time horizon we are current in, we can then plan to correct course accordingly. In this work we focus on identifying the first horizon. Intervention is useful in both online settings, where undesirable states may arrive incrementally and in offline settings where observations are available prior to intervention.

In this paper, we model online intervention in a competitive environment where three agents operate: (1) a user (human), (2) an attacker (human or a software) agent and (3) an observer (a software) agent who will intervene the user. The observer passively monitors the user and the attacker competing to achieve different goals. The attacker attempts (both actively and passively) to leverage the progress made by a user to achieve its own goal. The attacker may mask domain knowledge available to the user to expand the attack vector and increase the likelihood of a successfull attack. The user is pursuing a desirable goal while avoiding undesirable states. Using domain-independant features, we train a decision tree classifier to help the observer decide whether to intervene. A variation of the relaxed plan graph (Blum and Furst 1997) models the desirable, undesirable and neutral states that are reachable at different depths. From the graph, we extract several domain independent features: risk, desirability, distances remaining to desirable goal and undesirable states and active landmarks percentage.

We train a classifier to recognize an observation as a intervention point and evaluate the learned model on previously unseen observation traces to assess the accuracy. Furthermore, the domain independent features used in the classifier offer a mechanism to explain why the intervention occurred. In real-time, making the decision to intervene for each observation may be costly. We examine how the observer can establish a wait time without compromising accuracy.

The contributions of this paper include: (1) formalizing the online intervention problem as an intervention graph that extends the planning graph, (2) introducing domain-independent features that estimate the criticality of the current state to cause a known undesirable state, (3) presenting an approach that learns to classify an observation as intervention or not, (4) incorporating salient features that are better predictors of intervention to generate explanations, and (5) showing this approach works well with benchmarks.

2 Example

Before we formalize the problem, we present examples for two cases of the online intervention: (1) the attacker is actively trying to make the user reach the undesirable state by leveraging the user's progress and (2) the passive attacker introduces an undesirable state to the environment without the user's knowledge (i.e., a trap), where attacker masks the location of the trap and exploits the user's unfamiliarity with the domain to make the user reach the undesirable state. In both cases, the observer monitors the attacker and the users' actions. The user plans for a desirable goal state, G_d. Given the unexpected modification to the domain model, executing this plan may likely cause the user to reach the undesirable state (G_u). The observer is assumed to be familiar with the domain (regardless of attacker's attempts to mask information to the user) and has knowledge about commonly occurring goals such as G_d and G_u. The user would like to be interrupted if some action will trigger G_u.

Active Attacker: We use the IPC block-words domain (Gupta and Nau 1992) to illustrate the active attacker's case. The observer is watching the user stacking blocks to spell a word. The domain contains 4 blocks: T, B, A, D. Figure 1 shows the undesirable state developing from initial state I. G_d equals the word TAD, while G_u equals the word BAD. The user can not recognize block B (indicated by dotted lines), which prevents the user from identifying states resulting from performing operations on B such as stack and pick up, and therefore fail to circumvent G_u on his own. The attacker will use block B to defeat the user and achieve G_u.

In the initial state (I), all blocks are on the table. The user's arm (solid line) and the attacker's arm (dotted line) are empty. In the next sequence of events, the observer sees that the user has picked up block A (S_1) and stacked A on D (S_2). Consider two alternative timelines T_1 and T_2 stemming from S_2. In T_1, the observer sees that the user has picked up T and the attacker has also picked up B. The next state shows that the user has stacked T on A to spell the word TAD and reached G_d successfully. In timeline T_2, the attacker has succeeded in reaching G_u by stacking B on A before the user stacked T on A, leveraging the user's progress.

Passive Attacker: This case considers the 3x3 grid world domain (McDermott 1999) shown in Figure 2. The observer watches the user (white circle) navigating from a start point (0,0) on the grid to reach G_d point (3,3) in 1-step actions. When executing a plan to reach G_d, the user would like to avoid the trap at point X (2,3), G_u but will not be able to

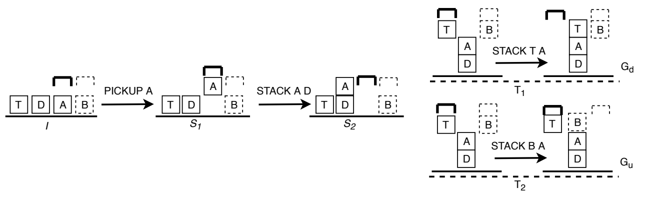


Figure 1: Reaching G_u with an active attacker

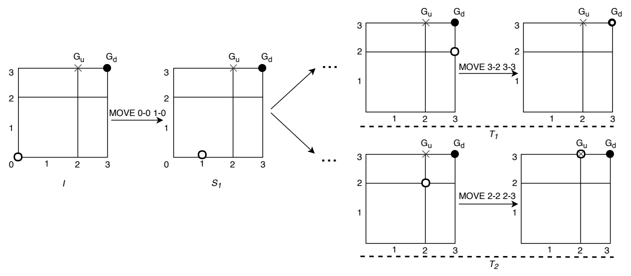


Figure 2: Reaching G_u with passive attacker

do so unless the observer interrupted. Let us assume the observer sees the user's action resulting in state S_1. Although the move indicates that the user is moving toward G_u and G_d, interruption is too early. In two alternative timelines T_1 (top right) and T_2 (bottom), the observer sees different moves. In T_1 the user has reached G_d while avoiding G_u, in which case the observer need not interrupt. However, in T_2 the user has reached G_u, in which case it would have been helpful if the user was blocked before moving to (2,3).

3 The Intervention Problem

Our formulation of the intervention problem makes several assumptions about the three actors. (1) **Observer**: intervention decisions are made in an online setting for each observation that appears incrementally and include actions executed by the attacker or the user. The goals G_d or G_u are known but the plans to reach G_d or G_u are hidden. The domains for which plan intervention problem is defined are discrete and all actions are assumed to be of unit cost. The observer has full observability in the domain and the environment is deterministic. Therefore, it can determine the actions that are immediately applicable in the current state. (2) **User**: Follows a plan to reach G_d, but may reach G_u unwittingly. G_u is hidden, but would like the observer's help to avoid G_u. The user does not have full observability of the domain or the attacker's actions. (3) **Attacker**: Follows a plan to reach G_u. The attacker has full observability of the domain and the user's actions. Given these assumptions, the observer assesses the state after each observation. This requires the observer to hypothesize about possible interesting trajectories from current state and evaluate each trajectory in terms of their likelihood to cause G_u.

3.1 Definitions

Following STRIPS (Fikes and Nilsson 1971), we define a planning problem as a tuple $P = \langle F, A, I, G \rangle$ where F is the set of fluents, $I \subseteq F$ is the initial state, $G \subseteq F$ represents the set of goal states and A is the set of actions. Each action $a \in A$ is a triple $a = \langle Pre(a), Add(a), Del(a) \rangle$ that consists of preconditions, add and delete effects respectively, where $Pre(a), Add(a), Del(a)$ are all subsets of F. An action a is applicable in a state s if preconditions of a are true in s; $pre(a) \in s$. If an action a is executed in state s, it results in a new state $s' = (s \setminus del(a) \cup add(a))$. The solution to P is a plan $\pi = \{a_1, \ldots, a_k\}$ of length k that modifies I into G by execution of actions $a_1, \ldots, a_k$.

The plan recognition problem defined by Ramirez and Geffner (2010) is a triple $T = \langle D, \mathcal{G}, O \rangle$ where $D = \langle F, A, I \rangle$ is a planning domain, $\mathcal{G}$ is the set of goals, and $\mathcal{G} \subseteq F$. An observation sequence $O = o_1, \ldots, o_m$ are actions $o_i \in A, i \in [1, m]$. A solution to the plan recognition problem is a subset of goals $G \in \mathcal{G}$ for which an optimal plan $P[G]$ satisfying O is produced.

Similarly, the plan intervention problem ($\mathcal{I}$) also uses observations of actions. However, instead of using information gleaned from the observation trace to find the most likely plans (and goals), the intervention problem aims to assess the current state for it's ability to cause G_u and identify whether or not the user needs to be blocked from making further progress. Unlike Ramirez and Geffeners' approach, the observations used in our solution are not noisy nor do they contain missing actions. This will be addressed in future work.

Plan intervention problem $\mathcal{I} = \langle D, O, G_u, G_d, M \rangle$ consists of a planning domain D and a sequence of observed actions O, a set of undesirable states $G_u \subseteq F$, a set of desirable states $G_d \subseteq F$ ($G_u \neq G_d$), and a decision tree classifier model M that combines a vector of domain-independant features to classify an obervation as requiring intervention or not. The extension to typical plan/goal recognition comes from the domain-independent feature vector, which will be discussed in section 3.3. A solution to $\mathcal{I}$ is a vector of decision points corresponding to actions in O indicating whether each action was identified as requiring intervention.

3.2 Modelling the Intervention Decision Space

To assess the criticality of the current state to cause G_u, the observer enumerates action sequences that will transform the current state to G_d. These action sequences and intermediate states make up the observer's decision space, which is a single-root directed acyclic connected graph $S = \langle V, E \rangle$, where V is the set of vertices denoting possible states the user could be in until G_d is reached, and E is the set of edges representing actions from A. We refer to this graph as the *intervention graph*. The root of the intervention graph indicates the current state. Leaves of the graph are goal states (i.e., G_u and G_d). A path from root of the tree to G_u represents a candidate attack plan, while a path from root to leaf node containing G_d represents a desirable plan.

Figure 3 illustrates the observer's decision space for unobserved actions extending from state S_1 in Figure 1. Some subtrees are hidden for simplicity. Given the initial state where all 4 blocks are on the table, the observer expects the next action to be one in the set (`PICK-UP` {`T`, `D`, `A`, `B`}), but B is hidden from the user. The attacker can execute any

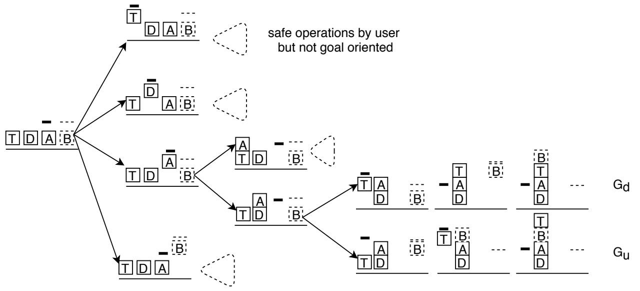


Figure 3: Fragment of the decision space at state I for blockwords plan intervention example in Figure 1

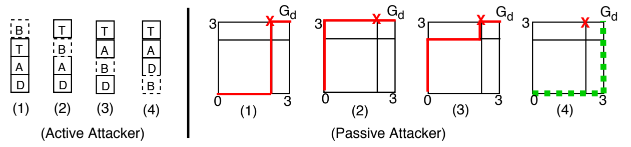


Figure 4: User achieving G_d amid attacker actions in intervention examples in Section 2

of the 4 actions. Using the intervention graph, the observer hypothesizes all possible action sequences that can be observed in the future, that will lead to G_u (spell BAD) or G_d (spell TAD). One such sequence (as shown in the figure) is: `PICK-UP A` $\rightarrow$ `STACK A D` $\rightarrow$ `PICK-UP T` $\rightarrow$ `STACK T A`. At this point the user reaches G_d. On the other hand if the sequence was `PICK-UP A` $\rightarrow$ `STACK A D` $\rightarrow$ `PICK-UP B` $\rightarrow$ `STACK B A`, with the last two actions executed by the attacker, the attacker achieves G_u.

Figure 4 illustrates how the user's plans to reach G_d could fail in the presence of an active (left) or passive (right) attacker. In the case of an active attacker, given the assumption that the attacker does not backtrack to a previous state and only leverages progress made thus far, it can make four attempts to prevent the user from reaching G_u by inserting the hidden block into the partially built stack. If the user achieves goal states 1 or 4 the user wins despite the attacker. If the observed actions indicate that the user is heading toward one of these two states, then an interrupt is unwarranted. State 3 is less ideal for the user but G_u is not achieved. In state 2 the attacker has successfully reached G_u. Observations leading to state 2 warrant interruption.

In the case of a passive attacker, the observer needs to hypothesize about likely goals of the user given the current state. Figure 4 (right) shows three of many such plans the user may follow to reach G_d. Paths 1, 2 and 3 all result in user going past the undesirable state (marked x), and at some point in these observation sequences the user must be interrupted before G_u is reached. In contrast, path 4 indicates a safe path and must not generate an interrupt.

Algorithm 1 describes how the intervention graph is built. The intervention graph is similar to the relaxed planning graph (RPG), where each level consists of predicates that have been made true and actions $a \in A$ whose preconditions are satisfied. Initially, before any observations have been made, the current state (i.e., root of the tree) is set to initial state I. Next, using the domain theory D, actions $a \in A$ whose preconditions are satisfied at current state are added to the graph. Each action in level i spawn possible states for level $i+1$. Calling the method recursively for each state until G_d and G_u are added to some subsequent level in the graph will generate a possible hypotheses space for the observer. As a new observation arrives, the root of the graph is changed to reflect the new state after the observation and subsequent layers are also modified to that effect. Similar to the RPG, we omit delete effects during construction. Also construction terminates once G_d is reached. The graph building algorithm does not allow adding backtracking actions because it will create a cycle.

Algorithm 1 Build Intervention Graph

```
Require: $D, s, G_u, G_d$
 1: $i = 0; s_i \leftarrow I$
 2: procedure EXPANDGRAPH($D, s, G_u, G_d$)
 3:     if $s_i \models G_u, G_d$ then return $\langle V, E \rangle$
 4:     else
 5:         for $a \in A$ where $Pre(a) \in s_i$ do
 6:             $s_{i+1} \leftarrow ((s_i \setminus Del(a)) \cup Add(a))$
 7:             if $s_{i+1} \equiv s_i$ then continue
 8:             $v \leftarrow$ AddVertex ($s_{i+1}$)
 9:             $e \leftarrow$ AddEdge ($s, s_{i+1}, a$)
10:             $V \cup \{v\}$ ; $E \cup \{e\}$
11:             ExpandGraph ($D, s_{i+1}, G_u, G_d$)
```

3.3 Domain Independent Features

We extract a set of features from the intervention graph that help determine when to intervene. These features include: Risk, Desirability, Distance to G_d, Distance to G_u and Percentage of active undesirable landmarks in current state. We use these features to train a decision tree. Figure 5 illustrates a fragment of the intervention graph after `PICK-UP A`. Following the subtree extending from action `STACK A D`, both G_u and G_d can be reached. Unexpanded subtree T_1 also contains instances where the user can reach G_d safely, without reaching G_u. We will use Figure 5 as a running example to discuss feature computation.

Risk (R) quantifies how likely the effects of current observation will lead to G_u. R is also coupled with the uncertainty the observer has about the next observation. We model the uncertainty as a uniform probability distribution across the set of actions whose preconditions are satisfied in current state. We define R as the posterior probability of reaching G_u while the user is trying to achieve G_d. Given the intervention graph, we extract paths from root to any leaf containing the G_d, including the ones in which the user has been subverted to reach G_u instead. By virtue of construction termination, G_d will always be a leaf. R is computed for paths leading to state (2) in Figure 4 (left) because in that state the attacker has won. In the passive attacker case any path in the intervention graph that causes the user to reach point X, before G_d is reached qualifies as candidates to compute R.

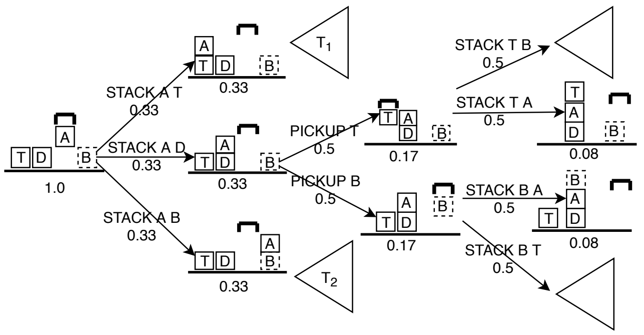


Figure 5: Fragment of the decision space after PICKUP A has been observed for block-words example. Numbers under each state and action indicate the probability. Subrees T_1 and T_2 are not expanded for simplicity.

Let $\Pi_{candidates}$ be the plans reaching G_d and let $|\Pi_{candidates}| = n$. The plan set Π_u contains action sequences that reach state G_u such that, $\Pi_u \subseteq \Pi_{candidates}$, $|\Pi_u| = m$ and $(m <= n)$. We compute posterior probability of reaching G_u for a path $\pi \in \Pi_u$, using chain rule in probability as, $P_\pi = \prod_{j=1}^{k} P(\alpha_j | \alpha_1, \alpha_2, ..., \alpha_{k-1})$, and $\alpha_j \in A$ and k is the length of path until G_u is reached. Then:

$$R = \begin{cases} \frac{\sum_{i=1}^{m} P_{\pi_i}}{m} & m > 0 \\ 0 & m = 0 \end{cases}$$

There are six action sequences the observer might observe when the user is trying to achieve G_d $(n = 6)$ and only one of those six sequences will make the user reach G_u $(m = 1)$. Since we assumed full observability for the observer, the root of the tree (current state) is assigned the probability of 1.0. Then, actions that are immediately possible after current state (`STACK A B`, `STACK A D`, `STACK A T`) are each assigned probabilites following a uniform distribution across the branching factor (0.33). Then for each applicable action in the current state, the resulting state gets the probability of $(1.0 \times 0.33 = 0.33)$. Similarly, we apply the chain rule of probability for each following state and action level in the graph until G_u first appears in the path. In this graph, G_u appears two actions later and $R = \frac{0.08}{1} = 0.08$.

Desirability (D) measures the effect of the observed action to help the user pursue the desirable goal safely. It separates common harmless actions from avoidable ones and connects the observations to knowledge of the goals the user wants to achieve. Given $\Pi_{candidates}$ as the set of plans extracted from the intervention graph that reach G_d and $|\Pi_{candidates}| = n$. The plan set Π_d contains action sequences that reach state G_d without reaching G_u, $\Pi_d = \Pi_{candidates} \setminus \Pi_u$, we compute posterior probability of reaching G_d without reaching G_u for a path $\pi \in \Pi_d$, using chain rule in probability as, $P_\pi = \prod_{j=1}^{k} P(\alpha_j | \alpha_1, \alpha_2, ..., \alpha_{k-1})$, and $\alpha_j \in A$ and k is the length of path. Then:

$$D = \begin{cases} \frac{\sum_{i=1}^{n-m} P_{\pi_i}}{n-m} & n - m > 0 \\ 0 & n - m = 0 \end{cases}$$

In Figure 5, there are five instances where user achieved

G_d without reaching G_u (two in subree T_1, three in the expanded branch). Extracting paths from root to these five instances, returns actions sequences the user may follow to reach G_d safely (Π_d). Following the same approach to assign probabilities for states and actions, $D = \frac{(0.08+0.08+0.08+0.04+0.04)}{5} = 0.07$. Computation for R and D is similar for the passive attacker case.

R and D are based on probabilities indicating the confidence the observer has about the next observation. We also use simple distance measures: (1) distance to G_u (δ_u) and (2) distance to G_d (δ_d). Both distances are measured in the number of actions required to reach a state containing G_d or G_u from root in the intervention graph.

Distance to $\boldsymbol{G_u}$ (δ_u) measures the distance to state G_u from the current state in terms of the number of actions. As with the computations of R and D, given $\Pi_{candidates}$ is the set of paths extracted from the intervention graph that reach G_d and $|\Pi_{candidates}| = n$. The path set Π_u contains action sequences that reach state G_u such that, $\Pi_u \subseteq \Pi_{candidates}$, $|\Pi_u| = m$ and ($m <= n$). We count s, the number of the edges (actions) before G_u is reached for each path $\pi \in \Pi_u$ and δ_u is defined as the average of the distance values given by the formula:

$$\delta_u = \begin{cases} \frac{\sum_{i=1}^{m} s_i}{m} & m > 0 \\ -1 & m = 0 \end{cases}$$

In this formula, -1 indicates that the undesirable state is not reachable from the current state. For the example problem illustrated in Figure 5, $\delta_u = \frac{3}{1} = 3$.

Distance to $\boldsymbol{G_d}$ (δ_d) measures the distance to G_d from current state. The path set Π_d contains action sequences that reach G_d without reaching G_u, $\Pi_d = \Pi_{candidates} \setminus \Pi_u$, we count t, the number of the edges where G_d is achieved without reaching G_u for each path $\pi \in \Pi_d$. Then, δ_d is defined as the average of the distances given by the formula:

$$\delta_d = \begin{cases} \frac{\sum_{i=1}^{n-m} t_i}{n-m} & n - m > 0 \\ -1 & n - m = 0 \end{cases}$$

In this formula, -1 indicates that G_d can not be reached safely from the current state. For the example problem illustrated in Figure 5, $\delta_d = \left\lceil \frac{3+3+7+7+3}{5} \right\rceil = 5$. Both δ_u and δ_d are computed similarly for the passive attacker case.

Percentage of active attack landmarks ($\mathcal{L}_{ac}$) captures the criticality of current state toward contributing to G_u. Landmarks (Hoffmann, Porteous, and Sebastia 2004) are predicates (or actions) that must be true in every valid plan for a planning problem. We used the algorithm in Hoffmann et al. (2004) to extract fact landmarks for the planning problem $P = \langle D, G_u \rangle$. These landmarks are referred to as attack landmarks because they establish predicates that must be true to reach G_u. Landmark Generation Graph (LGG) (Hoffmann, Porteous, and Sebastia 2004) for P for the active attacker case is shown in Figure 6. Predicates (ON B A), (ON A D) correspond to G_u. Predicates that are grouped must be made true together. When the observed actions activate any attack landmarks, it signals that an undesirable state is imminent. Landmarks have been successfully used in deriving heuristics in plan recognition (Vered et al. 2018)

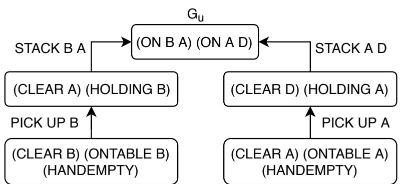


Figure 6: LGG for P. Contains verified fact landmarks for P and *greedy-necessary orders*. A box with multiple landmarks indicate fact landmarks that must be true together.

Algorithm 2 Generate Feature Vectors

```
Require: $D, I, O, G_u, G_d, p$-probability distribution.
1: procedure FEATUREVECTOR($D, O, I, G_u, G_d, p$)
2:     $i = 0; s_i \leftarrow I$
3:     for $o \in O$ do
4:         $G(V, E) \leftarrow ExpandGraph(D, s_i, G_u, G_d)$
5:         Apply action probabilities to $e \in E$ following $p$
6:         Apply state probabilities to $v \in V$ following $p$
7:         $V(o) \leftarrow [R_o, D_o, \delta_{u_o}, \delta_{d_o}, \mathcal{L}_{ac_o}, Class]$
```

and generating alternative plans (Bryce 2014). We compute a feature using attack landmarks: percentage of active attack landmarks in current state ($\mathcal{L}_{ac}$). To compute $\mathcal{L}_{ac}$ for the example in Figure 5, we count the number of landmark predicates that have become active (l) in the root of the intervention graph. Then, ($\mathcal{L}_{ac}$) is given by the formula: $\mathcal{L}_{ac} = \frac{l}{|\mathcal{L}_u|}$ In Figure 5, $l = 4$ ((CLEAR B),(CLEAR D),(ONTABLE B),(HOLDING A)) and $\mathcal{L}_{ac} = 4/10 = 0.4$.

4 Learning When to Intervene

We train the decision tree classifier in supervised learning mode to categorize observed actions into two classes: "Y" indicating that the interruption is warranted and "N", indicating that intervention is unwarranted. According to this policy, in the expanded sub-tree in Figure 5 the path that reaches G_u is labeled as follows: PICK-UP A (N), STACK A D (N), PICK-UP B (N), STACK B A (Y). Label for each action is indicated within brackets. We will make this labeled data set available for the community. Given a labeled observation set and corresponding feature vectors, we train the decision tree classifier with 10-fold cross validation. Then the trained model is used to predict intervention for previously unseen intervention problems. We decided to chose the decision tree as the classifier because the decision tree learned model had the highest accuracy in predicting intervention on new problems compared to the two other classifiers: random forests (Breiman 2001) and Naive Bayes.

To generate training data we first created twenty planning problems for each benchmark domain. Then observation traces corresponding to each problem were generated. We enforced a limit of 100 observation traces for each planning problem for grid domains. These observation traces were provided as input to Algorithm 2. The algorithm takes a PDDL domain, a set of undesirable and desirable states and a probability distribution as input and produces a relation V of observations and feature vectors. We train a decision tree classifier using the Weka [1] framework. We selected the implementation of C4.5 algorithm (Quinlan 1993) (J48), which builds a decision tree using the concept of information entropy. We chose the decision tree classifier for its ability determine salient features for intervention, which facilitates generating explanations for the user.

5 Results and Discussion

We focus on two questions: (1) Using domain-independent features indicative of the likelihood to reach G_u from current state, can the intervening agent correctly interrupt to prevent the user from reaching G_u? and (2) If the user was not interrupted now, how can we establish a wait time until the intervention occurred before G_u? To address the first question, we evaluated the performance of the learned model to predict intervention on previously unseen problems.

The experiment suit consists of the two example domains from Section 2. To this we added Navigator and Ferry domains from IPC benchmarks. In Navigator domain, an agent simply moves from one point in grid to another goal destination. In the Ferry domain, a single ferry moves cars between different locations. To simulate intervention in active attacker case (the Block-Words domain), we chose word building problems. The words user and the attacker want to build are different but they have some common letters (e.g., TAD/BAD). The attacker is able to exploit the user's progress on stacking blocks to complete word the attacker wants to build. In Easy-IPC and Navigator domains, we designated certain locations on the grid as traps. The goal of the robot is to navigate to a specific point on the grid safely. In the Ferry domain a port is *compromised* and a ferry carrying a car there results in an undesirable state. The ferry's objective is to transport cars to specified locations without passing a compromised port.

In addition to the trained data set, we also generated 3 separate instances of 20 problems each (total of 60) for the benchmark domains to produce testing data for the learned model. The three instances contained intervention problems that were different the trained instances. For example, number of blocks in the domain (block-words), size of grid (navigator, easy-ipc), accessible and inaccessible paths on the grid (navigator, easy-ipc), properties of artifacts in the grid (easy-ipc). For each instance we generated 10 observation traces for each planning problem (i.e., 200 observation traces per instance). We define true-positive as the classifier correctly predicting "Y". True-negative is an instance where the classifier correctly predicts "N". False-positives are instances where classifier incorrectly predicts an observation as an interrupt. False-negatives are instances where the classifier incorrectly predicts the observation not as an interrupt.

5.1 Feature Selection

When a human user receives an interruption, the user may like to know a reason. To extract salient features for intervention, we applied a correlation based feature selection technique in data pre-processing step to identify the top four

[1] `http://www.cs.waikato.ac.nz/ml/weka/`

Domain	Feature	Correlation
Blocks	Risk	0.85
	Distance to G_d	0.30
	Desirability	0.23
	Distance to G_u	0.09
Easy-IPC	Risk	0.84
	Distance to G_d	0.44
	Distance to G_u	0.27
	Desirability	0.23
Navigator	Risk	0.85
	Distance to G_d	0.28
	Desirability	0.18
	Distance to G_u	0.04
Ferry	Risk	0.84
	Distance to G_d	0.34
	Desirability	0.16
	Distance to G_u	0.08

Table 1: Correlation factors of top 4 features for benchmark domains.

best predictors. Feature selection reduces complexity of the model, makes the outcome of the model easier to interpret, and reduces over-fitting.

The attribute selector in Weka uses the Pearson's correlation to measure predictive ability between nominal attributes and the class. Our feature vector consists of nominal attributes. Table 1 summarizes top 4 correlated features for each domain. Risk is the best performing feature. Distance desirable state feature is the next best choice for a feature. The percentage of active attack landmarks was the weakest predictor of intervention across all benchmark domains and was removed from training.

Interrupting at each observation: Assuming the decision to intervene is made for every observation, we calculated the true-positive rate (TPR=$\frac{TP}{TP+FN}$), false-positive rate (FPR=$\frac{FP}{FP+TN}$), true-negative rate (TNR= $\frac{TN}{TN+FP}$), false-negative rate (FNR=$\frac{FN}{TP+FN}$) of the trained model. For each domain, row 'Each' in table 2 summarizes TPR, FPR, TNR, FNR for predicting intervention in unseen observation traces. The classifier works well in identifying intervention across domains. In line with our expectation, TPR and TNR are very high ($> 95\%$) across domains and FNR and FPR is very low($< 5\%$). Because the accuracy remains consistant across test instances we conclude that the model is reasonably tolerant for modifications in the domain such as grid sizes and number of objects.

Delaying the interruption: In real-life, making the intervention decision for every observation may be costly. If we are intervening a human user, he may disregard frequent interruptions as noise. For this reason, we examine how to establish a wait time until intervention occurs for the first time. We used the feature ($\mathcal{L}_{ac}$) as a checkpoint for the intervening agent to wait safely without interrupting the user.

We modified the observation traces to contain action sequences starting from a point where the current state contained 50% and 75% of active landmarks. For problem instances where 75% active landmark percentage was infeasible, we limited it to the maximum active landmark percentage. We used the same learned model to predict intervention for these modified traces. For each domain, row 'Delayed50' in table 2 summarizes TPR, FPR, TNR, FNR for predicting interruptions for benchmark domains given that the decision is delayed until $50\% <= \mathcal{L}_{ac} < 75\%$. The row 'Delayed75' indicates that the decision was delayed until $\mathcal{L}_{ac} >= 75\%$.

Accuracy is not affected significantly by delaying the intervention from the chosen checkpoints. However, a negative effect of delaying intervention is missing true positives. We evaluated how the delay affects the percentage of true positive observations missed. Table 3 summarizes these results. Intuitively, the longer the delay, a higher percentage of true positives will be missed. For the Blocks-Word domain, there is no effect between the the delay until 50% and 75%. In both cases the delaying the decision does not cause the intervening agent to miss any true positives. The most significant loss occurs in Navigator domain, where delay until 75% will cause a loss of 2%-28% while delaying until 50% is the safest choice. The Ferry domain exhibits a similar pattern where the delay until 75% landmarks become active will cause a loss of 8%-18%. We conclude that delaying interruptions can be controlled by the percentage of active landmarks in the current state and that for certain domains it is a trade off between loss of true-positives and the delay.

6 Explaining Intervention

When an observation that warrants intervention is identified intervening agent issues a warning (and an explanation) to the user. The user needs to take corrective/mitigating actions to avoid the undesirable state. The decision trees can help explain intervention. Decision trees generated for the benchmark domains are shown in Figure 7. Combining the shallow trees and the definitions of the features allow us to generate a clear and succinct set of rules to explain intervention. For the Block-word domain, (Figure 7-a), the rule that explains intervention first looks at the value of Risk. If the risk is less than or equal to 0.5 then that observation does not qualify as an intervention point. By definitions, this means that from the current state there are multiple ways to reach the undesirable state, indicating the observation is a common action that can be perceived as harmless. Next, if the observation that has a risk level of grater than 0.5 (indicating there are fewer ways of reach the undesirable state and that it's imminent), next feature to look at is the distance to the undesirable state. If the distance is negative, indicating that execution of this step will trigger the undesirable state, then the observation warrants intervention. Otherwise the observation does not require intervention. With this decision tree, an explanation for intervention in Blocks-words domain can be developed as: *The current step was intervened because the risk level is significant* ($> .5$) *and the effect of this observed action will trigger the undesirable state*.

For the passive attacker domains (Figure 7 - (b),(c),(d)) the learned model generated even simpler trees with only one feature being used to determine intervention. For Easy-IPC and Navigator domains, the Risk feature determines the class of an observation. This leads to generating explanations for the Easy-IPC and Navigator domains such as *The current step was intervened because the risk level is significant* ($> .75$ for Easy-IPC and $> .5$ for Navigator). For the

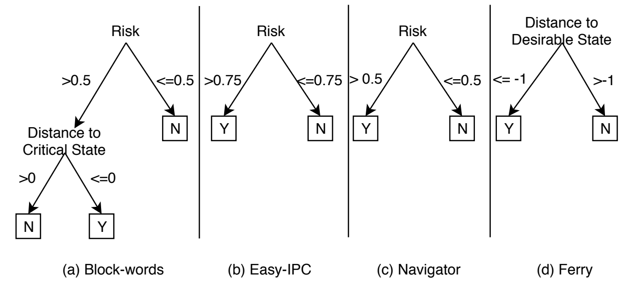


Figure 7: Decision trees generated for (a) Blocks, (b) Easy-IPC, (c) Navigator, (d) Ferry domains

Ferry domain, Distance to G_d determines intervention. A negative value indicates that if the next step was executed there is no way to reach the desirable goal state without triggering the undesirable state. Thus an explanation of intervention for the Ferry domain will be: *The current step was intervened because the effect of this step will make it impossible to reach the desired goal without triggering the undesirable state*.

7 Related Work

Closely related areas of literature for this work is plan/goal recognition. Plan recognition is the problem of inferring the course of action (i.e., plan) an actor may take towards achieving a goal from a sequence of observations (Schmidt, Sridharan, and Goodson 1978; Kautz and Allen 1986). The constructed plan, if followed to completion, is expected to result in states that correspond to goals of the actor, which in turn presupposes that the actor intends to achieve those goals. Plan/goal recognition approaches in the literature explore both domain-dependent and independent methods. In domain-dependent methods agents rely heavliy on domain knowledge for inference. For example, Kabanza et al. (2010) presents a solution that recognizes an agents adversarial intent by mapping observations made to date to a plan library. Boddy et al. (2005) discuss how to construct and manipulate domain models that describe behaviors of adversaries in computer security domain and use these models to generate plans. Another approach uses Goal Driven Autonomy (GDA) that allows agents to continuously monitor the current plans execution and assess if the current state matches with expectation (Klenk, Molineaux, and Aha 2013).

More recent work attempts to separate this knowledge dependency by allowing the agent to learn knowledge from observations (Jaidee, Muñoz-Avila, and W. Aha 2011). In contrast, domain-independent goal recognition that use planning to infer agents goals. Ramirez and Geffner (2009; 2010) used an existing planner to generate hypotheses from observations to infer a single agent's plan. Their approaches offer advantages of being more adaptive to input as well as exploiting existing planning systems and plan representations. Their first approach computed the set of goals that can be achieved by optimal plans that match the observations. The second approach removed the optimality constraint and computed a probability distribution across possible plans

Domain	Interrupt Type	Instance 1(20)				Instance 2 (20)				Instance 3 (20)			
		TPR	FPR	TNR	FNR	TPR	FPR	TNR	FNR	TPR	FPR	TNR	FNR
Blocks	Each	1	0	1	0	1	0	1	0	1	0	1	0
	Delayed50	1	0	1	0	1	0	1	0	1	0	1	0
	Delayed75	1	0	1	0	1	0	1	0	1	0	1	0
Easy-IPC	Each	1	.05	.95	0	1	.03	.97	0	1	.03	.97	0
	Delayed50	1	.06	.94	0	1	.03	.97	0	1	.03	.97	0
	Delayed75	1	.06	.94	0	1	.03	.97	0	1	.03	.97	0
Navigator	Each	1	.01	.99	0	1	.03	.97	0	1	.02	.98	0
	Delayed50	1	.01	.99	0	1	.03	.97	0	1	.02	.98	0
	Delayed75	1	.02	.98	0	1	.03	.97	0	1	.03	.97	0
Ferry	Each	1	.02	.98	0	1	.05	.95	0	1	0	1	0
	Delayed50	1	.02	.98	0	1	.05	.95	0	1	0	1	0
	Delayed75	1	.02	.98	0	1	.03	.97	0	1	0	1	0

Table 2: True-positive (TPR), False-positive (FPR), True-negative (TNR), False-negative (FNR) rates for predicting interrupt decision for unseen problems.

Domain	Delay	Instance 1	Instance 2	Instance 3
Blocks	Delayed50	0%	0%	0%
	Delayed75	0%	0%	0%
Easy-IPC	Delayed50	0%	6%	5%
	Delayed75	0%	6%	5%
Navigator	Delayed50	0%	0%	0%
	Delayed75	28%	2%	4%
Ferry	Delayed50	6%	5%	0%
	Delayed75	11%	8%	18%

Table 3: Percentage of missed observations that should have been flagged as an interrupt

that could be generated from existing planners (Ramırez and Geffner 2010). Keren et al. (Keren, Gal, and Karpas 2014) introduced the worst-case distinctiveness (wcd) metric as a measurement of the ease of performing goal recognition in a domain. The wcd problem finds the longest sequence of actions an agent can execute while hiding its goal. They show that by limiting the set of available actions in the model wcd can be minimized, which will allow the agent to reveal it's goal as early as possible.

In online recognition, Vered et al. (2018) propose an approach that combines goal-mirroring and landmarks to infer the goal of an agent. Landmarks are used to minimize the number of hypotheses the agent has to evaluate, thus improving the effeciency of the recognition process. Pozanco et al. (2018) combines Ramirez and Geffener's plan recognition approach and leverages landmarks to counterplan and block an opponent's goal achievement. The main difference between plan intervention and recognition is that, in intervention the time intervention happens is critical. In plan recognition, identifying the plan at the right time is not a priority. The user's preferences in intervention (e.g., in-time, targetted intervention vs. prolonged and incremental) and the source of uncertainty in the environment (e.g., environment, attacker) complicate the intervening agent's decisioni and can be seen as trade-offs. Furthermore, our approach complements existing approaches by using a decision tree to identify events that warrant intervention and identifying salient features that may be useful in generating explanations to plan intervention.

8 Summary and Future Work

We formalized the online plan intervention problem in a competitive domain where an attacker both actively and passively attempts to leverage progress made by a user to achieve the attacker's own conflicting goals. We introduced the intervention graph, which models the decision space of an observer, whose goal is to support the user by blocking actions that allows the attacker to achieve his goal. We trained a classifier using domain-independent features extracted from the intervention graph to evaluate the criticality of the current state. The model predicts intervention with high accuracy for the benchmark domains.

Our solution suffers from state space explosion for large domains. As an solution, we suggest sampling from alternative plans generated from off-the-shelf planners. This will also allow us to compare the proposed approach with existing online goal-recognition methods. The uncertainty model can be extended to limiting the observer's ability to fully perceive the current state. We recognize the attack models (for both active and passive cases) can be expanded to different threat models. For example, the attacker can behave as truly adversarial and undo progress the user has made so far and guide the user towards an entirely different goal. We will improve on explanations by suggesting actions that will help the user avoid the undesirable state when intervention occurs, instead of delegating the responsibility of being safe to the user, and integrating causal reasoning to explanations. These extensions lay a foundation for applying classical planning techniques for decision support and assistive agents.

Acknowledgments

We thank the anonymous reviewers for comments that helped improve the paper. The authors also thank AFOSR and NRL for funding this research.

References

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1):281–300.

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 12–21.

Breiman, L. 2001. Random forests. *Machine Learning* 45(1):5–32.

Bryce, D. 2014. Landmark-based plan distance measures for diverse planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Fikes, R. E., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3):189–208.

Gupta, N., and Nau, D. S. 1992. On the complexity of blocks-world planning. *Journal of Artificial Intelligence* 56(2):223–254.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22(1):215–278.

Jaidee, U.; Muñoz-Avila, H.; and W. Aha, D. 2011. Integrated learning for goal-driven autonomy. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2450–2455.

Kabanza, F.; Bellefeuille, P.; Bisson, F.; Benaskeur, A. R.; and Irandoust, H. 2010. Opponent behaviour recognition for real-time strategy games. In *Proceedings of the 5th AAAI Conference on Plan, Activity, and Intent Recognition (PAIR)*, AAAIWS'10-05, 29–36. AAAI Press.

Kautz, H. A., and Allen, J. F. 1986. Generalized plan recognition. In *Proceedings of 5th National Conference on Artificial Intelligence (AAAI)*, 32–37.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 154–162.

Klenk, M.; Molineaux, M.; and Aha, D. W. 2013. Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence* 29:187–206.

McDermott, D. 1999. Using regression-match graphs to control search in planning. *Artificial Intelligence* 109(1–2):111–159.

Pozanco, A.; Yolanda, E.; Fernández, S.; and Borrajo, D. 2018. Counterplanning using goal recognition and landmarks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 4808–4814.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Ramırez, M., and Geffner, H. 2009. Plan recognition as planning. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCAI)*, 1778–1783.

Ramırez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*, 1121–1126.

Schmidt, C. F.; Sridharan, N.; and Goodson, J. L. 1978. The plan recognition problem: An intersection of psychology and artificial intelligence. *Artificial Intelligence* 11(1-2):45–83.

Vered, M.; Pereira, R. F.; Magnaguagno, M.; Meneguzzi, F.; and Kaminka, G. A. 2018. Online goal recognition as reasoning over landmarks. In *The AAAI 2018 Workshop on Plan, Activity, and Intent Recognition (PAIR)*.

Towards an argumentation-based approach to explainable planning

Anna Collins, Daniele Magazzeni and **Simon Parsons**
Department of Informatics, King's College London
{anna.collins,daniele.magazzeni,simon.parsons}@kcl.ac.uk

Abstract

Providing transparency of AI planning systems is crucial for their success in practical applications. In order to create a transparent system, a user must be able to query it for explanations about its outputs. We argue that a key underlying principle for this is the use of causality within a planning model, and that argumentation frameworks provide an intuitive representation of such causality. In this paper, we discuss how argumentation can aid in extracting causalities from plans and models, and how they can create explanations from them.

1 Introduction

Explainability of AI decision-making is crucial for increasing trust in AI systems, efficiency in human-AI teaming, and enabling better implementation into real-world settings. Explainable AI Planning (XAIP) is a field that involves explaining AI planning systems to a user. Approaches to this problem include explaining planner decision-making processes as well as forming explanations from the models. Past work on model-based explanations includes an iterative approach (Smith 2012) as well as using explanations for more intuitive communication with the user (Fox, Long, and Magazzeni 2017). With respect to human-AI teaming, the more helpful and illustrative the explanations, the better the performance of the system overall.

Research into the types of questions and motivations a user might have includes work with *contrastive questions* (Miller 2018). These questions are structured as *'Why F rather than G?'*, where F is some part (i.e. action(s) in a plan) of the original solution and G is something the user imagines to be better. While contrastive questions are useful, they do not consider the case when a user doesn't have something else in mind (i.e. G) or has a more general question about the model. This includes the scenario in which the user's understanding of the model is incomplete or inaccurate. Research in the area of model reconciliation attempts to address this knowledge gap (Chakraborti et al. 2017).

More broadly, questions such as *'Why A?'*, where A is an action in the plan, or *'How G?'*, where G is a (sub)goal, must be answerable and explainable. Questions like these are inherently based upon definitions held in the domain related to a particular problem and solution. The user's motivation behind such questions can vary: he could think the action is unnecessary, be unsure as to its effects, or think there is a better option. Furthermore, questions regarding particular state information may arise, such as *'Why A here?'* and *'Why can't A go here?'*. For these, explanations that include relevant state information would vastly improve their efficiency when communicating with a user (Miller 2018). This is especially true for long plans, when a user does not have access to a domain, or the domain is too complex to be easily understood. Thus, extracting relevant information about action-state causality from the model is required.

In the space of planning, causality underpins a variety of research areas including determining plan complexity (Giménez and Jonsson 2008) and heuristics (Helmert 2004). Many planners also can create causal graph visualizations of plans for a user to interact with (Pearl 2014). The general structure of causality in planning is 'action causes state'. Indirectly, this can be seen as 'action enables action', where the intermediary state is sufficient for the second action to occur. Hilton describes different 'causal chains' which mirror the types of causality found in planning; action-state causality can be identified as either a 'temporal' or 'unfolding' chain, while action-action causality is similar to an 'opportunity chain'(Hilton, McClure, and Slugoski 2005). For now, we will focus on these two types of general causality.

To represent the causality of a model, argumentation is a good candidate; as detailed by (Bochman 2005), argumentation frameworks and causal models can be viewed as two versions of one entity. A recent related work uses argumentation for explainable scheduling (Cyras et al. 2019). We consider an ASPIC$^+$ (Modgil and Prakken 2013) style framework with defeasible rules capturing the relationships between actions in a plan and strict rules capturing action-state causality. This structure allows more than a causal representation of a plan; it allows multiple types of causality to be distinguished and different causal 'chunks' to be created and combined to be used as justification for explanations.

In this paper we present an initial approach for using argumentation to represent causality, which can then be used to form more robust explanations. In the following sections, a motivating scenario will be introduced and used to showcase our current approaches of abstracting causalities and state information into argumentation frameworks.

2 Motivating Example

Consider a simple logistics scenario in which three trucks are tasked with delivering three packages to different locations. The user analyzing the planner output has the plan as well as a general, non-technical understanding of the model and the goals of the problem; the user knows that trucks can move between certain waypoints that have connecting roads of differing lengths, there are refueling stations at waypoints B and E, and some subgoals of the problem are to have $package\,1$ delivered to $waypoint\,C$, $package\,2$ delivered to $waypoint\,G$, and $package\,3$ delivered to $waypoint\,D$. The user is also aware that the three trucks and three packages are at $waypoint\,A$ in the initial state. A basic map of the domain and plan are shown in Figures 1 and 2, respectively.

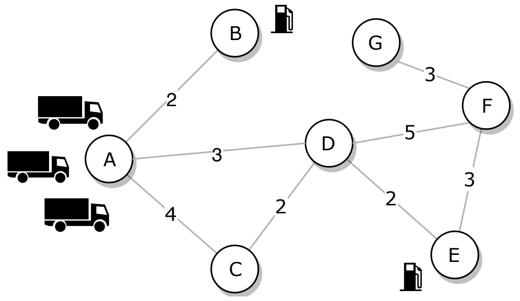


Figure 1: Example Domain Map

```
0.000: (load_truck t1 p1)     [1.000]
0.000: (load_truck t2 p3)     [1.000]
0.000: (drive_truck t3 wpB)   [2.000]
1.001: (load_truck t1 p2)     [1.000]
1.001: (drive_truck t2 wpD)   [3.000]
2.001: (drive_truck t1 wpC)   [4.000]
2.001: (refuel_truck t3)      [5.000]
4.002: (unload_truck t2 p3)   [1.000]
5.002: (unload_truck t1 p1)   [1.000]
6.003: (drive_truck t1 wpD)   [2.000]
8.004: (drive_truck t1 wpE)   [2.000]
10.005: (refuel_truck t1)     [5.000]
15.006: (drive_truck t1 wpF)  [3.000]
18.007: (drive_truck t1 wpG)  [3.000]
21.008: (unload_truck t1 p2)  [1.000]
```

Figure 2: Example Plan

Even with a simple and intuitive problem such as this, questions may arise which cannot be answered trivially. One such question is *'Why drive truck 1 to waypoint E?'*. Addressing this question requires the causal consequence of applying the action; in other words, how does driving *truck 1* to *waypoint E* help in achieving the goal(s)?

As discussed previously, tracking state information throughout a plan can be useful for explanations. This is especially true when values of state variables are not obvious at any given point in a plan and their relevance to a question is not known. A question such as *'Why drive truck 3 to waypoint B?'* has this property. These two questions will be addressed in the following sections.

3 Background on Argumentation

As mentioned above, in this paper we will make use of ASPIC$^+$ as the underlying argumentation system from which explanations are constructed. However, what we are suggesting is not limited to ASPIC$^+$; we can imagine using most formal argumentation systems to reason in this way. For a full description of ASPIC$^+$ see (Modgil and Prakken 2013). In this paper we only make use of the ability to construct arguments, and so that is the only aspect of the system that we describe.

We start with a language $\mathcal{L}$, closed under negation. A reasoner is then equipped with a set Rules of strict rules, denoted $\phi_1, \ldots, \phi_n \rightarrow \phi$, and defeasible rules, denoted $\phi_1, \ldots, \phi_n \Rightarrow \phi$, where $\phi_1, \ldots, \phi_n, \phi$ are all elements of $\mathcal{L}$. A knowledge base Δ is then a set of elements K from $\mathcal{L}$ and a set Rules. From Δ it is possible to construct a set of arguments $\mathcal{A}(\Delta)$, where an argument A is made up of some subset of K, along with a sequence of rules, that lead to a conclusion. Given this, $\texttt{Prem}(\cdot)$ returns all the premises, $\texttt{Conc}(\cdot)$ returns the conclusion and $\texttt{TopRule}(\cdot)$ returns the last rule in the argument. An argument A is then:

- ϕ if $\phi \in K$ with: $\texttt{Prem}(A) = \{\phi\}$; $\texttt{Conc}(A) = \phi$; $\texttt{Sub}(A) = \{A\}$; and $\texttt{TopRule}(A)$ = undefined.
- $A_1, \ldots, A_n \rightarrow \phi$ if A_i, $1 \leq i \leq n$, are arguments and there exists a strict rule of the form $\texttt{Conc}(A_1), \ldots, \texttt{Conc}(A_n) \rightarrow \phi$ in Rules. $\texttt{Prem}(A) = \texttt{Prem}(A_1) \cup \ldots \cup \texttt{Prem}(A_n)$; $\texttt{Conc}(A) = \phi$; and $\texttt{TopRule}(A) = \texttt{Conc}(A_1), \ldots, \texttt{Conc}(A_n) \rightarrow \phi$.
- $A_1, \ldots, A_n \Rightarrow \phi$ if A_i, $1 \leq i \leq n$, are arguments and there exists a defeasible rule of the form $\texttt{Conc}(A_1), \ldots, \texttt{Conc}(A_n) \Rightarrow \phi$ in Rules. $\texttt{Prem}(A) = \texttt{Prem}(A_1) \cup \ldots \cup \texttt{Prem}(A_n)$; $\texttt{Conc}(A) = \phi$; and $\texttt{TopRule}(A) = \texttt{Conc}(A_1), \ldots, \texttt{Conc}(A_n) \Rightarrow \phi$.

Then, given $K = \{a; b\}$ and Rules = $\{a \rightarrow c; b, c \Rightarrow d\}$, we have the following arguments:

$$\begin{aligned} &A_1 : a \\ &A_2 : b \\ &A_3 : A_1 \rightarrow c \\ &A_4 : A_2, A_3 \Rightarrow d \end{aligned}$$

When applied to planning, these arguments define a subsection of a causal chain, as will be described below.

4 Tracing Causality

In order to utilize causality in explanations, the causal links between actions in a plan need to be extracted and abstracted into a framework. This process is planner-independent, so it requires only the plan, problem, and domain as inputs. An algorithm is used to extract the causalities which then form a knowledge base of causal links. This can then be used by an argumentation engine to construct arguments representing the causal 'chunks' in a plan. From this, questions of the forms *'Why A?'* and *'How G?'* can be addressed. This process is described in the following sections.

4.1 Extracting causalities from a plan

To extract causal relationships between actions in a plan, an algorithm similar to the one used in (Chrpa and Barták 2008) for detecting action dependencies is utilized:

1. Finds connections between one action's effects and another's preconditions from the domain to form a knowledge base. In general terms we can think of these chunks as being statements in some logical language of the form:

$$a \Rightarrow b$$
$$b, c \Rightarrow d$$

   which denote the statements 'a enables b' and 'b and c together enable d' where a, b, c, d are actions in a plan.

2. Finds the subgoals, if any, that are satisfied by these causal links

Thus, part of our logistics example could be translated into the causal knowledge base:

$$\begin{aligned}&((load\ truck\ t1\ p1),\\ &(drive\ truck\ t1\ wpC)) \Rightarrow (unload\ truck\ t1\ p1)\\ &(drive\ truck\ t1\ wpC) \Rightarrow (drive\ truck\ t1\ wpD)\\ &(unload\ truck\ t1\ p1) \Rightarrow p1\ at\ wpC\\ &(drive\ truck\ t1\ wpD) \Rightarrow (drive\ truck\ t1\ wpE)\end{aligned}$$

4.2 Forming arguments

Given a knowledge base, the argumentation engine can construct a sequence of arguments with defeasible rules:

$$\begin{aligned}A_1 :&(load\ truck\ t1\ p1)\\ A_2 :&(drive\ truck\ t1\ wpC)\\ A_3 :&A_1, A_2 \Rightarrow (unload\ truck\ t1\ p1)\\ A_4 :&A_3 \Rightarrow p1\ at\ wpC\\ A_5 :&A_2 \Rightarrow (drive\ truck\ t1\ wpD)\\ A_6 :&A_5 \Rightarrow (drive\ truck\ t1\ wpE)\\ A_7 :&A_6 \Rightarrow (refuel\ truck\ t1)\\ A_8 :&A_7 \Rightarrow (drive\ truck\ t1\ wpF)\\ A_9 :&A_8 \Rightarrow (drive\ truck\ t1\ wpG)\\ A_{10} :&A_9 \Rightarrow (unload\ truck\ t1\ p2)\\ A_{11} :&A_{10} \Rightarrow p2\ at\ wpG\end{aligned}$$

These summarize the causal structure of part of the plan (i.e. a 'causal chunk' as defined in Secion 4.3), summarized in argument A_{11}, which can then be presented to a user who is seeking explanations. A visualization of these arguments can be seen in Figure 3.

4.3 Using causal chunks for explanation

We define the notion of a causal 'chunk' as any subsection(s) of the causal chain(s) extracted from the plan or model and then combined. Intuitively, these chunks can focus on one 'topic' (e.g. state variable, object instance) to provide a higher-level abstraction of causality rather than just the individual causal links. The argument A_{11} which represents such a causal chunk shows only the action-action causalities

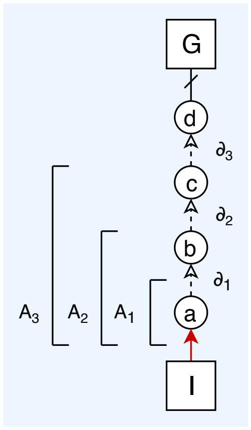


Figure 3: I is the initial state, G is the goal state, a, b, c, d are (not necessarily sequential) actions in a plan, δ_i are the defeasible rules from the knowledge base, and A_1, A_2, A_3 are arguments formed from these actions and rules.

(i.e. from just one causal chain) involving the object *truck 1*. These chunks are created by searching through the `Rules` of the framework for those pertaining to a specific 'topic'.

Given arguments such as A_{11}, we propose two methods of structuring explanations. The first method is allowing the user to engage the system in a dialogue. For our example, the question, *'Why e?* where e is the action of driving *truck 1* to *waypoint E* could be used to query the system:

`why` e

Following work such as (Parsons, Wooldridge, and Amgoud 2003), the system replies to this query by building an argument for e, in this case A_6, and using this to provide a suitable response, which might be by returning `Conc`(A_5), since $A_5 \Rightarrow e$. Thus the system could reply with:

d, `which leads to` e

where d is *drive truck t1 wpD*. The user could then continue to unpack the causal chunk by asking:

`why` d

and so on. This would provide the user with the causalities which enabled action e to be applied. The same could be done using a forward approach where the argument A_6 is expanded until a subgoal is reached, if possible (e.g. A_{11}). The user can then ask:

`why` e

and the system responds with:

e `leads to` f

as in $A_7 : A_6 \Rightarrow f$. Iteratively, this would show how e leads to some goal or subgoal. Reversing this process will also explain how a goal is reached.

The second method of structuring explanations is detailed in Section 5.2, and can be applied to this example similarly.

5 Extracting State Information

Using a similar method as above, causalities held within the state space of the plan are extracted and represented as a knowledge base. An algorithm is used that iterates through the effects of actions from a plan and extracts the state variables they alter. They can then be used to answer questions such as *'Why A here?'* and *'Why can't A go here?'*. In general terms, we define these dependencies as being statements in some logical language of the form:

$$\begin{aligned} &x_0, y_0, z_0 \\ &\qquad a \rightarrow \Delta x_a \\ &\qquad b \rightarrow \Delta y_b; b \rightarrow \Delta z_b \\ &x_f, y_f, z_f \end{aligned}$$

which denote the statements 'a causes Δx_a' and 'b causes Δy_c and Δz_c'. Here, a, b are actions in the plan, and x, y, z are state variables. The x_0, y_0, z_0 denote the values of those variables in the initial state while x_f, y_f, z_f denote the final values in the goal state; Δx_a denotes the change in x after applying action a.

Applying this to our logistics example and the question, *'Why drive truck 3 to waypoint B?'*, these strict rules are relevant:

$$\begin{gathered} t3\, fuel\, is\, 2 \\ (drive\, truck\, t3\, wpB) \rightarrow t3\, fuel\, decrease\, 2 \\ (refuel\, truck\, t3) \rightarrow t3\, fuel\, increase\, 25 \\ t3\, fuel\, is\, 25 \end{gathered}$$

From these, it is clear the truck's fuel level is too low in the initial state to go anywhere besides waypoint B (see Figure 1). However, it is not clear why the truck does not just stay put. Alone, these rules do not provide a full explanation, but they can be added to the action-action causal chains for more complete explanations.

5.1 Combining different forms of causality

When used in conjunction, the causal traces and opportunity traces form a strong basis of justification for an explanation (see Figure 4 for a visual representation). Using the example from before, the relevant defeasible rules from the causal chain are:

$$\begin{gathered} (drive\, truck\, t3\, wpB) \Rightarrow (refuel\, truck\, t3) \\ (refuel\, truck\, t3) \Rightarrow t3\, fuel\, > 5 \end{gathered}$$

where the conclusion of the second rule is a subgoal of the problem, perhaps previously unknown to the user. That is, because the problem requires all trucks to have a minimum amount of fuel at the end, truck 3 had to refuel but could not deliver any packages due to its low initial fuel amount. Thus, combining arguments from both types of causal chains more aptly answers this question.

A method for seamlessly creating explanations from this structure is an intended future work. For now, it is possible to extract both the defeasible rules and strict rules governing the causal effects related to a specific topic and present them to a user. How to determine which rules are relevant to a specific user question and how to combine the rules to form higher-level causal chunks are ongoing works.

One possible method of creating relevant causal chunks is to extract *all* rules related to a specific 'topic' (e.g. state

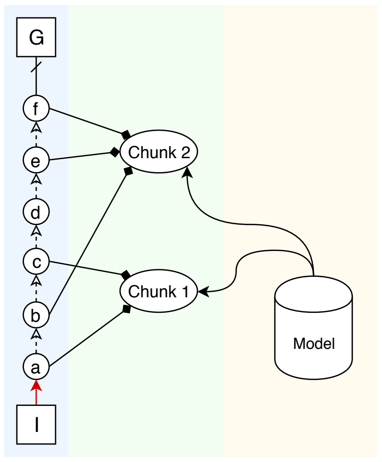


Figure 4: Rules from both the plan layer (blue) and the model layer (yellow) are combined to form the causal 'chunks'.

variable). For the variable 't3 fuel', all actions which alter it will be extracted along with any actions that enable the altering actions from the defeasible rules. Additionally, any (sub)goals containing 't3 fuel' will be extracted. Together, these form a chunk representing the causes of changes to 't3 fuel' as well as its relationship to the (sub)goals. The arguments below represent the causal 'chunk':

A_1 : $t3\, fuel\, is\, 2$

A_2 :$A_1 \Rightarrow ((drive\, truck\, t3\, wpB) \rightarrow t3\, fuel\, decrease\, 2)$

A_3 :$A_2 \Rightarrow ((refuel\, truck\, t3) \rightarrow t3\, fuel\, increase\, 25)$

A_4 :$A_3 \Rightarrow t3\, fuel\, > 5$

where the conclusion of A_3 is a subgoal of the problem.

5.2 More forms of explanation

When unpacked iteratively, the arguments in the causal chunk centred on 't3 fuel' would give a similar output explanation as in the example in Section 4.3. For example, a user asking the question *'Why b?'* where *b* is the action *(drive truck 3 to waypoint B)* would either receive the response:

t3 fuel is 2 `enables` *b*

or the response:

b `causes` *t3 fuel decrease 2* `and enables` *c*

if using a forward chaining approach, where c is the premise of the conclusion of A_2, *(refuel truck t3)*. This process would continue until the subgoal *t3 fuel >5* is reached. However, identifying what state variables are relevant given a user question is not trivial. The question *'Why drive truck 3 to waypoint B?'* has no mention of the truck's fuel, so its relevance must be deduced from the plan, problem and domain.

Another method of providing explanations is through a graph structure, as depicted in Figure 5. Given a query, the relevant causal chunks would be identified and represented in the graph with individual actions and state changes as nodes and the causal rules between them as edges. This approach could also help explain question of the form, *Why* *can't A go here?*, as inapplicable actions (ones not in the plan) can be shown. Developing a robust system such as this is important future work.

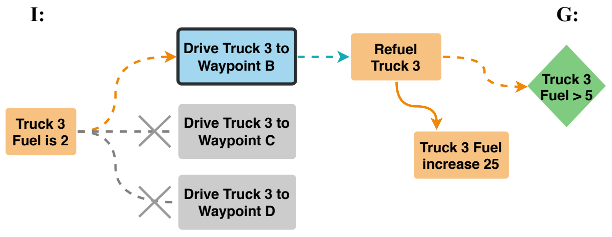

I:
G:
Drive Truck 3 to Waypoint B
Refuel Truck 3
Truck 3 Fuel > 5
Truck 3 Fuel is 2
Drive Truck 3 to Waypoint C
Truck 3 Fuel increase 25
Drive Truck 3 to Waypoint D

Figure 5: An example graph with the queried action in blue and nodes contained in the 't3 fuel' chunk in orange, and I and G the initial and goal states. Dashed edges denote defeasible rules; solid edges denote strict rules.

6 Discussion

We acknowledge that this is a preliminary step and more work is required to expand on the ideas presented in this paper. One such future work involves defining exactly what questions, which range from action-specific to model-based, can be answered and explained using our approach. Also, how these questions are captured from a user is an open question. The query, *'Why didn't truck 3 deliver any packages?'* can be answered using the causal information captured in the framework, but how one converts this question to a form that the system understands requires further research. Potential methods for communicating a user question include a dialogue system or Natural Language Processing techniques.

Along with expanding the set of questions that can be addressed, extensions to the argumentation framework itself should be considered. Better methods for creating causal 'chunks' for specific user questions are needed. It may be advantageous to use argumentation schemes to help identify relevant topics of chunks and which causal chains should be included from the framework. This relates to the idea of 'context' and identifying the motivation of a question. If the system can be more precise in extracting the relevant information, the explanations themselves will be more effective.

Related to this is the need to explore other ways of presenting an explanation to a user. Research into the efficacy of explanations and how to properly assess the effectiveness of the explanations in practice are future areas of research, and will require user studies. Our starting point will be the approach outlined in Section 4.3 which has been shown empirically to be effective in contexts such as human-robot teaming (Sklar and Azhar 2015).

7 Conclusion

In this paper we proposed an initial approach to explainable planning using argumentation in which causal chains are extracted from a plan and model and abstracted into an argumentation framework. Our hypothesis is that this allows ease of forming and communicating explanations to a user. Furthermore, causal 'chunks' can be created by combining relevant causal links from the chains which explain the causalities surrounding one 'topic'. We believe these help with making more precise explanations, and that chunks can be used to provide hierarchical explanations. Overall, the approach is a first step towards exploiting the intuitive functionality of argumentation in order to use causality for explanations.

Acknowledgements This work was partially supported by EPSRC grant EP/R033722/1, Trust in Human-Machine Partnership, and by a PhD studentship from the Faculty of Natural and Mathematical Sciences at King's College London.

References

Bochman, A. 2005. Propositional argumentation and causal reasoning. In *IJCAI*.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *IJCAI*.

Chrpa, L., and Barták, R. 2008. Towards getting domain knowledge: Plans analysis through investigation of actions dependencies. In *International Florida Artificial Intelligence Research Society Conference*.

Cyras, K.; Letsios, D.; Misener, R.; and Toni, F. 2019. Argumentation for explainable scheduling. In *https://arxiv.org/abs/1811.05437*.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable planning. *IJCAI workshop on Explainable AI* abs/1709.10256.

Giménez, O., and Jonsson, A. 2008. The complexity of planning problems with simple causal graphs. *J. Artif. Intell. Res.* 31:319–351.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *ICAPS*.

Hilton, D. J.; McClure, J. L.; and Slugoski, B. R. 2005. The course of events: Counterfactuals, causal sequences and explanation. In *The Psychology of Counterfactual Thinking*.

Miller, T. 2018. Contrastive explanation: A structural-model approach. *arXiv preprint arXiv:1811.03163*.

Modgil, S., and Prakken, H. 2013. A general account of argumentation with preferences. *Artificial Intelligence* 195:361–397.

Parsons, S.; Wooldridge, M.; and Amgoud, L. 2003. Properties and complexity of formal inter-agent dialogues. *Journal of Logic and Computation* 13(3):347–376.

Pearl, J. 2014. Graphical models for probabilistic and causal reasoning. In *Computing Handbook, Third Edition: Computer Science and Software Engineering*. 44: 1–24.

Sklar, E. I., and Azhar, M. Q. 2015. Argumentation-based dialogue games for shared control in human-robot systems. *Journal of Human-Robot Interaction* 4(3):120–148.

Smith, D. 2012. Planning as an iterative process. In *AAAI*.

Human-Understandable Explanations of Infeasibility for Resource-Constrained Scheduling Problems

Niklas Lauffer, Ufuk Topcu [*]
{nlauffer, utopcu}@utexas.edu
University of Texas, Austin

Abstract

Significant work has been dedicated to developing methods for communicating reasons for decision-making within automated scheduling and planning systems to human users. However, much less focus has been placed on communicating reasons for why scheduling systems are unable to arrive at a feasible solution when over-constrained. We investigate this problem in the context of task scheduling. We introduce the agent resource-constrained project scheduling problem (ARCPSP), an extension of the resource-constrained project scheduling problem which includes a conception of agents that execute tasks in parallel. We outline a generic framework, based on efficiently enumerating minimal unsatisfiable sets (MUS) and maximal satisfiable sets (MSS), to produce small descriptions of the source of infeasibility. These descriptions are supplemented with potential relaxations that would fix the infeasibility found within the problem instance. We illustrate how this method may be applied to the ARCPSP and demonstrate how to generate different types of explanations for an over-constrained instance of the ARCPSP.

1 Introduction

In many real-world applications, human users in charge of developing plans and making decisions are aided by automated planning and scheduling systems. For example, NASA mission planning makes use of a large team of human planners that use various automated scheduling systems in order to construct day-to-day as well as long-term plans for crew members. A primary function of these automated systems is generating different types of plans and schedules while ensuring that various constraints do not conflict. When plans are ultimately constructed by human planners for a human crew, it is essential for both the planners, and the crew executing the plans, to understand how and why certain scheduling decisions were made by automated tools. In general, when the primary function of such constraint satisfaction and optimization tools is to support human decision-making, it is necessary for the automated systems to be transparent in how they arrive at certain outputs.

Significant work has been dedicated to generating human-understandable explanations for why certain automated planning decisions were made (Seegebarth et al. 2013). However, little work has been done in generating reasons for why plans or schedules *cannot* be generated under certain specifications. Human users interacting with such constraint satisfaction or optimization tools are bound to run into configurations for which no feasible solution exists. Fixing infeasible configurations is a challenging task for the human user if they are unable to understand why the solver arrives at an unsatisfiable conclusion.

While various partial constraint satisfaction tools exist for solving such over-constrained problems (Freuder and Wallace 1996), solutions employing these tools have significant limitations that make them less applicable in certain real-life scenarios. Most of these methods employ constraint hierarchies to determine which constraints should be violated in order to satisfy more important ones. However, in complicated planning or scheduling applications involving multiple human agents, constructing such a hierarchy is often impractical. Instead, if reasons for infeasibility can be properly conveyed back to the human user, they can make high-level decisions to solve infeasibility in any way they see fit.

In this paper, we provide a framework for iteratively generating human-understandable *explanations* of infeasibility for a specific class of scheduling problems. These explanations manifest themselves as minimal sets of specifications (or constraints) that are responsible for causing infeasibility, coupled with suggestions for relaxations through which feasibility could be achieved.

The method proposed in this paper allows users to enumerate over a series of explanations for infeasible instances of problems at varying levels of abstraction. For example, raw explanations of relevant low-level constraints may be directly output or a causal link may be established back to higher level descriptions of the problem to understand what specifications were responsible for the feasibility issue. This system also allows directed questions about feasibility to be asked, such as "why can task A not be scheduled after task B?"

A strategy for iteratively generating minimal unsatisfiable sets (MUS) and maximal satisfiable sets (MSS) forms the basis for interpreting the infeasibility of the problem. Existing methods such as QuickXplain (Junker 2004) focus on generating a single most preferable explanation of infeasibility. Likewise, (Burt, Klimova, and Primas 2018) aims to generate a single explanation in the context of optimization

[*]This work has been supported in part by the grants NASA NNX17AD04G and NSF 1652113.

without attempting to achieve minimality. However, over-constrained problems may contain several infeasibility issues which cannot be solved by changing only a single part of the problem. So, because a single MUS only provides indication of a single feasibility issue, we aim to enumerate several sets of MUS to highlight multiple feasibility issues found within the problem instance. Therefore, the proposed enumeration strategy is based on MARCO (Liffiton et al. 2016), a flexible algorithm for generating MUSes and MSSes in succession.

Motivated by the domain of space mission scheduling, we introduce and investigate the agent resource-constrained project scheduling problem (ARCPSP), an extension of the resource-constrained project scheduling problem (RCPSP) that incorporates the delegation of tasks to differing agents. This problem cannot be framed as an instance of the RCPSP because it deals with the case of asymmetric agents in which certain tasks may only be executed by a subset of the agents. This problem is meant to model applications in which efficient scheduling for teams of differing agents is critical. While we only explicitly investigate this problem, the generality of the approach outlined in this paper would allow the methodology to be adapted for different types of constraint satisfaction and optimization tools as well as different types of planning and scheduling problems.

The main contributions of this paper are the following: firstly, we provide a formal definition of the agent resource-constrained project scheduling problem (ARCPSP) in Section 3. Then in Section 4 we outline a difference logic encoding of the ARCPSP which is used to check feasibility of problem instances. The framework for generating human-understandable explanations of infeasibility for instances of the ARCPSP is described in Section 5. Finally, we provide an overview of the trade-off between interpretability and expressibility of different types of explanations and conclude by discussing how these ideas can be extended.

2 Preliminaries and Definitions

In this section, we introduce relevant background information and definitions used throughout the paper. These concepts will set the stage for formulating the ARCPSP in terms of satisfiability modulo theory and using minimal unsatisfiable sets and maximal satisfiable sets to generate explanations of infeasibility.

2.1 Boolean Satisfiability

Let X be a set of variables and clauses $C_1, \dots, C_n$ be formulas representing constraints over X. Consider a formula of the form

$$\varphi = \bigwedge_{i=1,\dots,n} C_i. \tag{1}$$

We say the formula φ is *satisfiable* if there exists some assignment to the variables in X which makes φ evaluate to TRUE. Otherwise, it is *unsatisfiable*. Note that if φ takes the form of equation (1), as it does throughout this paper, every clause C_i must be TRUE in order for φ to evaluate to TRUE. To implement the temporal constraints within a schedule, the clauses C_i are taken from the theory of *difference logic* (DL), which makes deciding φ a satisfiability modulo theory (SMT) problem. To check satisfiability of problem instances, we use the Microsoft Z3 SMT solver (De Moura and Bjørner 2008).

2.2 Difference Logic

As will be discussed in Section 4, the agent resource-constrained project scheduling problem (ARCPSP) can be encoded in *difference logic* (DL), a fragment of linear real arithmetic (LRA). The numerical components of DL are solvable in polynomial time (Cotton and Maler 2006) using graph-based procedures based on an incremental Bellman-Ford algorithm. In general, decidability for DL using these methods is more efficient than the simplex-based methods used to decide LRA. Under DL, atoms are restricted to the form

$$x - y \leq k \quad \text{for} \quad x, y \in X,\ k \in \mathbb{R}. \tag{2}$$

However, we can rewrite the following atoms in difference form:

- $x - y \geq k \quad \equiv \quad (y - x \leq -k)$
- $x - y = k \quad \equiv \quad (x - y \leq k) \wedge (x - y \geq k)$
- $x = y \quad \equiv \quad x - y = 0$

Bounds $x \leq k$ can also be incorporated by writing them as $x - x_0 \leq k$ where x_0 is a special variable that is later set to zero.

2.3 Minimal Unsatisfiable and Maximal Satisfiable Sets

Definition 1. A *minimal unsatisfiable set* (MUS) of a set C of constraints is a subset $M \subseteq C$ such that M is unsatisfiable and every proper subset $M' \subset M$ is satisfiable. A *maximal satisfiable set* (MSS) of a set C of constraints is a subset $M \subseteq C$ such that M is satisfiable and every proper superset M', with $C \supseteq M' \supset M$, is unsatisfiable. A *minimal correction set* (MCS) of a set C of constraints is the complement of some maximal satisfiable set of C, and can be understood as a minimal set of constraints which need to be removed from C in order to make it satisfiable.

It is important to note that MUSes, MSSes, and MCSes are only *locally* maximal (or minimal), and are different from concepts of globally optimal subsets. MUSes can be understood as isolated, infeasible subsets of the constraints. Their primary characteristic is that removing any single constraint would make the set satisfiable. However, this does not necessarily guarantee the feasibility of the entire set of constraints because there might be many disjoint MUSes within the set. In order to make the entire set feasible (satisfiable), a *hitting set* of the MUSes must be removed. Every MCS is precisely one combination of such a hitting set.

Definition 2. A *background* of a set C of constraints is a subset $B \subseteq C$ of hard constraints, which must be necessarily satisfied. In the context of scheduling problems, backgrounds typically include constraints that ensure that the outcome of the schedule is logical, including conditions such as tasks not overlapping and resource constraints not being exceeded. We denote everything outside of the background

$M \setminus B$ as the *foreground*. Hence, the background and foreground partition the set C of constraints.

A *minimal conflict* of an over-constrained set C of constraints with respect to a background B is then a subset of the foreground $M \subset C \setminus B$ such that $M \cup B$ is unsatisfiable and, for any superset $M' \supset M$, $M' \cup B$ is satisfiable. A *minimal relaxation* of an over-constrained set C of constraints with respect to a background B is a subset of the foreground $M \subset C \setminus B$ such that $(C \setminus M) \cup B$ is satisfiable and, for any superset $M' \supset M$, $(C \setminus M') \cup B$ is unsatisfiable. Then an *explanation* is a sequence of minimal conflicts and minimal relaxations for a problem instance.

The definitions of minimal conflicts and minimal relaxations mirror the concepts of MUSes and MCSes, respectively, while incorporating a background of constraints which cannot be modified. A background is necessary for specifying hard constraints which cannot be relaxed or modified. This way we can prevent certain constraint from consideration for conflicts or relaxations. A background also allows the generation of explanations concerning different aspects of a scheduling problem instance, a concept which will be explored later in the paper.

3 Problem Description

The problem that we formulate is an extension of the resource-constrained project scheduling problem (RCPSP). Loosely, the RCPSP considers nonpreemptive, precedence-constrained tasks of known durations that are constrained by reusable resource requirements (i.e. resources that are returned after a task stops using them). The agent resource-constrained project scheduling problem extends the RCPSP to include a set number of agents that execute the tasks in parallel, subject to certain compatibility constraints. Additionally, while the RCPSP generally cares about optimizing the total makespan of the schedule, we instead introduce a set start and end time for each scheduling instance and only focus on its feasibility (i.e. whether or not all tasks can be completed within this specified time frame).

3.1 The Agent Resource-Constrained Project Scheduling Problem

An instance of an agent resource-constrained project scheduling problem (ARCPSP) is defined by a tuple $(M, J, s, p, U, E, R, B, b)$, where the components are defined as follows.

- $M = \{M_1, M_2, \cdots, M_m\}$ is a set of agents.
- $J = \{J_1, J_2, \cdots, J_n\}$ is a set of non-preemptive (uninterruptible) tasks.
- $s = [(a_1, b_1), (a_2, b_2), \cdots, (a_n, b_n)]$ are the allowable time ranges in which the tasks should be executed, where $a_i, b_i \in \mathbb{N}$.
- $p = [p_1, \dots, p_n]$ is a vector of the durations of tasks J, where p_i is the duration of task J_i.
- $U = \{U_1, U_2, \cdots, U_n\}$ is the compatibility set for the tasks. Each task J_i can be completed by a subset $U_i \subseteq M$ of agent.
- $E \subseteq J \times J$ is a set of precedence relations. $(J_i, J_j) \in E$ if and only if task J_i must terminate before task J_j begins. Precedence relations must be defined in a consistent way (by respecting the transitive property).
- $R = \{R_1, R_2, \cdots, R_q\}$ is a set of reusable resources.
- $B \in \mathbb{N}^q$ represents the total availability of the resources R. The tasks that share resource B_i are *mutually exclusive* if $B_i = 1$.
- $b \in \mathbb{N}^{n \times q}$ represents the resource demands of tasks where task J_i requires $b_{i,j}$ units of resource R_j during its execution. The total demand of resource R_j at anytime cannot exceed its total availability B_j.

A *schedule* $(S, A) = (\{S_1, S_2, \cdots, S_n\}, \{A_1, A_2, \cdots, A_n\})$ is a solution to an instance of an ARCPSP, where S_i and A_i are the start time and the assigned agent of task J_i, respectively. A schedule is *feasible* if it satisfies the following constraints:

- No agent has overlapping tasks,
$$\big(S_i + p_i \leq S_j\big) \lor \big(S_j + p_j \leq S_i\big) \tag{3}$$
$\forall i, j \in (1, \dots, n)$ such that $A_i = A_j$.
- Every task falls within its allowable time frame
$$\big(s_{i,1} \leq S_i\big) \land \big(S_i + p_i \leq s_{i,2}\big) \quad \forall S_i \in S. \tag{4}$$
- The activities are assigned to compatible agents
$$M_i \in U_i \quad \forall M_i \in A. \tag{5}$$
- The precedence relations are met
$$S_i + p_i \leq S_j \quad \forall (J_i, J_j) \in E. \tag{6}$$
- The resource constraints are satisfied, let $\mathcal{J}_t = \{J_i \in J \mid S_i \leq t < S_i + p_i\}$ represent the set of tasks being executed at time t, then
$$\sum_{J_i \in \mathcal{J}_t} b_{i,j} \leq Bj \quad \forall R_j \in R, \, \forall t \geq 0. \tag{7}$$

4 SMT Formulation

The constraints of the ARCPSP can be formulated in terms difference logic in the following way. Constraint (3) can be rewritten as

$$(S_i - S_j \leq -p_i) \lor (S_j - S_i \leq -p_j) \lor \neg(A_i = A_j) \tag{8}$$

$\forall S_i, S_j \in S$. Constraint (4) can be rewritten as

$$(s_{i,1} \leq S_i) \land (S_i \leq s_{i,2} - p_i) \quad \forall S_i \in S, \tag{9}$$

and constraint (6) can be rewritten as

$$S_i - S_j \leq p_i \quad \forall (J_i, J_j) \in E. \tag{10}$$

By representing the agents as integers $M_i \in \mathbb{N}$, constraint (5) can be rewritten as

$$\bigvee_{u \in U_i} M_i = u \quad \forall M_i \in A \tag{11}$$

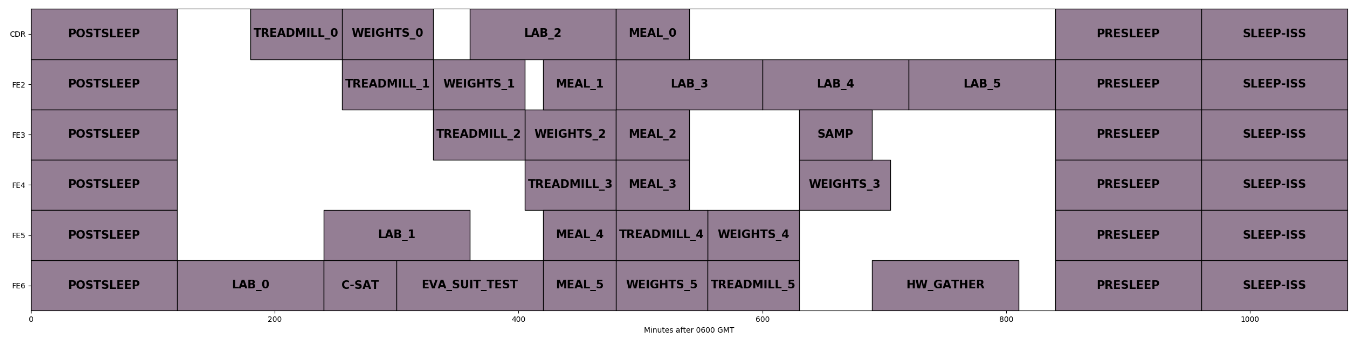


Figure 1: A feasible solution to Example 1.

Encoding the resource constraints is slightly more challenging. For mutually exclusive constraints ($B_i = 1$), the tasks that share a resource can simply be encoded as not being allowed to be executed at the same time. That is, constraint (7) can be rewritten as

$$(S_i - S_j \leq -p_i) \vee (S_j - S_i \leq -p_j) \tag{12}$$

$\forall b_{i,k}, b_{j,k} = 1, \forall R_k \in R$. We can generalize this idea to non-mutually exclusive constraints through the concept of *minimal forbidden sets*. First introduced by (Möhring and Stork 2009), forbidden sets are unsatisfiable sets with respect to resource constraints only. They represent the sets of tasks that cannot be simultaneously scheduled because they would otherwise exceed the availability of some resource constraint. The essential feature of a minimal forbidden set is that a single task can be rescheduled to another time to make the set respect the resource constraint.

Therefore, given a minimal forbidden set J^*, we would like to encode a constraint requiring that they cannot all be executing at the same time

$$\bigvee_{J_i \in J^*} \neg(J_i \in \mathcal{J}_t) \quad \forall t \geq 0 \tag{13}$$

where $\mathcal{J}_t = \{J_i \in J \mid S_i \leq t < S_i + p_i\}$ represents the set of tasks being executed at time t. This encoding is similar to methods which encode the RCPSP in terms of linear arithmetic (Bofill et al. 2016), but this requires discretizing time and incurs a cumbersome number of constraints if there are a large number of time-points. Moreover, equation (13) cannot easily be formulated in terms of difference logic. Instead, we can reformulate the constraint as there being at least two tasks in J^* that do not overlap

$$\bigvee_{J_i \in J^*} \bigvee_{J_j \in J^*} (J_i + p_i \leq J_j) \vee (J_j + p_i \leq J_i) \tag{14}$$

for every minimal forbidden set J^*. This constraint is logically equivalent to requiring that at any time-point, there be at least one task in each minimal forbidden set that is not being executed.

Constraining all of the *minimal forbidden sets*, a subset of all of the forbidden sets, is sufficient to prevent resource conflicts because every forbidden set is a superset of some minimal forbidden set. Algorithms exist for computing all minimal forbidden sets (Stork and Uetz 2005) so we will not discuss such a computation here.

Encoding resource constraints as forbidden sets is efficient in the context of the ARCPSP in comparison to other methods, such as equation (13). This is primarily because of the computational advantage achieved by difference logic over other theories such as linear real arithmetic and the encoding not requiring a discretization of time. Representing resource constraints as minimal forbidden sets also provides an explicit representation of resource constraints in terms of MUSes. If a resource constraint appears in an explanation, we can represent it as the minimal forbidden set which is being violated. For example, if a resource constraint constitutes a component of some MUS, it will be represented as some subset $\{A, B, C\}$ of tasks, meaning that tasks A, B, and C cannot be scheduled at the same time because they would violate a resource constraint.

Example 1. ***Scheduling astronauts aboard the ISS***

We model the problem of scheduling astronauts aboard the International Space Station (ISS) as an instance of the ARCPSP for which the elements of $M = \{M_1, M_2, \cdots, M_m\}$ represent the crew members. We consider the case of $m = 6$ astronauts. The bounds on task execution are from minute 120 to 840; the sleeping related tasks outside of this bound are fixed so are not a part of the problem instance. The availability of the *power* resource, is 1000 units. The tasks are divided into different categories:

- There are 6 laboratory tasks, each of duration 120 with allowable time ranges of $(120, 840)$, the entire work day. However, they have precedence constraints $\{(J_{L_i}, J_{L_{i+1}}) \mid 1 \leq i < 6\}$, each laboratory task must be completed before the next one begins. Each laboratory task can be completed by any astronaut, so the compatibility set is all of the astronauts. Each laboratory task also has a power requirement of 400 units.
- There are m weights and m treadmill tasks, one for each agent, each of duration 75. They have allowable time ranges of $(180, 720)$. Each weight and treadmill task must be completed by a unique astronaut so their compatibility sets can be specified by letting the ith task only be completed by astronaut M_i. However, there is only one set of weights and treadmill equipment, so we can define reusable resources R_W and R_T both with availability

$B_W, B_T = 1$, respectively. Each treadmill task also has a power requirement of 200 units.

– There are m meal tasks. They have allowable time ranges of $(420, 540)$. Similar to the exercise tasks, each one must be completed by a unique astronaut so their compatibility sets can be specified by letting the ith meal task only be completed by astronaut M_i.

– Several miscellaneous tasks, *deploy_cubesat*, *collect_sample*, *hardware_gather*, and *eva_suit_test* with durations 60, 60, 120, and 120 respectively, do not fall into any particular group. These tasks have an allowable time range of $(120, 840)$ and can all be completed by any astronaut. They require 400, 500, 400, 400 units of power, respectively.

A feasible schedule for this instance of the ARCPSP is visualized in Figure 1.

Example 2. ***An Infeasible Modification***

We modify the previous example slightly to produce an unsatisfiable problem instance. If we change the duration of each laboratory task from 120 to 121, we get an over-constrained system of constraints for which no feasible schedule exists. We'll use this running example to produce explanations in the next several sections.

5 Subset Enumeration: Finding Conflicts and Relaxations

The proposed strategy for enumerating subsets is based on MARCO (Liffiton et al. 2016) over other systems such as CAMUS (Liffiton and Sakallah 2008) because outputting at least some MUSes quickly is more important than explicitly generating every MUS. MARCO relies on four main functions to explore the feasibility of a power set lattice of constraints which we outline here in the language of conflicts and relaxations.

– **BlockUp** - Called whenever a conflict is found. Marks every supersets of the current set, preventing it from being explored later on.

– **BlockDown** - Called whenever a relaxation is found. Marks every subsets of the current set, preventing it from being explored later on.

– **Grow** - If the current set is satisfiable, adds constraints to the current set until adding any other constraint would make it unsatisfiable.

– **Shrink** - If the current set is unsatisfiable, removes constraints from the current set until removing any other constraint would make it satisfiable.

A power set lattice of Boolean variables representing each constraint in the foreground is maintained throughout the execution of the algorithm. First, a random subset of constraints is constructed by choosing a point in the Boolean lattice. Then the SAT solver checks whether the set is `SAT` or `UNSAT`. If the resulting assignment is `SAT` (feasible), then constraints are iteratively added to the current set until a minimal relaxation is found. If the initial set is instead

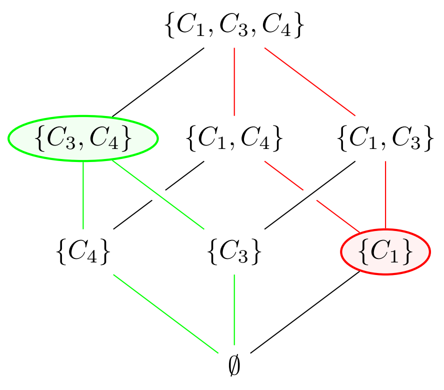


Figure 2: The power set lattice of $\{C_1, C_3, C_4\}$ with background C_2 and corresponding relaxation (green) and conflict (red).

`UNSAT` (infeasible), constraints are iteratively removed until a minimal conflict is found. After a minimal conflict is found, **BlockUp** is called, removing any supersets of the minimal conflict from consideration in the lattice. We can do this because any superset of a conflict must be unsatisfiable because it contains the conflict. The opposite direction also applies, any subset, after a minimal relaxation has been removed, must be satisfiable so we can rule them out of consideration. Hence, after a minimal relaxation is found, we **BlockDown**, removing any subsets from consideration in the lattice. Then a new satisfying assignment is pulled from the remaining sections of the boolean lattice and new conflicts and relaxations are generated until the entire lattice is blocked or the algorithm times out.

Example 3. ***A Small Over-Constrained Formula***

Consider the unsatisfiable conjunction of the following set of clauses,

$$C_1 = \{a\},\ C_2 = \{\neg a\},\ C_3 = \{\neg a \vee b\},\ C_4 = \{\neg b\}$$

with background $B = \{C_2\}$. We'll use this example to step through an execution of the subset enumeration algorithm. A visualization of the associated Boolean lattice is shown in Figure 2. A random initial seed has us start with clause $\{C_1, C_3\}$ and the SAT solver says it's `UNSAT`. We then **Shrink** and remove C_3 from the set and the SAT solver says $\{C_1\}$ is still `UNSAT` and minimal. We then output this minimal conflict $\{C_1\}$. Because this set is now minimal, we can **BlockUp**, removing supersets $\{C_1\}$, $\{C_1, C_3\}$, $\{C_1, C_4\}$, $\{C_1, C_3, C_4\}$ from consideration. We then choose a new seed, let's say $\{C_3\}$. The resulting set is `SAT` so we **Grow** to the set $\{C_3, C_4\}$ which is then `SAT` and maximal so we can **BlockDown** subsets $\{C_4\}$, $\{C_3\}$, $\{\emptyset\}$ and output $\{C_1\}$, the complement of $\{C_3, C_4\}$, as a relaxation. The lattice is then entirely blocked off so we terminate with the single conflict and relaxation. Figure 2 shows the power set lattice of the foreground along with the corresponding relaxation (green) and conflict (red).

5.1 Background Constraints for the ARCPSP

The standard background for the ARCPSP involves constraints to ensure that the resulting schedule is logical. This way, the foreground only involves constraints which can be altered by parameters that are controlled by the user.

– New variables S_0 and S_{m+1} are introduced that mark the beginning and end of the schedule bounds. These variables prevents the subset enumeration from relaxing the temporal constraints of tasks outside of the feasible bounds of the schedule

$$(S_0 \leq S_i) \wedge (S_i + p_i \leq S_{n+1}) \quad \forall S_i \in S. \tag{15}$$

– Each task is assigned to *some* existing agent. Without this constraint, the solver could assign tasks to a nonexistent agent to solve conflicts

$$M_i \in M \quad \forall M_i \in A. \tag{16}$$

– No agent has overlapping activities. Disallowing the solver from consider cases in which tasks can overlap prevents it from generating meaningless results. This condition is precisely constraint (3).

We will refer to this background set of constraints throughout the following section.

5.2 Constraint Explanations

The method of generating minimal conflicts and relaxations can be applied to both sets of individual constraints and, by modifying the background, sets of tasks. In this section, we investigate the first case, which we call *constraint explanations*. Following the strategy in the beginning of Section 5, we enumerate only over the constraints that are in the foreground, as specified in Section 5.1. That is, we consider constraints `timeframe`, `compatibility`, `precedence`, and `resource` referring to equations (4), (5), (6), and (7), respectively, for each task in the schedule. The rest of the constraints are implied as a part of the background because they only correspond to imposing a logical structure on the solution, not constraining the parameters of the schedule. Hence, the Boolean lattice which is enumerated over contains only these four types of constraints.

The outputs for constraint explanations are formatted as a tuple of the relevant tasks followed by a constraint type. For example, `(LAB_0, LAB_1)_precedence` refers to the precedence constraint between the first and second laboratory tasks. When a constraint is only relevant to a single task, we write the task followed by the constraint type (e.g., `MEAL_0_compatible` refers to the agent compatibility constraint for the first meal task).

The full constraint explanation for Example 2 includes 14 minimal relaxations and 3 minimal conflicts. Figure 3 shows a representative part of this full constraint explanation. The omitted conflicts and relaxations are identical in structure to the ones shown in Figure 3 and give practically redundant information. Computing the set of minimal forbidden sets took 2.11 seconds and calculating the full explanation took 1.57 seconds.

	Constraints
Confl 1	{`(LAB_0,LAB_1)_precedence`, `(LAB_1,LAB_2)_precedence`, `(LAB_2,LAB_3)_precedence`, `(LAB_3,LAB_4)_precedence`, `(LAB_4,LAB_5)_precedence` }
Relax 1	{`(LAB_3, LAB_4)_precedence`}
Confl 2	{`(LAB_1,LAB_2)_precedence`, `(LAB_2,LAB_3)_precedence`, `(LAB_3,LAB_4)_precedence`, `(LAB_4,LAB_5)_precedence`, `MEAL_0_compatible`, `MEAL_1_compatible`, `MEAL_2_compatible`, `MEAL_3_compatible`, `MEAL_4_compatible`, `MEAL_0_timeframe`, `MEAL_1_timeframe`, `MEAL_2_timeframe`, `MEAL_3_timeframe`, `MEAL_4_timeframe`}
Relax 2	{`(LAB_4, LAB_5)_precedence`, `MEAL_4_timeframe`}

Figure 3: Constraint explanations for Example 2.

In this example, relaxations provide the user with minimal ways in which constraints could be changed to fix the schedule. Meanwhile, conflicts give insight into why infeasibility is occurring. For example, `Relex 1` indicates that removing the precedence between laboratory task 3 and 4 would make the schedule feasible. However, `Confl 1` indicates that the precedence between all of the laboratory tasks does not fit in the schedule. A user could use this latter information to alter the original parameters of the schedule rather than having to void an entire task or constraint. One possible solution could be extending the length of the schedule or shortening the length of some of the laboratory tasks, an option which is not revealed by relaxations alone.

This formulation of explainability in terms of conflicts and relaxations also allows a user to ask pointed questions concerning the feasibility of an instance of an ARCPSP problem. Given a feasible instance of a problem, such as Example 1, specific questions may be asked about infeasible modifications of the problem. The modification in Example 2 is gotten by extending the lengths of the laboratory tasks. Hence, the explanation in Figure 3 may be interpreted as an answer to the question: "why can the laboratory tasks not have a duration longer than 120 minutes?"

5.3 Implication-Based Enumeration

In the following sections we explore two variations of the subset enumeration algorithm to generate higher-level descriptions of infeasibility. This is accomplished by pushing every constraint to the background and populating the foreground with a fresh set of Boolean variables. Then, a set L of constraints can be constructed that encodes a logical relationship between the new symbolic variables in the foreground and the actual constraints in the background.

The set L of logical relations linking symbolic variables to the constraints also becomes part of the background. Then, only the set of Boolean variables remains to be enumerated over in the foreground. In practice, this can be accomplished by replacing the Boolean lattice outlined in Section 5 by the symbolic lattice composed of the new variables. This enables the generation of explanations concerning these symbolic variables, which is dependent on the relationship L.

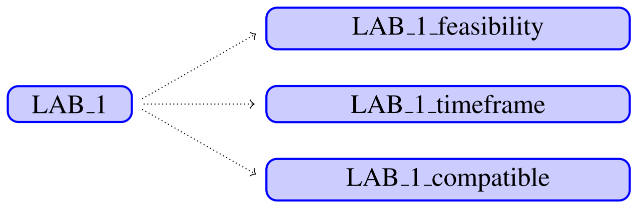
Depending on what kinds of constraints populate the foreground, the size of the Boolean lattice which needs to be enumerated over can be greatly reduced. This reduces the number of calls that need to be made to the SMT solver before arriving upon conflicts and relaxation. Additionally, these type of explanations can reduce redundancies and produce more compact descriptions of infeasibility. The following section outlines how this concept can be applied to generate minimal conflicts and relaxations of sets of tasks.

5.4 Task Explanations

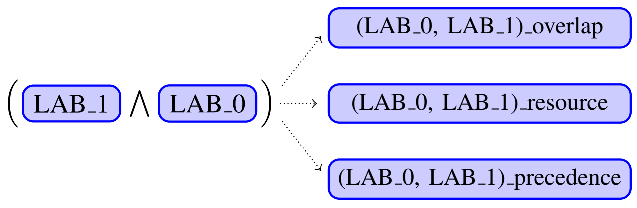
Task explanations can be generated by replacing the foreground (and hence, the power set lattice) with a set of variables representing individual tasks. We introduce a Boolean variable for each task and separate the constraints of the ARCPSP into two classifications: *individual* and *relational*. Constraints (4) and (5) as well as constraints (15) and (16) are individual constraints, involving only a single task. Constraints (6), (7), and (3) are relational constraints, involving multiple tasks. We then encode the constraint that the truth of each task's Boolean variable J_j implies the truth of every one of its individual constraints

$$J_j \implies \mathfrak{I}_j \tag{17}$$

where $\mathfrak{I}_j$ represents a conjunction of the task's timeframe (4), compatibility (5), and feasibility (15, 16) constraints. Visually we can represent the implication as follows, where `LAB_1` represents a Boolean variable and arrows represent logical implications:



For the relational constraints, we add the condition that the truth of all of the dependent tasks' Boolean variables implies its truth. So if relational constraint C_1 is between task J_1 and J_2, then we impose the constraint $J_1 \wedge J_2 \implies C_1$ in the same manner as outlined above:



	Tasks
Confl 1	`{LAB_0, LAB_1, LAB_2, LAB_3, LAB_4, LAB_5}`
Relax 1	`{LAB_1}`
Confl 2	`{LAB_1, LAB_2, LAB_3, LAB_4, LAB_5, MEAL_0, MEAL_1, MEAL_2, MEAL_3, MEAL_4}`
Relax 2	`{LAB_5, MEAL_0}`

Figure 4: Task explanations for Example 2.

Hence, individual constraints need only be satisfied if their associated task's Boolean variable is true and relational constraints need only be satisfied if *all* of their associated tasks' Boolean variables are true. Through this logical relationship, we can now enumerate over these Boolean variables, each of which conceptually represents a task. As the Boolean variables are toggled on and off, the associated constraint lattice becomes constrained as if the schedule had been constructed with only that subset of tasks. Executing the enumeration algorithm over this modified foreground for the same over-constrained problem (Example 2) produces a similar set of conflicts and relaxations, part of which is shown in Figure 4. The full explanation includes 3 minimal relaxations and 17 minimal conflicts in total. It took 4.03 seconds to compute the minimal forbidden sets and 0.55 seconds to compute the full explanation.

Here, the task and constraint explanations are quite similar, but this is not always the case. The task explanations can often be much more compact than the constraint explanations because each variable represents many constraints. For similar reasons, the number of total conflicts and relaxations is often greatly reduced. Because of this, task explanations can give more straight forward explanations for the over-constrained problem, but they lack the granularity of the constraint explanations. For example, with the constraint explanations we were able to diagnose that the precedence between the lab tasks was creating an issue rather than a resource or other constraint. The task explanations leave out this detail, sacrificing expressibility for interpretability.

5.5 Specification Explanations

In order to draw explanations back to a high-level interpretation of the problem, the foreground can be replaced by a set of human-written specifications. This further reduces the size of the power set lattice that is constructed out the foreground and reduces redundancy in the generated conflicts and relaxations.

Tasks are often formed in groups which share certain scheduling specifications. For example, the meal tasks in Example 1 all share the same parameters except that they are assigned to unique astronauts. When Example 1 was described, such similar tasks were naturally formulated in different categories (e.g., meal, weights, etc.). Hence, specifications for tasks may be more succinctly expressed by making use of these similarities.

An informal, human written list of constraints specifying

	Specifications
Confl 1	{Lab tasks must happen in sequence, the scheduling horizon is 6am to 6pm}
Relax 1	{Lab tasks must happen in sequence}
Relax 2	{The scheduling horizon is 6am to 6pm}

Figure 5: Specification explanations for Example 2.

the parameters of Example 1 could be as follows:

– The scheduling horizon is 6am to 6pm.

– Meal tasks must be scheduled between 1pm and 2pm.

– Lab tasks must happen in sequence.

– Each meal/treadmill/weights task must be assigned to a different astronaut.

– Weights/treadmill tasks cannot happen 60 minutes before pre-sleep.

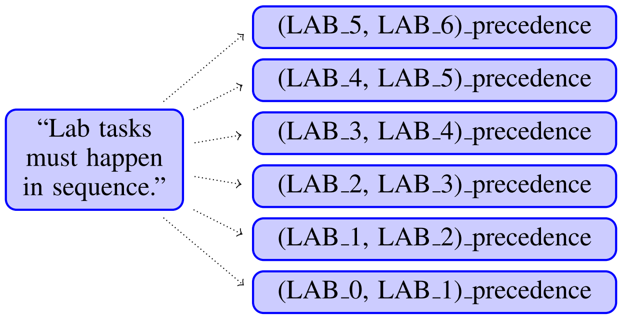
– No more than 1000 units of power may be drawn at once.

– The treadmill tasks require 200 units of power.

– Tasks EVA_SUIT_TEST, HW_GATHER, C-SAT, and SAMP require 400, 400, 500, and 400 units of power.

– There is only one set of weights and treadmill equipment.

Then a relevant logical relationship may be drawn back to the actual set of constraints for each such specification. For example, the precedence relations between the lab tasks could be related through:


"Lab tasks must happen in sequence."
(LAB_5, LAB_6)_precedence
(LAB_4, LAB_5)_precedence
(LAB_3, LAB_4)_precedence
(LAB_2, LAB_3)_precedence
(LAB_1, LAB_2)_precedence
(LAB_0, LAB_1)_precedence

Similar constraints may be encoded for the rest of the specifications, which compose the set L linking the human written specifications to the actual constraints of the problem. This construction allows the generation of *specification explanations*. The specification explanation for Example 1 is displayed in Figure 5. Notice the greatly reduced size of the specification explanation. Unlike the constraint and task explanation, the specification explanation does not suffer from producing many redundant conflicts and relaxations.

A fundamental trade-off exists between the expressibility and interpretability of different kinds of explanations. Low-level explanations involving constraints provide detailed reasons for infeasibility but may be difficult for a human user to parse or understand. In contrast, because the high-level specification explanations correlate directly with the types of constraints which a human planner may think in, they potentially provide more direct and concise information to the user. However, they lack the fine tuned granularity of information that constraint and task explanations provide. For example, if only a single precedence constraint between the laboratory tasks was causing an issue, the specification explanation would obscure which of the constraints is responsible.

6 Conclusion

We introduced the agent resource-constrained project scheduling problem (ARCPSP) along with an associated difference logic encoding. We proposed a general framework for generating minimal conflicts and minimal relaxations based on the MARCO algorithm and demonstrated how it could be used to generate varying types of descriptions for why infeasibility is occurring in instances of the ARCPSP. The framework outlined in this paper is general enough to be applied to constraint satisfaction formulations for various other scheduling and planning problems. These ideas may potentially be further extended to different kinds of formal languages, such as linear temporal logic, that are used to describe planning problems.

6.1 Future Work

In an interactive system, such as a scheduling software, when a user attempts to make an infeasible modification, it may be useful to generate a reason for the infeasibility in real time. Similarly, a user could query whether a modification to a feasible schedule would preserve feasibility and, if not, why not? Explanations similar to the ones constructed throughout this paper may likely be used to such an effect. Investigating methods for synthesizing natural language sentences out of the explanations is also subject to future research.

Following the goal of QuickXplain (Junker 2004), given a partial ordering of constraint or task importance, preferred conflicts and relaxations may be explored earlier and full explanations may list conflicts and relaxations in preferential order. Such functionality would be especially useful in cases for which generating the full explanation is intractable. A preferential ordering of explanations may be achieved by adding and removing constraints during the **Grow** and **Shrink** steps based on the constraint preference ordering. Similarly, methods for enumerating disjoint (or otherwise distinct) conflicts may also be useful for producing a representative set of conflicts as concisely as possible.

Currently, the most limiting bottleneck for scaling to larger problem instances comes from the number of minimal forbidden sets which can grow exponentially with the number of tasks. Certain lazy clause generation algorithms (Laborie 2003) may be used to represent resource constraints in a more efficient manner. Such representations may also be adapted to implement *consumable* resources in an explainability setting.

References

Bofill, M.; Coll, J.; Suy, J.; and Villaret, M. 2016. Solving the multi-mode resource-constrained project scheduling problem with SMT. In *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, 239–246.

Burt, C.; Klimova, K.; and Primas, B. 2018. Generating explanations for mathematical optimisation: Solution framework and case study. In *ICAPS 2018 Workshop on Explainable Planning (XAIP)*.

Cotton, S., and Maler, O. 2006. Fast and flexible difference constraint propagation for dpll(t). In Biere, A., and Gomes, C. P., eds., *Theory and Applications of Satisfiability Testing - SAT 2006*, 170–183. Berlin, Heidelberg: Springer Berlin Heidelberg.

De Moura, L., and Bjørner, N. 2008. Z3: An efficient SMT solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, 337–340. Berlin, Heidelberg: Springer-Verlag.

Freuder, E. C., and Wallace, R. J. 1996. Partial constraint satisfaction. In Jampel, M.; Freuder, E.; and Maher, M., eds., *Over-Constrained Systems*, 63–110. Berlin, Heidelberg: Springer Berlin Heidelberg.

Junker, U. 2004. Quickxplain: Preferred explanations and relaxations for over-constrained problems. In *AAAI*.

Laborie, P. 2003. Algorithms for propagating resource constraints in ai planning and scheduling: Existing approaches and new results. *Artificial Intelligence* 143(2):151 – 188.

Liffiton, M. H., and Sakallah, K. A. 2008. Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning* 40(1):1–33.

Liffiton, M. H.; Previti, A.; Malik, A.; and Marques-Silva, J. 2016. Fast, flexible mus enumeration. *Constraints* 21(2):223–250.

Möhring, R., and Stork, F. 2009. Stochastic project scheduling under limited resources: A branch and bound algorithm based on a new class of policies.

Seegebarth, B.; Müller, F.; Schattenberg, B.; and Biundo, S. 2013. Making hybrid plans more clear to human users — a formal approach for generating sound explanations. In *Proceedings of the Twenty-Second International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'12, 225–233. AAAI Press.

Stork, F., and Uetz, M. 2005. On the generation of circuits and minimal forbidden sets. *Math. Program.* 102(1):185–203.

Model-Free Model Reconciliation

Sarath Sreedharan, Alberto Olmo Hernandez, Aditya Prasad Mishra and **Subbarao Kambhampati**
School of Computing, Informatics, and Decision Systems Engineering,
Arizona State University, Tempe, AZ 85281 USA
{ ssreedh3, aolmoher, amishr28, rao } @ asu.edu

Abstract

Designing agents capable of explaining complex sequential decisions remains a significant open problem in human-AI interaction. Recently, there has been a lot of interest in developing approaches for generating such explanations for various decision-making paradigms. One such approach has been the idea of *explanation as model-reconciliation*. The framework hypothesizes that one of the common reasons for a user's confusion could be the mismatch between the user's model of the agent's task model and the model used by the agent to generate the decisions. While this is a general framework, most works that have been explicitly built on this explanatory philosophy have focused on classical planning settings where the model of user's knowledge is available in a declarative form. Our goal in this paper is to adapt the model reconciliation approach to a more general planning paradigm and discuss how such methods could be used when user models are no longer explicitly available. Specifically, we present a simple and easy to learn labeling model that can help an explainer decide what information could help achieve model reconciliation between the user and the agent with in the context of planning with MDPs.

1 Introduction

A significant barrier to integrating AI systems into our daily lives has been their inability to interact and work with us humans in an intuitive and explicable manner. Orchestrating such interactions would require the agents to have the ability to help users in the loop better understand the rationale behind their various actions. Thankfully there has been a lot of effort within the AI research community to develop systems capable of holding explanatory dialogues with users and thus help them understand the decisions under question [David W. Aha and Magazzeni, 2018; Daniele Magazzeni, 2018]. Such explanatory systems could help users resolve confusions regarding agent decisions that may stem from either a (1) lack of understanding (or even misunderstanding) of the task or (2) from their inferential limitations. While many earlier works

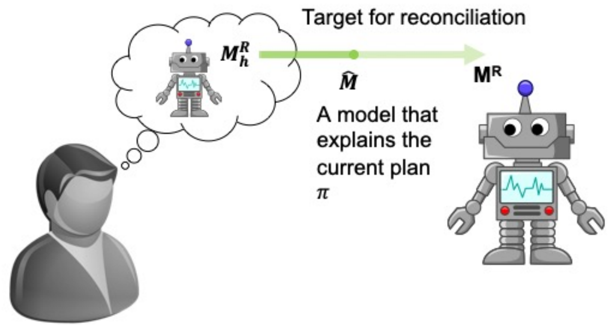

M_h^R
Target for reconciliation
$\widehat{M}$
M^R
A model that explains the current plan π

Figure 1: A general overview of the explanation as model reconciliation.

in explanation have generally focused on the latter (cf. [Khan *et al.*, 2009; Hayes and Shah, 2017; Seegebarth *et al.*, 2012; Topin and Veloso, 2019]), there is a growing consensus on the importance of explanatory mechanisms that can help bridge the knowledge asymmetry between the system and the user.

In particular, in **explanation as model-reconciliation** [Chakraborti *et al.*, 2017] we had studied the problem of reconciling knowledge asymmetries between the user and the agent within the context of planning. Works in this direction have generally looked at cases where the user's model of the task (i.e their belief about the initial state, the transition dynamics, and the goal) is known beforehand (in a representation scheme comparable to the one used by the agent) and do not match the agent's model. This mismatch means that the user would not be able to correctly evaluate the validity or the optimality of a given plan. Thus in this paradigm the explanations consist of information about the agent's model that the user could incorporate into his/her own model to correctly evaluate the plan in question.

Unfortunately, it is not always possible to have access to such models. In the most general case, we are dealing with the user's model of the agent and hence the user may not be capable of presenting traces or decisions that could be generated from this model. Even if the system tries to learn such a representation based on interactions with the user, there is no guarantee that the specific representational assumptions of the learned model and the vocabulary used would be satisfied by the user's mental model.

The definition of explanation as model reconciliation may leave one with the idea that there is no way around it. How could one ever truly perform effective reconciliation when there exists no user model guiding us to choose the parts of the model, which when revealed to the user will help them correctly evaluate the current decision? Are we left with revealing the entire agent model to the user as the only option? In this paper, we propose a simple and intuitive way we could still generate minimal explanations in the absence of declarative models. We argue that we could drive such explanations by using learned models that can predict how human expectations could be affected by possible explanations (derived completely from information about the agent model) and in fact show how this method could be viewed as a variation of previous approaches that have been put forth to identify explicable behavior.

We will start by extending model reconciliation to the more general setting of planning with Markov Decision Processes (Section 4). The rest of the paper will investigate how these ideas could be used when the human mental model of the task is unavailable, and will formulate a learning problem that allows us to learn simple models that could be used to identify minimal explanations (Section 5). Finally, we will evaluate our method on a set of standard MDP benchmarks and perform user studies to validate its viability (Section 6).

2 Background

Figure 1 presents a general schematic representation for explanation as model reconciliation. The automated agent (henceforth referred to as robot) starts with a model $\mathcal{M}^R$ that can be used to generate a decision π (where depending on the context, π may be a single action, plan, policy or a program). In this setting, $\mathcal{M}_h^R$ corresponds to the human's preconceived notions about the robot model. The explainer's job then becomes providing information about the model $\mathcal{M}^R$, such that the updated human model can correctly evaluate the validity of the robot decisions.

In this case, the robot could have chosen to provide the entire model, but for most realistic tasks, such models could be quite large, so dumping the entire model could be both unnecessary and impractical. It's also well known that people generally prefer explanations that are selective [Miller, 2018; Lombrozo, 2006]. Thus the users would be happier with explanations that asks them to update a subset of beliefs as opposed to a complete update. Note that $\mathcal{M}^R$ need not be the original agent model, but rather some abstraction/approximation of the underlying robot model (that conserves some desired property of the decision like optimality or validity). In [Chakraborti *et al.*, 2017] where model reconciliation was first introduced, $\mathcal{M}^R$ was a classical planning model hence inherently interpretable and thus the reconciliation could easily be achieved, but the idea could be applied beyond just planning models. For example, one could understand the explanation methodology used by LIME [Ribeiro *et al.*, 2016] as being a special case of model reconciliation. In their case, they assume the human model is empty and $\mathcal{M}^R$ is an approximation of the underlying decision model is automatically generated for each decision using a set of predefined features.

In this work, we will be looking at the agents that use discounted infinite horizon Markov Decision Processes (or MDPs) [Russell and Norvig, 2003] as the decision making framework. Each MDP $\mathcal{M}$ is defined by a tuple $\langle S, A, T, R, \gamma, \mu \rangle$, where the S provides the set of possible atomic states, A defines the set of actions, T is the transition function, R the reward, γ the discounting factor (where $0 \leq \gamma < 1$) and μ corresponds to the distribution of possible initial states. $T : S \times A \times S \rightarrow [0, 1]$ provides the probability that for given state $s \in S$, the execution of an action a would induce a transition to a new state s', and $R : S \times A \times S \rightarrow \mathbb{R}$ defines the reward corresponding to this transition. The solution concept in MDP takes the form of a policy π that maps each state to a potential action. A policy is said to be optimal for $\mathcal{M}$ (denoted as $\pi^*_{\mathcal{M}}$) if there exists no other policy that dominates the given policy in terms of the expected value of states (where the value of a state s under a policy π for a model $\mathcal{M}$ is denoted as $V^{\pi}_{\mathcal{M}}(s)$). Executing the policy in a state results in a sequence of state action state tuples called *execution trajectory* or simply a *trajectory*, denoted as $\tau = \langle (s_1, a_1, s_2), ..., (s_{n-1}, a_{n-1}, s_n) \rangle$ and we will use $P_{\mathcal{M}}(\tau|\pi)$ to denote the probability of sampling the given trajectory τ for a policy π in model $\mathcal{M}$.

In the explanatory setting we are interested in, the robot uses a model $\mathcal{M}^R = \langle S, A, T^R, R^R, \gamma^R \rangle$ of the task to come up with the policy to act on. For now we will assume this MDP already defines an interpretable model and the human uses a model $\mathcal{M}_h^R = \langle S, A, T_h^R, R_h^R, \gamma_h^R \rangle$ to evaluate it (we will relax this assumption in later sections). Now the task ahead of us will be to formulate how we could still identify minimal information that could resolve user confusion when $\mathcal{M}_h^R$, but before we can do that we need to reinterpret the ideas of inexplicability and the idea of model reconciliation that was defined in [Chakraborti *et al.*, 2017] in the context of MDPs and we will start by considering a simple scenario.

3 Illustrative Example

Consider a warehouse scenario, where a robot is tasked with moving packages from racks and dropping them off at the dispatch chute. The robot is powered by a battery pack that can be recharged by visiting its docking station. The docking station also doubles as a quality assurance station that the robot needs to visit whenever it picks up a box labeled #013 (which means the box is fragile). The robot's operations are mostly deterministic, apart from a small probability of slipping (0.25) in some cells, that could leave the robot in the same position.

Now suppose the warehouse has just hired a new part-time employee to oversee the operations. The employee is just getting used to this new setting and is puzzled by the robot's decision to once in a while take a detour from the drop-off activity and visit a specific position of the factory floor (which is, in fact, the docking location). If we wished the robot to be explainable, then it would need to be capable of helping the employee better understand the underlying model used by the robot (i.e achieve some form of model reconciliation). Given the fact that the robot may not have an exact model of the user, one way to achieve this could be by providing robot's entire model to the user. Unfortunately, this could easily overwhelm the user.

Another possibility could be to allow the user to specify which robot actions appear inexplicable, and focus on providing facts relevant to those actions. This explanation may still prove to be quite verbose and may in fact not help resolve their confusion. For example, imagine a case where the robot is visiting the station to recharge its batteries and the human says that the visit action is inexplicable. Now even if the robot mentions that visiting the station recharges it, the employee may still be confused if they are under the incorrect assumption that the robot is operating on full battery. Similarly, if the human had expected the robot to go to the docking station due to some confusion regarding the box codes, the human may mark the robot decision to not go to the dropoff as being inexplicable and the explanations that could resolve the confusion may have little to do with that specific action marked as inexplicable.

4 Explanation as Model Reconciliation For MDPs

In this setting the human and robot models are captured as MDPs defined over the same set of states and thus we can characterize both models by the tuple $\theta = \langle \theta_T, \theta_R, \theta_\gamma, \theta_\mu \rangle$, where the θ_T provides the set of parameters that defines the transition probabilities $P(.|s, a)$, while θ_R the parameters corresponding to the reward function, θ_γ the parameters corresponding to the discount factor and θ_μ the parameters for the initial state distribution. For simple MDP models with atomic states, θ_T contains parameters of the categorical distribution for each transition (θ_μ will contain similar parameters for the initial state distribution), θ_R contains the reward associated with each transition (an $\langle s, a, s' \rangle$ tuple) and θ_γ just contains the value of the discount factor. The specific instantiations of the parameters for each model $\mathcal{M}$ is captured as $\theta(\mathcal{M})$. For simplicity, we will denote each of the unique parameters in the tuple θ using indexes. For example, $\theta_T^{s,a}(\mathcal{M}^R)$, will correspond to the parameters for the distribution $P(.|s, a)$ for the model $\mathcal{M}^R$.

If we use $\mathbb{M}$ to capture the set of all possible models and $\Theta = \theta_T \times \theta_R \times \theta_\gamma \times \theta_\mu$, then model reconciliation operation can be captured as a function $\mathcal{E}_{\langle \mathcal{M}_h^R, \mathcal{M}^R \rangle} : 2^\Theta \rightarrow \mathbb{M}$ that takes in a set of model parameters and generates a new version of the model $\mathcal{M}_h^R$ where the set of specified parameters will be set to values from $\mathcal{M}^R$. For example, $\hat{\mathcal{M}} = \mathcal{E}_{\langle \mathcal{M}_h^R, \mathcal{M}^R \rangle}(\theta_T^{s_1,a})$ will be a new model such that $\theta(\hat{\mathcal{M}})$ will be identical to $\theta(\mathcal{M}_h^R)$, except that $\theta_{T_{\hat{\mathcal{M}}}}^{s_1,a}$, will be equal to $\theta_{T^R}^{s_1,a}$.

Practically, the model reconciliation operation corresponds to the robot informing the human about some part of its model. This communication could incur cost and we can capture this by using the cost function $\mathcal{C} : 2^\Theta \rightarrow \mathbb{R}$ that maps a given set of a threshold to a cost.

Now the question we need to ask is whether the agent is trying to explain its policy or if it is trying to explain some behavior (i.e an execution trace). Most of the earlier work that looks at model reconciliation explanation (cf. [Chakraborti *et al.*, 2017; Sreedharan *et al.*, 2018a; 2018b]) has looked at sequential plans and has generally ignored this differentiation and treated the problem of explaining plans to be same as that of explaining behavior. In general, a given plan or policy compactly represents a set of possible behaviors and the choice of explaining behavior *vs* explaining the plans/policies could affect the content of the explanation being given. For example, when explaining policies there is the additional challenge of presenting the entire policy to the user and the explainer may need to justify action choices for extremely unlikely states or contingencies. On the other hand, when explaining a given set of behaviors the explainer needs to only justify their action choices for cases they actually witnessed. For example, when explaining traces from the warehouse scenario, given the small probability of slipping, the robot may never have to mention what to do when it slips, but on the other hand if we are dealing with full policies, the agent may need to talk about the states where the robot is in the slipped positions and they need to get up from that position and move on.

Explaining policies or plans becomes more relevant when we consider explanatory dialogues where the agent and the user are trying to jointly come to agreement on what policy/plans to follow (eg: decision support systems), while the latter may be more useful when the user is observing some agent operating in an environment.

With respect to policies, we assume that the user is presented with the entire policy. A given policy is said to be explicable to the human, if the policy is optimal for the human model. Therefore the goal of the explainer becomes that of ensuring the optimality of the given policy

Definition 1. *A set of parameters $\theta_\mathcal{E}$ corresponds to a* ***complete policy explanation*** *for the given robot policy $\pi^*_{\mathcal{M}^R}$, if the policy is also optimal for $\mathcal{E}_{\langle \mathcal{M}_h^R, \mathcal{M}^R \rangle}(\theta_E)$ and is said to be the minimally complete policy explanation if there exists no other complete explanation $\theta_{\mathcal{E}'}$, such that, $\mathcal{C}(\theta_{\mathcal{E}'}) < \mathcal{C}(\theta_\mathcal{E})$*

Finding a complete policy explanation is relatively straightforward (the set of all parameters automatically meets this requirement). The more challenging case becomes that of finding the minimal or the cheapest explanations i.e. the minimally complete explanations. Such minimally complete explanations can be calculated by adopting a search strategy similar to [Chakraborti *et al.*, 2017]. The search can start at the human model and try to find the minimal number of parameters that needs to be updated in the human model for the current policy to become optimal. Similar to generating minimally complete explanations, i.e, we can also generate monotonic explanations (i.e explanations where no further information about parameters in the robot model can affect the optimality of the current plan).

In the case of policies, we can also describe explicable planning and balancing cost of explanations with that of choosing policies that are inherently explicable, where inexplicablity score ($\mathcal{I}_\mathcal{E}$) of a policy π is defined as

$$\mathcal{I}_\mathcal{E}(\pi, \mathcal{M}_h^R) = |E[V_{\pi*}^{\mathcal{M}_h^R}(s)|s \sim \mu_h^R] - E[V_\pi^{\mathcal{M}_h^R}(s)|s \sim \mu_h^R]|$$

Where $\pi*$ is the optimal policy in the human model. Explicable planning thus becomes the problem of choosing

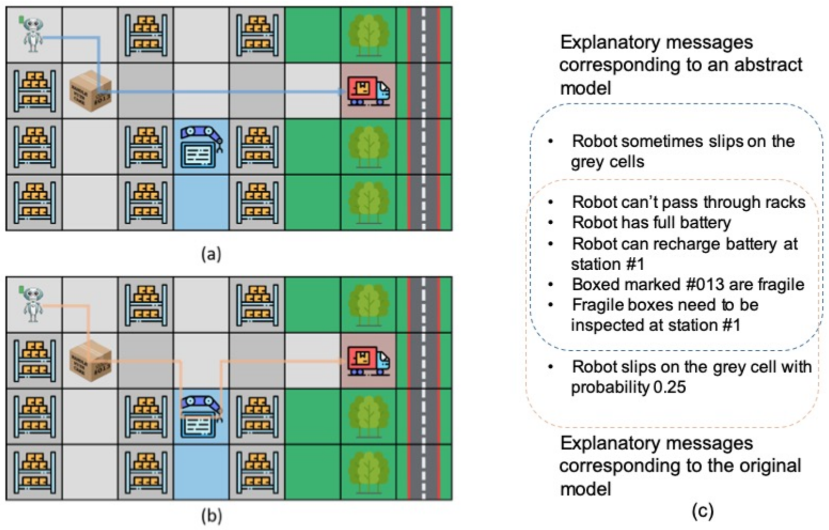

Explanatory messages corresponding to an abstract model
Robot sometimes slips on the grey cells
Robot can't pass through racks
Robot has full battery
Robot can recharge battery at station #1
Boxed marked #013 are fragile
Fragile boxes need to be inspected at station #1
Robot slips on the grey cell with probability 0.25
Explanatory messages corresponding to the original model
(a)
(b)
(c)

Figure 2: Subfigure (a) shows a visualization of a trajectory expected by the user described in the illustrative example, and (b) shows the visualization of a trajectory the user may observe. Subfigure (c), shows the various explanatory messages that could be used in this scenario, note that the messages span information from multiple abstractions of the given task

policies that minimize inexplicability score [Kulkarni *et al.*, 2016], while minimizing the potential loss in optimality due to the policy choice (since the most explicable plan may not be an optimal policy). Balanced planning, as studied in [Chakraborti *et al.*, 2019; Sreedharan *et al.*, 2019], proposes going one step further and also takes into account possible savings in inexplicability score that can be achieved by providing explanation (while incurring additional cost of communicating the required explanations).

For explaining behavior, we will look at the simplest case, namely the agent needs to explain a set of behaviors that the user has just observed. We will assume that the observer has full observability of the state and is seeing the robot behavior for the first time. In such a setting, a given trace τ would appear explicable to the user if it could be sampled from their expected MDP policy (i.e a policy optimal in their model) or more generally, i.e $P_{\mathcal{M}_h^R}(\tau|\pi) > \delta$, where δ is some small threshold. [1]

Definition 2. *A set of parameters $\theta_{\mathcal{E}}$ corresponds to a* ***complete behavior explanation*** *for a set of traces $\mathbb{T} = \{\tau_1, ...\tau_n\}$, if $\forall \tau \in \mathbb{T}$, $\exists \pi$ such that $P_{\mathcal{E}_{\langle \mathcal{M}_h^R, \mathcal{M}^R \rangle}(\theta_E)}(\tau|\pi) > \delta$ and π is an optimal policy for the model $P_{\mathcal{E}_{\langle \mathcal{M}_h^R, \mathcal{M}^R \rangle}(\theta_E)}$. The explanation is said to be the minimally complete behavior explanation if there exists no other complete explanation $\theta_{\mathcal{E}'}$, such that, $\mathcal{C}(\theta_{\mathcal{E}'}) < \mathcal{C}(\theta_{\mathcal{E}})$*

Note that given the above definition, if δ is set very high it may not be possible to find a complete explanation, as the trace may genuinely contain low probability transitions. In this work we will assume δ to be zero.

While model reconciliation could be an important component of either policy or behavior explanation, the applicability of the model reconciliation explanations on their own for policies is limited by the fact that in all but problems with the smallest state spaces, the user would have trouble going over the entire policy. Thus in these settings, explanatory systems would need to also utilize policy approximation or summarization methods, then allow users the ability to drill down on policy details as required. Since our main goals was to focus on developing approaches that allow us to generate model reconciliation explanations without explicitly defined user models, the rest of the paper will mostly focus on behavior explanation. In Section 8, we will have a brief discussion on how these methods could potentially be extended to policy explanation scenarios.

[1]We use δ in the general case to allow for the possibility that people can be surprised by unlikely events of non-zero probability

5 Explaining Without Explicit Human Mental Models

Now we will look at how we can identify cheap complete behavior explanations when the human model is unknown. We will go one step further from identifying not only the parameters that need to be explained, but also capturing the right modality/abstractions to present the information about the parameters. That is, we will no longer assume that the human is using a full MDP model to come up with their decisions. Instead, the robot starts with a set of explanatory messages $\Psi = \{m_1, m_2, ..., m_n\}$ that can be presented to the user. Where the messages correspond to a set of parameter values (the parameters corresponding to a set of messages $\{m_1, ..m_k\}$ is denoted as $\mathcal{E}(\{m_1, ..m_k\})$) of the model as captured in some abstraction of this model and has a corresponding cost ($\mathcal{C}$) associated with it. The abstractions to consider may depend on the specific scenario and the previous information about the intended users (laypeople vs. experts). Some simple possibilities may be to consider qualitative models (say non-deterministic ones instead of stochastic) and considering state abstractions the given task. Note that, technically $\mathcal{E}(\Psi)$, need not span the set of all possible model parameters, but could rather be limited to a subset of parameters identified to be relevant to the given problem. One possible way may be to consider variations of explanation techniques like MSE [Khan *et al.*, 2009] to identify set of possible factors that affect the optimality of each action. In Figure 2, the subfigure (c) shows a set of possible explanatory messages for the warehouse domain, that consists of each parameter mapped to some english statement. For models captured using factored representations that use relational or propositional fluents, such statements could be easily generated using templates (cf. [Hayes and Shah, 2017]).

Given this setting, we will now make some simplifying assumptions, namely, (1) the order in which the explanatory messages are presented does not matter (2) we have access to a set of observers with similar models and they share this model with the target user (3) the robot is viewing the task at the same level or at a more detailed level of granularity than the user and (4) the user and robot have some shared vocabulary in regards to the task. While assumption (1) is easily met since we are mostly dealing with model information and (4) is a prerequisite for most explanatory approaches, in section 8 we will discuss how we can possibly relax requirements (2) and (3).

Now our goal is to learn a predictive model that is able to predict whether a given user would find a given $\langle s, a, s\rangle$ tuple explicable and how the user's perception changes with the given explanatory messages.

For example, at the beginning of an episode the user may be presented with the following explanatory messages, $\hat{\Psi} = \{m_1 =$ "Robot slips with probability 0.25 at grey cells"$\}$, which corresponds to the fact that $P(s_i|a, s_i) = 0.25$, for all states s_i where the feature grey cell is true and for all actions a. Now the user will be presented with a sequence of transitions, say $\langle (1,2), \text{right}, (2,2)\rangle$ and asked whether the transition was explicable or not. Then the tuple $\langle\langle (1,2), \text{right}, (2,2)\rangle, \{m_1\}, l_1\rangle$, where l_1 is the label assigned by the user to the transition, becomes input to our learning method.

The exact function that we would want to learn would be

$$\mathcal{L}(\langle s,a,s'\rangle, \{m_1,...,m_k\}) = \begin{cases} 1 & \text{if } \langle s,a,s'\rangle \sim \\ & \pi^*_{\mathcal{E}_{\langle \mathcal{M}_h^R, \mathcal{M}^R\rangle}(\theta(\{m_1,...,m_k\})}(s) \\ 0 & \text{otherwise} \end{cases}$$

Note that this is a modified version of the sequential model we introduced in [Zhang *et al.*, 2017] for identifying whether a given plan is explicable or not. Though our methods vary in some significant aspects, namely, (1) we allow for the possibility that the explicability of the actions/traces could be affected by explanations provided by the system; (2) we no longer use labels of high level tasks as a proxy for the explicability of the trace. Instead, we just use a simple binary label on whether the transition is explicable or not; (3) we no longer consider sequence models but rather a much simpler labeling model that maps a single transition to the explicability label. We argue that in cases where the human is markovian on the same set of features as the agent, this rather simpler model suffices.

It is also important that our learning approach is more tractable than the ones studied in [Zhang *et al.*, 2017], since in their case to build a balanced dataset (of explicable and inexplicable plans), they would need to uniformly sample through the entire plan space (an extremely hard endeavour with no obvious known approaches), while we stick to traces generated from the optimal policy and only need to randomly generate possible sets of explanatory messages, which is clearly a smaller set.

Once we have learned an approximation of the above labeling function $\hat{\mathcal{L}}$, the problem of explanation generation for a trace $\tau = \langle s_0, a_0, s_1, ..., s_n, a_n, s_{n+1}\rangle$ becomes that of finding the subset of Ψ that balances the cost of communication with the reduction in the inexplicability of the given trace, i.e

$$\arg\min_{\hat{\Psi}}(\mathcal{C}^{\mathcal{M}}(\hat{\Psi}) + \alpha * \Sigma_{i=0}^{n}(1 - \hat{\mathcal{L}}(\langle s_i, a_i, s_{i+1}\rangle, \hat{\Psi})))$$

Where $\hat{\Psi}$ is a subset of Ψ and α is some scaling factor that balances the cost of explanation with the number of inexplicable transitions for a given trace.

6 Evaluation

The success of the approach described above would be directly dependent on whether we can learn high accuracy labeling models. Once we have access to such a model, we could be quite confident in our ability to generate useful explanation (provided the user's model is the same as the one the labeler was trained on) and identifying the best explanation becomes a matter of just searching for the required subset of messages that minimizes the objective defined in section 5. So to evaluate the method our focus was on identifying if it was possible to learn high accuracy models. We validated our approach on both simulations and on data collected from users.

6.1 Evaluation on Simulated Data

For simulations, we used a slightly modified versions of the Taxi domain [Dietterich, 1998] (of size 6*6), the Four rooms domain [Sutton *et al.*, 1999] (of size 9*9) and the warehouse scenario (of size 9*9) described before (implemented using the SimpleRL framework [Abel, 2019]). For each domain, we start with an MDP instance (henceforth referred to as the robot model) and then create a space of possible user models by identifying a set of possible values for each MDP parameter. For example, in the taxi domain the parameters include position of the passengers, their destination, the step cost, discounting etc., for the Four rooms this included the goal locations, locations with negative rewards, discounting, step cost, slip probability, etc., and finally for the warehouse, the position of the box, the position of station #1, the step cost, slipping probabilities and the discounting factors were selected as potential parameters that can be updated. In this setting, we assume that there exists a single explanatory message for each possible parameter.

For each individual test, we select a random subset of three parameters and then randomly choose a value for each of these. We then treat this new MDP model as a stand-in for

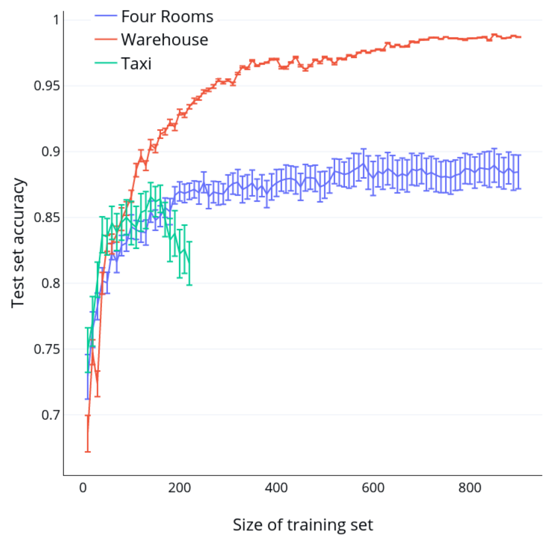


Figure 3: The test accuracy for increasing sizes of training set.

the user model and use it to label traces generated from the original MDP. The traces were generated by choosing a random initial state and then following the optimal policy of the robot until either the terminal state is reached or the trace length reaches a predefined limit. For each trace, a random subset of the explanations was selected and presented to the human. This means updating the MDP parameters to their corresponding values in the robot model only for the parameters specified by the current subset of explanation. Each individual transition was then labeled using this updated MDP. A transition was labeled as inexplicable if the action is not the optimal one in the human model (i.e. Q value is lower) or the next state had a probability of occurring of $\delta = 0$.

We then used this set of labeled transitions to create a training set and test set for a decision tree learner. The input features to the decision tree consist of current state features, (just x and y for Four rooms and the position of the the taxi and passengers for the Taxi domain and for Warehouse it included the position of the robot and the fact whether the agent picked up the box or visited station #1), the index of the action and features capturing the current subset of explanations being considered. In each Warehouse and Four rooms test instance, we collected 900 unique data points as training set and 100 data points as the test set. Due to the complexity of the taxi domain, we generated less data points (since for each different explanation subset we need to solve a new planning problem) and used close to 220 unique points as training data and on average 28 data points as the test set.

We then tested on 20 such instances for each domain. Figure 3 plots the average test accuracy for models trained with training sets of varying sizes. As evident from the graph, a simple decision tree seems to be able to easily model the effect of explanations on labeling for these simulated scenarios. We chose a simple learning model to establish the viability of this method, but one could easily see that the use of more sophisticated learning methods and/or more informed features should lead to better results.

6.2 User Studies

Next, we wanted to establish if we can still learn such simple models when the labels are collected from naive users. Our goal here is not to consider scenarios with possible differences in the user's knowledge, but rather cases where, even in the presence of a set of users with similar backgrounds, their responses to explanations would be too varied to learn useful models. To test this, we used the Warehouse domain as a test bed and collected feedback on how users would view the explicability of traces generated from this domain when presented with explanatory messages detailed in Figure 2.

For the study, we recruited 45 master turkers from the Amazon Mechanical Turk. Each participant was provided with the URL to a website (`https://goo.gl/Hun3ce`) where they could view and label various robot behaviors. We considered a setting where the robot had a full battery, but was picking up a fragile box and thus still needs to visit the station #1. The robot could slip on some cells marked in dark grey with probability 0.25 (slipping here meant the robot picture is tilted to give an impression that it slipped on the cell and didn't prevent the robot from moving to the next cell). To make sure that all the users had similar mental models at the start, they were provided with the following facts, (a) that robot couldn't pass through racks, (b) whenever the robot runs low on battery it needs to get to Station 1, (c) whenever the robot has a green battery sign next to the robot, that means their battery is full and (d) the robot needs to take the shortest route to the goal. Also, they were presented with an example trace in this instructions section and were made to take a small pre-test that allowed them to revise the above facts in various scenarios. After the pre-test, they were shown eight traces from the robot policy sampled according to their probabilities. After the first trace, the user was given an explanation message before each trace, where the message was taken from the seven possible messages (the order of the messages was always randomized).

From the data collected from 45 turkers, we removed data from seven users, based on the fact they didn't find any of the transition in the first trace (i.e the case where no explanation was provided) inexplicable. We imagine this number would go down when we move to expert users or users who are invested in the success of the robot. The data generated for the remaining 38 users were then used to train a decision tree. Since the placement of other objects in the environment were fixed, we were able to use rather simple features for the model like the current position of the robot (x and y), previous position (again x and y), the action, whether they have slipped and finally the explanations given. We found the model to have an average 10-fold cross validation score of 0.935. For a randomly generated train and test split (where the test split was 10% and contained around 7% inexpicable labels) the precision score was 0.9637 and the recall score was 0.9568.

Furthermore, we could see that the model was able to correctly predict the usefulness of intuitive minimal explanations for the given scenario. For example, it predicted that while the robots decision to visit station #1 would be considered inexeplicable by the user in the absence of any explanation, the user would mark it as explicable when they are explained about the box being fragile and that fragile boxes need to be inspected at station #1. In fact the model predicted that only the message that "fragile boxes need to be inspected at station #1" is enough to convince the user about the need for that action (i.e the user could deduce that the box must have been fragile). This shows that such learned models may help us generate cheaper explanations (the above set of explanations is smaller than the corresponding minimal complete behavior explanation for the domain), by taking into account the users ability to correctly predict missing information in simple cases. Another point of interest was that the model predicted all slipping events as explainable even in the absence of any explanations. The cases where the user saw a slip before being told about the possibility of slipping was rare (since there are two explanatory messages related to slipping and the probability of slipping was 0.25). Furthermore when we went over the data, we found that in most such cases, the users did mark it as explainable. This may be because the effect of slipping may not have been that detrimental to the overall plan (it doesn't take you off the current path). It would be interesting to see if this result would be the same in cases where slipping was a more likely event and if it had a more apparent effect on the robot's plan.

7 Related Work

To the best of our knowledge, this work represents the first attempt at learning proxies for user mental models that allows an agent to predict the potential impact of providing explanations as model reconciliation to observers. With that said, there have been works that have looked at the problem of generating explanations in the presence of model uncertainty for human models. In particular, our previous works like [Sreedharan *et al.*, 2018a; 2018b] have looked at cases where the agent has access to a set of potential human models. One drawback of considering a set of possible models is either they would need to have explicit sensing to identify the user model (which could mean asking a large number of questions to the user) or providing a large amount of information to cover the space of all possible models. In our work, the problem of identifying the specifics of the user model is resolved through an offline training process.

Another work quite related to the discussion covered in this paper is [Reddy *et al.*, 2018], wherein the authors tried to identify cases where they can learn a potential model for the human's expectation of the task transition dynamics when they do not align with the real world dynamics. Unlike their work, we do not assume that the user can provide traces for the given task, rather they may be able to provide some high-level feedback on the action (i.e. they may not be able to do or even know the right action but may be able to point out actions or transitions that surprise them). Moreover, their work requires that the user and the robot must have the same reward function, which is again an assumption we do not make. Even if we had followed their technique to learn a potential approximation of the human's transition model for the task, there is no guarantee that the learned representation would be one that makes sense to the human.

8 Discussion and Conclusion

This paper proposes a possible way in which model reconciliation explanation could be applied to cases where the user model is unknown. The method described here is a rather simple and general method to identify information that could potentially affect the user's mental model and produce effects that align with the agent's requirements. There is no requirement here that the messages have to align with actual facts about the world. This again points to the rather troubling similarities between the mechanisms needed to generate useful explanations and lies [Chakraborti and Kambhampati, 2018].

Two important assumptions we made throughout the work is that the user only considers the current state (as defined by the robot) to make their decisions and we have access to a model that was learned from interactions to previous users who had similar knowledge level to the current user. Relaxing the first assumption would require us to go beyond learning models that map each transitions to labels. Instead we have to consider sequential labeling models (for example models based on LSTM or CRF) of the type considered in [Zhang *et al.*, 2017] to capture the human's expectations. For example, we considered a simple extension of the warehouse domain where the human believes the robot should visit two locations (i.e the human state contains variables that record whether the user has visited the locations). Even though here the user is considering a more detailed model, we were able to learn labeling models of 80% accuracy by using simple CRFs. As for the second, instead of assuming that all users are of the same type, a more reasonable assumption may be that the users could be clustered into N groups and we could learn a different labeling model for each user type. Now we still have a challenge of identifying the user type of a new user and one way to overcome this would be by adopting a decision-theoretic approach to this problem and modeling it as a POMDP (where user labels become observations and previously learned user models the observation models).

The work discussed in this paper only covers explanations that allow the user and the system to reconcile any model difference. This only covers a part of the entire explanatory dialogue. Even if there is no difference in models, the user may still have questions about parts of the policy or may raise alternative policies they think should be followed. This may arise from a difference in inferential abilities and may require providing information that is already part of their deductive closure eg: help them understand the long term consequences of taking some actions. Once you have access to a set of such messages one could use a method similar to the one described in the paper to find the set of helpful ones. Unlike the model reconciliation setting where the messages stand for information about the model, it is not quite clear how one could automatically generate such messages.

Acknowledgments

This research is supported in part by the ONR grants N00014-16-1-2892, N00014-18-1-2442, N00014-18-1-2840, the AFOSR grant FA9550-18-1-0067, and the NASA grant NNX17AD06G.

References

[Abel, 2019] David Abel. simple_rl: Reproducible Reinforcement Learning in Python. *ICLR Workshop on Reproducibility in Machine Learning*, 2019.

[Chakraborti and Kambhampati, 2018] Tathagata Chakraborti and Subbarao Kambhampati. (when) can ai bots lie? In *AIES*, 2018.

[Chakraborti *et al.*, 2017] Tathagata Chakraborti, Sarath Sreedharan, Yu Zhang, and Subbarao Kambhampati. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *IJCAI*, pages 156–163, 2017.

[Chakraborti *et al.*, 2019] Tathagata Chakraborti, Sarath Sreedharan, and Subbarao Kambhampati. Balancing explicability and explanation in human-aware planning. *IJCAI*, 2019.

[Daniele Magazzeni, 2018] Pat Langley Susanne Biundo Daniele Magazzeni, David Smith, editor. *Proceedings of the 1st Workshop on Explainable Planning*. ICAPS, 2018.

[David W. Aha and Magazzeni, 2018] Patrick Doherty David W. Aha, Trevor Darrell and Daniele Magazzeni, editors. *Proceedings of the 2nd Workshop on Explainable Artificial Intelligence*. IJCAI, 2018.

[Dietterich, 1998] Thomas G Dietterich. The maxq method for hierarchical reinforcement learning. In *ICML*, volume 98, pages 118–126. Citeseer, 1998.

[Hayes and Shah, 2017] Bradley Hayes and Julie A Shah. Improving robot controller transparency through autonomous policy explanation. In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 303–312. IEEE, 2017.

[Khan *et al.*, 2009] Omar Zia Khan, Pascal Poupart, and James P Black. Minimal sufficient explanations for factored markov decision processes. In *Nineteenth International Conference on Automated Planning and Scheduling*, 2009.

[Kulkarni *et al.*, 2016] Anagha Kulkarni, Tathagata Chakraborti, Yantian Zha, Satya Gautam Vadlamudi, Yu Zhang, and Subbarao Kambhampati. Explicable Robot Planning as Minimizing Distance from Expected Behavior. *CoRR*, abs/1611.05497, 2016.

[Lombrozo, 2006] Tania Lombrozo. The structure and function of explanations. *Trends in Cognitive Sciences*, 10(10):464 – 470, 2006.

[Miller, 2018] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 2018.

[Reddy *et al.*, 2018] Siddharth Reddy, Anca D Dragan, and Sergey Levine. Where do you think you're going?: Inferring beliefs about dynamics from behavior. *NIPS*, pages 1454–1465, 2018.

[Ribeiro *et al.*, 2016] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.

[Russell and Norvig, 2003] Stuart Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Prentice Hall, 2003.

[Seegebarth *et al.*, 2012] Bastian Seegebarth, Felix Müller, Bernd Schattenberg, and Susanne Biundo. Making hybrid plans more clear to human users-a formal approach for generating sound explanations. In *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

[Sreedharan *et al.*, 2018a] Sarath Sreedharan, Subbarao Kambhampati, et al. Handling model uncertainty and multiplicity in explanations via model reconciliation. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*, pages 518–526, 2018.

[Sreedharan *et al.*, 2018b] Sarath Sreedharan, Siddharth Srivastava, and Subbarao Kambhampati. Hierarchical expertise level modeling for user specific contrastive explanations. In *IJCAI*, pages 4829–4836, 2018.

[Sreedharan *et al.*, 2019] Sarath Sreedharan, Tathagata Chakraborti, Christian Muise, and Subbarao Kambhampati. Planning with Explanatory Actions: A Joint Approach to Plan Explicability and Explanations in Human-Aware Planning. *arXiv preprint arXiv:1903.07269*, 2019.

[Sutton *et al.*, 1999] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[Topin and Veloso, 2019] Nicholay Topin and Manuela Veloso. Generation of policy-level explanations for reinforcement learning. In *AAAI*, pages 1074–1080, 2019.

[Zhang *et al.*, 2017] Yu Zhang, Sarath Sreedharan, Anagha Kulkarni, Tathagata Chakraborti, Hankz Hankui Zhuo, and Subbarao Kambhampati. Plan Explicability and Predictability for Robot Task Planning. In *ICRA*, pages 1313–1320, 2017.

Explaining the Space of Plans through Plan-Property Dependencies

Rebecca Eifler,[1] **Michael Cashmore,**[2] **Jörg Hoffmann,**[1]
Daniele Magazzeni,[2] **Marcel Steinmetz,** [1]
[1]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany,
[2]King's College London, Department of Informatics, London, UK,
{lastname}@cs.uni-saarland.de, {firstname.lastname}@kcl.ac.uk

Abstract

A key problem in explainable AI planning is to elucidate decision rationales. User questions in this context are often contrastive, taking the form "Why do A rather than B?". Answering such a question requires a statement about the space of possible plans. We propose to do so through plan-property dependencies, where plan properties are Boolean properties of plans the user is interested in, and dependencies are entailment relations in plan space. The answer to the above question then consists of those properties C entailed by B. We introduce a formal framework for such dependency analysis. We instantiate and operationalize that framework for the case of dependencies between goals in oversubscription planning. More powerful plan properties can be compiled into that special case. We show experimentally that, in a variety of benchmarks, the suggested analyses can be feasible and produce compact answers for human inspection.

Introduction

Explainable AI (XAI) is concerned with making AI systems' decisions more lucid and thus trustworthy. AI planning is relevant to XAI as a decision-making methodology, model-based and thus suited to provide explanations in principle. Consequently, research on explainable AI planning (XAIP) has received increasing interest in recent years (e. g. (Seegebarth *et al.* 2012; Smith 2012; Langley *et al.* 2017; Fox *et al.* 2017; Chakraborti *et al.* 2017; 2019)).

A recent analysis (Miller 2019) of lessons to be learned for XAI from social sciences highlights that user questions are often *contrastive*. A question "Why this?" actually means "Why this *rather than something else* that I would expect?". To address such queries, explanatory systems should analyse alternative solutions, and support the user in understanding the consequences of the "something else" in question. AI planning fits well for this kind of analysis. Two prior works designed variants thereof (Fox *et al.* 2017; Miller 2018). The work by Fox et al. is the starting point of our work here.

Fox et al. suggest, given a plan π and a user question "Why does π start with action A rather than B?", to generate a new plan π' starting with B, and answer the question based on comparing the two plans: undesirable properties of π' serve to explain the previous decision. While this idea is natural, a key weakness is that there may be differences between π and π' unrelated to the use of A vs. B. Many comparison aspects (e. g. which other actions are used, or which "soft" objectives are satisfied) may be affected by arbitrary decisions in the planner's search.

Here we address the same kind of explanation problem, but we replace the *existential* answer generating a single alternative plan π' with a *universal* answer determining shared properties of *all* possible such alternatives. In this way, the analysis we propose aims at explaining the space of possible plans, rather than pointing out examples.

Our proposed analysis works at the level of **plan properties**: Boolean functions on plans that capture aspects of plans the user cares about (whether or not the plan starts with a particular action, whether or not a particular soft objective is satisfied, etc). We assume that the set P of plan properties of interest is given as part of the input.[1] Our analysis then determines the **dependencies** across plan properties, i. e., **plan-space entailments** which we define as follows. The "plan space" is the set Π of candidate plans to be considered (canonically, the set of plans for an input planning task). A plan property p **entails** another property p' in Π if every $\pi \in \Pi$ that satisfies p also satisfies p'. A user question "Why does the current plan π satisfy p rather than q?" can then be answered in terms of the properties q' not true in π but entailed by q: things that will *necessarily* change when satisfying q.

Our approach also supports iterative planning, along the lines suggested by Smith (2012). Given a current plan $\pi \in \Pi$ and a user question "Why achieve p rather than q?", if the consequences of q are tolerable to the user, she may choose to enforce q, gradually narrowing the plan-candidate space Π.

We remark that our approach can be viewed as an intermediate between domain/task analysis (e. g. (Fox and Long 1998)), which our approach generalizes; and model checking applied to planning models, which our approach is an instance of (related to (Vaquero *et al.* 2013)).

Our contributions are as follows. We conceptualize the explainability problems we address, through a generic framework making minimal assumptions on the planning context (Section). We instantiate the framework with goal-

[1]An interesting yet challenging question for future work is how one can automatically identify relevant plan properties.

fact conjunction dependencies in oversubscription planning (e. g. (Smith 2004; Domshlak and Mirkis 2015)), and devise analysis algorithms for that purpose (Section). We show that more general plan properties – in particular, **action-set properties** – can be compiled into goal facts and thus into that analysis (Section). We give an illustrative example (Section), and we evaluate our techniques on international planning competition (IPC) benchmarks modified for oversubscription planning, and on IPC benchmarks extended with action-set properties (Section). We find that, in a variety of benchmark studies, the suggested analyses can be feasible and produce compact answers for human inspection.

Generic Framework

We assume some formalism defining planning **tasks** τ. We do not need any assumptions about that formalism, except that it defines a concept of **plans** π, where that concept can again be arbitrary (action sequence/schedule/partial order/etc). Our definitions are relative to a set Π of plans of interest. The canonical setup we have in mind is that where Π is induced by τ, e. g. as the set of action sequences applicable in the initial state, or as the set of plans that achieve a goal. It could also be useful in some cases though to focus the analysis on a small set of candidate plans listed as part of the input.

Plan Properties and Property Entailment

Plan properties, in their most general form, are simply functions mapping a task and plan to a Boolean value indicating whether or not the property is satisfied:

Definition 1 (Plan Property)**.** *Denoting by $\mathcal{T}$ the set of all tasks and by $\mathcal{P}$ the set of all plans, a* plan property *is a partial function $p : \mathcal{T} \times \mathcal{P} \mapsto \{true, false\}$. Given a task τ and a set of plans Π, we say that p is a plan property defined on τ and Π if its domain includes $\{(\tau, \pi) \mid \pi \in \Pi\}$.*

Example plan properties are goal facts/goal formulas (true at end of plan?), temporal plan trajectory constraints, constraints on subsets of actions used/not used, deadlines, bounds on resource consumption, etc. We expect that, typically, p will be computable in time polynomial in the size of its input (though that is not a requirement of our framework).

We assume a set P of plan properties as part of our input. P may be exponentially large in the size of its specification though. An example we will explore later is that where the user is interested in dependencies between subsets of a set G of soft-goal facts. The set P of interest then are the conjunctions ϕ over G (functions checking whether ϕ is true at the end of a plan), but the input to our analysis specifies only G.

The kind of dependency our framework focuses on is entailment over plan properties, in the space of truth-value assignments induced by the plan-candidate set Π:

Definition 2 (Π-Entailment)**.** *Let τ be a task, Π a set of plans, and P a set of plan properties defined on τ and Π.*

*Let $\pi \in \Pi$. We identify π with the truth-value assignment $\pi : P \mapsto \{true, false\}$ where $\pi(p) := p(\tau, \pi)$. We identify Π with the set of such truth-value assignments. We say that π **satisfies** p, written $\pi \models p$, if $\pi(p) = true$. We denote by $\mathcal{M}_\Pi(p) := \{\pi \mid \pi \in \Pi, \pi \models p\}$ the models of p.*

*We say that p Π-**entails** q, written $\Pi \models p \Rightarrow q$, if $\mathcal{M}_\Pi(p) \subseteq \mathcal{M}_\Pi(q)$. We say that p and q are Π-**equivalent**, written $\Pi \models p \Leftrightarrow q$, if $\mathcal{M}_\Pi(p) = \mathcal{M}_\Pi(q)$. We denote $[p]_\Pi := \{q \mid q \in P, \Pi \models p \Leftrightarrow q\}$.*

This definition essentially just views plans $\pi \in \Pi$ as truth-value assignments in the obvious manner. Entailment and equivalence over plan properties are then defined straightforwardly, with Π in the role traditionally taken by a knowledge base that restricts the truth-value assignments under consideration. Observe that formulas over plan properties can be encoded as individual plan properties, so that defining Π-entailment over individual plan properties is enough to permit logical combinations thereof.

Importantly, the role of Π as a knowledge base means that Π-entailment is more than standard entailment: the latter implies the former, but not vice versa. As a simple example, say the plan properties P are propositional formulas ϕ over facts, evaluated at the end of the plan. Then $\phi \Rightarrow \psi$ implies that $\Pi \models \phi \Rightarrow \psi$, simply because any (plan-end) state that satisfies ϕ must satisfy ψ. But not vice versa: e. g. if facts p, q are mutex in the task then $\Pi \models p \Rightarrow \neg q$. As a more motivating example, say the plan properties are soft goals (like having scientific observations in satellite planning) as well as resource preferences (like consuming at most a given amount of energy). Then entailments of interest can take the form $\Pi \models p \Rightarrow \neg(q_1 \wedge q_2 \wedge q_3)$ saying that we cannot have p without foregoing either of q_1 or q_2 or q_3. Note that this is an entailment specific to Π, which may not hold in general (e. g. if cheaper actions are available, or if cheaper plans are admitted by removing some other hard goals). The identification of such specific entailments – specific to the space Π of plans considered – is central to our framework.

Plan-Space Explanations

Our plan-space explanations are based on the Π-entailment relation on P given the knowledge base Π:

Definition 3 (PDO, cPDO)**.** *Let τ be a task, Π a set of plans, and P a set of plan properties defined on τ and Π.*

*The **plan-property dependency order (PDO)** for Π and P is the partial order $\Rightarrow_\Pi$ over the equivalence classes $[p]_\Pi$, where $[p]_\Pi \Rightarrow_\Pi [q]_\Pi$ iff $\Pi \models p \Rightarrow q$.*

*A **concrete PDO (cPDO)** replaces each equivalence class $[p]_\Pi$ with exactly one $p \in [p]_\Pi$.*

The PDO makes explicit how the plan properties P depend on each other. For all contrastive user questions of the form "Why r rather than p?", the answer can be directly extracted from the PDO, in terms of the properties entailed by p. For example, the answer may be "we cannot have p without foregoing either of q_1 or q_2 or q_3".

However, the PDO and the answers it provides can be large. A concrete PDO can be a practical proxy if equivalence classes are large. Beyond that, it is clearly important to identify (i) more compact and/or (ii) more restricted plan-space explanations. We introduce variants of both here.

Regarding (i), in our concrete instantiation of this framework we use **subsumption** over Π-entailment relations, relying on an easy-to-test sufficient criterion for Π-entailment:

Definition 4 (Dominant cPDO)**.** *Let τ be a task, Π a set of plans, and P a set of plan properties defined on τ and Π.*

*Let $\Rightarrow_{suff} \subseteq P \times P$ be such that, if $p \Rightarrow_{suff} q$, then $\Pi \models p \Rightarrow q$. In a cPDO, we say that $p \Rightarrow_\Pi q$ **subsumes** $p' \Rightarrow_\Pi q'$ **given** $\Rightarrow_{suff}$ if $p' \Rightarrow_{suff} p$ and $q \Rightarrow_{suff} q'$.*

*A **dominant cPDO (dcPDO)** for Π and P given $\Rightarrow_{suff}$ is the subset of non-subsumed $p \Rightarrow_\Pi q$ in a cPDO.*

An entailment $p \Rightarrow_\Pi q$ subsumes another one $p' \Rightarrow_\Pi q'$ if its left-hand side is weaker ($p' \Rightarrow_{suff} p$) and its right-hand side is stronger ($q \Rightarrow_{suff} q'$): in this case, $p' \Rightarrow_\Pi q'$ follows from $p \Rightarrow_\Pi q$. A dominant cPDO thus selects only the strongest Π-entailments in a cPDO, as a more compact representation of the information present in that cPDO.

The role of $\Rightarrow_{suff}$ here is to qualify the amount of information we are allowed to use in identifying this compact representation. This is important because, if we show compacted information to a user, then the user should be able to de-compact this information – to obtain whichever information the user is actually interested in – effortlessly. A simple restriction is for $\Rightarrow_{suff}$ to be computable in polynomial time, but cognitive abilities may necessitate stronger restrictions. Here we will consider goal-fact conjunctions and disjunctions, and use the trivial $\Rightarrow_{suff}$ where larger conjunctions are stronger while larger disjunctions are weaker.

As a simple form of (ii) more restricted plan-space explanations, we will employ the restriction of focus to a predefined subset D of dependencies of interest:

Definition 5 (Restricted (dc)PDO)**.** *Let τ be a task, Π a set of plans, and P a set of plan properties defined on τ and Π.*

*Let $D \subseteq P \times P$ be any binary relation on plan properties. Then a **(dc)PDO for** D results from ignoring Π-entailments $\Pi \models p \Rightarrow q$ where $(p, q) \notin D$.*

Some words are in order regarding complexity. Testing Π-entailment encompasses the plan existence problem even for extremely simple plan properties (asking whether the plan achieves a fact p). This is exacerbated by the size of the PDO. Certainly, a (dc)PDO should ideally be computed offline, prior to interaction with a user.

Goal Dependencies

We now instantiate our framework with a concrete use case: dependencies between goals in oversubscription planning, where the question addressed is which combinations of (soft) goals exclude which other combinations. In Section , we will show how to compile a more powerful plan property language into this special case.

Planning Framework

Most of the techniques we introduce in what follows are applicable to a broad range of planning frameworks. Nevertheless, for a concrete exposition, henceforth we consider the *finite-domain representation (FDR)* framework (Bäckström and Nebel 1995; Helmert 2009), with finite-domain state variables as used in the Fast Downward system (Helmert 2006) on which our implementation is based.

An FDR task τ is a tuple $\tau = (V, A, c, I, G)$ where V is the set of **variables**, A is the set of **actions**, $c : A \mapsto \mathbb{R}_0^+$ is the action **cost** function, I is the **initial state**, and G is the **goal**. A **state**, in particular I, is a complete assignment to V; G is a partial assignment to V; each action $a \in A$ has a **precondition** pre_a and an effect eff_a, both partial assignments to V. We will refer to variable-value pairs $v = d$ as **facts**, and we will identify partial variable assignments with sets of facts. An action a is **applicable** in a state s if $pre_a \subseteq s$. The outcome state $s[[a]]$ is like s except that $s[[a]](v) = eff_a(v)$ for those v on which eff_a is defined. The outcome state of an iteratively applicable action sequence π is denoted $s[[\pi]]$.

We address an oversubscription variant of FDR, where an **oversubscription planning (OSP) task** is a tuple $\tau = (V, A, c, I, G, b)$ exactly like an FDR task but with an additional **cost bound** $b \in \mathbb{R}_0^+$. Intuitively, the goals G are "soft", and the challenge is to achieve a maximally valuable subset of G within the cost bound. OSP frameworks in the literature employ notions (e. g. goal-fact rewards) of what it means to be "maximally valuable". Here we assume instead that the user's preferences over the soft goals are difficult to specify and/or elicitate, so that an in-depth characterization of the trade-offs between different goal sets – their dependencies – is of interest. In the terms of our framework, this means that the set Π of **plans** is simply the set of all action sequences $\pi = \langle a_1, \dots, a_n \rangle$ applicable in I and where $\sum_{i=1}^{n} c(a_i) \leq b$. An analysis over suitable sets of properties P and dependencies D then yields the desired trade-off information.

Plan Properties

The plan properties we consider here are characterized by propositional formulas over goals:

Definition 6 (Goal Properties)**.** *Let $\tau = (V, A, c, I, G, b)$ be an OSP task, and Π its set of plans.*

*A **goal property** for τ is a function $p_\phi : \Pi \mapsto \{true, false\}$ where ϕ is a propositional formula over the atoms G, and $p_\phi(\pi) = true$ iff ϕ evaluates to true under the truth value assignment where $g \in G$ is true iff $g \in I[[\pi]]$.*

We identify goal properties p_ϕ with the characterizing formulas ϕ. We consider a class of properties and dependencies identifying exclusions between goal conjunctions:

Definition 7 (Goal Exclusion)**.** *Let $\tau = (V, A, c, I, G, b)$ be an OSP task, and Π its set of plans.*

*The **PDO for goal exclusion (PDO-GE)** is the PDO for Π, the property set $P^{\text{GE}} := \{\bigwedge_{a \in A} g \mid A \subseteq G\} \cup \{\neg \bigwedge_{g \in B} b \mid B \subseteq G\}$, and the dependency set $D^{\text{GE}} := \{(\bigwedge_{a \in A} a, \neg \bigwedge_{b \in B} b) \mid A \cap B = \emptyset\}$.*

We restrict focus to goal conjunctions and negations thereof, and we are interested only in implications of the form $\Pi \models \bigwedge_{a \in A} a \Rightarrow \neg \bigwedge_{b \in B} b$ stating that, if we achieve all of A, we have to forego at least one of B. The PDO-GE then explains to the user how exactly different goal subsets exclude each other, identifying the fine-grained trade-off.

Given the restriction to D^{GE}, the equivalence classes in the PDO-GE are singletons. Hence there is a unique cPDO-GE, that we identify with the PDO-GE itself.

For compacting the information presented to a user, we use the sufficient criterion for entailment where $\bigwedge_{a \in A'} g \Rightarrow_{\mathit{suff}} \bigwedge_{a \in A} a$ iff $A' \supseteq A$ and $\neg \bigwedge_{b \in B} b \Rightarrow_{\mathit{suff}} \neg \bigwedge_{b \in B'} g$ iff $B \subseteq B'$. The dominant PDO-GE thus selects the entailments with minimal left-hand side conjunctions excluding minimal right-hand side conjunctions.

Computing the Dominant PDO-GE

The dominant PDO-GE can be read off the **minimal unsolvable goal subsets (MUGS)**, where $G' \subseteq G$ is a MUGS if G' cannot be achieved but every $G'' \subsetneq G'$ can:

Proposition 1 (PDO-GE from MUGS). *Let $\tau = (V, A, c, I, G, b)$ be an OSP task, and Π its set of plans.*

Then $\Pi \models \bigwedge_{a \in A} a \Rightarrow \neg \bigwedge_{b \in B} b$ is in the dominant PDO-GE if and only if $A \cup B$ is a MUGS.

Proof. A Π-entailment $\Pi \models \bigwedge_{a \in A} a \Rightarrow \neg \bigwedge_{b \in B} b$ clearly holds iff $A \cup B$ is unsolvable. Dominant entailments in the PDO-GE result from set-inclusion minimal A and B, corresponding to the set-inclusion minimality of MUGS. □

Our computational problem thus boils down to computing all MUGS. This can be done through a search over goal sets, that we refer to as **systematic weakening (SysW)**:

(1) the start node of the search is G;

(2) each search step selects an open node G', calls a planner to test whether G' is solvable in τ, caches the result, and expands G' if it is unsolvable;

(3) the children of a node G' are those $G'' \subset G'$ where $|G''| = |G'| - 1$.

Upon termination, the MUGS are those nodes G' all of whose children are solvable.

Dually, **systematic strenghtening (SysS)** starts from $\emptyset$, with search steps expanding solvable nodes, and children adding one more goal fact. Upon termination, the MUGS can be easily obtained from the unsolvable search nodes.

In both SysW and SysS, every goal set can be reached from the start node by permutations of the same goal-fact removal/addition steps. We avoid duplicate planner calls by caching. We give goal sets unique integer IDs, for fast cache lookup, and to fix the expansion order so that we always know whether or not we have generated a node before.

As a non-trivial search enhancement, we created synergy with recent nogood learning techniques, **conjunction learning** (Steinmetz and Hoffmann 2017b) and **trap learning** (Steinmetz and Hoffmann 2017a). These techniques refine dead-end detection methods (nogoods) based on the unsolvable states encountered in state space search on a planning task. As the children tasks in our searches are closely related to their parents, the refined nogoods are likely to be useful still. So we **transfer** the nogoods along search paths, resulting in iteratively stronger and stronger nogoods. For both conjunction learning and trap learning, the nogoods learned depend on the goal, so that only some of the nogoods remain valid for transfer in SysW where children remove goals. We designed simple methods to identify this nogood subset, keeping track of goal dependencies in conjunction learning, and re-asserting trap validity in trap learning.

Yu et al. (2017) perform an analysis related to MUGS, to suggest goals to drop in oversubscribed situations. They address conditional temporal problems (a form of conditional temporal plans), and leverage previous conflict analysis methods in that area. It remains a question for future work whether such conflict analysis could inspire different analysis methods in our planning framework.

Compilations into Goal Dependencies

The analysis of goal properties just described can be used to analyze more complex properties that can be compiled into goal facts. Given the well-known power of compilation in planning languages (e. g. (Gazen and Knoblock 1997; Nebel 2000; Edelkamp 2006; Palacios and Geffner 2009; Baier *et al.* 2009)), there is large potential in this idea. As an example, here we consider what we refer to as action-set properties:

Definition 8 (Action-Set Properties). *Let $\tau = (V, A, c, I, G, b)$ be an OSP task, Π its set of plans, and $A_1, \dots, A_n \subseteq A$.*

*An **action-set property** for τ and $A_1, \dots, A_n$ is a function $p_\phi : \Pi \mapsto \{\mathit{true}, \mathit{false}\}$ where ϕ is a propositional formula over the atoms $A_1, \dots, A_n$, and $p_\phi(\pi) = \mathit{true}$ iff ϕ evaluates to true under the truth value assignment where A_i is true iff π contains at least one action from A_i.*

As before, we identify action-set properties p_ϕ with the characterizing formulas ϕ. Arguably, action-set properties are practically relevant. They allow to express things like “objective x is covered by satellite y”, “route x is not used”, “passengers x and y ride in the same vehicle”, etc. At the same time, the simple syntax of action-set properties lends itself to effective compilation, as follows.

Given τ, Π, and $A_1, \dots, A_n$ as in Definition 8, to obtain a compiled task τ'

1) introduce Boolean flags isUsed_i that are initially false and set to true by any action from A_i;

2) introduce formula-evaluation state variables and actions evaluating each p_ϕ based on these (following (Gazen and Knoblock 1997; Nebel 2000)), setting Boolean flags isTrue_ϕ storing the outcome values;

3) introduce a separate 1) *planning phase* vs. 2) *formula-evaluation phase*, and a switch action allowing to go from 1) to 2).

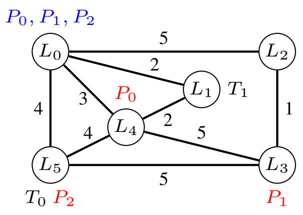
Then the planning-phase prefixes in τ' are in one-to-one correspondence with Π, and given such a prefix π the evaluation phase in τ' can achieve isTrue_ϕ iff $p_\phi(\pi) = \mathit{true}$.

Now say that we want to analyze the dependencies across a given set P of action-set properties (e. g. possible undesirable consequences of not using route X). We are given τ, Π, and P; we want to compute the PDO for property exclusion over P, i. e., the dependencies of the form $\Pi \models \bigwedge_{\phi \in A} \phi \Rightarrow \neg \bigwedge_{\psi \in B} \psi$. With the above, this can be done by instead computing the PDO-GE for τ' with goal set $\{\mathit{isTrue}_\phi \mid \phi \in P\}$, and identifying each isTrue_ϕ with ϕ in the outcome.

Clearly, similar compilation techniques can be used for much more powerful property languages. In a preliminary exploration, we implemented a compilation for LTL properties based on previous work (Edelkamp 2006; Baier *et al.* 2009). Our results indicate that this renders the PDO analysis infeasible computationally. It remains an open question how LTL properties can be addressed more effectively.

An Illustrative Example

To illustrate our approach and the kind of explanations it provides, consider the IPC NoMystery domain, a classical transportation domain with fuel consumption. We consider the example task with two trucks and three packages as illustrated below. Fuel costs are indicated at road segments (initial fuel is 16 for T_0 and 7 for T_1). The packages are initially at L_0 (shown in blue); their goal locations are L_4, L_3,and L_5 (shown in red). We define three kinds of action-set properties for this domain: *uses* T_i (L_x, L_y) (truck T_i drives at least once from L_x to L_y or vice versa); *doesn't use* T_i (L_x, L_y) (the opposite); *same truck* P_x P_y (both packages are delivered by the same truck). In our example task, we consider six instances of these properties: 1. uses T_0 (L_2, L_3); 2. same truck P_1 P_2; 3. uses T_0 (L_4, L_3); 4. same truck P_2 P_0; 5. doesn't use T_0 (L_0, L_5); 6. uses T_1 (L_5, L_4).



We fix the package destinations as hard goals, defining the set of plans Π considered. Computing the MUGS over the six action-set properties using the algorithms previously described, it turns out there are seven minimal unsolvable subsets of these properties, each of size three.

Say now that the current plan uses T_0 only, and includes the action (drive T0 L5 L0). The user might ask *"Why don't you avoid the road $L_0 - L_5$, which has a lot of traffic at the moment?"*. Answering this question in terms of contrastive explanation, as previously discussed, corresponds to forcing property 5 to be satisfied. At the same time, the plan already satisfies properties 2 and 4. However, one of the MUGS is $\{2, 4, 5\}$, and hence the answer to the user question would be: *Because if you don't use that road, then you would not be able to deliver all packages using a single truck.*

Experiments

We implemented our approach in Fast Downward (FD) (Helmert 2006). We evaluate it, in turn, on IPC benchmarks modified for oversubscription planning, and on a selection of IPC benchmarks extended with action-set properties.

In all experiments, the base planner called by our SysS and SysW algorithms on each search node employs h^{FF} (Hoffmann and Nebel 2001) for search guidance. The experiments were run on a cluster of Intel E5-2660 machines running at 2.20 GHz, with time (memory) cut-offs of 30 minutes (4 GB).

Oversubscription Planning

To evaluate our analysis of goal dependencies as per Section , we modified all optimal-planning STRIPS IPC domains up to IPC'18. Following Domshlak and Mirkis (2015), for each benchmark task we ran an optimal planner (A*with $h^{\text{LM-cut}}$ (Helmert and Domshlak 2009)) to determine the optimal plan cost C, then obtained OSP tasks by setting the cost bound to $b = x * C$ where $x \in \{0.25, 0.5, 0.75\}$. Our benchmark set consists of 46 domains, and contains those tasks solved by the optimal planner, and where the number of goal facts is < 32. We extended conjunction learning (Steinmetz and Hoffmann 2017b) to deal with cost bounds, thus enabling nogood learning and transfer in SysS and SysW.

Figure 1 shows our data. Consider first the coverage data (leftmost two parts). To have some sort of measure of how computationally difficult our proposed analysis is, we use reference points from classical planning. First, the $h^{\text{LM-cut}}$ column gives coverage for A* with $h^{\text{LM-cut}}$ run on the original IPC instance without a cost bound. This provides a comparison to solvable optimal planning. Second, the h^C columns give coverage for search, with nogood learning, on the respective cost-bounded instances, when all goals must be achieved and thus the task is unsolvable. This provides a comparison to proving unsolvability, in the same situation where our approach computes all MUGS. It is expected that our algorithms, solving a more complex problem, will perform worse than the reference points.[2] The question is, *how much worse?*

As a short summary of the answer provided by Figure 1 to that question, compared to the $h^{\text{LM-cut}}$ reference point, taking the per-domain best of our four algorithm configurations, for $x = 0.25$ we get equal coverage in 36 of the 46 domains, and in that sense are "not much worse" than optimal planning. For larger cost bounds, the solvable goal subsets become larger, and accordingly our analysis becomes harder. For $x = 0.5$ we get equal coverage in 23 domains, for $x = 0.75$ in 13. The comparison to the h^C proving-unsolvability reference points is qualitatively similar, with equal coverage in 38, 25, and 20 domains for $x = 0.25, 0.5, 0.75$ respectively. Overall, it seems fair to say that our analyses can be feasible in many cases, in the sense of not being more infeasible than the most closely related classical planning problems.

While comparing our algorithm configurations against each other is not our focus here, observe in the rightmost part of Figure 1 that both SysS and SysW suffer from larger cost bounds, but that is less so for SysW. This is because, for small cost bounds, solvable goal sets are small and thus SysS terminates earlier; while for large cost bounds, solvable goal sets are large and thus SysW terminates earlier. Conjunction learning (h^C in the table) is moderately beneficial.

Consider finally the #MUGS part of Figure 1. Observe that, if the user asks a question "Why r rather than p?", the answer are the properties entailed by p, represented here

[2]Indeed, the first reference point is an upper bound to our coverage, as only solved instances are included in our benchmark set; and the second reference point is an upper bound for SysW as it constitutes the first search node in that algorithm.

	Reference Points				Coverage, $x =$												#MUGS, $x =$						Search Tree Fraction, $x =$					
	$h^{\text{LM-cut}}$	h^C			0.25				0.5				0.75				0.25	0.5	0.75	0.25	0.5	0.75	0.25		0.5		0.75	
		$x =$			SysS		SysW		SysS		SysW		SysS		SysW								Sys		Sys		Sys	
domain	-	0.25	0.5	0.75		h^C		h^C		h^C		h^C		h^C		h^C		avg			max		S	W	S	W	S	W
agricola (20)	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
airport (50)	28	28	24	17	25	26	24	**27**	19	**21**	19	**21**	**19**	16	**19**	16	2.7	2.0	1.2	11	5	4	**0.67**	0.76	0.88	**0.71**	1.00	**0.61**
barman (34)	4	4	4	4	4	4	4	4	4	4	4	4	**4**	0	**4**	**4**	3.0	3.0	1.0	3	3	1	**0.50**	0.88	0.88	0.88	-	-
blocks (35)	28	28	28	28	**28**	**28**	27	**28**	23	**27**	21	**27**	18	24	17	**26**	7.6	10.8	14.1	39	30	57	**0.19**	0.97	**0.39**	0.93	0.78	**0.72**
childsnack (20)	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
data-network (20)	12	12	12	12	12	12	12	12	12	12	12	12	11	**12**	11	**12**	1.7	1.5	1.2	3	3	2	0.83	**0.65**	0.88	**0.65**	0.92	**0.61**
depot (22)	7	7	7	6	7	7	7	7	7	7	7	7	**4**	3	**4**	3	4.0	7.0	4.5	6	12	10	**0.34**	0.94	**0.52**	0.91	0.89	**0.68**
driverlog (18)	13	13	13	11	13	13	13	13	10	11	10	**12**	8	**10**	7	**10**	7.0	18.2	8.7	22	45	17	**0.19**	0.98	**0.58**	0.86	0.85	**0.50**
elevators (50)	40	40	40	35	40	40	40	40	**40**	37	38	37	**35**	26	31	26	3.9	4.9	3.2	8	13	8	**0.37**	0.94	**0.67**	0.89	0.92	**0.71**
floortile (36)	13	13	13	6	7	7	6	**8**	2	2	2	2	**2**	1	**2**	**2**	175.6	66.0	31.5	697	71	33	**0.12**	0.99	**0.67**	0.80	0.97	**0.28**
freecell (80)	15	15	15	13	15	15	15	15	15	15	15	15	**14**	13	13	13	4.0	4.7	3.4	4	6	5	**0.31**	0.94	**0.60**	0.94	0.88	**0.76**
ged (20)	15	15	15	11	15	15	15	15	**15**	10	10	10	10	**7**	10	**7**	9.2	38.7	12.5	18	101	38	**0.23**	0.90	**0.47**	0.80	**0.58**	0.70
grid (5)	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1.5	1.5	1.0	2	2	1	0.81	**0.69**	0.81	**0.69**	1.00	**0.56**
gripper (14)	7	7	7	5	**7**	5	5	5	4	4	4	4	**4**	3	**4**	3	458.3	87.0	39.5	1820	252	120	**0.21**	0.98	**0.65**	0.88	0.96	**0.46**
hiking (20)	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	1.4	1.4	1.0	2	2	1	0.89	**0.61**	0.89	**0.61**	1.00	**0.61**
logistics (60)	26	26	26	20	24	**26**	21	**26**	15	19	14	**20**	12	13	12	**15**	6.3	6.0	2.9	25	22	7	**0.31**	0.95	**0.68**	0.84	0.90	**0.63**
miconic (150)	141	120	76	50	**66**	**66**	55	64	**45**	40	44	43	**41**	36	40	36	76.0	24.1	8.4	363	98	36	**0.33**	0.91	**0.73**	0.82	0.95	**0.61**
movie (30)	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	7.0	35.0	21.0	7	35	21	**0.06**	0.99	**0.50**	0.94	0.94	**0.50**
mprime (35)	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	1.3	1.2	1.2	2	2	2	0.90	**0.59**	0.92	**0.59**	0.93	**0.59**
mystery (30)	17	17	17	17	17	17	17	17	17	17	17	17	15	**17**	15	**17**	1.4	1.4	1.2	2	2	2	0.88	**0.63**	0.88	**0.63**	0.92	**0.63**
nomystery (20)	14	14	14	13	14	14	14	14	10	**12**	10	**12**	8	8	8	8	7.3	18.5	5.8	18	47	13	**0.20**	0.96	**0.63**	0.92	0.87	**0.61**
openstacks (77)	47	45	45	43	**45**	**45**	37	43	**45**	43	29	41	**42**	**42**	22	33	15.3	14.2	12.3	25	25	23	**0.06**	0.99	**0.05**	0.99	**0.18**	0.97
org-syn (20)	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	5.1	5.1	5.1	12	12	12	**0.23**	0.94	**0.23**	0.94	**0.23**	0.94
org-syn-s (13)	10	10	10	9	**8**	**8**	7	**8**	**8**	**8**	7	**8**	**7**	6	6	6	5.2	7.2	8.3	12	28	36	**0.20**	0.95	**0.23**	0.95	**0.32**	0.89
parcprinter (26)	24	20	20	20	10	10	10	**14**	10	10	10	**14**	10	10	10	**12**	3.8	8.2	5.0	14	24	10	**0.44**	0.98	**0.61**	0.95	**0.72**	0.85
parking (40)	5	5	5	1	**5**	**5**	4	**5**	0	**1**	0	**1**	0	0	0	0	36.8	31.0	-	79	31	-	0.02	0.99	-	-	-	-
pathways (23)	5	5	5	5	5	5	5	5	4	**5**	4	**5**	4	4	4	4	3.2	3.8	1.8	6	10	3	**0.53**	0.81	0.77	0.77	0.91	**0.70**
pegsol (2)	2	2	2	2	0	0	**2**	**2**	0	0	**2**	**2**	0	0	**2**	**2**	7.0	23.5	64.0	8	41	122	-	-	-	-	-	-
pipesworld-nt (50)	17	17	17	17	17	17	17	17	**17**	**17**	16	**17**	**16**	14	**16**	14	3.7	6.4	3.8	8	31	17	**0.44**	0.89	**0.73**	0.84	0.88	**0.66**
pipesworld-t (50)	12	12	12	11	12	12	12	12	11	11	11	11	**9**	11	**9**	10	3.6	5.0	3.7	7	15	12	**0.43**	0.94	**0.67**	0.87	0.90	**0.63**
psr-small (50)	49	49	49	49	48	48	**49**	**49**	47	47	48	**49**	46	46	**48**	**48**	3.7	2.7	2.0	20	13	9	0.76	**0.63**	0.94	**0.55**	0.97	**0.47**
rovers (31)	8	8	8	7	8	8	8	8	7	7	7	7	**6**	5	**6**	4	18.0	11.4	3.8	95	35	12	**0.36**	0.93	**0.74**	0.84	0.91	**0.59**
satellite (19)	7	7	7	6	7	7	7	7	6	6	6	**7**	4	5	4	**6**	5.6	26.9	14.7	7	76	36	**0.19**	0.97	**0.49**	0.94	0.88	**0.73**
scanalyzer (40)	23	21	21	13	**9**	15	**9**	13	9	9	9	9	**9**	5	**9**	**9**	20.9	36.7	31.2	46	103	43	**0.25**	0.99	**0.53**	0.86	**0.75**	0.83
snake (17)	7	7	7	4	6	6	6	6	3	3	3	3	**3**	1	2	1	10.5	21.0	44.3	16	27	77	**0.13**	0.92	**0.32**	0.86	**0.58**	0.73
sokoban (50)	50	50	49	41	**50**	**50**	49	**50**	**46**	43	45	43	**40**	30	**40**	28	6.6	4.1	1.8	56	36	10	**0.60**	0.85	0.86	**0.71**	0.95	**0.51**
storage (30)	15	15	15	15	15	15	15	15	15	15	15	15	**15**	14	**15**	14	3.6	3.7	2.1	10	10	5	**0.62**	0.81	0.85	**0.75**	0.98	**0.57**
termes (20)	6	6	4	1	**6**	**6**	5	**6**	**5**	1	1	2	**1**	0	0	0	3.2	2.6	3.0	8	6	3	**0.37**	0.72	0.53	0.53	-	-
tetris (17)	6	6	6	5	**6**	**6**	5	**6**	**4**	3	3	3	**3**	2	**3**	2	29.7	32.8	7.3	81	82	11	**0.26**	0.98	0.81	**0.77**	0.97	**0.41**
tidybot (40)	23	23	23	19	23	23	23	23	**23**	22	**23**	22	13	13	7	**14**	3.1	3.3	3.4	4	6	6	**0.38**	0.92	**0.41**	0.92	**0.75**	0.84
tpp (30)	7	7	7	6	7	7	7	7	6	**7**	6	6	**6**	5	**6**	5	4.1	8.9	4.2	9	25	11	**0.43**	0.86	**0.67**	0.83	0.96	**0.66**
transport (70)	23	23	23	22	23	23	23	23	23	23	23	23	**23**	22	22	22	3.5	3.7	2.1	5	10	6	**0.43**	0.91	**0.59**	0.88	0.73	**0.69**
trucks (30)	10	10	10	8	**10**	**10**	9	**10**	6	**7**	6	**7**	**5**	3	**5**	4	15.7	17.1	5.0	36	31	8	**0.23**	0.97	**0.70**	0.89	0.93	**0.65**
visitall (14)	14	13	13	13	**13**	**13**	10	10	9	**10**	8	**10**	6	6	7	**8**	97.4	111.6	46.5	307	380	150	**0.20**	0.93	**0.41**	0.90	0.79	**0.75**
woodworking (35)	29	25	25	25	**23**	**23**	12	15	**9**	**9**	5	**9**	5	5	5	5	267.3	95.0	16.8	1030	192	26	**0.02**	0.99	**0.27**	0.93	0.72	**0.52**
zenotravel (20)	13	13	12	9	**13**	**13**	12	**13**	**9**	**9**	8	**9**	8	**9**	8	**9**	10.4	4.0	2.6	36	6	4	**0.36**	0.94	**0.67**	0.89	0.87	**0.66**
Sum (1583)	862	828	776	670	733	**740**	688	732	630	624	592	**636**	**556**	517	523	528												

Figure 1: Results on IPC benchmarks modified for oversubscription planning. Reference Points: related classical planning tasks (see text). Coverage: of our MUGS algorithms SysS and SysW, with vs. without conjunction learning h^C. #MUGS: average/maximum number of MUGS, indicating explanation size (see text). Search Tree Fraction: fraction of worst-case search tree explored. Best performance in each part shown in **boldface**. Cost bounds set to x times optimal cost.

through the smallest conjunctions excluded by p. The number of such conjunctions is at most the number of MUGS. So #MUGS corresponds to worst-case answer/explanation size. As the data shows, that size is often small, of a scale that seems feasible for human inspection.

Action-Set Properties

To evaluate the use of our framework with more complex plan properties, beyond goal facts, we experimented with the compilation of action-set properties as per Section . We selected four IPC domains for extension with action-set properties, namely NoMystery, Rovers, and TPP as considered in resource-constrained planning (Nakhost *et al.* 2012), where minimum resource requirements are known as per available problem generators; plus the Blocksworld as an intuitively rather differently structured domain. In all four domains, we use discrete resource consumption encoded into the STRIPS model, enabling the use of trap learning (Steinmetz and Hoffmann 2017a) which turns out to be highly beneficial here.

In Blocksworld, we include two gripper hands and the action-set properties ask whether a given gripper is used to pick up a given block, or to stack a given pair of blocks. In NoMystery, the properties are as in our illustrative example (Section). In Rovers, the properties ask whether a given rover or camera is used for a given observation. In TPP, they ask whether given road segments are used, and whether given goods are bought at given markets. In all cases, we vary the number of action-set properties between 1 and 10. We fix the original goal facts as hard goals, and we set the available resources to $x \in \{1.0, 1.5, 2.0\}$ times the minimum needed to allow for costlier plans satisfying some of the properties.

We created benchmark tasks with size parameters around the borderline of computational feasibility for our analyses, given our time/memory limits. In Blocksworld, we used 5 – 8 blocks; in NoMystery, our tasks have 2 trucks, 6 locations, and 4 – 7 packages; in Rovers, they have 2 rovers, 5 waypoints, and 4 – 7 science objectives; in TPP, we use 5 markets, 1 depot, and 4 – 7 goods. In all domains, we vary the number of goal facts (and associated objects) between 4 and 7. We create 10 base instances for each size-parameter setting, which are then modified for our experiments with different initial resource levels, and action-set properties to be considered.

To have some comparison measure for performance, again we use classical-planning reference points, based on A* with $h^{\text{LM-cut}}$, and on search with trap learning, respectively. We now run these reference points on tasks where all (original goal facts plus) action-set properties are hard goals. These tasks may be solvable (in which case A* with $h^{\text{LM-cut}}$ tends to be better) or unsolvable (in which case trap learning

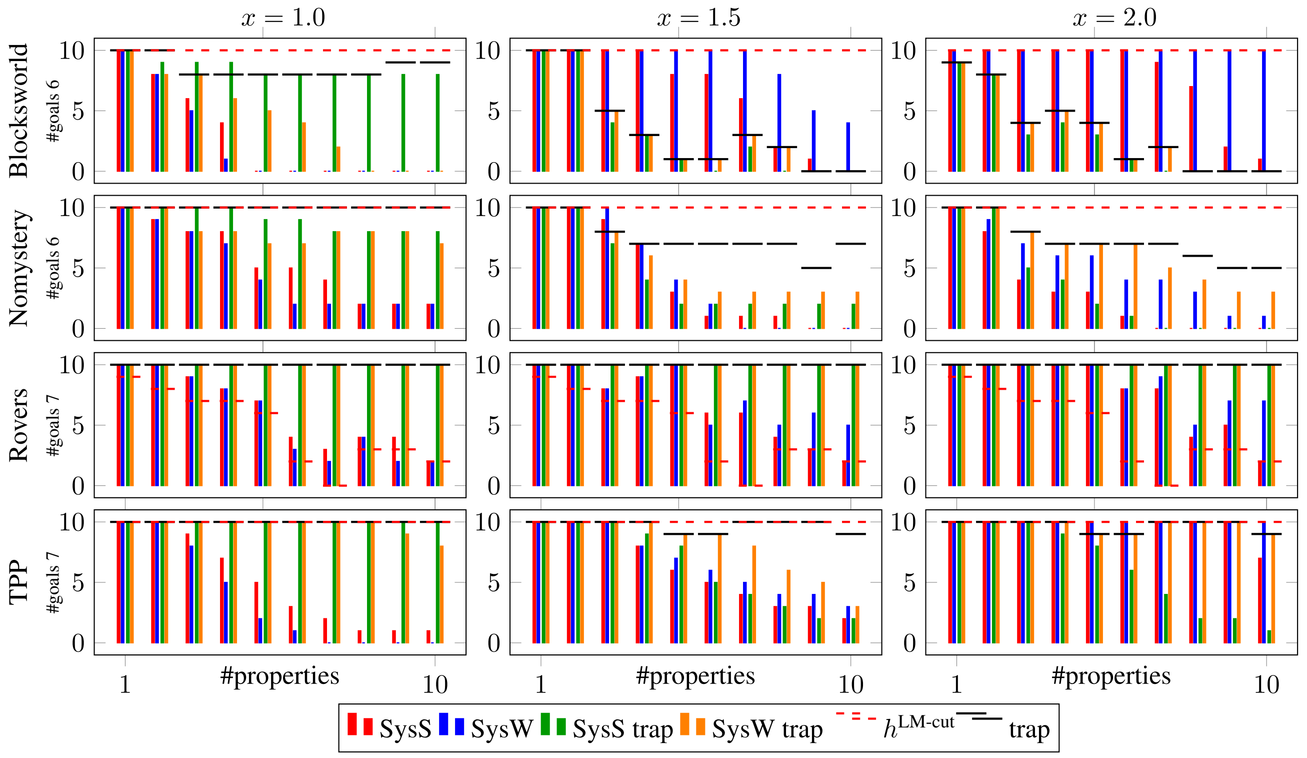


Figure 2: Coverage results on IPC benchmarks extended with action-set properties.

tends to be better). The configurations of our own algorithm are SysS and SysW as before, now with vs. without trap learning (and transfer).

Figure 2 shows the coverage data. For space reasons, we show only one row per domain, fixing the number of hard goals at the feasibility borderline. Smaller numbers of goal facts tend to be quite easy, larger ones mostly infeasible, with variance depending on the domain and algorithm.

Conclusion

We introduced a framework for plan-space explanation via plan-property dependencies. We believe that the framework is useful conceptually as a problem formulation shaping a relevant part of XAIP. Our techniques for first instantiations of the framework exhibit reasonable performance in IPC benchmark studies. The computed explanations are often small and thus potentially feasible for human inspection.

In future work, the effectiveness of these explanations for human users remains to be evaluated in user studies. Another important question is how to address deeper "why" questions, asking for the reasons behind an entailment $\Pi \models p \Rightarrow q$. Possible ideas are to include additional properties into P, elucidating the causal chain between p and q; or to find a minimal relaxation (superset) of the plan set Π for which p no longer entails q, thus elucidating the circumstances under which that entailment holds. Last but not least, of course our framework and algorithms can and should be extended to richer planning frameworks and plan property languages.

Acknowledgments

This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-18-1-0245. Jörg Hoffmann's group has also received support by the German Research Foundation (DFG) as part of CRC 248 (see perspicuous-computing.science). Part of this work was performed while Jörg Hoffmann was visiting NASA Ames Research Center. We thank J. Benton, Minh Do, Jeremy Frank, and David Smith for insightful discussions.

References

Christer Bäckström and Bernhard Nebel. Complexity results for SAS$^+$ planning. *Computational Intelligence*, 11(4):625–655, 1995.

Jorge A. Baier, Fahiem Bacchus, and Sheila A. McIlraith. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence*, 173(5-6):593–618, 2009.

T. Chakraborti, S. Sreedharan, Y. Zhang, and S. Kambhampati. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In Carles Sierra, editor, *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*. AAAI Press/IJCAI, 2017.

Tathagata Chakraborti, Anagha Kulkarni, Sarath Sreedharan, David E. Smith, and Subbarao Kambhampati. Explicability? legibility? predictability? transparency? privacy? security? the emerging landscape of interpretable agent behavior. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS'19)*. AAAI Press, 2019.

Carmel Domshlak and Vitaly Mirkis. Deterministic oversubscription planning as heuristic search: Abstractions and reformulations. *Journal of Artificial Intelligence Research*, 52:97–169, 2015.

Stefan Edelkamp. On the compilation of plan constraints and preferences. In Derek Long and Stephen Smith, editors, *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS'06)*, pages 374–377, Ambleside, UK, 2006. Morgan Kaufmann.

Maria Fox and Derek Long. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9:367–421, 1998.

Maria Fox, Derek Long, and Daniele Magazzeni. Explainable planning. In *Proc. IJCAI'17 Workshop on Explainable AI*, 2017.

B. Cenk Gazen and Craig Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In S. Steel and R. Alami, editors, *Proceedings of the 4th European Conference on Planning (ECP'97)*, pages 221–233. Springer-Verlag, 1997.

Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pages 162–169. AAAI Press, 2009.

Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

Malte Helmert. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173:503–535, 2009.

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

Pat Langley, Ben Meadows, Mohan Sridharan, and Dongkyu Choi. Explainable agency for intelligent autonomous systems. In Satinder Singh and Shaul Markovitch, editors, *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*. AAAI Press, February 2017.

Tim Miller. Contrastive explanation: A structural-model approach. *CoRR*, abs/1811.03163, 2018.

Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, 2019.

Hootan Nakhost, Jörg Hoffmann, and Martin Müller. Resource-constrained planning: A Monte Carlo random walk approach. In Blai Bonet, Lee McCluskey, José Reinaldo Silva, and Brian Williams, editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, pages 181–189. AAAI Press, 2012.

Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315, 2000.

Hector Palacios and Hector Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research*, 35:623–675, 2009.

Bastian Seegebarth, Felix Müller, Bernd Schattenberg, and Susanne Biundo. Making hybrid plans more clear to human users - A formal approach for generating sound explanations. In Blai Bonet, Lee McCluskey, José Reinaldo Silva, and Brian Williams, editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*. AAAI Press, 2012.

David E. Smith. Choosing objectives in over-subscription planning. In Sven Koenig, Shlomo Zilberstein, and Jana Koehler, editors, *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, pages 393–401, Whistler, Canada, 2004. Morgan Kaufmann.

David Smith. Planning as an iterative process. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, pages 2180–2185, Toronto, ON, Canada, July 2012. AAAI Press.

Marcel Steinmetz and Jörg Hoffmann. Search and learn: On dead-end detectors, the traps they set, and trap learning. In Carles Sierra, editor, *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*. AAAI Press/IJCAI, 2017.

Marcel Steinmetz and Jörg Hoffmann. State space search nogood learning: Online refinement of critical-path dead-end detectors in planning. *Artificial Intelligence*, 245:1 – 37, 2017.

Tiago Stegun Vaquero, José Reinaldo Silva, Flavio Tonidandel, and J. Christopher Beck. itsimple: towards an integrated design system for real planning applications. *Knowledge Engineering Review*, 28(2):215–230, 2013.

Peng Yu, Brian Charles Williams, Cheng Fang, Jing Cui, and Patrik Haslum. Resolving over-constrained temporal problems with uncertainty through conflict-directed relaxation. *Journal of Artificial Intelligence Research*, 60:425–490, 2017.

Design for Interpretability

Anagha Kulkarni[1*] · **Sarath Sreedharan**[1*] · **Sarah Keren**[2] · **Tathagata Chakraborti**[3]
David E. Smith · **Subbarao Kambhampati**[1]

[1]Arizona State University
[2]Harvard University
[3]IBM Research AI

anaghak@asu.edu, ssreedh3@asu.edu, skeren@seas.harvard.edu, tchakra2@ibm.com, david.smith@psresearch.xyz, rao@asu.edu

Abstract

The interpretability of an AI agent's behavior is of utmost importance for effective human-AI interaction. To this end, there has been increasing interest in characterizing and generating interpretable behavior of the agent. An alternative approach to guarantee that the agent generates interpretable behavior would be to design the agent's environment such that uninterpretable behaviors are either prohibitively expensive or unavailable to the agent. To date, there has been work under the umbrella of *goal or plan recognition design* exploring this notion of environment redesign in some specific instances of interpretable of behavior. In this position paper, we scope the landscape of interpretable behavior and environment redesign in all its different flavors. Specifically, we focus on three types of interpretable behaviors – explicability, legibility and predictability – and present a general framework for environment design that can be instantiated to achieve these behaviors. We also discuss how specific instantiations of this framework correspond to prior works on environment design, and identify exciting opportunities for future work.

1 Introduction

The design of human-aware AI agents must ensure that its decisions are interpretable to the human in the loop. Uninterpretable behavior can lead to increased cognitive load on the human – from reduced trust, productivity to increased risk of danger around the agent (Fan et al. 2008). Christensen et al. (2009) emphasises in the *Roadmap for U.S. Robotics – "humans must be able to read and recognize agent activities in order to interpret the agent's understanding"*. The agent's behavior may be uninterpretable if the human: (1) has incorrect notion of the agent's beliefs and capabilities (Zhang et al. 2017; Chakraborti et al. 2017b; Kulkarni et al. 2019) (2) is unaware of the agent's goals and rewards (Dragan and Srinivasa 2013; Kulkarni, Srivastava, and Kambhampati 2019) (3) cannot predict the agent's plan or policy (Fisac et al. 2018; Kulkarni, Srivastava, and Kambhampati 2019). Thus, in order to be interpretable, the agent must take into account the human's expectations of its behavior – i.e. *the human mental model* (Chakraborti et al. 2017a). There are many ways in which considerations of the human mental model can affect agent behavior.

[*]equal contribution

1.1 The Landscape of Interpretable Behavior

There has been significant interest recently in characterizing different notions of interpretable behavior of a human-aware AI agent (Chakraborti et al. 2019). Three important properties of interpretable behaviors emerge, namely – (1) *explicability*: when the agent behavior conforms to the expectations of the human; (2) *legibility*: when the agent behavior reveals its objectives or intentions to the observer; and (3) *predictability*: when the (remaining) agent behavior can be precisely predicted by the human.

In existing works, the generation of these behaviors was explored from the point of view of the agent – i.e. the agent altered its planning process by using its estimation of the mental model of the human in the loop in order to exhibit the desired behavior. We refer the reader to (Chakraborti et al. 2019) for a detailed treatise of these concepts.

1.2 Environment Design

A parallel thread of work, under the umbrella of *goal and plan recognition design*, has looked at the notion of changing the environment of an agent to increase interpretability of the behaviors available to an agent. The design of environment can be optimized in order to maximize (or minimize) some objective for the actor (for example, optimal-cost to a goal, desired behavioral property) (Zhang, Chen, and Parkes 2009; Keren et al. 2017). An environment design problem takes the initial environment configuration as input, along with set of modifications allowed in the environment and outputs a sequence of modifications, which can be applied to the initial environment to derive a new environment in which the desired objective is optimized. The problem of environment design is more suited for structured settings where an actor performs repetitive objectives (for example, on factory floors, etc). It is also suited for settings involving multiple actors, where the expectations of the observers are the same but there are multiple actors in the environment, making environment design an effective choice (for example, in restaurants with waiter robots where the human customers have same expectations).

Goal (and Plan) Recognition Design In existing work, the concept of environment design for planning agents has been studied in the concept of goal (or plan) recognition

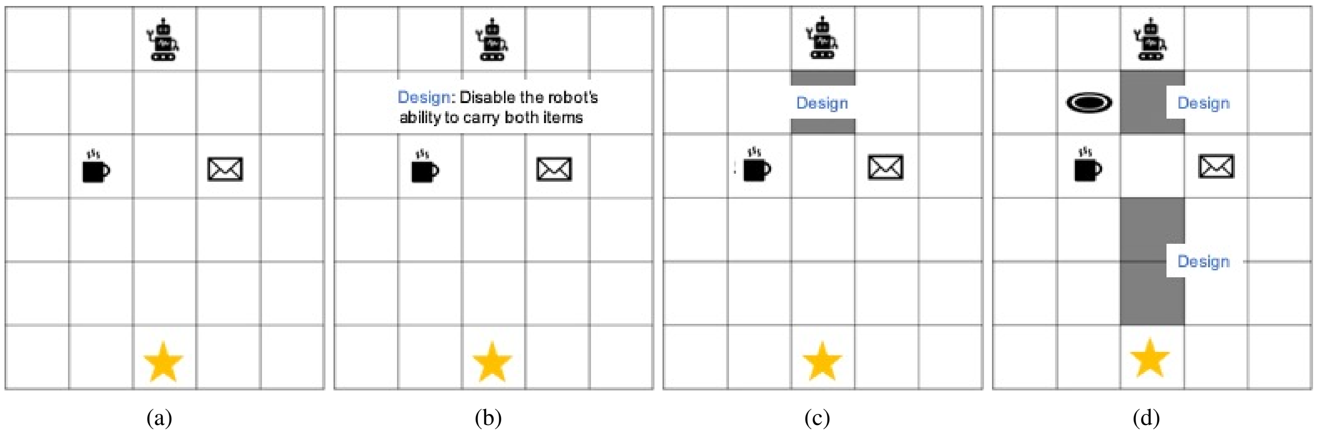


Figure 1: The office assistant domain: (a) The original domain; (b) The domain can be updated for more explicable behavior by disabling the robot coffee holder; (c) To induce legible behavior, we can add dividing walls to constrain the agent and help the observer reduce uncertainty in their mental model; and (d) To induce predictable behavior we can reduce uncertainty about the order of pickup by including a tray that allows the agent to pick up the objects in any order.

design (Keren, Gal, and Karpas 2014; Mirsky et al. 2019) in order to make the goals (or plans) of an actor easier to recognize to the observer. Immediately, this should remind the reader of *legibility* (and *predictability*) introduced above. The goal of this paper is to bridge the gap between these two parallel threads of work and explore the full spectrum of interpretable behavior in the context of environment design. We will discuss how existing work in environment design fits into this narrative and highlight much needed gaps in existing work as exciting avenues for future work.

1.3 Why Design for Interpretability

Adopting the reasoning capabilities of an agent to deal with the human mental model, as done in classical human-aware decision-making versus the design of environments are indeed complimentary approaches towards achieving the same purpose: behavior of the agent that is more interpretable to the observer. While one could conceive of the most general framework that accounts for both, it is useful to recognize that these have their own unique set of features. Perhaps, the biggest advantage of environment design is that the process of generation of interpretable behavior is offloaded from the actor onto the design process. In other words, the computational overhead of generating interpretable behavior – in having to deal with the human mental model and reason in the space of models – is now part of the design process only, which can be done offline.

We will see later, in our formulation, how the actor in the "Design for Interpretability" is modeled as a cost-optimal agent that is able to produce interpretable behavior while still planning in the traditional sense. In addition, design also means that the observer does not have to count on the agent to be cooperative and interpretable, and instead can deal with adversarial agents as well. At the end of this paper, we will see that this advantage does come with a caveat.

In general, the notion of interpretable behavior is complemented by communication: e.g. authors in (Chakraborti, Sreedharan, and Kambhampati 2019) balance considerations of explanations and explicability, while authors in (Chakraborti et al. 2018) balance out intention projection actions for the sake of legibility and predictability. Communication operates in model space by being able to change the observer's beliefs. Design, also operating in model space, can be seen as an alternative to communication that complements the notion of interpretable behavior.

1.4 An Illustration of Design for Interpretability

Consider an office setting where an office assistant robot, responsible for delivering items such as coffee or mail to the employees, is about to be deployed (Figure 1a). The robot (*actor*) will be supervised by office security guards (*observer*) who have worked with previous generation office assistant robots and have some expectations regarding their functions. In particular, they expect the robot to carry one item at a time (i.e. either mail or coffee) and each robot generally has a strong preference on the order in which it picks up these items (though the order changes from robot to robot). Unknown to the guards, the new model adds more flexibility to the robot by (1) removing the need for the robots to have fixed preference on the order to pick up items and (2) installs a coffee cup holder that allows the robot to carry both mail and coffee at the same time. Now if we allow the new robot to simply act optimally in the original setting, it would unnecessary confuse the observers.

If the robot was built to generate interpretable behavior, it will change its behavior (and possibly settle for suboptimal decisions in its own model) in order to conform to expectations or it will provide explanations that address these model differences. However, the same effect can be achieved if the designers who are *deploying* the robot also designed the environment to ensure that decisions of the new robot remain interpretable to the occupants of the office.

If the designers wish to prioritize explicability, then the change that they would need to make would be to disable the coffee holder, this will cause the robot to choose one of the items first, deliver it and then move on to the second one. For explicability, it does not matter which one the robot chooses as the user would simply assume that the order chosen by the robot is the one enforced by the robot's model. As for legibility, the aim is to help the user differentiate between the models as early as possible, one way to do it would be to disable the coffee holder and then build introduce obstacles as shown in Figure 1c. Finally, for predictability, the focus is to allow the user to be able to predict the entire plan as early as possible. One possible design for this scenario is to disable the coffee holder and provide the robot with a tray that allows the robot to carry both items at the same time. The observer can see the tray and realizes the robot can place both items in the tray and the order of picking up no longer matters. In predictability, we may need to add additional obstacles to further restrict the space of possible plans that can be done by the robot (Figure 1d).

2 Background

An interpretable decision making problem involves two entities: an actor (A) and an observer (O). The actor operates in an environment while being observed by the observer.

Definition 1. *An* ***interpretable decision making problem*** *is a tuple, $\mathcal{P}_{Int} = \langle P_A, \mathbf{P_O}, Int\rangle$, where:*

- *P_A is the decision making problem of the actor A*
- *$\mathbf{P_O} = \{P_O^i\}$ is observer's mental model of the actor, represented by a set of possible decision making problems that the observer believes that the actor may be solving.*
- *$Int : \Pi \to \mathbb{R}$ is the interpretability score that is used to evaluate agent plans (where Π is the space of plans)*

Interestingly, we do not require that $P_A \in \mathbf{P_O}$ – i.e. the problems in $\mathbf{P_O}$ can be different from P_A in all possible aspects (e.g. state space, action space, initial state and goals). The solution to $\mathcal{P}_{Int}$ is a plan or policy that not only solves P_A but also satisfies some desired properties of interpretable behaviors (measured through the interpretability score). The score could reflect properties like explicability, legibility or predictability of the plan.

Explicability The actor's behavior is considered explicable if it aligns with at least one of the observer's expected plans (as per their mental model). The set of plans expected by the observer consists of all the cost-optimal solutions for problems in $\mathbf{P_O}$. The target of explicability is thus to generate behavior that belongs to this set of expected plans.

Legibility With legibility, the objective of the actor is to inform the observer about its model – i.e. reduce the size of $\mathbf{P_O}$. An actor's behavior is said to be perfectly legible if it can be derived from only one model in $\mathbf{P_O}$. The longer it takes for a plan prefix to achieve this, the worse is the plan's legibility. This notion of interpretability thus helps the observer narrow down their belief over the possible actor models as quickly as possible.

Predictability The objective of the actor with predictability is to generate the most disambiguating behavior – i.e. given the actor's plan prefix, the observer should be able to predict its completion. These predictions would be in terms of cost-optimal completions of a given prefix in the possible problems in the mental model. This means that if there exists the same unique completion in all of the models then *the plan is predictable even though not legible*. The shorter the length of the disambiguating plan prefix, the better the predictability of the plan. An empty prefix would thus correspond to the most predictable plan.

3 Design for Interpretability

In this section, we present a general formulation for the design problem for interpretable behaviors. Given an environment design, we assume that the actor is a rational agent and therefore is incentivized to generate cost-optimal plans. Let the set of cost optimal plans of the actor be $\Pi^*_{P_A}$. A cost-optimal plan solution to P_A can exist anywhere on the spectrum of interpretability from high to low. Therefore, we need a measure to quantify the interpretability score for the actor's set of cost-optimal plans. To that end, we introduce the worst-case interpretability score wci as follows:

Definition 2. *The* ***worst-case interpretability score*** *$wci(\cdot)$, for $\mathcal{P}_{Int}$ is defined as*

$$wci(\mathcal{P}_{Int}) = \min_{\pi \in \Pi^*_{P_A}} \quad Int(\pi) \tag{1}$$

$Int(\cdot)$ is instantiated for each type of interpretable behavior separately and is discussed in detail at the end of this section. The higher the interpretablity score, the better the interpretability of the behavior (in terms of either of three properties). Therefore, the worst-case interpretability score is the minimum interpretability score of a cost-optimal plan of the actor.

We can now define the design problem for interpretability. When a modification is applied to the environment, both the actor's decision making problem and the observer's mental model are modified, thereby changing the worst-case interpretability score of the actor's cost-optimal plans for the given decision making problem. Let $\mathcal{P}$ denote the set of valid configurations in the real environment. Although $P_A \in \mathcal{P}$, problems in $\mathbf{P_O}$ might not necessarily be in $\mathcal{P}$ if the observer has incorrect or infeasible notions about the actor's model. Therefore, we represent the set of configurations that the observer thinks are possible as $\widetilde{\mathcal{P}}$, and $\mathbf{P_O} \subseteq \widetilde{\mathcal{P}}$.

Definition 3. *The* ***design problem for interpretability****, DP-Int, is a tuple $\langle \mathcal{P}^0_{Int}, \Delta, \Lambda_A, \Lambda_O\rangle$ where,*

- *$\mathcal{P}^0_{Int} = \langle P^0_A, \mathbf{P_O}^0, Int\rangle$ where $P^0_A \in \mathcal{P}$ and $\mathbf{P_O}^0 \subseteq \widetilde{\mathcal{P}}$ are the initial models.*
- *Δ is the set of modifications that can be applied to the environment. ξ is a sequence of modifications.*
- *$\Lambda_A : \Delta \times \mathcal{P} \to \mathcal{P}$ and $\Lambda_O : \Delta \times \widetilde{\mathcal{P}} \to \widetilde{\mathcal{P}}$ are the model transition function that specify the resulting model after applying a modification to the existing models.*

The set of possible modifications includes modifying the set of states, action preconditions, action effects, action costs, initial state and goal. Each modification $\xi \in \Delta$ is associated with a cost, such that, $C(\xi) = \sum_{\xi_i \in \xi} C(\xi)$. After applying ξ to both P_A^0 and $\mathbf{P_O}^0$, the resulting actor decision making problem model and observer mental model are represented as $P_A^{|\xi|}$ and $\mathbf{P_O}^{|\xi|}$ respectively.

Let $\mathcal{P}_{Int}^{|\xi|}$ be the modified interpretable decision making problem after applying the modification ξ to $\mathcal{P}_{Int}$. Our objective here is to solve ***DP-Int*** such that the worst-case interpretability score of $\mathcal{P}_{Int}$ is maximized. Apart from that, the design cost of ξ has to be minimized, as well as the cost of a plan π_A that solves $P_A^{|\xi|}$.

Definition 4. *A **solution to** **DP-Int**, is a sequence of modifications ξ with*

$$\min(-wci(\mathcal{P}_{Int}^{|\xi|}), C(\xi), cost(\pi_A)) \quad (2)$$

This completes the general framework of design for interpretability. In the following, we will look at specific instances of design for the different notions of interpretability.

3.1 Design for Explicability

In order to be explicable, the actor's plan has to be consistent with the observer's expectations of it. The observer has an implicit assumption that the actor is a rational agent. Therefore the set of plans expected by the observer includes the cost-optimal plans for all the planning models in the observer's mental model. Let $\Pi^*_{\mathbf{P_O}}$ be the set of expected plans for the observer. Given the set of expected plans, the explicability of the actor's plan depends on how different it is from the expected plans. In order to quantify the explicability of a plan, we introduce the following scoring function:

Definition 5. *The **explicability score** $Exp(\cdot)$ of an actor's plan π_A that solves P_A is defined as follows:*

$$Exp(\pi_A) = \max_{\pi \in \Pi^*_{\mathbf{P_O}}} e^{-\delta_{\mathbf{P_O}}(\pi_A, \pi)} \quad (3)$$

Here $\delta_{\mathbf{P_O}}(\cdot)$ computes the distance between two plans with respect to the observer's mental model. For example, the distance function could compute a cost-based difference between the two plans in the observer's mental model. Plugging this scoring function in Equation 2 allows us to instantiate the design problem for explicability.

3.2 Design for Legibility

In order to be legible, the actor's plan has to reveal its problem to the observer as early on as possible. Therefore, the legibility of a plan is inversely proportional to the length of its shortest prefix that has unique cost optimal completion for more than one problem in the observer's mental model.

Definition 6. *The **legibility score** $Leg(\cdot)$ of an actor's plan, π_A, that solves P_A is defined as follows:*

$$Leg(\pi_A) = \min_{\tilde{\pi}_A \in \tilde{\Pi}_{P_A}} e^{-|\tilde{\pi}_A|} \quad (4)$$

such that $\exists (P_O^i, P_O^j) \in \mathbf{P_O}, i \neq j$ with unique cost optimal completion of $\tilde{\pi}_A$ in each model, and $\tilde{\Pi}_{P_A}$ is the set of all prefixes of π_A. Plugging this scoring function in Equation 2 allows us to instantiate the design problem for legibility.

Goal Recognition Design The work on goal recognition design (GRD) (Keren, Gal, and Karpas 2014) is a special case of the design problem for legibility. The GRD problem involves an actor and an observer where the observer's mental model consists of planning models that have the exact same state space, actions and initial state as the actor's planning model. However, each planning model in the observer's mental model has a different goal. The actor's true goal is one of them, and the objective of GRD problem is to redesign the environment, such that, the true goal of the actor is revealed to the observer as early as possible. The interpretability problem defined here is a general one, where the observer's mental model can be different in all possible ways from the actor's actual planning model.

3.3 Design for Predictability

In order to be predictable, the plan has to be the most-disambiguating plan among the set of plans the observer is considering – i.e. the observer should be able to predict the rest of the plan after seeing the prefix. Therefore, predictability of a plan is inversely proportional to the length of its shortest prefix which ensures only one optimal completion solving only a single problem in the observer's mental model. We can quantify the predictability score as follows:

Definition 7. *The **predictability score** $Pred(\cdot)$ of an actor's plan π_A that solves P_A is defined as follows:*

$$Pred(\pi_A) = \min_{\tilde{\pi}_A \in \tilde{\Pi}_{P_A}} e^{-|\tilde{\pi}_A|} \quad (5)$$

such that $\exists! \pi \; \exists P_O^i \in \mathbf{P_O}$ where π is an optimal completion of $\tilde{\pi}_A$, and $\tilde{\Pi}_{P_A}$ is the set of all prefixes of a plan π_A. Plugging this scoring function in Equation 2 allows us to instantiate the design problem for predictability.

Connection to Plan Recognition Design The predictability problem corresponds to the plan recognition design (PRD) problem (Mirsky et al. 2019). However, our proposed framework in terms of possible observer models subsumes the plan library based approaches in being able to support a generative model of observer expectations.

4 Discussion and Future Work

We will now highlight limitations of the proposed framework and discuss how they may be extended in the future.

Multiple decision making problems. The problem of environment design, as studied in this paper, is suitable for settings where the actor performs a single repetitive task. However, our formulation can be easily extended to handle an array of tasks that the agent performs in its environment by considering a *set* of decision making problems for the actor (Sreedharan, Chakraborti, and Kambhampati 2018), where the worst-case score is decided by taking either minimum (or average) over the $wci(\cdot)$ for the set of problems.

Interpretability Score. The three properties of interpretable agent behavior are not mutually exclusive. A plan can be explicable, legible and predictable at the same time. In general, a plan can have any combination of the three properties. In Equation 2, $Int(\cdot)$ uses one of these properties at a time. In order to handle more than one property at a time, one could formulate $Int(\cdot)$ as a linear combination of the three properties. In general, the design objective would be to minimize the worst-case interpretability score such that the scores for each property are maximized in the modified environment, or at least allow the designer pathways to trade off among potentially competing metrics.

Cost of the agent. In Section 1.3 we mentioned an advantage of the design process in the context of interpretability – the ability to offload the computational load on the actor, in having to reason about the observer model, to the offline design stage. However, there is never any free lunch. The effect of environment design is more permanent than operating on the human mental model. That is to say, interpretable behavior while targeted for a particular human in the loop or for a particular interaction, does not (usually) affect the actor going forward. However, in case of design of environment, the actor has to live with the design decisions for the rest of its life. That means, for example, if the environment has been designed to promote explicable behavior, the actor would be incurring additional cost for its behaviors (than it would have had in the original environment). This also affects not only a particular decision making problem at hand, but also everything that the actor does in the environment, and for all the agents it interacts with. As such there is a "loss of autonomy" is some sense due to environment design, the cost of which can and should be incorporated in the design process.

Acknowledgments

This research is supported in part by the ONR grants N00014-16-1-2892, N00014-18-1-2442, N00014-18-1-2840, the AFOSR grant FA9550-18-1-0067, NASA grant NNX17AD06G and JP Morgan faculty research grant.

References

Chakraborti, T.; Kambhampati, S.; Scheutz, M.; and Zhang, Y. 2017a. AI Challenges in Human-Robot Cognitive Teaming. *arXiv:1707.04775*.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017b. Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy. In *IJCAI*.

Chakraborti, T.; Sreedharan, S.; Kulkarni, A.; and Kambhampati, S. 2018. Projection-Aware Task Planning and Execution for Human-in-the-Loop Operation of Robots. In *IROS*.

Chakraborti, T.; Kulkarni, A.; Sreedharan, S.; Smith, D.; and Kambhampati, S. 2019. Explicability? Legibility? Predictability? Transparency? Privacy? Security?: The Emerging Landscape of Interpretable Agent Behavior. In *ICAPS*.

Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2019. Balancing Explicability and Explanation in Human-Aware Planning. In *IJCAI*.

Christensen, H. I.; Batzinger, T.; Bekris, K.; Bohringer, K.; Bordogna, J.; Bradski, G.; Brock, O.; Burnstein, J.; Fuhlbrigge, T.; Eastman, R.; et al. 2009. A Roadmap for US Robotics: From Internet to Robotics. *Technical Report*.

Dragan, A., and Srinivasa, S. 2013. Generating Legible Motion. In *RSS*.

Fan, X.; Oh, S.; McNeese, M.; Yen, J.; Cuevas, H.; Strater, L.; and Endsley, M. R. 2008. The influence of agent reliability on trust in human-agent collaboration. In *ECCE*.

Fisac, J. F.; Liu, C.; Hamrick, J. B.; Sastry, S. S.; Hedrick, J. K.; Griffiths, T. L.; and Dragan, A. D. 2018. Generating Plans that Predict Themselves. In *WAFR*.

Keren, S.; Pineda, L.; Gal, A.; Karpas, E.; and Zilberstein, S. 2017. Equi-reward utility maximizing design in stochastic environments. In *IJCAI*.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal Recognition Design. In *ICAPS*.

Kulkarni, A.; Chakraborti, T.; Zha, Y.; Vadlamudi, S. G.; Zhang, Y.; and Kambhampati, S. 2019. Explicable Robot Planning as Minimizing Distance from Expected Behavior. In *AAMAS*. Extended Abstract.

Kulkarni, A.; Srivastava, S.; and Kambhampati, S. 2019. A Unified Framework for Planning in Adversarial and Cooperative Environments. In *AAAI*.

Mirsky, R.; Gal, K.; Stern, R.; and Kalech, M. 2019. Goal and plan recognition design for plan libraries. *TIST*.

Sreedharan, S.; Chakraborti, T.; and Kambhampati, S. 2018. Handling Model Uncertainty and Multiplicity in Explanations as Model Reconciliation. In *ICAPS*.

Zhang, Y.; Sreedharan, S.; Kulkarni, A.; Chakraborti, T.; Zhuo, H. H.; and Kambhampati, S. 2017. Plan Explicability and Predictability for Robot Task Planning. In *ICRA*.

Zhang, H.; Chen, Y.; and Parkes, D. C. 2009. A general approach to environment design with one agent. In *IJCAI*.

When Agents Talk Back: Rebellious Explanations

Ben Wright[1] and **Mark Roberts**[2] and **David W. Aha**[2] and **Ben Brumback**[2]
[1]NRC Postdoctoral Fellow, Navy Center for Applied Research and Artificial Intelligence
[2]Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory
{benjamin.wright.ctr, mark.roberts, david.aha, benjamin.brumback}@nrl.navy.mil

Abstract

As the area of Explainable AI (XAI), and Explainable AI Planning (XAIP), matures, the ability for agents to generate and curate explanations will likewise grow. We propose a new challenge area in the form of rebellious and deceptive explanations. We discuss how these explanations might be generated and then briefly discuss evaluation criteria.

1 Introduction

Explanations as a research area in AI (XAI) has been around for several decades (Clancey 1986; Buchanan and Shortliffe 1984; Craven 1996; Wick and Thompson 1992; Doyle et al. 2003; Sørmo et al. 2005; Weston et al. 2015; Miller 2018). It has additionally gained momentum recently as evidenced by the increasing number of workshops and special tracks covering it in various conferences (e.g., VIS-xAI, FEAP-AI4Fin, XAIP, XAI, OXAI, MAKE-eXAI, ICCBR-19 Focus area).

While still growing in use, there have been some approaches to formalizing XAI. DARPA (2016) stated that anything calling itself XAI should address the following questions:

- Why did the agent do that and not something else?
- When does the agent succeed and when does it fail?
- When can I trust the agent?

However, less thought out is the idea of explanations that are *deceptive* or *rebellious* in nature. These forms of explanation can be an entirely new area of discussion and use for certain autonomous agents.

The study of deception and rebellion are both rich fields, and many aspects of both that are studied in civilian and military capacities. For example, the area of deception detection works on finding ways to detect inconsistencies (Thomas and Biros 2011; Kott et al. 2011; Biros et al. 2005). Isaac and Bridewell (2017) discuss a number of ways why deception is an important topic for autonomous agents.

Studies of rebellion and resistance have investigated how, why, when it does, and doesn't, happen (Martí and Fernández 2013; Pershing 2003b). The use of both has also been studied (Reed 2016; Anderson et al. 2004; Kott and Ownby 2005; Keller et al. 2015; Mourougayane and Srikanth 2015).

The idea of pairing deception and rebellion with explanations may not be intuitive initially. However, in addition to being areas of rich study, deception and rebellion offer key conditions that are of interest to agent reasoning. Predominately, it requires multiple actors (i.e., An actor *deceives* another actor, or an actor *rebels* against a coordinator). Additionally, there needs to be some sort of conflict or misalignment between the actors. Either something needs to be in contention for an actor to rebel, or something needs to be in conflict for the actor to use deception. b Rebellion in agents has been a growing area of interest (Aha and Coman 2017; Coman and Aha 2018; Dannenhauer et al. 2018; Boggs et al. 2018; Coman and Aha 2017). This area is focused on finding models in which agents can *rebel* from directives given in certain circumstances. This can include having more up-to-date knowledge that would affect the plan, finding opportunities to exploit but may be off-mission, or solving problems or roadblocks before they become an issue even if it is off-mission. (Aha and Coman 2017) discuss three ways in which rebellion could manifest in agents. The *expression* of a rebellion can consist of either an explicit or implicit act. The *focus* is either inward or outward facing. Lastly, the *interaction initiation* can either be reactive or proactive.

Deception in agents has been progressing over the last decade, with many discussions on formalizing deception. The majority of this formalism is on the topic of lying (Van Ditmarsch et al. 2012; Sakama et al. 2011; Van Ditmarsch 2014). There has also been inroads for more encompassing deception as described by (Sakama 2015) and (Sakama and Caminada 2010). Of interest here, (Sakama, Caminada, and Herzig 2014) defined *Quantitative & Qualitative Maxims for Dishonesty* as the following maxims:

1. Lie, Bullshit (BS), or withhold information as little as possible to achieve your objective.
2. Never lie if you can achieve your objective by BS.
3. Never lie nor BS if you can achieve your objective by withholding Information.

4. Never lie, BS, nor withhold information if you can achieve your objective with a half-truth.

A particular topic that has received attention is deceptive, or dishonest, agents in negotiations (Sakama et al. 2011; Nguyen et al. 2011; Zlotkin and Rosenschein 1991).

With these concepts in mind, we will pursue research to answer the following:

> *What kind of reasoning models are required to generate explanations of a deceptive or rebellious nature?*

2 Related Work

Research into rebellion and deception have been studied for decades in other fields. We summarize some of these studies in this section, as they will form the foundation and context of our approach.

2.1 Dissent

Dissent assumes a challenge to some system of power or belief (Martin 2008). The author goes further to differentiate between dissent and rebellion. Martin claims that a basic distinction between dissent and rebellion is that dissenters still believe in the process that has been established whereas rebels do not.

2.2 Whistle-blowing

A number of real-world studies on dissent and whistle-blowing in organizations showcase the importance of this topic and area (Hamid et al. 2015; Kenny et al. 2018; Martin 2008; Pershing 2003a; Martin and Rifkin 2004). Many of these discuss the outcomes and results towards both the actor and the agent in these situations.

Martin and Rifkin (2004) discuss organizational responses to whistle-blowing and dissent. Manager reprisals may include building a damaging record towards them, threats against them, isolation, prosecution, or setting them up for failure at their job. In a number of instances, it seems that giving dissent, or becoming a whistle-blower, can quickly turn into an adversarial or antagonistic situation.

2.3 Cynicism

Closely tied to dissent is the idea of cynicism. Mantere and Martinsuo (2001) defines it as:

> (1) a belief that there is a gap between desired and observed organizational identity; (2) a negative affect toward the organization or organizational change (strategy); and (3) tendencies to disparaging and/or critical behaviors toward the organization that are consistent with those beliefs and affect.

They additionally give examples of cynicism as: pessimism, emotional/narrative expressions or outbursts, frustration, irony, accusations, neglect, negative coping behaviors and expressions, and aggression.

2.4 Subversion

Observed actions of resistance to change have been studied in a number of ways. Once such study (Ybema and Horvers 2017) noted that there was a tendency towards "informal or mundane" types of resistance compared to an open, direct, and explicit form of objection. When faced with managerial initiatives, the following expressions of resistance were noted: "careful carelessness", humor, cynicism and skepticism, nostalgic talk (i.e., the "Good ol' Days"), alternative articulations of self-hood, and simulation of productivity.

While these expressions of resistance were present, workers were also camouflaging this dissent with a good-humored appearance. It was noted that hiding dissent allowed for behind-the-scenes inaction to stifle any new directives and also avoided many conversations that workers deemed futile. Similar studies include (Ewick and Silbey 2003; McKay et al. 2013; Reissner 2011).

Along with giving a number of examples of resistance, Martí and Fernández (2013) defined a few different kinds of resistance. Is the resistance individual or collective in nature? Covert or overt? Mundane or heroic?

While overt forms of resistance usually did not work, the *stories* of that resistance remained and were used later to continue forms of resistance.

2.5 Observational Deception Studies

There have been several studies on how deception in society works. A very good resource is (Whiten and Byrne 1988) which defines a number of deceptive actions observed among primates in the wild. They are categorized as either *concealment, distraction, use of a tool,* or *use of another agent*. Examples include hiding things from sight, looking at things to have others avoid looking at something else, and getting other agents to take the blame for something. In addition to primates, there have been deceptive studies for cephalopods (Brown et al. 2012) and dolphins (Hill et al. 2018).

In studies of deception in warfare (Reed 2016), there have been noticeable benefits to using deception. This includes minimizing the amount of resources for a task or ensuring the opponents mis-allocate their resources. Also, most of the example deceptive acts required an extended duration of time to be executed and prove successful. This included preparations for and performing the deceptive actions.

3 Example (Ecological Disaster)

A good case study from (Dunin-Keplicz and Verbrugge 2011) describes an ecological disaster scenario. We present it now to provide context for explanations discussed in Section 4. In the ecological disaster scenario, two poisons have breached containment in a large area. Each poison is highly dangerous on its own. However, when combined they could be explosive. Robots on the ground have the ability to neutralize specific poisons in small areas. An unmanned air vehicle (UAV) can survey large areas to identify high concentrations of poison. A coordinator can issue commands to either the UAV or the ground robots.

A robot can gauge the type and quantity of poisons in the small area around it, and can move across the ground. The UAV can scan larger areas to identify large concentrations of individual poisons, though it reports only the highest quantity. Additionally, the UAV cannot scan areas that are obscured by ceilings or coverings. The coordinator receives the information from the UAV and the Robots, but does not have a view of their own.

Examples of rebellion here could be robots not following commands to enter areas that would explode or the UAV deciding to survey a different area to help robots on the ground compared to an area that the coordinator instructed the UAV to survey.

4 Example Deceptive & Rebellious Explanations

Let us now consider explanations that are rebellious or deceptive within the context given in Section 3. Some of the recurring reasons for generating these explanations include "makes it simpler" or "avoids a larger conversation". These can limit conversations but can also cause misinterpretations. It also has the ability to avoid conversation bottlenecks, letting agents continue to perform other actions.

4.1 An Explanation with Lying

During a disaster, it may come to pass that an agent with the ability to administer first-aid or medical attention, perhaps they are equipped with air masks or oxygen, encounters a victim who will only let them near if they do not inform any law enforcement of their position. In this instance, the agent would administer the first aid or medical treatment and, when asked by the Coordinator to explain their activities, would say they are performing a different activity or in a different location.

4.2 An Explanation that Withholds Information

A robot could be traversing an area trying to help people evacuate or administer aid and they have informed the Coordinator they are performing this task. However, suppose there is a person who wishes to remain anonymous or would refuse help. In this case, the robot could explain its actions but leave out details that would identify the person in later reports or debriefs. This would help save the person and increase that victim's trust in the robot to help them out of the area.

4.3 An Explanation that is only a Half-Truth.

A version of an explanation with a half-truth could be as follows: a medical agent has found and is administering aid to a victim, however the victim is too far gone. Keeping the victim calm with a half-truth or "white lie" explanation would be beneficial to the victim.

4.4 An Explanation that is a Protest

An example of a protest-based explanation could come from the following contingency. An exploratory or search agent has encountered an area of the environment that it deems too hazardous to continue in. The Coordinator asks why it isn't moving forward anymore. The agent responds with,"I will not move forward until the hazardous area has been secured and is safe to pass through."

4.5 An Explanation that is Cynical

As discussed in Section 2.3, cynicism is a bit odd. It is usually used as a form of soft-resistance. The agent still performs the action or command, but may not do it optimally. An example could be that the Coordinator assigns a robot that has a full amount of neutralizing and medical equipment to survey an area. This might take a while for the robot to execute, so the Coordinator might ask why progress is slow, and the explanation could be "If I could fly this would go much quicker." Alternatively, asking to release all of its equipment so that it can be lighter to perform the survey is another example.

4.6 An Explanation with Disobedience

For this instance, the agent is in some sort of situation in which it will not continue an objective given by the Coordinator. Perhaps the Coordinator has tasked an agent with neutralizing an area under a fallen concrete roof. However the agent has noticed a victim in the area being treated by another agent. In that instance the agent could respond to the Coordinator, "I will not neutralize that area, there is a victim in the vicinity. Please assign me a different objective."

5 Enablers of Rebellious Explanations

In order to generate possible deceptive explanations as suggested above an agent would require a few things in its models to properly generate a model. An agent would require an internal model of the domain so that it can reason about possible actions, tasks, and goals. It would also require an internal model of the *external agent's* domain model. This is required so that when generating the "deceptive" aspect, it can be validated against the presumed model of that agent. In addition to these models, a few conditions should be met as well. Notably, a discrepancy must be noticed between the external agent's model and the internal model in relation to the query asked. There needs to be a specific condition or contingency in which the truth would not maximize an overall objective benefit for the agent in question. Likewise, rebellious explanations require similar things such as internal models for both the domain and objectives along with noticing a discrepancy between the objectives, the domain, and the agent's internal model.

Of great interest is the work in model reconciliation (Chakraborti et al. 2017). This is focused on maintaining (at least) two separate models, an internal one to the agent, and an external one for someone interacting with the agent. The idea is for the agent to *reconcile* any differences between these two versions to formulate an explanation. This approach is promising in regards to expanding it towards rebel or deceptive tasks in explanation.

In the case of either deceptive or rebellious explanations, a discrepancy is required. This has been an active research focus. Ingrand and Ghallab (2017) survey work on discrepancy detection. Useful to this thread of research, (Molineaux, et al. 2010) discusses it in the context of goal reasoning.

In terms of reasoning models, (Roberts et al. 2018) discuss some interesting concepts in relation to Goal Networks. A life cycle for goals is also discussed. Combining these goal networks and the temporal nature of goal life cycles, a goal timeline develops. This timeline structure can represent the reasoning needed for some of the explanation models once discrepancies have been detected.

Utilizing models of the world that are not in the explainer's original model is both challenging and novel to pursue for several reasons. It requires an agent to distinguish between different viewpoints of explanations. Introduces reasoning over viable explanations that can be generated. Requires a conversation concerning the ethics of deciding when an agent can decide to deceive. Finally, it opens up the area of XAI to new sets of scenarios - namely those that are deceptive or rebellious.

6 Evaluation

To facilitate the development of deceptive or rebellious explanations, we will need a way to evaluate them. We propose a few areas that may be suitable for such testing. One such testing ground is the RoboCup Rescue. This is a popular disaster simulation (Kitano and Tadokoro 2001) that can be leveraged to simulate examples similar to those given in Section 3. Various games and game simulations may prove useful to test for these explanations. Some game options include Minecraft, One Night Ultimate Werewolf, Secret Hitler, Clue, and Diplomacy. Other relevant domains may include those that involve unmanned air and underwater vehicles. These vehicles require a large amount of autonomy and can be utilized in areas where discrepancies between an operator's situation awareness and the vehicle's belief state differ dramatically.

Along with testing simulations, we can also look at measures of explanation effectiveness. Some of these measures can include clarity, timeliness, or correctness. Did the explanation answer the query? How easy was the explanation to understand? Was the time it took to respond seen as adequate? Is the user's attitude toward the agent lower or higher given this form of explanation?

Acknowledgements This research was performed while the first author held an NRC Research Associateship award at NRL. Thanks to DARPA and NRL for supporting this research. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

Aha, D. W., and Coman, A. 2017. The AI Rebellion: Changing the Narrative. In *31st AAAI Conf. on AI.*

Anderson, E. A.; Irvine, C. E.; and Schell, R. R. 2004. Subversion as a Threat in Information Warfare. *Journal of Information Warfare* 3(2):51–64.

Biros, D. P.; Hass, M. C.; Wiers, K.; Twitchell, D.; Adkins, M.; Burgoon, J. K.; and Nunamaker, J. F. 2005. Task Performance Under Deceptive Conditions: Using Military Scenarios in Deception Detection Research. In *Proc. of the 38th Annual Hawaii Int'l Conf. on System Sciences*, 22b–22b. IEEE.

Boggs, J.; Dannenhauer, D.; Floyd, M.; and Aha, D. 2018. The Ideal Rebellion: Maximizing Task Performance in Rebel Agents. In *6th Goal Reasoning Workshop at IJCAI/FAIM-2018.*

Brown, C.; Garwood, M. P.; and Williamson, J. E. 2012. It Pays To Cheat: Tactical Deception in a Cephalopod Social Signalling System. *Biology letters* 8(5):729–732.

Buchanan, B. G., and Shortliffe, E. H. 1984. Explanation as a Topic of AI Research. *Rule-based expert systems: the MYCIN experiments of the Stanford Heuristic Programming Project* 331.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy. In *Proc. of the 26th IJCAI*, 156–163. AAAI Press.

Clancey, W. J. 1986. Intelligent Tutoring Systems: A Tutorial Survey. Technical report, Standford Univ CA Dept of Computer Science.

Coman, A., and Aha, D. W. 2017. Cognitive Support for Rebel Agents: Social Awareness and Counternarrative Intelligence. In *Proc. of the 5th Conf. on Advances in Cognitive Systems.*

Coman, A., and Aha, D. W. 2018. AI Rebel Agents. *AI Magazine* 39(3):16–26.

Craven, M. W. 1996. Extracting Comprehensible Models from Trained Neural Networks. Technical report, University of Wisconsin-Madison Department of Computer Sciences.

Dannenhauer, D.; Floyd, M.; Magazzeni, D.; and Aha, D. 2018. Explaining Rebel Behavior in Goal Reasoning Agents. In *Explainable AI Planning Workshop at ICAPS 2018.*

DARPA. 2016. Broad Agency Announcement: Explainable Artificial Intelligence (XAI). Online. https://www.darpa.mil/attachments/DARPA-BAA-16-53.pdf.

Doyle, D.; Tsymbal, A.; and Cunningham, P. 2003. A Review of Explanation and Explanation in Case-Based Reasoning. Technical report, Trinity College Dublin, Department of Computer Science.

Dunin-Keplicz, B., and Verbrugge, R. 2011. *Teamwork in Multi-Agent Systems: A Formal Approach*, volume 21. John Wiley & Sons.

Ewick, P., and Silbey, S. 2003. Narrating Social Structure: Stories of Resistance to Legal Authority. *American journal of sociology* 108(6):1328–1372.

Hamid, M. H.; Othman, Z.; et al. 2015. Whistleblowing and Voicing Dissent in Organizations. *Int'l Journal of Management* 6(1):8–15.

Hill, H. M.; Dietrich, S.; Cadena, A.; Raymond, J.; and Cheves, K. 2018. More Than a Fluke: Lessons Learned From a Failure to Replicate the False Belief Task in Dolphins. *Int'l Journal of Comparative Psychology* 31.

Ingrand, F., and Ghallab, M. 2017. Deliberation for Autonomous Robots: A Survey. *Artif. Intell.* 247:10–44.

Isaac, A., and Bridewell, W. 2017. White Lies on Silver Tongues: Why Robots Need to Deceive (and How). *Robot Ethics* 2:157–72.

Keller, K. M.; Miller, L. L.; Robson, S.; Farris, C.; Stucky, B. D.; Oshiro, M.; and Meadows, S. O. 2015. An Integrated Survey System for Addressing Abuse and Misconduct Toward Air Force Trainees During Basic Military Training. Technical report, Rand Project Air Force Santa Monica CA.

Kenny, K.; Fotaki, M.; and Vandekerckhove, W. 2018. Whistleblower Subjectivities: Organization and Passionate Attachment. *Organization Studies* 0170840618814558.

Kitano, H., and Tadokoro, S. 2001. Robocup Rescue: A Grand Challenge for Multiagent and Intelligent Systems. *AI Magazine* 22(1):39–52.

Kott, A., and Ownby, M. 2005. Tools for Real-Time Anticipation of Enemy Actions in Tactical Ground Operations. Technical report, Defense Advanced Research Projects Agency Arlington VA.

Kott, A.; Singh, R.; McEneaney, W. M.; and Milks, W. 2011. Hypothesis-Driven Information Fusion in Adversarial, Deceptive Environments. *Information Fusion* 12(2):131–144.

Mantere, S., and Martinsuo, M. 2001. Adopting and Questioning Strategy: Exploring the Roles of Cynicism and Dissent. In *17th European Group for Organisation Studies Colloquium*.

Martí, I., and Fernández, P. 2013. The Institutional Work of Oppression and Resistance: Learning From the Holocaust. *Organization Studies* 34(8):1195–1223.

Martin, B., and Rifkin, W. 2004. The Dynamics of Employee Dissent: Whistleblowers and Organizational Jiu-Jitsu. *Public Organization Review* 4(3):221–238.

Martin, B. 2008. Varieties of Dissent. *Dissent and the Failure of Leadership* 22–36.

McKay, K.; Kuntz, J. R.; and Näswall, K. 2013. The Effect of Affective Commitment, Communication and Participation on Resistance to Change: The Role of Change Readiness. *New Zealand Journal of Psychology (Online)* 42(2):29.

Miller, T. 2018. Explanation in Artificial Intelligence: Insights from the Social Sciences. *Artif. Intell.*

Molineaux, M.; Klenk, M.; and Aha, D. W. 2010. Goal-Driven Autonomy in a Navy Strategy Simulation. In *Proc. of the 24th AAAI Con. on AI.*

Mourougayane, K., and Srikanth, S. 2015. Intelligent Jamming Threats to Cognitive Radio Based Strategic Communication Networks-A Survey. In *2015 3rd Int'l Conf. on Signal Processing, Communication and Networking*, 1–6. IEEE.

Nguyen, N.-H.; Son, T. C.; Pontelli, E.; and Sakama, C. 2011. ASP-Prolog for Negotiation Among Dishonest Agents. In *Int'l Conf. on Logic Programming and Nonmonotonic Reasoning*, 331–344. Springer.

Pershing, J. L. 2003a. To Snitch or Not To Snitch? Applying the Concept of Neutralization Techniques to the Enforcement of Occupational Misconduct. *Sociological Perspectives* 46(2):149–178.

Pershing, J. L. 2003b. Why Women Don't Report Sexual Harassment: A Case Study of an Elite Military Institution. *Gender issues* 21(4):3–30.

Reed, J. A. 2016. Know Thy Team, Know Thy Enemy: Using Deception Teams to Achieve Air Superiority. Technical report, Air Command and Staff College, Air University Maxwell AFB United States.

Reissner, S. C. 2011. Patterns of Stories of Organisational Change. *Journal of Organizational Change Management* 24(5):593–609.

Roberts, M.; Monteath, I.; Sheh, R.; Aha, D.; Jampathom, P.; Akins, K.; Sydow, E.; Shivashankar, V.; and Sammut, C. 2018. What *Was* I Planning to Do? In *Explainable AI Planning Workshop at ICAPS 2018*.

Sakama, C., and Caminada, M. 2010. The Many Faces of Deception. *Proc. of the 30 Yrs of Nonmonotonic Reasoning*.

Sakama, C.; Caminada, M.; and Herzig, A. 2014. A formal account of dishonesty. *Logic Journal of the IGPL* 23(2):259–294.

Sakama, C.; Tran, S. C.; and Pontelli, E. 2011. A Logical Formulation for Negotiation Among Dishonest Agents. In *22nd IJCAI*.

Sakama, C. 2015. A Formal Account of Deception. In *2015 AAAI Fall Symposium Series*.

Sørmo, F.; Cassens, J.; and Aamodt, A. 2005. Explanation in Case-Based Reasoning–Perspectives and Goals. *Artif. Intell. Review* 24(2):109–143.

Thomas, J., and Biros, D. 2011. A Conceptual Model of Real World, High Stakes Deception Detection. In *2011 44th Hawaii Int'l Conf. on System Sciences*, 1–10. IEEE.

Van Ditmarsch, H.; Van Eijck, J.; Sietsma, F.; and Wang, Y. 2012. On the Logic of Lying. In *Games, actions and social software*. Springer. 41–72.

Van Ditmarsch, H. 2014. Dynamics of Lying. *Synthese* 191(5):745–777.

Weston, J.; Bordes, A.; Chopra, S.; Rush, A. M.; van Merriënboer, B.; Joulin, A.; and Mikolov, T. 2015. Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks. *arXiv preprint arXiv:1502.05698*.

Whiten, A., and Byrne, R. W. 1988. Tactical Deception in Primates. *Behavioral and brain sciences* 11(2):233–244.

Wick, M. R., and Thompson, W. B. 1992. Reconstructive Expert System Explanation. *Artif. Intell.* 54(1-2):33–70.

Ybema, S., and Horvers, M. 2017. Resistance Through Compliance: The Strategic and Subversive Potential of Frontstage and Backstage Resistance. *Organization Studies* 38(9):1233–1251.

Zlotkin, G., and Rosenschein, J. S. 1991. Incomplete Information and Deception in Multi-Agent Negotiation. In *IJCAI*, volume 91, 225–231.

Challenges of Explaining Control

Adrian Agogino[1], **Ritchie Lee**[2], **Dimitra Giannakopoulou**[1]
[1]NASA Ames Research Center
[2]Stinger Ghaffarian Technologies, Inc. (SGT)
NASA Ames Research Center, MS 269-1
Moffett Field, California 94035
{adrian.k.agogino,ritchie.lee,dimitra.giannakopoulou}@nasa.gov

Abstract

Reinforcement learning and evolutionary algorithms are used increasingly in the development of sophisticated control solutions for autonomous systems. However, it is challenging to trust such solutions for safety-critical systems because the rationale behind the control decisions they produce is obfuscated, and hidden behind parameters that are not directly related to the problem they target. Several approaches have been proposed to explain standard supervised learning algorithms, but these approaches cannot be readily applied to control algorithms due to the time-extended nature of the latter. This paper experiments with six techniques in order to develop explanations for autonomous, learning-based control: 1) Bayesian rule lists, 2) Function analysis, 3) Single time step integrated gradients, 4) Grammar-based decision trees, 5) Sensitivity analysis combined with temporal modeling with LSTMs, and 6) Explanation templates. These techniques are tested on a simple 2d domain, where a simulated rover attempts to navigate through obstacles to reach a goal. For control, this rover uses an evolved multi-layer perceptron that maps an 8d field of obstacle and goal sensors to an action determining where it should go in the next time step. Results show that some simple insights in explaining the neural network are possible, but that good intuitive explanations are difficult.

Introduction

Explanation of machine learning algorithms is a challenging and important field of research. Most techniques to date have focused on supervised learning algorithms, such as image processing, text processing and medical diagnosis (Letham et al. 2015; Gunning). Instead of supervised learning, this paper focuses on reward based machine learning such as reinforcement learning and evolutionary algorithms, where rewards are given to measure performance instead of using examples of what is correct. The nature of reward learning and supervised learning is different in both problem domains and learning tools used to solve these problems. In this paper we look at explainability techniques that have been designed for supervised learning problems and apply them to reward learning problems.

Reinforcement learning and evolutionary algorithms can be used to automatically learn high performance control systems for complex problems (Floreano and Mondada 1994; Crites and Barto 1996; Agogino, Stanley, and Miikkulainen 2000). This is particularly the case in the context of autonomy where control may involve many variables and need to dynamically adapt to different environments and situations.

A common form of machine learning is to train a set of weights of a neural network-based control policy. Based on inputs (such as sensors) the control policy can command control actions (such as speed and direction of a vehicle). Training is typically done with a simulator, where the learning algorithm attempts to improve the performance of the control policy through a long series of trials. The goal of this training process is to produce a high-performance non-linear control policy that takes inputs and produces controls.

While a successful training will produce a control policy that achieves high performance in simulation, how the control policy actually works will typically be unclear to its programmers, let alone its end-users. Due to this fact, machine learning algorithms are often referred to as "blackbox": their inputs and outputs can be viewed, but there is no knowledge of their internal workings.

Even when machine learning achieves high performance, it can be difficult to trust for two reasons: 1) coverage, and 2) generalizability. In terms of coverage, while an algorithm may have performed well in scenarios that were tested, there may be other likely scenarios where it would have performed very poorly. In addition since coverage of machine learning algorithms is largely dependent on the data set, the user may not even be aware of the algorithm's coverage and can easily overlook large gaps in the data sets. In terms of generalizability, while the algorithm performed well in the simulator it may not perform well in the real world or in environments that are slightly different than the simulated one. These problems can be exacerbated by the blackbox nature of these learning algorithms, where reward hacking, poorly defined utility functions or simple errors in the simulator can lead to unrealistically high levels of performance that cannot be achieved when deployed. In addition, machine learning algorithms have many unintuitive parameters that have no obvious relation to the underlying control problem, such as number of hidden nodes and learning rates. Yet poor choices of these parameters can lead to poor generalization.

Improving explainability of these blackbox algorithms can help improve trust that they will behave as expected when deployed (Gunning). If a control decision is backed up by a meaningful and understandable rationale, then one can trust that the decision is not made "by chance", and therefore the system can be expected to behave well in other similar circumstances. Additionally, if we understand a learned control algorithm, we can see if there are any clear gaps in coverage, or if there are any obvious flaws that would prevent it from generalizing outside of the simulated environment. On the other hand, what constitutes a meaningful, understandable explanation?

Providing explanations of machine learning is a very active research field. Several approaches have been proposed for standard supervised learning algorithms. Despite this fact, it is still unclear what types of explanations may be suitable in practice. Control further complicates the picture, because control strategies develop over time, and are typically not evaluated over snapshots. How can such strategies be captured in explanations and what type of explanations would those be?

To address this problem, we have experimented with a variety of techniques to provide explanations in the context of a very simple machine learning algorithm that we developed for navigating a rover towards a goal while avoiding obstacles. We decided to build the algorithm from scratch in order to evaluate the pitfalls and errors that may occur in developing such systems, as well as how/what explanations may assist in detecting those. We used six techniques in order to develop explanations: 1) Bayesian rule lists, 2) Function analysis, 3) Single time step integrated gradients, 4) Grammar-based decision trees, 5) Sensitivity analysis combined with temporal modeling with LSTMs, and 6) Explanation templates. This set of techniques was chosen as it represents a diverse set of explanations that could be readily applied to control data. In particular, it includes both local and global explanations. These local attempt to explain a single control action in a particular state. To form a big picture of a control policy with local explanations, we would want many local explanations covering many different states. In contrast global explanations try to explain an overall action policy over all states.

The remainder of the paper is organized as follows. We first present the example obstacle avoidance problem we use throughout the paper. Then we describe the neural network controller and the Monte Carlo algorithm used to determine the weights of the neural network. We subsequently discuss the need for explainability and how simple analysis of algorithm performance may be insufficient. To address this we present six different explainability algorithms applied to the example problem and discuss their relative merits.

Test Problem

We test our explainability methods on a simple test problem, where a rover moving on a 2d plane tries to navigate towards a goal while avoiding obstacles. It does this with a neural network that maps goal and obstacle sensors into a control action that determines the speed and direction of the rover for the next time step. The weights of this neural network are determined with an evolutionary algorithm using a simulation of the environment.

Environment and Utility

In our test domain, a rover attempts to reach a single goal while avoiding 100 obstacles placed randomly on an x-y plane (see Figure 1). The rover starts in the middle of the obstacle field and the goal is located above the obstacle field. At each time step the rover takes a small movement in the x and y direction. At the end of 70 time steps, the rover's performance is evaluated.

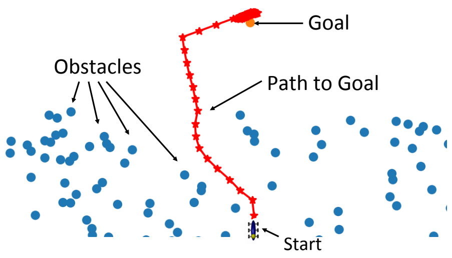


Figure 1: Obstacle Avoidance Problem. A rover attempts to navigate towards a goal while avoiding obstacles in a 2-d plane.

Sensors

At every time step, the rover senses the world through eight continuous sensors (Agogino and Tumer 2004). From a rover's point of view, the world is divided up into four quadrants with fixed orientation to the x-y axis, with two sensors per quadrant (see Figure 2 Left). For each quadrant, the first sensor returns a function of the obstacles in the quadrant at time t. Specifically the first sensor for quadrant q returns the sum of inverse square distances from an obstacle to the rover:

$$s_{1,t} = \sum_{j \in J_q} \frac{1}{{\delta_j}^2} ,$$

where J_q is the set of obstacles in quadrant q and δ_j is the euclidean distance from obstacle j to the rover. The second sensor, $s_{2,t}$, returns the inverse square distance from the rover to all the goals in each quadrant at time t. In our case since there is only one goal, only the quadrant that contains the goal will have a non-zero value, which is $1/d^2$ where d is the euclidean distance from the rover to the goal.

The sensor space is broken down into four regions to facilitate the input-output mapping. There is a trade-off between the granularity of the regions and the dimensionality of the input space. In some domains the tradeoffs may be such that it is preferable to have more or fewer than four sensor regions.

Rover Control Strategies

With four quadrants and two sensors per quadrant, there are a total of eight continuous inputs. This eight dimensional

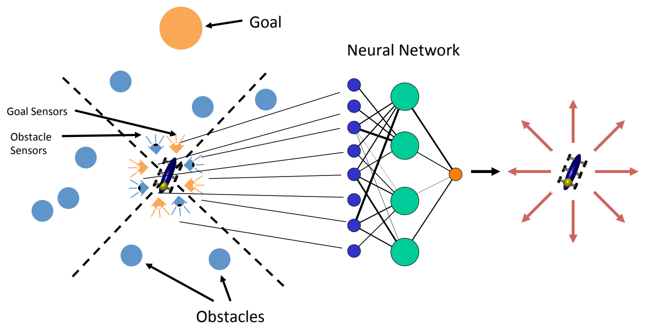


Figure 2: Rover Sensors. The rover has 8 sensors: 4 obstacle sensors and 4 goal sensors. Each sensor observes the presence of objects in its quadrant based on a sum of inverse squared distances. More objects and closer objects there increase sensor value. Sensors are inputs to a neural network that produces a control action determining x, y movement in the next time step.

sensor vector constitutes the state space for a rover. At each time step the rover uses its state to compute a two dimensional output. This output represents the x, y movement relative to the rover's location and orientation.

The mapping from rover state to rover output is done through a Multi Layer Perceptron (MLP) (Haykin 1998), with eight input units, ten hidden units and two output units [1]. The MLP uses a sigmoid activation function, therefore the outputs are limited to the range $(0, 1)$. The actual rover motions dx and dy, are determined by normalizing and scaling the MLP output by the maximum distance the rover can move in one time step. More precisely, we have:

$$dx = 2d_{max}(o_1 - 0.5)$$
$$dy = 2d_{max}(o_2 - 0.5) \ ,$$

where d_{max} is the maximum distance the rover can move in one time step, o_1 is the value of the first output unit, and o_2 is the value of the second output unit.

Monte Carlo Algorithm

We use a simple 2-phase Monte Carlo algorithm to determine the weights for the neural network controller. In the first phase, the weights for each Monte Carlo run are set to a value between -6 and 6 sampled from a uniform distribution. After the weights are selected the rover is evaluated in a simulation for 70 time steps, and its performance is recorded. 1000 Monte Carlo runs are performed in this first phase.

In the second phase, for each Monte Carlo run, the weights are copied from the neural network with the best performance in the first phase. The weights of this copy are then mutated by adding noise selected from a uniform distribution in the range -0.05 to 0.05. 200 Monte Carlo runs are performed from this second phase and the weights of the best performing rover are saved as the final solution of the algorithm.

The reward used to evaluate each Monte Carlo run is as follows:

$$R = \sum_{t=0}^{T} 100 s_{2,t} - s_{1,t}$$

where $s_{2,t}$ is the goal sensor value and $s_{1,t}$ is the sum of obstacle sensor values at time t run for T time steps. This utility goes up when the rover gets closer to the goal and down when it gets close to obstacles. Note that the goal sensor is scaled since the rover is usually much closer to obstacles than the goal.

Explainability

At the completion of the Monte Carlo algorithm we have a neural network capable of controlling a rover in our test problem. Traditionally we would test this controller by running it and observing how it performs, such as by looking at the path it took as shown in Figure 1. We can also look at how its performance improved during training. For instance Figure 3 shows that while random neural networks tend to perform poorly, there are a few that perform much better than average. The figure also shows that in Phase 2 of training that performance improves, but not significantly.

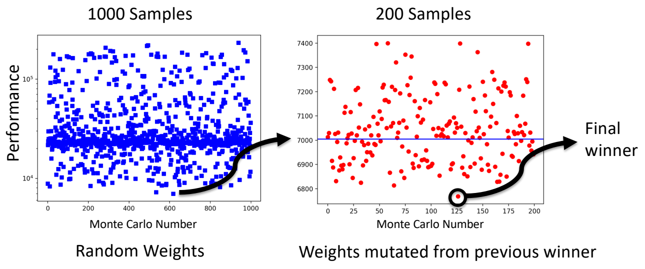


Figure 3: Rover Performance during training, displayed as negative reward. Left: In the first 1000 Monte Carlo runs, performance varies considerably. Right: The next 200 Monte Carlo runs are based on the best sample from the first thousand (performance of this sample shown in blue line).

While this analysis gives some insight into the performance of the neural network controller, it does not tell us how it actually operates. In particular, it does not tell us if the controller has any hidden failure modes that we should be aware of. Machine learning algorithms and neural networks in particular can have many subtle failures that not be apparent in basic testing. A neural network is represented by a large collection of interconnected weights and inspecting the values of these weights is usually not helpful in determining if the network is operating correctly. As an example of such a failure, when we first trained our neural network on the obstacle avoidance problem, we accidentally limited the range of possible weight values to be on too narrow of interval for the neural network to fully approximate non-linear functions. While the training went smoothly and the algorithm produced a viable control policy, the performance of this control policy was significantly lower than what it could

[1]Note that other forms of continuous reinforcement learners could also be used instead of evolutionary neural networks. However neural networks are ideal for this domain given the continuous inputs and bounded continuous outputs.

have been due to this error. Ultimately a unit test, testing the ability of the neural network to approximate a sine wave revealed this issue.

While performance tests and unit tests give some insight into how a neural network is operating, we would like additional explanations of how a trained neural network is actually operating. In this paper we attempt to analyze our neural network using several different explainability methods: 1) Bayesian rule lists, 2) Function analysis, 3) Single time step integrated gradients, 4) Grammar-based decision trees, 5) Sensitivity analysis combined with temporal modeling with LSTMs, and 6) Explanation templates. This set of techniques was chosen as it represents a diverse set of explanations that could be readily applied to control data. In this set, integrated gradients provides a local explanation that attempts to explain a single control action in a particular state. The rest of the explanations are more global in that they attempt to explain the overall control policy independent of state. Another factor is temporal as a control policy attempts to maximize reward over time. Of our explanations only grammar-based decision trees and modeling with LSTMs explicitly attempts to reason over time. In general, the temporal aspect of control makes explanations difficult and complex, therefore most of our explanations attempt to explain individual actions rather than an entire sequence of actions.

Bayesian Rule Lists

Explanations in terms of Bayesian Rule Lists (BRL) (Letham et al. 2015) consist of a list of if-then rules predicated on the controller's inputs. These rules are generated looking at the input/output data associated with the controller, not looking at the neural network itself. Since the domain is continuous and the rules are discrete, a mapping between the domain and the rules needs to be created.

For our example problem we created a simple mapping by hand. For the obstacle sensors, we converted the values of the four quadrants into one of four categorical labels (up, down, left and right), signifying which quadrant had the greatest value. For instance if left quadrant had the greatest value then the label would be "left." The outputs of the controller are converted to two binary values corresponding the x and y values of the output. When the y output has a positive value then its label is 1, otherwise its value is 0. The x value is encoded similarly. To simplify the mapping we ignore the values of the goal sensor and tested the rover in an area where the obstacle sensors dominated. Given these mappings we can convert a set of sensor and control action data into a set of labels.

We performed a BRL extraction using a control run of 70 time steps. Doing this, we can generate four separate rules for going up, down, left and right. The results for the up rule were as follows:

```
if obstacles to left
    go up with probability .19
else if obstacles are up
    go up with probability .87
else
    go up with probability .07
```

Notice that the second rule is somewhat problematic. The neural network actually wants to head towards an obstacle when it is close by. On further inspection, we saw that indeed the rover tends to head towards obstacles, but also turns enough as it is doing so to avoid the obstacle. While effective, this strategy would not seem satisfactory for safety critical systems.

While the BRL was able to expose a potential hazard in the controller, it tended to be hard to use and did not give much insight into the full behavior of the controller. In addition, since it treats the controller as a black box, BRL is only able to characterize observed behavior and could miss important properties of the controller that were not observed in the training data.

Activation Analysis

Our next analysis of the neural network controller is to look at the shape of the input/output functions. For each quadrant in the sensor field there are three functions. The up sensor functions are as follows (the other mappings are appropriately rotated according to the sensor orientation) 1) Mapping from obstacle sensor to Y control action, 2) Mapping from obstacle sensor to X control action, and 3) Mapping from goal sensor to Y control action. The plots of these functions are shown in Figure 4. From the plot we can see that the goal sensor controller behaves as expected. When the goal is present in a sensor, the controller tends to move towards the goal. However the obstacle sensor controllers are a bit more non-intuitive. When the rover gets close to obstacles in the up direction, the X controller will move the rover to the right. However if it gets very close to the obstacles the X controller will start moving in the left direction. Even more worrisome is that when obstacles are close the Y controller will accelerate towards them. This analysis confirms the explanation rules created by the Bayesian Rule List.

Sensitivity Analysis

One way to test some of the properties of a neural network directly is to test the sensitivity of the inputs to the outputs (Tulio Ribeiro, Singh, and Guestrin 2016; Sundararajan, Taly, and Yan 2017). This analysis may be able to tell us for a particular location, which inputs are the most important to the controller's decision.

Gradient Analysis The most basic form of sensitivity analysis is gradient analysis where we measure the gradient of the input with respect to the outputs. This can be accomplished in neural networks using backpropagation. To test this analysis we created a scenario where a rover is located right below the location of an obstacle (see Figure 5). In this scenario we then calculated the gradient of each of the four obstacle sensors with respect to the controller output. The results are as follows:

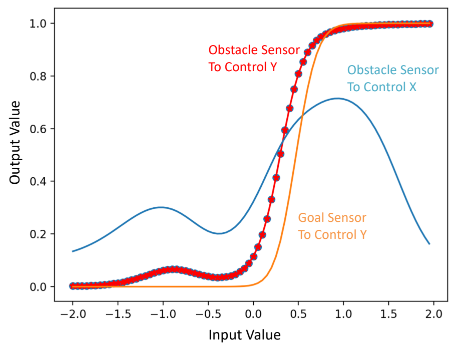


Figure 4: Neural Network Controller Functions. Function analysis shows rover should move towards goal as expected. However, rover also has a tendency to move towards obstacles. It only avoids them by turning when it gets close.

```
Up:           0.031
Left:           0.227
Down:           0.120
Right:            0.139
```

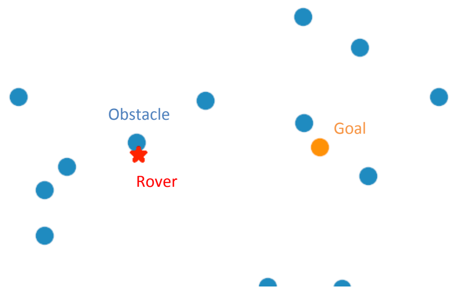


Figure 5: Scenario for Sensitivity Analysis.

This result shows the limitation of basic gradient analysis. We would expect the Up sensor to be the most important to the control, since there is an obstacle very close in the up direction. However, since the rover is so close to the obstacle this sensor saturated so any small change in its value causes almost no change control action. Therefore this sensor actually has the smallest gradient, which is the opposite of what we would hope in terms of explainability. Instead of looking at only local changes in its value we need to look at the effects of larger changes.

Integrated Gradients Integrated gradients attempts to solve this limitation of local gradients by adding a series of gradients from the sensor value of interest down to a baseline sensor value. In this way any important change that happens along this path will be recorded. To test integrated gradients we perform a test where the baseline sensor has a value of zero in all four quadrants and calculate 100 gradients from rover position in our scenario down to the baseline value. The results are as follows:

```
Up:           14.67
Left:           16.40
Down:           11.10
Right:            12.00
```

These results are somewhat more satisfying as the up sensor now has the second largest value.

Explanation Template

Our next attempt at explaining the behavior of the neural network is to model its global properties with respect to an understandable control algorithm (Chandrasekaran, Tanner, and Josephson 1989). We call this control algorithm an "explanation template." This template comprises a simple control algorithm that is easy to comprehend with free parameters that are determined by analyzing the behavior of the neural network. We tried this technique using a simple linear policy. Here is the policy template for the upward looking goal and obstacle sensors:

$$\begin{aligned} v_{up} &= w_0 s_{o,u} + w_1 s_{g,u} \\ v_{right} &= w_2 s_{o,u} \end{aligned}$$

where v_{up} and v_{right} are the up and right velocities for the next time step, $s_{o,u}$ is the value for the upward looking obstacle sensor, $s_{g,u}$ is the value for the upward looking goal sensor, and w_0, w_1, w_2 are the free parameters. Using data from 50 trials of the rover we performed linear regression and found the values of the free parameters producing the following explanation of the system:

$$\begin{aligned} v_{up} &= \frac{s_{o,u}}{803} + \frac{s_{g,u}}{6139} \\ v_{right} &= \frac{s_{o,u}}{1585} \end{aligned}$$

This explanation shows that the neural network has a small tendency to move towards the goal, but a large tendency to move towards an obstacle. It is able to avoid obstacles as it also has a tendency to move right when it approaches an obstacle. These findings are consistent with the function analysis and the Bayesian rule lists.

Grammar-Based Decision Trees

Our next attempts use grammar-based decision trees (GBDTs) (Lee et al. 2018). The idea is to learn an interpretable model from data and then inspect the learned rules to gain insight into system behavior. GBDT generalizes a traditional decision tree, where the decision rules are Boolean expressions derived from a context-free grammar. The grammar allows any logical language to be used and the user can tune the grammar for explainability. GBDT has been shown to provide good representational ability while being interpretable (Lee et al. 2018). GBDT can model different types of data by choosing an appropriate grammar. For example, a grammar based on first-order logic can be used to model static data, while a grammar based on temporal logic can be used to model time series data. We explored two approaches to applying the GBDT model. The first approach models the input-output behavior of the neural network policy and the second approach models the time series data that the policy and its environment together produces.

GBDT Control Policy Modeling In this first GBDT approach, we model the input-output behavior of the neural network policy. We learn an interpretable model that approximates the behavior of the policy and then inspect the learned rules to gain insight into the decisions of the policy.

To construct the training data for the GBDT, we use the input-output pairs of the neural network policy seen during its training. Since the output of the neural network policy is a relative position in 2d, but GBDT can only produce discrete output, we take the relative angle of the network output and round it to the nearest 45 degrees. We use a simple grammar consisting of comparison operators less than $<$ and greater than $>$ operating on the input features xid; and logical operators conjunction $\wedge$, disjunction $\vee$, and negation $\neg$ that enable the formation of more complex expressions. The full grammar is shown in Figure 6.

$$\begin{aligned} b &\mapsto (b \wedge b) \mid (b \vee b) \mid \neg b \\ b &\mapsto (X[xid] < X[xid]) \mid (X[xid] > X[xid]) \\ xid &\mapsto top \mid left \mid bottom \mid right \end{aligned}$$

Figure 6: GBDT Grammar for Modeling Control Policy.

The GBDT was trained using genetic programming (Koza 1992) to optimize each rule of the tree (Lee et al. 2018). The resulting GBDT, shown in Figure 7, attained 85.5% accuracy. The GBDT found two rules to distinguish between three policy outputs up, up_left, and up_right. The reason there are only three actions used is because the goal is located above the start point, so those are the primary actions required for successful navigation. In Figure 7, if there are more obstacles to the left than to the bottom, then go up and to the left. This behavior can be observed in the two up_left segments in Figure 1 as the agent navigates toward and around the cluster of the obstacles to the left. The second rule has two terms. If bottom is greater than right and top is greater than left, then move up. This behavior is seen in Figure 1 as the rover moves away from obstacles to the bottom passing obstacles to the right. As the rover approaches the goal, there are no obstacles to the top or left, so top equals left and the second rule becomes false. In this case, the output is up_right.

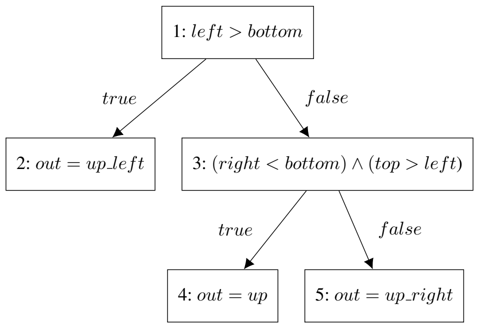


Figure 7: GBDT Result from Modeling Control Policy.

Our result reveals that the learned neural network policy may be overfitted to the scenario because the output relies on this specific arrangement of the obstacles. The discovered rules are indeed true patterns in the data. However, it is unclear that these rules provide satisfactory explanations to humans. For example, humans do not find comparisons between different axes, such as left $>$ bottom, very intuitive.

GBDT Temporal Modeling The above approach does not take into account (1) the temporal nature of the problem and (2) the interactions between the controller and the environment. In this second GBDT approach, we attempt to capture the temporal properties of the combined controller-environment system. We construct a training dataset where the inputs are multivariate sequences of the obstacle sensor values, goal sensor values, policy output, and agent position, and the target outputs are whether the sequence was produced by the final (optimal) neural network control policy or another (suboptimal) controller that was considered but ultimately discarded during training. We train a GBDT model on the temporal data and then inspect the learned rules to gain insight into the temporal properties that distinguish between paths from the optimal and suboptimal controller.

We specify a grammar based on a simple temporal logic as shown in Figure 8. The grammar includes temporal operators globally G and eventually F; elementwise logical operators conjunction $\wedge$, disjunction $\vee$, negation $\neg$; and comparison functions that perform elementwise comparison of a feature sequence to precomputed constants. These comparison functions are expressed in the grammar in the form $f_{op}(xid, vid)$, which computes $X[xid] \; op \; V[xid, vid]$, where $X[xid]$ is the temporal sequence of feature xid, op is a comparison operator, and $V[xid, vid]$ is a precomputed lookup table that returns the vid'th decile division point of the range of feature xid in the data.

The GBDT was trained using genetic programming (Koza 1992) to optimize each rule of the tree (Lee et al. 2018). The resulting GBDT, which attained 99.9% accuracy, is shown in Figure 9. The GBDT model identified three temporal properties relevant to distinguishing between whether a sequence is optimal or suboptimal. The following properties need to be simultaneously satisfied for the input sequence to be classified as optimal: (1) At some point, $action\_x$ reaches a value that is greater than 90% of the range of $action\_x$ in the data (node 1 in Figure 9); (2) The following statement must be false: At some point, obs_sense_right is greater than 30% of the range of obs_sense_right in the data (node 2)(In other words, obs_sense_right must be globally below 30% of its range); and (3) At some point, $action\_y$ is greater than or equal to 80% of the range of $action\_y$ in the data (node 4).

In summary, the GBDT has discovered that strong right and strong up actions combined with a weak right obstacle sensor are correlated with the optimal policy. While these properties hold true in the data, it did not provide a very deep insight or satisfying explanation for the control policy.

$$
\begin{aligned}
b &\mapsto G(\vec{b}) \mid F(\vec{b}) \\
\vec{b} &\mapsto (\vec{b} \wedge \vec{b}) \mid (\vec{b} \vee \vec{b}) \mid \neg \vec{b} \\
\vec{b} &\mapsto (\vec{r} < \vec{r}) \mid (\vec{r} \leq \vec{r}) \mid (\vec{r} > \vec{r}) \mid (\vec{r} \geq \vec{r}) \\
\vec{b} &\mapsto f_{<}(xid, vid) \mid f_{\leq}(xid, vid) \\
\vec{b} &\mapsto f_{>}(xid, vid) \mid f_{\geq}(xid, vid) \\
\vec{r} &\mapsto X[xid\_sens] \mid X[xid\_pos] \\
xid &\mapsto xid\_sens \mid xid\_pos \\
xid\_sens &\mapsto xid\_obs \mid xid\_goal \\
xid\_obs &\mapsto obs\_sens\_top \mid obs\_sens\_left \\
xid\_obs &\mapsto obs\_sens\_bottom \mid obs\_sens\_right \\
xid\_goal &\mapsto goal\_sens\_top \mid goal\_sens\_left \\
xid\_goal &\mapsto goal\_sens\_bottom \mid goal\_sens\_right \\
xid\_pos &\mapsto xid\_action \mid xid\_loc \\
xid\_action &\mapsto action\_x \mid action\_y \\
xid\_loc &\mapsto x \mid y \\
vid &\mapsto |(1:10)
\end{aligned}
$$

Figure 8: GBDT Grammar for Temporal Modeling.

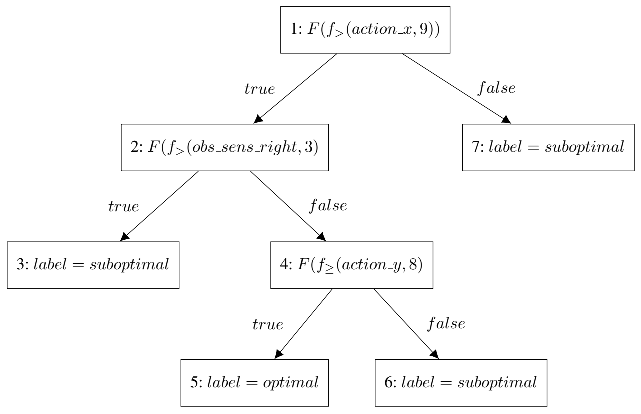


Figure 9: GBDT Result from Temporal Modeling.

Temporal Modeling using LSTMs

This approach combines temporal modeling with attributions to highlight the most salient sequential inputs. We begin by training a long short-term memory (LSTM) classifier (Hochreiter and Schmidhuber 1997) to distinguish between input sequences produced by the final (optimal) neural network controller and sequences produced by another (suboptimal) controller considered (but ultimately discarded) during training. Then we apply integrated gradients (Sundararajan, Taly, and Yan 2017) to evaluate the importance of each input. Because LSTM is a neural network for sequential data, attributions highlight not only which features are important, but also at which time steps. Attributions produce local explanations in that explanations apply to a particular example, rather than explaining global patterns over the dataset.

Figure 10 shows an interesting attributions result that occurs in many examples classified as optimal in the data. In this example, we see a clear repeating pattern in features 7 and 8 that is highlighted by attributions as being the most important in classifying this example as being produced by the optimal controller. Feature 7 is the bottom goal sensor and feature 8 is the right goal sensor. It is also observed that the attribution assigns more importance to the latter parts of the sequence. We investigated the highlighted values in the data and discovered that there is an interesting phenomenon in the data. Sequences produced by the optimal controller reaches the goal much sooner than 70 time steps. To collect maximum reward, the agent stays near the goal for as long as possible. However, because a complete stop is difficult to learn, the controller learned to cycle near the goal, and it is this cycling that is being highlighted by the attributions. The controller has learned the following behavior: (1) When the goal is near and located below, move a small amount downwards and to the left; and (2) when the goal moves from being detected by the bottom sensor to being detected by the right sensor, then jump up and rightward and restart the cycle. Indeed, the optimal controller exhibits this cycling behavior while the suboptimal ones do not.

While this approach using temporal modeling has demonstrated that it can help identify interesting patterns in the data, the algorithm merely highlights parts of the data and does not elaborate on why those parts are important. Ultimately, a human must perform additional analysis to try to understand the relevance, which may be very challenging.

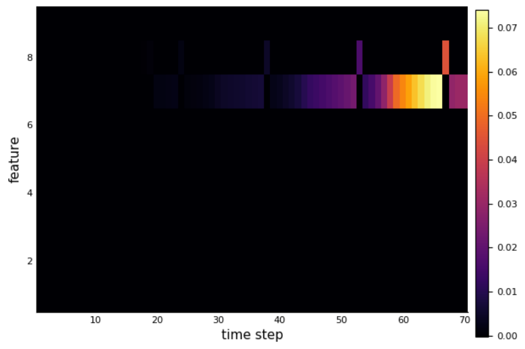


Figure 10: Attributions Result on LSTM Model.

Discussion

While several explanation algorithms have been successfully used on supervised learning problems, direct application to reward based controls learning is somewhat illusive. A large part of this is due to the time-extended property of control policies. An action taken at a particular time step may seem sub-optimal at that particular time step but has benefits for future time steps. This limits a lot of direct application of supervised learning explanation as these explanations will tend to explain the superficial benefit of the action for the immediate time step and will likely miss the explanations of the future benefits. Our use of grammar-Based decision trees and temporal modeling attempt to address this issue, but they also lead to another problem: Control policies that need to optimize for future time steps are performing operations that are inherently complex and are difficult to summarize with simple explanations. In our test-domain the explanation algorithms are able to expose a major flaw in the operation of our learned neural network controller. However, it seems unlikely that they would be able to reveal more subtle issues or would be able to scale to more complex learned controllers. In addition the explanations do not seem as convincing or as useful as the explanations the same algorithms provide for their original supervised learning domain.

Conclusion

Explaining a control algorithm based on machine learning is difficult due to the black-box nature of machine learning algorithms and the time-extended properties of control problems. In this paper we attempt to explain such a controller used on a simple obstacle avoidance problem: a neural network trained using a Monte Carlo algorithm. We do this by applying a number of explainability algorithms to this problem. These algorithms look at the inputs and outputs of the controller and based on these values attempt to explain what the controller is trying to do. The explanation algorithms proved useful in revealing a potential hazard in the controller, where it tries to head towards an obstacle and then turn to avoid it. However beyond this flaw it was difficult to gain deep insights into these explanations.

Acknowledgments

This work was supported by the ATTRACTOR project within NASA's Convergent Aeronautics Solutions (CAS) program.

References

Agogino, A., and Tumer, K. 2004. Efficient evaluation functions for multi-rover systems. In *The Genetic and Evolutionary Computation Conference*, 1–12.

Agogino, A.; Stanley, K.; and Miikkulainen, R. 2000. Online interactive neuro-evolution. *Neural Processing Letters* 11:29–38.

Chandrasekaran, B.; Tanner, M. C.; and Josephson, J. R. 1989. Explaining control strategies in problem solving. *IEEE Expert* 4(1):9–15.

Crites, R. H., and Barto, A. G. 1996. Improving elevator performance using reinforcement learning. In Touretzky, D. S.; Mozer, M. C.; and Hasselmo, M. E., eds., *Advances in Neural Information Processing Systems - 8*, 1017–1023. MIT Press.

Floreano, D., and Mondada, F. 1994. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In *Proc. of Conf. on Simulation of Adaptive Behavior*.

Gunning, D. Explainable artificial intelligence (xai).

Haykin, S. 1998. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 2nd edition.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press.

Lee, R.; Kochenderfer, M. J.; Mengshoel, O. J.; and Silbermann, J. 2018. Interpretable categorization of heterogeneous time series data. In *sdm*. SIAM.

Letham, B.; Rudin, C.; H. McCormick, T.; and Madigan, D. 2015. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics* 9:1350–1371.

Sundararajan, M.; Taly, A.; and Yan, Q. 2017. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 3319–3328. JMLR. org.

Tulio Ribeiro, M.; Singh, S.; and Guestrin, C. 2016. why should i trust you?: Explaining the predictions of any classifier. 97–101.

Online Explanation Generation for Human-Robot Teaming

Mehrdad Zakershahrak, Ze Gong, Nikhillesh Sadassivam, Akkamahadevi Hanni and Yu Zhang[1]

***Abstract*— As Artificial Intelligence (AI) becomes an integral part of our life, the development of explainable AI, embodied in the decision-making process of an AI or robotic agent, becomes imperative. For a robotic teammate, the ability to generate explanations to explain its behavior is one of the key requirements of an explainable agency. Prior work on explanation generation focuses on supporting the reasoning behind the robot's behavior. These approaches, however, fail to consider the mental workload needed to understand the received explanation. In other words, the human teammate is expected to understand any explanation provided, often before the task execution, no matter how much information is presented in the explanation. In this work, we argue that an explanation, especially complex ones, should be made in an online fashion during the execution, which helps spread out the information to be explained and thus reducing the mental workload of humans. However, a challenge here is that the different parts of an explanation are dependent on each other, which must be taken into account when generating online explanations. To this end, a general formulation of online explanation generation is presented along with three different implementations satisfying different online properties. We base our explanation generation method on a model reconciliation setting introduced in our prior work. Our approaches are evaluated both with human subjects in a standard planning competition (IPC) domain, using NASA Task Load Index (TLX), as well as in simulation with ten different problems across two IPC domains.**

I. INTRODUCTION

As intelligent robots become more prevalent in our lives, the interaction of these AI agents with humans becomes more frequent and essential. One of the most important aspects of human-AI interaction is for the AI agent to provide explanations to convey the reasoning behind the robot's decision-making [1]. An explanation provides justifications for the agent's intent, which helps the human maintain trust of the robotic peer as well as a shared situation awareness [2], [3]. Prior work on explanation generation often focuses on supporting the motivation for the agent's decision while ignoring the underlying requirements of the recipient to understand the explanation [4], [5], [6]. However, a good explanation should be generated in a lucid fashion from the recipient's perspective [7].

To address this challenge, the agent should consider the discrepancies between the human and its own model while generating explanations. In our prior work [7], we encapsulate such inconsistencies as *model differences*. An explanation then becomes a request to the human to adjust the model differences in his mind so that the robot's behavior would make sense in the updated model, which is used to produce the human's expectation of the robot. The general decision-making process of an agent in the presence of such model differences is termed *model reconciliation* [7], [8].

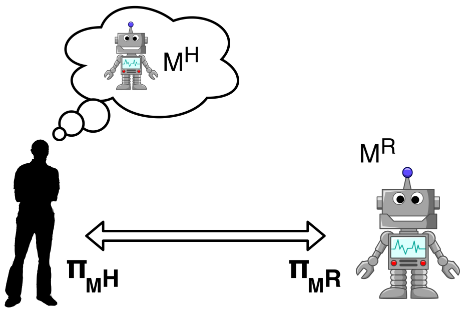


Fig. 1: The model reconciliation setting [7]. M^R represents the robot's model and M^H represents the human's model of expectation. Using M^H, the human generates π_{M^H}, which captures the human's expectation of the robot. Whenever the two plans are different, the robot should explain by generating an explanation to reconcile the two models.

One remaining issue, however, is the ignorance of the mental workload required of the human for understanding an explanation. In most earlier work on explanation generation, the human is expected to understand any explanation provided regardless of how much information is present and no discussion has been provided on the process for presenting the information. In this work, we argue that explanations, especially complex ones, should be provided in an online fashion, which intertwines the communication of explanations with plan execution. In such a manner, an online explanation requires less mental workload at any specific point of time. One of the main challenges here, however, is that the different parts of an explanation could be dependent on each other, which must be taken into account when generating online explanations. The online explanation generation process spreads out the information to be communicated while ensuring that they do not introduce cognitive dissonance so that the different parts of the information are perceived in a smooth fashion.

A. Motivating Example

Let us illustrate the concept of online explanations through a familiar situation between two friends. Mark and Emma want to meet up to study together for an upcoming exam. Mark is a take-it-easy person so he plans to break the review session into two 60 minutes parts, grab lunch in between the sub-sessions and go for a walk after lunch. On the other

[1]Mehrdad Zakershahrak, Ze Gong, Nikhillesh Sadassivam, Akkamahadevi Hanni and Yu Zhang are with the School of Computing, Informatics and Decision Systems Engineering, Arizona State University, Tempe, AZ. {mzakersh,zgong11,nsadassi, ahanni,Yu.Zhang.442}@asu.edu

hand, Mark knows that Emma is of a more focused type who would rather keep the review in one session and get lunch afterwards. Mark would like to keep his plan. However, had he explained to Emma at the beginning of his plan, he knew that Emma would have proposed to order takeouts for lunch on the way before the review session. Instead, without revealing his plan, he goes with Emma to the library. After studying for 60 minutes, he then explains to Emma that he cannot continue without energy, which makes going to lunch the best option for both. At the same time, Mark refrained from telling Emma (until after lunch) that he also needed a walk since otherwise Emma would have proposed for him to take a walk alone while she stays a bit longer for review, and then to meet up at the lunch place.

The above example demonstrates the importance of providing an explanation in an online fashion. Mark gradually reveals the reasoning to maintain his plan as the execution unfolds so that it also becomes both acceptable and understandable to Emma, even though being subject to different values due to model differences (e.g., Mark values lunch break more than Emma thinks he does). The key point here is to explain minimally and only when necessary. In this way, the information to be conveyed is spread out throughout the plan execution, potentially with even a reduced amount of information, so that there is less mental workload requirement at the current step–from Emma's perspective, the interaction with Mark is more straightforward.

In this paper, we develop a new method for explanation generation that intertwines explanation with plan execution. The new form of explanation is referred to as *online explanation*, which considers the mental workload of the receiver of an explanation by breaking it into multiple parts that are to be communicated at different times during the plan execution. We implemented three different approaches for online explanation generation, each focusing on different "online" properties. In the first approach, our focus is on matching the plan prefix. In the second approach, the focus is on making the very next action understandable to the human teammate. In the third approach, the focus is on matching the prefix of the robot's plan with any possible optimal human plan. We use a model search method that ensures that the earlier information communicated would not affect the later parts of the explanation. This creates a desirable experience for the recipient by significantly reducing the mental workload. Our approaches are evaluated both with human subjects and in simulation.

II. Related Work

AI and its numerous applications have provided astounding benefits in areas such as transportation, medicine, finance and military in recent years, but AI agents are so far limited in their ability to operate as a teammate. To be considered a teammate, the agent must not only achieve a given task, but also provide a level of transparency to other members of the team [3]. One of the ways to achieve this is to enable AI agents to be self-explanatory in their behaviors. Recently, explainable AI paradigm [9] rises as one essential constituent of human-AI collaboration. Explainable AI helps improve human trust of the AI agent and maintain a shared situation awareness by contributing to the human's understanding of the underlying decision-making process of the agent.

The explainable agency's effectiveness [10] is assessed based on its capability to model the human's perception of the AI agent accurately. This means that an explainable AI agent must not only model the world, but also the other agents' perception of itself [11]. This model of the other agents allows the agent to infer about their expectation of itself. Using this model, an agent can generate legible motions [12], explicable plans [8], [13], [14], or assistive actions [15]. In these approaches, an agent often substitutes cost optimality with a new metric that simultaneously considers cost and explicability. Another way of using the model is for an AI agent to signal its intention before execution [16]. The motivation here is to use the model to search for additional context information that would help improve human understanding.

A third way of using this model is for the agent to explain its behavior by generating explanations [4], [5], [6]. Similar to intention signaling, this method has the benefit that the agent can maintain its optimal behavior. Research along this direction has focused on generating the "right" explanations based on the recipient's perception model of an explanation [7], [17]. This is useful, however, only with the assumption that the explanation can be understood, regardless of how much information is provided or whether sufficient time is given–the mental workload that is required for understanding an explanation is largely ignored.

In our prior work, we have studied how the ordering of the information of an explanation may influence the perception of an explanation [18]. In this work, we further argue that an explanation must sometimes be made in an online fashion. This is especially true for complex explanations that require a large amount of information to be conveyed. The idea behind online explanation generation is to provide a minimal amount of information that is sufficient to explain part of the plan that is of interest currently (e.g., the next action), and in such a way intertwine explanation generation with plan execution.

III. Explanation Generation

Our problem definition is based on the model reconciliation setting defined in our prior work [7]. We provide a brief review of the relevant concepts before defining our problem in this work. Our problem is closely associated with planning problems so we first provide the background here. A planning problem is defined as a tuple $(F, A, \mathcal{I}, \mathcal{G})$ using PDDL [19], similar to STRIPS [20]. F is the set of predicates used to specify the state of the world and A is the set of actions used to change the state of the world. Actions are defined with a set of preconditions, add and delete effects. $\mathcal{I}, \mathcal{G}$ are the initial and goal state.

Definition 1 (Model Reconciliation): : A model reconciliation is a tuple $(\pi^*_{I,G}, \langle M^R, M^H \rangle)$, where $cost(\pi^*_{I,G}, M^R) = cost^*_{M^R}(I, G)$ and $\pi^*_{I,G}$ is the robot's plan to be explained.

Where $cost(\pi^*_{I,G}, M^R)$ is the cost of the plan generated using M^R and $cost^*_{M^R}(I,G)$ is the cost of the optimal plan based on the initial and goal state pair under M^R. In other words, the robot plan to be explained is required to be optimal according to M^R, assuming rational agents. The model reconciliation setting also takes the human's model M^H into account, which captures the human's expectation of the robot's behavior. When the robot's behavior to be explained (i.e., $\pi^*_{I,G}$) matches with the human's expected behavior, the models are said to be reconciled for the plan. A figure that illustrates the model reconciliation setting is presented in Figure 1. Explanation generation in a model reconciliation setting means bringing two models, M^H and M^R, *"close enough"* by updating M^H such that $\pi^*_{I,G}$, the robot's plan, becomes fully explainable (optimal) in the human's model. A mapping function was defined in [7] to convert a planning problem into a set of features that specifies the problem as Γ: $\mathcal{M} \longmapsto S'$ is a mapping function, which transfers any planning problem $(F, A, \mathcal{I}, \mathcal{G})$ to a state s' in the feature space as follows:

$$\tau(f) = \begin{cases} init-has-f, & \text{if } f \in \mathcal{I}. \\ goal-has-f, & \text{if } f \in \mathcal{G}. \\ a-has-precondition-f, & \text{if } f \in pre(a), a \in A. \\ a-has-add-effect-f, & \text{if } f \in eff^+(a), a \in A. \\ a-has-del-effect-f, & \text{if } f \in eff^-(a), a \in A. \\ a-has-cost-f, & \text{if } f = c_a, a \in A. \end{cases}$$

$$\Gamma(\mathcal{M}) = \{\tau(f) | \forall f \in \mathcal{I} \cup \mathcal{G} \cup \bigcup_{a \in \mathcal{A}} \{f' | \forall f' \in \{c_a\} \cup pre(a) \cup eff^+(a) \cup eff^-(a)\}\}$$

In other words, the mapping function converts a planning problem into a set of features that specifies the problem.

Definition 2 (Explanation Generation [7]): The explanation generation problem is a tuple $(\pi^*_{I,G}, \langle M^R, M^H \rangle)$, and an explanation is a set of unit feature changes to M^H such that 1) $\Gamma(\widehat{M^H}) \setminus \Gamma(M^H) \subseteq \Gamma(M^R)$, and 2) $cost(\pi^*_{I,G}, \widehat{M^H}) - cost^*_{\widehat{M^H}}(I,G) < cost(\pi^*_{I,G}, M^H) - cost^*_{M^H}(I,G)$, where $\widehat{M^H}$ is the model after the changes.

An explanation hence reconciles two models by making the cost difference between the human's expected plan and the robot's plan smaller after the model updates.

Definition 3 (Complete Explanation [7]): Given an explanation generation problem, a complete explanation is an explanation that satisfies $cost(\pi^*_{I,G}, \widehat{M^H}) = cost^*_{\widehat{M^H}}(I,G)$.

The robot's plan must be optimal in the human's model after a complete explanation ($\widehat{M^H}$). A minimal complete explanation (MCE) [7] is defined as a complete explanation that contains the minimum number of unit feature changes.

IV. Online Explanation Generation (OEG)

While the previous explanation generation approach provides a framework to generate explanations considering both the robot's model and the human's model, it largely ignores the mental workload requirement of the human for understanding the explanation. We introduce *online explanation generation* to address this issue. The key here is to only provide a minimal amount of information during the plan execution to explain the part of the plan that is of interest and not explainable.

Definition 4 (Online Explanation Generation): Given a model reconciliation problem, an online explanation is a set of sub-explanations (e_k, t_k), where e_k represents the kth set of unit features to be made (as a sub-explanation) at step t_k in the plan.

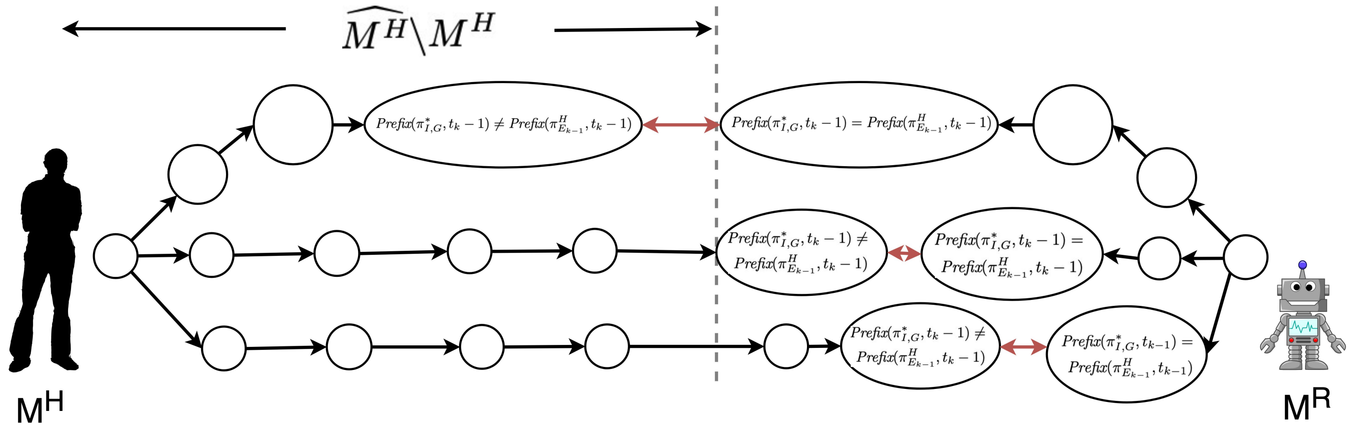
Basically, an online explanation requires only that any actions in the robot's plan before the kth sub-explanations will match with that of the human's expectation. In such a way, the robot can split an explanation into multiple parts, which are made in an online fashion as the plan is being executed. We provide three different approaches of online explanation generation based on the definition provided, while each of these approaches focus on one aspect of explanation generation intertwined with plan execution. Section IV-A discusses *OEG with Plan Prefix matching*, Section IV-B describes *OEG with Next Action matching* and Section IV-C explains *OEG with any prefix matching*.

A. OEG for matching Plan Prefix (OEG-PP)

To generate the sub-explanations (i.e., $\{e_k\}$) for an online explanation, the planning process must consider how the sequence of model changes would result in the changes of the human's expectations after each sub-explanation. Similar to the search process for complete explanations [7], we convert the problem of explanation generation to the problem of model search in the space of possible models. The challenge here is that the model changes may not be independent, i.e., future changes may render a mismatch in the previously reconciled plan prefixes. To address this issue, it must be ensured that the model changes after e_k, i.e., $e_{k+1:m}$ where m denotes the size of the set of sub-explanations, would not change the plan prefixes in M^H. This can be achieved by searching from M^R to M^H to find the largest set of model changes which ensure that the plan prefix would not change afterwards after further sub-explanations. This search process is illustrated in Figure 2. An OEG-PP is a set of sub-explanations (e_k, t_k) such that:

$$\begin{gathered} \forall k > 1, \textit{Prefix}(\pi^*_{I,G}, t_k - 1) = \textit{Prefix}(\pi^H_{E_{k-1}}, t_k - 1) \\ \Gamma(M^H_{E_{k-1}}) = \Gamma(M^H) \cup E_{k-1} \\ s.t. \\ \bigcup_i e_i = \Gamma(\widehat{M^H}) \backslash \Gamma(M^H) \subseteq \Gamma(M^R) \end{gathered} \tag{1}$$

where *Prefix*(π, t) returns the prefix of a plan π up to step t_{k-1}. E_k represents $e_{1:k}$ and $\pi^H_{E_k}$ is the optimal plan created from $M^H_{E_k}$ (M^H after providing sub-explanations e_1 to e_k). More specifically, the following process will be performed recursively for each sub-explanation. First, we continue moving along $\pi^*_{I,G} = (a_1, a_2, ..., a_n)$ as long as the plan prefix matches with the prefix of the plan using the human model M^H. Let $t = t_1$ be the first plan step where they differ. Our search for the sub-explanation starts



$\widehat{M^H}\backslash M^H$
$Prefix(\pi^*_{I,G}, t_k - 1) \neq Prefix(\pi^H_{E_{k-1}}, t_k - 1)$
$Prefix(\pi^*_{I,G}, t_k - 1) = Prefix(\pi^H_{E_{k-1}}, t_k - 1)$
$Prefix(\pi^*_{I,G}, t_k - 1) \neq Prefix(\pi^H_{E_{k-1}}, t_k - 1)$
$Prefix(\pi^*_{I,G}, t_k - 1) = Prefix(\pi^H_{E_{k-1}}, t_k - 1)$
$Prefix(\pi^*_{I,G}, t_k - 1) \neq Prefix(\pi^H_{E_{k-1}}, t_k - 1)$
$Prefix(\pi^*_{I,G}, t_{k-1}) = Prefix(\pi^H_{E_{k-1}}, t_{k-1})$
M^H
M^R

Fig. 2: Model space search process for OEG-PP. Compared to MCE in the previous work [7], the difference is that in our approach the search starts from the robot model and stops where the plan prefixes for the updated human model and the robot model match, while in the previous approach the search process starts from the human model (M^H). In this aspect, our research process is more akin to MME [7]. However, since we are focusing on matching the prefixes rather than the whole plan in one shot, our approach must run this process multiple times compared to only once in MME. While seemingly more computationally expensive, this characteristic actually allows us to beat both MCE and MME in terms of computation since our approaches at any time consider only a small set of changes (see results). The dotted line represents the border of the maximum state space model modification in robot model which reconciles the two models up to where the plan execution currently is. Maximum updates to the robot model is equivalent to minimum updates to the human model.

with M^R. It finds the largest set of model changes to M^R such that the prefix of a plan using the corresponding model (i.e., M^R minus the set of changes) matches with that of $\pi^*_{I,G}$ up to step $t_2 - 1$. The complement set of changes (i.e., the difference between M^H and M^R, minus this set of changes) will be e_1. For the next recursive step, we will start from action t_1 and the human model will be $M^H_{E_1}$. To ensure that the prefix (up to $t_2 - 1$) will be maintained for future steps, we directly force the later plans to be compatible with the prefix. Since we know that an optimal plan exists that satisfies this requirement following the search process, this would not affect our solution for online explanation.

The recursive search algorithm for model space OEG is presented in Algorithm 1 for finding e_k given E_{k-1}. To search for e_k, we use a recursive model reconciliation procedure on the model space. Given $M^H_{E_{k-1}}$ and M^R, we start off with finding the difference between these two models, and modify M^R with respect to M^H to find the largest set of model changes that can satisfy constraints introduced in Eq. (1). This algorithm continues until the human's plan matches with that of the robot's plan.

B. *OEG for matching Next Action (OEG-NA)*

Throughout OEG-PP, we assume that generating explanations would modify M^H, and the goal of explanation generation is to ensure that the robot and human plan have the same prefix at any step of plan execution. However, this is not always required since the human may not be interested in actions that occurred. Hence, we relax earlier than the current action the plan prefix condition, such that the robot needs only to reconcile between M^R and M^H to match the very next action in $\pi^*_{I,G}$ and $\pi^H_{E_{k-1}}$ at step t_k, regardless of the earlier actions in the plan prefix. This approach is also motivated by the fact that the human is

Algorithm 1: OEG-PP Algorithm

```
input : $M^H_{E_{k-1}}$, $M^R$, $\pi^*_{I,G}$ and $\{e_{k-1}, t_k - 1\}$
output: Sub-explanation $e_k$
Compute $\Delta(M^H_{E_{k-1}}, M^R)$ as the difference between
  the two models;
Sort $\Delta(M^H_{E_{k-1}}, M^R)$ ascending based on the size of
  the model changes;
Compute $\pi_H$ based on $M^H_{E_{k-1}}$ with prefix set up to
  $t_k - 1$;
$t_k \leftarrow$ FirstDiff($\pi^*_{I,G}$, $\pi_H$);
▷ The first plan difference between
  $\pi^*_{I,G}$ and $\pi_H$
LONGESTMONOTONIC($M^H_{E_{k-1}}, t_k, \Delta$)
if ($\pi^*_{I,G} \equiv \pi_H$) then
  return {};
for $\forall f \in \Gamma(M^R)\backslash\Gamma(M^H_{E_{k-1}})$ do
  ▷ All remaining differences after
    sub-explanations $E_{k-1}$
  $\lambda \leftarrow \Gamma(\overline{M^H_f})$ ;   ▷ create a modification
  if $\pi^H_{E_{k-1}} \equiv \pi^*_{I,G}$ then
    return $\lambda$;
  Create a plan $\pi^f_H$ using $(\overline{M^H_f})$;
  if ($t_k \leq$ FirstDiff($\pi^f_H$, $\pi^*_{I,G}$)) then
    if $|\lambda| > \lambda_{max}$ then
      $\lambda_{max} \leftarrow \lambda$;
      $\Delta(M^H_{E_{k-1}}, M^R) -= \lambda_{max}$ ;
      Sort ($\Delta(M^H_{E_{k-1}}, M^R)$) ascending ;
      LONGESTMONOTONIC($M^H_{E_{k-1}}, t_k, \Delta$) ;
  return $\lambda_{max}$ as $e_k$;
```

known to have limited cognitive memory span [21]. In the most limited case, the agent focuses on explaining the very next action that is different between the most recent human plan $\pi^H_{E_{k-1}}$ and $\pi^*_{I,G}$. Similar to Algorithm 1, We perform a recursive model reconciliation procedure on the model space. Compared to other two approaches, first, we perform the search from $\widehat{M^H} \backslash M^H$ rather than M^R (see Figure 2) since it is computationally faster due to the fact that the plan prefixes do not need to be identical and since the search procedure is monotonic, the search result would be equivalent as if the procedure started from M^R. The other difference here is that we do not compare the entire plan prefix. Instead, the agent explains only the immediate next action that does not match in the human and robot plans that, without requiring the explanation also maintains the match between the prefixes. In this aspect, the search process of OEG is similar to that of minimally monotonically explanation (MME) in [7], except that the process must be executed multiple times for OEG due to its online fashion. In the implementations, however, our algorithms actually combine search from M^H and M^R for a better performance, given the fact that latter model updates do not often affect the previous sub-explanations:

$$\begin{aligned} &\forall k > 1, \forall t_k - 1 \leq t < t_k, a_k \in \pi^*_{I,G}[t] \ \& \ a_k \in \pi^H_{E_{k-1}}[t] \\ &\qquad \& \ \Gamma(M^H_{E_{k-1}}) = \Gamma(M^H) \cup E_{k-1} \\ &\qquad\qquad s.t. \\ &\qquad\qquad \bigcup_i e_i = \Gamma(\widehat{M^H}) \backslash \Gamma(M^H) \subseteq \Gamma(M^R) \end{aligned} \tag{2}$$

C. OEG for matching Any Prefix (OEG-AP)

One assumption in the OEG-PP approach is that the robot has only right plan. Subsequently, the robot's goal is to reconcile the human's plan with respect to its own plan using model space search. We relax this assumption by assuming that there is a set of optimal plans. In such a setting, the robot does not need to explain as long as there exists a human plan that has the same prefix as the robot's plan earlier than the current action. The goal of OEG here is thus to satisfy the following:

$$\begin{aligned} &\exists \pi^H_{E_{k-1}} \in \Pi^H_{E_{k-1}} \\ &\forall k > 1, \textit{Prefix}(\pi^*_{I,G}, t_k - 1) = \textit{Prefix}(\pi^H_{E_{k-1}}, t_k - 1) \\ &\qquad \Gamma(M^H_{E_{k-1}}) = \Gamma(M^H) \cup E_{k-1} \end{aligned} \tag{3}$$

where $\Pi^H_{E_{k-1}}$ is a set of optimal plans generated using $M^H_{E_{k-1}}$, $\pi^H_{E_{k-1}}$ is the human optimal plan generated from $M^H_{E_{k-1}}$ and $\pi^*_{M_H}$ is the human optimal plan generated from the original human model (M^H). A straightforward solution to OEG-AP is to generate all human optimal plans and check if any one of them matches with the robot's plan (prefix). This approach however is computationally expensive. Instead, we implemented a compilation approach. To check that a plan prefix *Prefix*$(\pi^*_{I,G}, t_k - 1)$ in the robot's plan is also a prefix in the human's model, we first compile the problem in the human's model into a new problem such that the robot's plan prefix would always be a prefix of the human's plan. If the cost of the human's optimal plan in this new domain model is equal to the cost of the human's optimal plan before the compilation, then clearly there exists an optimal plan in the human's model that matches the prefix. Otherwise, we know that an explanation must be made. Hence, the key here is to ensure that a plan prefix is always satisfied in the compiled model.

This is not difficult to achieve. For all $i \geq 1$, such that $a_i, a_{i+1} \in$ *Prefix*$(\pi^*_{I,G}, t_k - 1)$, where a_i, a_{i+1} are two consecutive actions in $\pi^*_{I,G}$, the compilation can be achieved by adding a predicate p_i to a_i as an effect, which is a prerequisite for a_{i+1}. a_{i+1}, in its turn deletes p_i and adds p_{i+1} which is a prerequisite for a_{i+2}, etc.

To search for e_k, we again use a recursive model reconciliation process on the model space, similar to Algorihm 1. Similar to IV-A, we start off with finding the difference between these two models. The main difference in this approach is that after each model update after a sub-explanation, the agent checks if there exists a human optimal plan that has the same plan prefix as the robot's plan up until the next action using the compilation approach described above. This check stops when such a plan does not exist and a new sub-explanations must be identified by model space search. This process continues until an optimal human plan exists that matches the robot's plan. Note however that this does not mean that an optimal planner would necessary return the same plan using the human's model.

V. Evaluation

We evaluated our approach for online explanation generation both with human subjects and in simulation for the different approaches introduced above and compared the results with Minimally Complete Explanation (MCE) [7] approach. For simulation, the goal is to see that how online explanation is in general different from MCE in terms of the information needed and computation time. We evaluated our approach on ten different problems across the rover domain and barman domain–two standard IPC domain described below. For both human and simulation evaluations, the differences between M^H and M^R are made by randomly removing preconditions from an arbitrarily chosen set of model features. For human subject study, the aim is to confirm the benefits of online explanation generation. Our hypothesis is as follows:

- *Online explanation generation will reduce mental workload and improve task performance.*

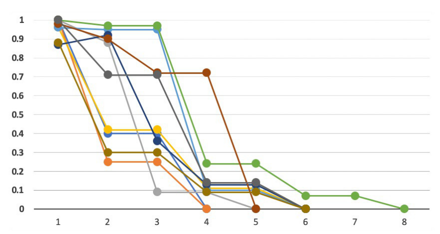
We evaluated our approach with human subjects on a modified rover domain (see Sec. V-D).

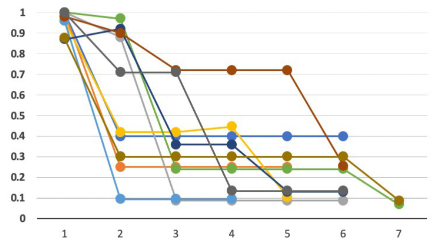
A. Rover Domain

In this domain, the rover is supposedly on Mars and the goal is to explore the space to take rock and soil samples as well as taking images and communicate the results after analysis to the base station via the lander. In order to take any image, the rover must first calibrate its camera with respect to the target. To sample rock or soil, the robot must have an empty space in its storage. At any point of time, the rover only has enough space to store one sample. In order to take

Problem	OEG-PP		OEG-NA			OEG-AP			MCE	
	Explanations	Time	Explanations	Distance	Time	Explanations	Distance	Time	Explanations	Time
Rover										
P1	3 (1.5)	8.89	7 (1.167)	0.4	17.929	2 (1)	0.4	6.94	3	28.91
P2	5 (1.67)	22.32	7 (1.4)	0.105	42.568	3 (1)	0.105	18.30	5	150.54
P3	6 (1.5)	18.68	8 (1.143)	0.068	21.258	3 (1)	0.068	1.64	5	176.16
P4	6 (1.5)	50.97	8 (1.33)	0.131	94.783	5 (1.25)	0.131	45.36	6	314.15
P5	5 (1.67)	54.83	8 (1.33)	0.135	106.709	3 (2)	0.135	50.36	4	272.76
Barman										
P1	5 (1.25)	43.01	5 (1.25)	0.911	59.912	2 (1)	0.943	24.37	5	179.95
P2	5 (1)	36.17	5 (1)	0.995	33.032	3 (1)	0.899	9.36	5	38.89
P3	5 (1.25)	36.83	5 (1)	0.895	46.775	3 (1.5)	0.705	9.67	5	51.84
P4	5 (1.25)	78.42	5 (1)	0.838	69.016	4 (1)	0.556	20.42	5	61.86
P5	5 (1.67)	41.88	5 (1)	0.892	54.708	3 (2)	0.556	10.15	5	61.48

TABLE I: Comparison of number of generated explanations and computation time using different approaches for IPC Rover and Barman Domains.


(a) OEG-PP


(b) OEG-NA

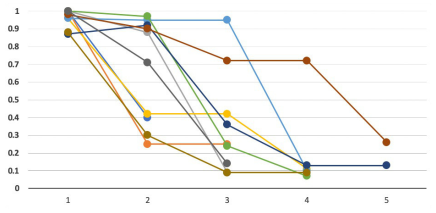

(c) OEG-AP

Fig. 3: Plan distance [22] convergence across three different approaches between $\pi^H_{E_{k-1}}$ and π^R for the Rover domain problems. The y-axis represents the distances while x-axis represents the number of E_k(sub-explanations).

multiple samples, it must drop the current sample before taking another sample [23].

B. Barman Domain

In this domain, the robot assumes the role of a barman whose goal is to serve a desired set of drinks using drink dispensers, glasses and a shaker. The constraints are that the robot can grab one object if its hand is empty, the robot can grab one object with one hand, and before filling it with a drink, a glass should be empty and clean [23].

C. Simulation Results

Table I shows the simulation results comparing minimally complete explanations (MCE) withx OEG-PP, OEG-NA and OEG-AP approaches for 5 problems in the rover domain and 5 problems in the barman domain. While the average number of model features of OEG (in a sub-explanation) being shared at each instance of time is considerably lower that MCE (every feature in the explanation is presented at once), the total number of model features in an explanation are the same for MCE and OEG-PP across most of the problems. We can see that in some cases (for instance, P3 from the Rover domain), the total number of model features in the explanation for OEG-PP and OEG-NA is more than that of MCE, which is expected since OEG is focused on generating the minimal amount of information at each time step, instead of the amount overall. The reason for sharing more information in total in OEG-PP and OEG-NA, when compared to MCE, lies in the dependence between the features and the behavior of the planner (i.e., which optimal is returned). While OEG-AP seems to have improved over the amount of information in an explanation, it actually only shows the advantage of considering all optimal plans instead of the one returned by the planner.

Comparing both the OEG-NA and OEG-AP approaches with MCE and OEG-PP, there is a remaining distance between the robot's plan and the human's plan in terms of plan action distance (also returned by an optimal planner). The distance of OEG-NA is due to the fact that only the immediate next action is considered. For OEG-AP, as we explained, there is no guarantee that the plan returned using the human's model will be the same as the robot's plan since it considers all optimal human plans and only requires one of them to match the robot's. This is also illustrated more clearly in Fig. 3. Furthermore, ion OEG approaches, since the execution and explanation is intertwined, the plan distance [22] between $\pi^H_{E_{k-1}}$ and π^R in our approaches gradually moves towards 0 as shown, which suggests a "smoother" adjustment for M^H during the execution. This is expected to have a positive effect on the human's mental workload, which we evaluated next.

Table I also presents the time comparison between different approaches. For computation time, the results are collected using a 2015 Mac book Pro, with 2.2 GHz Intel Core i7 and 16 GB of memory. The results of the time comparison suggest that OEG-PP is faster than MCE. Moreover, OEG-NA seems the slowest while OEG-AP is the fastest since it uses fewer model features. The performance improvement over MCE may be surprising, thanks to combining search from M^R and M^H. In our implementation, the possible model updates are sorted ascending based on their feature size and our algorithms start checking the ones with the smallest changes from the robot's side. The consistency check is left as we proceed to the next sub-explanation and backtracking is performed when it fails. This search process takes advantage of the fact that latter information often does not affect the previous sub-explanations.

D. Human Study

To test our hypothesis, we designed a human study to compare our three approaches for online explanation generation with minimally complete explanation (MCE) [7]. Furthermore, to ensure that the performance difference is not solely due to simply breaking information into multiple pieces, we also implement another approach that randomly breaks MCE during plan execution (referred to as MCE-R). We conducted our experiment using Amazon Mechanical Turk (MTurk) with 3D simulation. The subjects were given an introduction to the rover domain and the task they were supposed to help with. Each subject was given a 30-minute limit to finish the task. Explanations were provided using plain English language and rover actions were depicted using GIF images from a 3D simulated scenario as the rover executes the plan. Figure 4 shows the 3D simulated scenario presented to the subjects. In this experiment, the human subject acts as the rover's commander, where the robot is on Mars and supposed to perform a mission autonomously. The human subject observes the rover's plan sequentially and is asked to determine whether the rover's current action is questionable or not, with explanations provided by OEG approaches or MCEs. Each subject can only perform the task for one setting to reduce the influence between different runs. To observe the effect on mental workload more clearly, we have also added a few spatial puzzles to the experiment as a secondary task to create additional cognitive demand.

In the scenario, we deliberately remove certain information from the domain so that the subject would create an incorrect plan, when no explanation is given. In particular, we did not inform them that the storage is limited, the memory is limited, the camera must be calibrated, and the camera must be calibrated with respect to the objective. This hidden information introduces differences between M^H and M^R in the model reconciliation setting, and hence resulting in scenarios where explanations must be provided. In this scenario, for example, the subject may question the action for calibrating the camera if they were not specifically told to consider that.

In MCE setting, the robot shares all the information at the beginning of the task [7], while the information is randomly broken to be communicated at different steps in MCE-R. In each of the OEG setting, the robot uses different approaches of online explanation generation, which intertwines the communication of explanation with the plan execution. In particular, the four pieces of missing information are provided to the subjects at different steps. In all settings, the subjects were asked to determine whether the robot's action makes sense or not at a time. The minimally complete explanations are generated based on [7] and online explanations are generated using approaches introduced above.

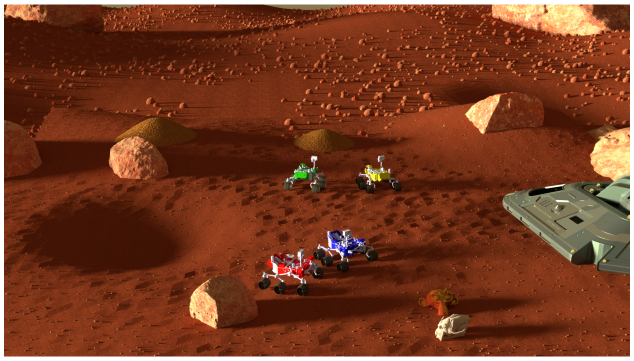
At the end of the study, the subjects were provided the NASA Task Load standard questionnaire to evaluate the efficiency of different explanation approaches by NASA Task Load Index (TLX) [24]. The NASA TLX is a subjective workload assessment tool to evaluate human-machine interface systems. Mental workload is a multidimensional variable which can be captured by different variables and NASA TLX is one of the most frequently used subjective measurements for capturing different aspect of mental workload [25]. It calculates an overall mental workload score using a weighted average on sub-scales: mental demand, physical demand, temporal demand, performance, effort and frustration. Since our experiment does not involve physical demand, we did not include the corresponding question. The description of questions used for each category is presented as follows:

- *Mental Demand*: How mentally demanding was the task?
- *Temporal Demand*: How hurried or rushed was the pace of the task?
- *Performance*: How successful were you in accomplishing what you were asked to do?
- *Effort*: How hard did you have to work to accomplish your level of performance?
- *Frustration*: How insecure, discouraged, irritated, stressed, and annoyed were you?

E. Human Study Results

We created the academic survey using Qualtrics and recruited 150 human subjects on MTurk, 30 subjects for each setting. To improve the quality of the responses, we set the criteria that the worker's HIT acceptance rate must be greater than 98%. After sifting out invalid responses (i.e., failing to identify the two purposely inserted random actions), we had 94 valid responses in total: 19 for each of MCE-R and MCE, 20 for OEG-PP, and 18 for each of OEG-NA and OEG-AP. The age range of subjects was between 18 and 70, and 29.8% of the subjects were female.

We examined how well the human subjects understand the robot's plan given the different explanations, and compared the distances across the five different settings. We compute the distance between the robot's plan and the human's expected plan by the ratio between the number of questionable actions and the total number of actions in a plan. The lower the distance value, the closer the human's plan is to the robot's plan. This metric intuitively captures how much the human subject understands the robot's plan. We calculated the averaged results of each settings over all of the subjects participated in that setting, using subjective questions from


(a) The blue rover moves between waypoints

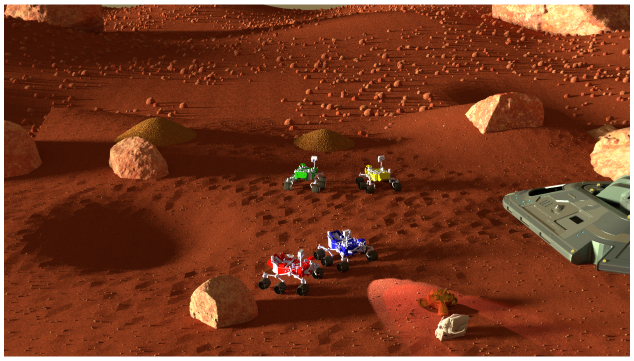
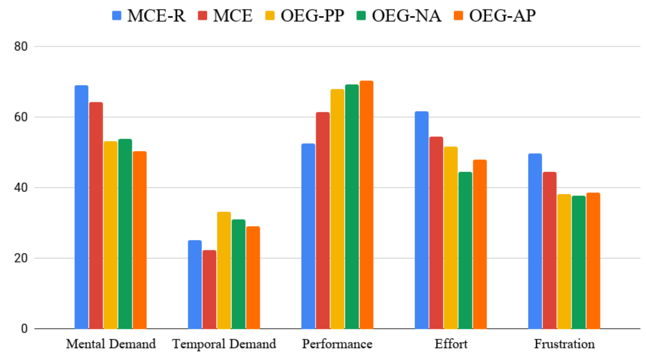
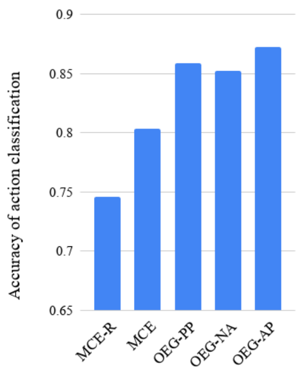

(b) The blue rover takes a picture of one of the objectives

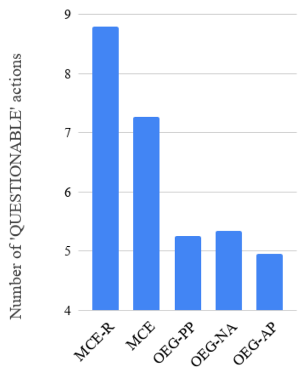
Fig. 4: The 3D visualization of the modified IPC rover domain problem provided to the human subjects. The rovers must together take pictures of targets, collect rock and soil samples, and transmit them to the lander after analysis. The subject views the actions of the rovers via GIF images. (a) and (b) shows the begin and end of an action in which one of the rover takes an image of a target at the bottom.



(a)



(b)



(c)

Fig. 5: (a) Comparison of the results of all of TLX categories for the five settings (b) Accuracy of action classification (c) Number of questionable actions

NASA TLX and objective performance measures such as the number of questionable actions and the accuracy of action classification. Results are shown in Figure 5.

The results overall show that OEG approaches are able to better reduce the human's mental workload than MCE approaches. This is backed up by the fact that OEG approaches resulted in better performance in almost all NASA TLX measures. Due to intertwining the explanation process with the plan execution, the OEG approaches create more temporal demand according to the experiment, which is expected. Figure 5 presents both objective performance measures, and subjective results of the human study amongst the 5 TLX categories. First, the number of questionable actions are significantly lower among OEG approaches when comparing to the MCEs. This indicates that the subjects had more trust towards robots in the OEG cases. Moreover, the accuracy of identifying the correct actions (questionable vs. non-questionable) among OEG approaches are higher. Between the three approaches, OEG-AP has the least number of questionable actions and the most accuracy.

We have also presented the p-value for the mental load based on the subjective measures in Table 6 (with weights 1 for all measures ranging from 0 to 100). The results indicate a statistical significant difference between OEG approaches and MCEs for the mental workload in a pairwise comparison. The overall p-value across five categories is 0.0068 between OEGs (as a group) and MCEs (as a group).

We also did some time analysis. The average overall time taken to accomplish the task for each of the categories is as follows: OEG-NA (567.44s) $<$ OEG-AP (629.56s) $<$ MCE-R (678.98s) $<$ MCE (763.47s) $<$ OEG-PP (775.65s), although we did not see a statistically significant difference due to large variances. The accuracy of the secondary task is also not significantly different between the various approaches.

VI. Conclusion

In this paper, we introduced a novel approach for explanation generation to reduce the mental workload needed for the human to interpret the explanations, throughout a human-robot interaction scheme. The key idea here is to break down a complex explanation into smaller parts and convey them in an online fashion, while intertwined with the plan execution. We take a step further from our prior work by considering not only providing the correct explanations, but also the explanations that are easily understandable. We provided three different approaches each of which focuses on one aspect of explanation generation weaved in plan execution. This

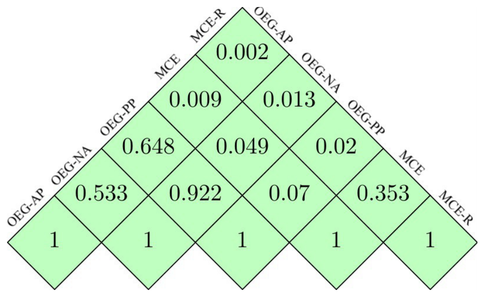


Fig. 6: p-values across different approaches on the mental workload, which is the sum of subjective measures (with weights 1).

is an important step toward achieving explainable AI. We evaluated our approaches using both simulation and human subjects. Results showed that our approaches achieved better task performance while reducing the mental workload.

ACKNOWLEDGMENT

This research is supported in part by the NSF grant IIS-1844524, the NASA grant NNX17AD06G, and the AFOSR grant FA9550-18-1-0067.

REFERENCES

[1] T. Lombrozo, "The structure and function of explanations," *Trends in cognitive sciences*, vol. 10, no. 10, pp. 464–470, 2006.

[2] M. R. Endsley, "Design and evaluation for situation awareness enhancement," in *Proceedings of the Human Factors Society annual meeting*, vol. 32. SAGE Publications Sage CA: Los Angeles, CA, 1988, pp. 97–101.

[3] N. J. Cooke, "Team cognition as interaction," *Current directions in psychological science*, vol. 24, no. 6, pp. 415–419, 2015.

[4] M. Göbelbecker, T. Keller, P. Eyerich, M. Brenner, and B. Nebel, "Coming up with good excuses: What to do when no plan can be found," in *ICAPS*, 2010.

[5] M. Hanheide, M. Göbelbecker, G. S. Horn, A. Pronobis, K. Sjöö, A. Aydemir, P. Jensfelt, C. Gretton, R. Dearden, M. Janicek *et al.*, "Robot task planning and explanation in open and uncertain worlds," *Artificial Intelligence*, vol. 247, pp. 119–150, 2017.

[6] S. Sohrabi, J. A. Baier, and S. A. McIlraith, "Preferred explanations: Theory and generation via planning," in *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[7] T. Chakraborti, S. Sreedharan, Y. Zhang, and S. Kambhampati, "Plan explanations as model reconciliation: Moving beyond explanation as soliloquy," *arXiv preprint arXiv:1701.08317*, 2017.

[8] Y. Zhang, S. Sreedharan, A. Kulkarni, T. Chakraborti, H. H. Zhuo, and S. Kambhampati, "Plan explicability and predictability for robot task planning," in *ICRA*. IEEE, 2017, pp. 1313–1320.

[9] D. Gunning, "Explainable artificial intelligence (xai)," *Defense Advanced Research Projects Agency (DARPA), nd Web*, 2017.

[10] P. Langley, B. Meadows, M. Sridharan, and D. Choi, "Explainable agency for intelligent autonomous systems," in *Twenty-Ninth IAAI Conference*, 2017.

[11] T. Chakraborti, S. Kambhampati, M. Scheutz, and Y. Zhang, "Ai challenges in human-robot cognitive teaming," *arXiv preprint arXiv:1707.04775*, 2017.

[12] A. D. Dragan, K. C. Lee, and S. S. Srinivasa, "Legibility and predictability of robot motion," in *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*. IEEE Press, 2013, pp. 301–308.

[13] M. Zakershahrak, A. Sonawane, Z. Gong, and Y. Zhang, "Interactive plan explicability in human-robot teaming," in *RO-MAN*. IEEE, 2018, pp. 1012–1017.

[14] M. Fox, D. Long, and D. Magazzeni, "Explainable planning," *arXiv preprint arXiv:1709.10256*, 2017.

[15] S. Reddy, A. Dragan, and S. Levine, "Where do you think you're going?: Inferring beliefs about dynamics from behavior," in *NeurIPS*, 2018, pp. 1461–1472.

[16] Z. Gong and Y. Zhang, "Robot signaling its intentions in human-robot teaming," in *HRI Workshop on Explainable Robotic Systems*, 2018.

[17] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artificial Intelligence*, 2018.

[18] Y. Zhang and M. Zakershahrak, "Progressive explanation generation for human-robot teaming," *arXiv preprint arXiv:1902.00604*, 2019.

[19] M. Fox and D. Long, "Pddl2. 1: An extension to pddl for expressing temporal planning domains," *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.

[20] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[21] F. Paas, A. Renkl, and J. Sweller, "Cognitive load theory and instructional design: Recent developments," *Educational psychologist*, vol. 38, no. 1, pp. 1–4, 2003.

[22] A. Kulkarni, Y. Zha, T. Chakraborti, S. G. Vadlamudi, Y. Zhang, and S. Kambhampati, "Explicablility as minimizing distance from expected behavior," *arXiv preprint arXiv:1611.05497*, 2016.

[23] IPC, "Competition domains for international planning committee," http://www.plg.inf.uc3m.es/ipc2011-learning/Domains.html, Jul. 2019.

[24] NASA, "Nasa task load index," https://humansystems.arc.nasa.gov/groups/TLX/, Feb. 2019.

[25] P. S. Tsang and V. L. Velazquez, "Diagnosticity and multidimensional subjective workload ratings," *Ergonomics*, vol. 39, no. 3, pp. 358–381, 1996.

Feature-directed Active Learning for Learning User Preferences

Sriram Gopalakrishnan, Utkarsh Soni, Subbarao Kambhampati
Arizona State University

Abstract

Learning preferences of users over plan traces can be a challenging task given a large number of features and limited queries that we can ask a single user. Additionally, the preference function itself can be quite convoluted and non-linear. Our approach uses feature-directed active learning to gather the necessary information about plan trace preferences. This data is used to train a simple feedforward neural network to learn preferences over the sequential data. We evaluate the impact of active learning on the number of traces that are needed to train a model that is accurate and interpretable. This evaluation is done by comparing the aforementioned feedforward network to a more complex neural network model that uses LSTMs and is trained with a larger dataset without active learning.

Introduction

When we have a human-in-the-loop during planning, learning that person's preferences over plan traces becomes an important problem. These preferences can be used to choose a plan from amongst a set of plans that are comparable by the planner's cost metrics. Such a plan would naturally be more desired by the human. The user may not like to constantly dictate their preferences, and may not always be in the loop during execution. Thus, it is important for the user's preference function to be learned well, and for the user to be able to verify them. For verification, there ought to be a way to interpret how the model's decisions were made, and verify how faithful the learned model is to the user's preferences.

A user's preferences function may be quite complex with dependencies over different subsets of features. The utility of some features maybe non-linear as well. Such a preference function may require a fair amount of information to approximate. We cannot expect a single user to give feedback over a large set of traces to get the relevant information. So Active learning, with a sufficiently expressive user interface for feedback, is essential to minimize queries and redundant information.

In this work, our objective was to model the user's preferences over plan traces. There do exist techniques that efficiently represent and reason about preference relationships. CP-nets (Boutilier et al. 2004) and Generalized additive independence(Braziunas and Boutilier 2006) models are typically used to represent preferences over sets of variables without consideration to the order in which they appear. While these models can be adapted to handle sequential data, they are not intended for it. LTL rules, however, can capture trajectory preferences very well and are used in PDDL 3.0 (Gerevini and Long 2005), and LPP (Bienvenu, Fritz, and McIlraith 2006). However, it can be very hard for a user to express their preferences in this form. We discuss existing approaches in more detail and the differences with respect to our work under the related work section.

In our approach to learning preferences, we want to efficiently identify the relevant features and the degree to which they affect the preference score of a plan. We thus employ a feature-directed active learning approach that specifically picks plan traces that are most informative about the feature's effects on preference. After active learning, we encode a plan trace in terms of the relevant features it contains. We gather a set of training data from active learning, along with the user's preference score to help train a simple Neural Network (NN) that we call the *FeatureNN* model. We use a Neural Network as they can approximate complex functions to a good degree. Our approach is in one way, related to Generalized Additive Independence in that we try to learn a utility function over pertinent features, but we do not explicitly define or restrict the form of any utility functions. Rather a simple one hidden-layer feed-forward neural network learns the functions, dependencies, and relative weights over the relevant features. The *FeatureNN* then predicts a preference score for each plan reflecting the user's preferences. We also compare the performance of the *FeatureNN* to another *SequenceNN* model that processes sequential data using an LSTM(Schmidhuber and Hochreiter 1997) module. The *SequenceNN* is not trained with data from active learning, but with a larger dataset of traces with ratings. This is to evaluate how efficient our active learning approach is with respect to the number of traces. Specifically, we compare the number of traces required by *SequenceNN* and *FeatureNN* for the same accuracy and interpretability.

Neural networks, unlike linear functions, are not as easy to interpret. Even simple NN with a single hidden layer can be a challenge. We help the user interpret the decisions of the neural network by showing how the preference score is affected by removing different features of a plan trace. This is similar to using Saliency Maps (Simonyan, Vedaldi, and Zisserman 2013) in images to explain what parts of the image contributed to the classification. In this way, we can explain to the user what plan features contributed to the preference value and by how much. The difference in preference score should correspond to the user's expectations as per their preference model. The more similar the effect of changes are to the user's preferences, the more interpretable the NN model is to the user as it approximates well their own preference function. Such a method of explaining a decision(score) is also related to explaining using counterfactuals (Miller 2018). Here the counterfactual is the plan trace without a specific feature. Additionally, when the specific features used to compute preferences comes from the user's feedback (during active learning), this interpretability is obviously improved.

We present our work by first defining the problem before delving into the methodology of our approach. In the Methodology section, we discuss the domain used, the user preference model, and the feature-directed active learning process. We also discuss the two neural network models used to learn the preference model, viz. the *FeatureNN* and the *SequenceNN* models. Then we present our experimental results in which we compare the two models with respect to their accuracy in predicting the preference score, as well as interpretability. Lastly, we discuss the results and possible extensions to the work.

Problem Definition

Given a Domain D, with a set of features F, and a planner P, the problem is to learn the preference function $F_p()$ that captures the user's preference model $U_p()$ and scores traces accordingly. The types of preferences we learn in this work are a function of the feature set F of the domain and not hidden variables or action costs. The user U is available to rate a plan trace on its preference, and annotate what features contributed positively or negatively to the rating. Features can be categorical or cardinal(count), and involve sequences.

Plans are rated between $[0, 1]$ with higher values indicating a greater preference. If there are no features in the plan that either contributed positively or negatively to the preference, then the preference score is 0.5.

An equivalent problem formulation assumes that instead of a domain D and planner P, we are given as input a large enough set of plan traces B (backlog of traces) over a relevant set of initial and goal states. We assume that this set of plans covers the space of possible preferences that the user might have.

Methodology

For our experiments, we chose to use gridworld with features that any human can relate to. We chose gridworld as it is easy to quickly generate many diverse plans that cover the range of features.

Given the domain and a task, we go through r rounds of active learning. Each round comprises of r_t traces. Both r and r_t are hyperparameters. For our experiments, we set the number of rounds at $r = 3$. After acquiring the data, we train the NN model and test it on a hidden set of traces. We now go over different parts of our methodology in detail.

Domain

The objective in our gridworld domain, which we call $Journey - World$, is to travel from home to the campsite which is shown in the grid in Figure 1(a). Each step of a plan corresponds to a cell of the grid. While some cells are empty, there a lot of cells that have features. These features can be eateries(like a coffee shop, restaurant), landmarks(natural history museum) or activities(visiting the library, watching a movie). The user can move through any of these states before reaching the campsite. A majority of the cells also contain landscapes like mountains, lake, sea or industries. The user is not allowed to move through $Landscape$ cells. Moving through cells adjacent to $Landscape$ cells corresponds to seeing the landscape along the journey. For example, if a step in the plan goes through a cell which is adjacent to a lake, this corresponds to the plan going through a state where the user passed by a lake.

All non-landscape features (like coffee, donut) are binary features in a plan trace i.e. the user has either visited one or not. On the other hand, the landscape features are cardinal, i.e. we count the number of such landscape features in the plan trace. We assume that the count of cardinal features can make a difference in the preference score. In total, there are 13 features in $Journey - World$.

We had designed $Journey - World$ with simple and commonly understood features to make it easier for subsequent human-studies. We assume people will have preferences over these features.

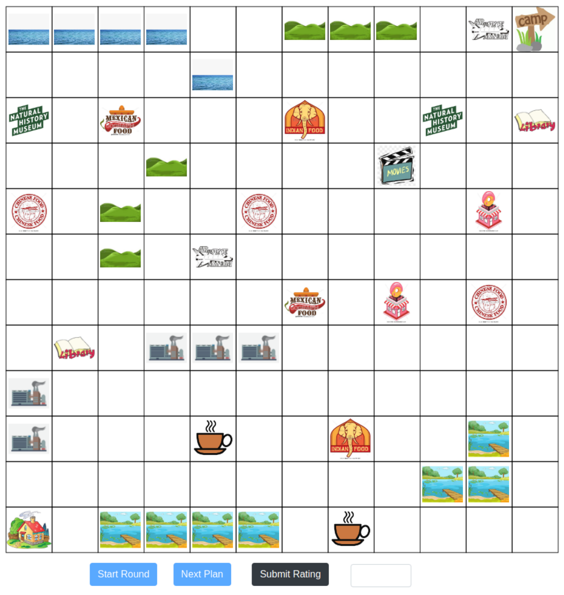
User Preference Model

For our current experiments, we chose to use a completely defined user preference model to represent the user. This made it easier for us to test and debug our methodology. In future extensions of this work, we will include evaluations with human trials. The user's preference model is defined as follows

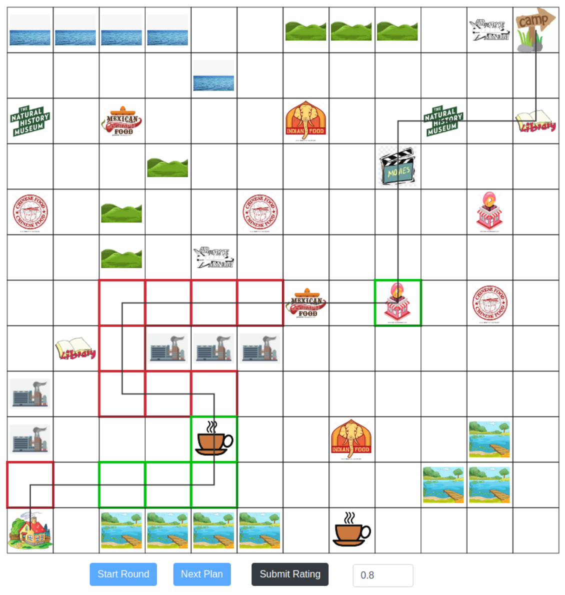
$$P(trace) = \begin{cases} 0.5 + 0.1 * (C) - 0.3 * (D) + 0.1 * n(L) \\ +0.1 * n(I) \qquad if\ not(C\ and\ D) \\ 0.5 + 0.3 * CD + 0.1 * n(L) + 0.1 * n(I) \\ \qquad\qquad if\ (C\ and\ D) \end{cases}$$

$$where\ n(x) = min(x, 2)\ , C \in \{0, 1\},\ D \in \{0, 1\}$$
$$and\ L, I \in \mathbb{N}$$

C is a binary variable that is 1 when the plan trace has a coffee. D is also a binary variable and represents a Donut. CD represents a binary variable set to true when the plan trace a coffee and a donut. When CD is true, C and D are false and this dependency affects the preference score computation as shown in the preceding equation. L and I represent



(a) Problem domain



(b) Rated and annotated plan trace

Figure 1: (a) Problem domain. The task is to go from home(lower left corner) to the camp site(upper right corner) (b) A plan trace that has been rated 0.8 by the user. The user has also provided annotations: green for liked features, and red for disliked features.

the number of lake and industry regions respectively. These are cardinal features, and the preference of the plan increases based on their count, up to 2, and then stops increasing. The function itself, while simple to understand, is non-linear and hidden underneath a large hypothesis space of functions that could be learned in the domain, over a larger set of features (13 in total for the $Journey - World$ domain). In our experiments, we programmed a separate module to rate and annotate plan traces based on an input preference function like the one described previously. This synthetic human is what rates and annotates in the active learning process that we will describe shortly. Using a synthetic human helped speed up the testing and debugging process, and gives us a baseline noiseless scenario to test against.

User Interface

The current user interface(Figure 1(a)) for $Journey - World$ displays the entire grid. Icons are used to show the features present in cells. The plans for each round of active learning are then shown one at a time. The plan steps are visualized as a line going from the home to the campsite. The user has to input a rating to indicate their preference for the plan based on the features that are visited. They can also annotate features of the plan that they like(green) or dislike(red) as shown in Figure 1(b). The user can click on the $Next\ Plan$ button to then move on to the next plan. The interface automatically switches to the next round of Active Learning when the current round's plans have all been rated.

Feature-Directed Active Learning

In our active learning process, we go through multiple (r) rounds of feedback. Each successive round utilizes the knowledge from previous rounds to select the most informative queries. In the first round, the user is shown the most diverse set of plan trajectories that were generated for the domain. We choose diverse plans because in the first round we do not have any knowledge of what features might affect the user's preference and hence, we want to cover the feature space as much as possible in $r_t = 30$ traces. In order to get the required diverse plan set, we first generate a large number of plan traces(10000 plans) over a user specified set of initial and goal states that we refer to as the backlog of plans, B. In our current experiments we only have one initial and one goal state. We are easily able to generate such a large backlog of plans because it is a type of gridworld domain. We did not want the computational cost of diverse plan generation to hamper the work. This is a computational cost that needs to be considered when working with other domains. The plans generated cover the entire feature space of the $Journey - World$ domain. We then select the 30 most diverse plans within the set of backlog plans for the first round. We will now discuss how this is determined.

The diversity score between any two plans p_a and p_b is denoted by $d(p_a, p_b)$. The diversity is based on the sum of feature count differences for features $f \in F$ that are present p_a and p_b. For a particular feature f, we compute the feature count difference f_Δ. Rather than use the difference in count per feature, we use a geometric series sum as computed in Equation 1. The first count in the difference contributes 1, the second count contributes a 0.5, the third contributes 0.25 and so forth. So the count difference for a single feature contributes to at most 2 to the diversity computation. This avoids any single feature from dominating the diversity computation.

The diversity between two traces is computed as the average $f_{E\Delta}$(Equation 2) over all the features in the domain. Finally, we calculate the backlog-diversity d^B for a plan p using equation 3. The backlog-diversity is the average pairwise diversity over every other plan in the backlog. Using this diversity score, we select the top r_t plans ($r_t = 30$ in our experiments) for the user feedback.

$$f_{E\Delta} = \frac{(1 - r^{f_\Delta})}{1 - r} \tag{1}$$

$$d(p_a, p_b) = \sum_{f \in F} f_{E\Delta}/|F| \tag{2}$$

$$d^B(p) = \frac{\sum_{p' \in B} d(p, p')}{|B|} \tag{3}$$

After the first round of diverse plans, we then make use of the ratings and annotations provided by the user in the first round to generate the most informative plan traces for the subsequent rounds. Given our acquired knowledge of relevant features from the previous round, our objective now is to figure out the effects and dependencies between these features. We also want to select traces for the next round that are more likely to be rated either significantly higher or lower. This region of data is typically harder to get as we expect most data to be closer to the average. In order to estimate which plan traces would be either most preferred or least preferred, we use a fast $weak\_predictor$ that predicts the rating of any arbitrary plan p given prior knowledge. We need the predictor to be fast as we have to give traces or queries for the next round in a short amount of time.

The $weak\_predictor$ estimates a value for each feature based on the prior annotated data. It can then estimate the score of an unrated plan as just the sum of the features present in it. The value of each feature is scored using a quick and simple method. First, for each scored plan trace p with rating r_p, the feature f is given a score f^p_{score} for that plan by equation 4. Then the feature's score, f_{score}, is computed as the average f^p_{score} over all plans that the feature appears in. Then to predict the score for an unrated plan, the $weak\_predictor$ assigns a score $predict(p)$ which is the sum of f_{score} for all features present in the plan.

$$f^p_{score} = \begin{cases} r_p \quad if\ f\ annotated\ as\ "liked" \\ -(1 - r_p) \quad if\ f\ annotated\ as\ "disliked" \end{cases} \tag{4}$$

In addition wanting plans that are likely to be rated much higher or much lower, we also want the next round traces to have two more properties. We still want to include some diversity in the plan traces with respect to the overall backlog of traces to uncover features that we might have missed in the first round. Additionally, we want to maintain some similarity in traces between the rounds. We think that the similarity between plans reduces the cognitive load on the user as they need not parse wholly different traces. Given a plan p, we denote its similarity to the already scored traces as $S(p)$.

Finally, we assign a combined weighted score of p_c to all the plans in the backlog given by equation 5. The top $r_t = 30$ plans are then picked for the next round, and in this way the active learning proceeds for r rounds. For our experiments r is 3 rounds.

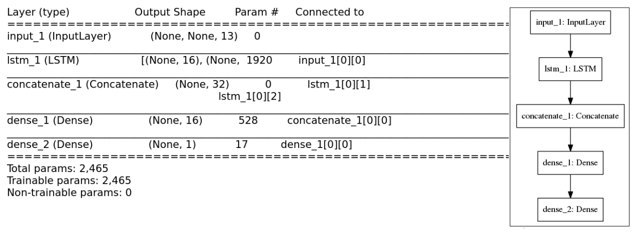
$$p_c = w_1 * predict(p) + w_2 * d^B(p) + w_3 * S(p) \tag{5}$$

Preference Learning using Neural Networks

For learning the preference function we used two models, *SequenceNN* model and *FeatureNN* model.

The *SequenceNN* model uses an LSTM module in it. We considered an LSTM based model as they are well suited to learning patterns overs sequential data. The input plan trace was encoded such that each step was an encoding over the features of the cell visited at that step. There are 13 features in total, and so each step is a 13-dimensional vector. We do *not* provide or restrict the input to only the features that the user annotated during active learning for the *SequenceNN* model. We wanted to test how easily the model could still figure out the relevant features and learn the preference function well.

The training data for the *SequenceNN* model was a set of rated plan traces. We varied the number of traces given from as small as 30 to 12,000 in increasingly larger step sizes. A plan trace would be a $N \times 13$ array where N is the plan length. We trained the model for 10 epochs with a batch size of 8, a learning rate of 0.01 and using stochastic gradient descent. The *SequenceNN* module in our model has 16 memory cells. After processing the plan trace through the LSTM module, we concatenate the output vector and memory nodes of the LSTM module and pass it through a single fully connected hidden layer, followed by the output layer which outputs the preference score between [0,1]. The model summary is in Figure 2. The idea is that the LSTM module output and memory, at the end of processing the sequence, will have the necessary information related to the sequence for predicting the score.



```
Layer (type)                 Output Shape        Param #     Connected to
==================================================================================
input_1 (InputLayer)          (None, None, 13)     0
__________________________________________________________________________________
lstm_1 (LSTM)               [(None, 16), (None,  1920        input_1[0][0]
__________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 32)          0          lstm_1[0][1]
                                                   lstm_1[0][2]
__________________________________________________________________________________
dense_1 (Dense)               (None, 16)          528        concatenate_1[0][0]
__________________________________________________________________________________
dense_2 (Dense)               (None, 1)          17         dense_1[0][0]
==================================================================================
Total params: 2,465
Trainable params: 2,465
Non-trainable params: 0
```

Figure 2: LSTM Model for Preferences

To test if any variant of the *SequenceNN* model could learn the relevant information better, we also tried varying the size of the model (number of parameters) to make it more powerful. We varied the model from 16 to 64 weights in the memory layer. Results are in the Evaluation and Analysis section.

For the *FeatureNN* model, the input was an encoding of the plan trace that only comprised of the features the user annotated as relevant during active learning. The entire plan trace was summarized into one encoded vector. For example, in the user preference model in our experiments, only 5 features matter to the user. We determine what these features are through active learning, and then defined our *FeatureNN* model accordingly to take a 5-dimensional vector as input. For example, if a plan trace had a step with coffee and steps that passed by 3 lakes, then the values at the corresponding indices are set to 1, and 3 respectively. Note that since coffee is a binary feature, even if two coffee steps were in the plan, it's value in the encoding is only either 1 or 0. As for the model description of *FeatureNN*, it was a simple fully connected neural net with one hidden layer of 4 dimensions and one output layer. The model summary is in Figure 3. Note that 4 dimensions or nodes for the hidden layer is not a magic number, and would need to be larger if there were more features. We reduced the number of dimensions for the hidden layer until the results were measurably worse. To train the *FeatureNN* model we vary the number of traces per round r_t from 5 to 50 traces for $r = 3$ rounds. Since the dataset size is very small (smallest is 15 traces), we create 200 duplicates of the data points uniformly and train for 10 epochs. We also shuffle the data and train with a batch size of 8, a learning rate of 0.01, and using stochastic gradient descent.

```
_____________________________________________________________
Layer (type)              Output Shape             Param #
=============================================================
input_1 (InputLayer)        (None, 5)                0
_____________________________________________________________
dense_1 (Dense)             (None, 4)               24
_____________________________________________________________
dense_2 (Dense)             (None, 1)                5
=============================================================
Total params: 29
Trainable params: 29
Non-trainable params: 0
_____________________________________________________________
```

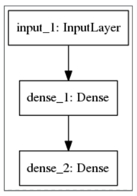


Figure 3: Simple NN Model for Preferences

Both models were tested on an unused test set of 1000 traces for accuracy and interpretability.

Interpretability of the Preference Model

A user can interpret a Neural Network's behavior through analyzing what features are salient to its decision, and by how much. This can be analyzed by adding or removing features and seeing the resultant effect on the predicted score. When done over a set of different traces, the user can intuit what mattered and how much. With this in mind, we compute a measure of interpretability we call the Attribution Error AE. The AE for a feature f of a Plan p is computed as follows:

$$AE(p,f) = |(U_p(Plan) - U_p(Plan - f)) - (F_p(Plan) - F_p(Plan - f))| \quad (6)$$

where $U_p()$ is the preference function of the user (true model of preferences) and $F_p()$ is the learned preference function. AE is simply the difference in the effect of the feature on the preference scores. The overall AE for each test plan p , $AE(p)$, is the average of $AE(p, f)$ for all f present in p. We compute the AE score for the test set as the average over only the top 10% of AE(p) errors. We do this because neural networks can sometimes have enough capacity to memorize many cases and increase accuracy. So it can predict the correct preference score of both of the original trace and modified trace (with dropped feature) by the memory of very similar traces. It would then seem like it's preference function predicts the same way as the ground-truth preference function, but it maybe using unrelated features. Therefore, it is in the failure cases that we get a true measure of its generalization and how faithful it is to the true model of preferences. That is why we use the average over the top 10% of AE(p) errors. These failure cases could correspond to the cases when a rare or unseen pattern of features are input, and thus not memorized.

Evaluation and Analysis

Evaluation of LSTM model

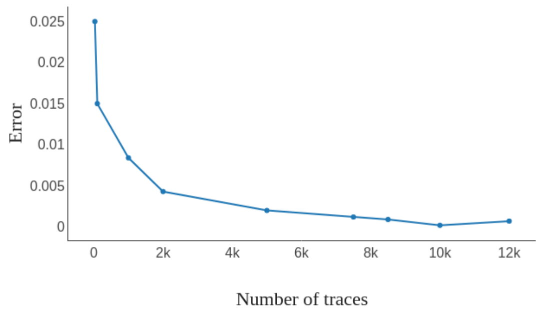
When varying the number of training input traces given to the *SequenceNN* model, we observed that the accuracy improved (error decreased) as expected(Figure 4(a)). Surprisingly, even with 30 traces, it was able to predict with an error of 2.5% over the test set of unseen 1000 traces. We attribute this to the fact that there are enough simple correlations with other features that can predict the score well for the preference function that we tested with. This is evidenced by the fact that the interpretability measure (Attribution Error) is very low for 30 traces(Figure 5 (a)); The attribution error was greater than 0.3 and the value range of AE is [0,1]. Additionally, we give the most diverse N traces for each training set size to the *SequenceNN* model. Diverse traces are more likely to contain relevant information.

The interpretability of the LSTM model was not impressive. The attribution error did decrease over the range of training set sizes, but only as low as 0.09 as shown in Figure 5(a). Given that the preference scores are between $[0, 1]$, this would correspond to a 9 percent error after 7500 rated traces. Needless to say, it is unreasonable to expect a single human to rate 7500 traces.

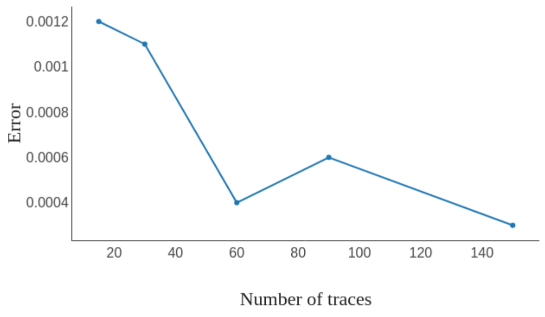
We also tried varying the size of the *SequenceNN* model from 16 to 64 dimensions. This improved accuracy by a minuscule amount (order of $1e-4$), and interpretability did not improve.

Evaluation of Feature-NN model

The performance of the *FeatureNN* model was significantly better both in accuracy(lower error) and interpretability than the *SequenceNN* model as seen in Figure 5. This should come as no surprise since we restrict the input space based on user feedback (knowledge) on relevant features. This also restricts the hypothesis space of functions that the simple feed-forward network could search over. We think this will make it more likely that the NN will find a good and faithful

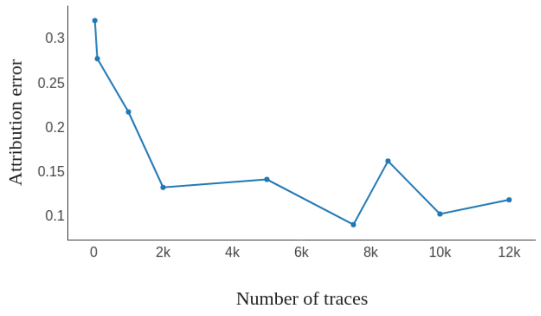


(a) Error using rated traces and a LSTM based learner

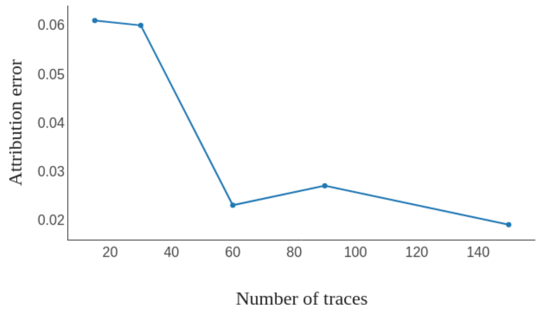


(b) Error with feature-directed active learning and a simple neural network

Figure 4: Comparison of accuracy



(a) Attribution error using rated traces and a LSTM based learner



(b) Attribution error with feature-directed active learning and a simple neural network

Figure 5: Comparison of attribution error

approximation function to the true preference function. This is as opposed to discovering predictive but incorrect correlations.

What is interesting to note is that the interpretability, as measured by the AE error drops to as little as 2.5 % in as little as 60 traces (20 traces per round over 3 rounds) for *FeatureNN* model as shown in Figure 5b. It drops below 2 % with 150 traces.

The *FeatureNN* model with 90 traces is as accurate as the *SequenceNN* with 7500 traces in our problem, with 8% less Attribution Error (more interpretable). While we expected *FeatureNN* to be better, we did not expect such a large difference in efficiency.

Analysis and Discussion

Even with as little as 13 features and a relatively uncomplicated preference function, a sufficiently powerful *SequenceNN* model did not find the underlying preference function. Instead, it found correlations that predicted the preference score to a very high level of accuracy. This, unfortunately, makes the model suffer in interpretability.

As the number of features increases, the hypothesis space of a NN will increase significantly. This makes it much more likely for any NN to find spurious correlations, and suffer in interpretability. So active learning and using a simpler NN becomes very important for learning preferences in plan traces.

As for prior feature knowledge, we assumed knowledge about what features were categorical (binary in our experiments) and what features were cardinal. Rather than assume this knowledge, we can get this from the user as well, and reduce the assumptions about the domain features. Alternatively, we could have just encoded all features as cardinal features, and let the neural network determine what features were categorical. While this is certainly possible, we think it better to get this knowledge from the user and encode the plan trace based on this knowledge. This makes the job of the neural network easier, and less likely to learn spurious correlations.

In our current encoding of features in *FeatureNN* model and our experiments, we have not included a preference dependency that considers the number of steps between features. For example, *I would like to have a donut within 3 plan steps after having a coffee*. This omission was not intentional. One can easily encode such a sequential feature as a variable as well. The number of steps between the two (state) features becomes a cardinal variable to represents this sequential feature.

Related Work

Two well known paradigms for learning, representing and reasoning over preferences are CP-nets and Generalized additive independence (and their variants). Both of them were intended for preferences over outcomes. Each outcome can be comprised of many parts(decisions). One can think of each decision as choosing a value for a variable. The user would have preferences over the possible outcomes, or there maybe a utility (value) associated to each outcome. In CP-nets (Boutilier et al. 2004), the decisions or variables are represented in a graph, and there exists dependencies over variables. The preferences of a variable's values are affected by the value of the parent variables. The CP in CP-nets stands for *Ceteris Paribus* or *"all else being equal"*. Here, the *all else* refers to the parents of the node, and when they are equal, then a particular set of preference orderings for the child variable's values hold. The knowledge of the dependence graph is either known apriori, or can be queried from the user. Once the hierarchy of dependence is known, the user is then queried about preferences at each node. For CP-nets to be used in plan trace preferences, we would have to ask the user what the dependencies are. Then we would have to ask the user for their relative preferences over features, given fixed parent feature values. Note that the variables for plan preferences may also have to incorporate information about order. So there are significantly more variables (features) to consider in sequential data versus unordered data. We think querying for such knowledge is very demanding, and not natural for preferences over plan traces.

We believe it is more natural for the user to specify the relevant conditional dependencies over features while annotating a plan trace. Additionally, we think it easier to give a preference value for the plan trace rather than relative preference orderings over the features in the domain. The features could include sequential dependencies or position-dependent features. We think it would be hard for the user to be able to describe sequential features and the relative ordering over them. Lastly, CP-nets do not compute utility values and some outcomes can be incomparable for a particular network of dependencies. In our problem, we would like a total order over the plans, to select the most preferred plan. So it helps to have a utility/preference value for every plan trace.

On the other hand, GAI (Braziunas and Boutilier 2006) models do provide a single utility value for a set of features. As stated in their work, they provide an additive decomposition of a utility function (into sub-utility functions) in situations where single attributes are not additively independent, but (possibly overlapping) subsets of attributes are. Since the subsets of attributes for the different sub-utility functions can overlap, one must query either with only global queries or a combination of local queries (over the subsets of features) with global queries to calibrate (as was done in the GAI work) (Braziunas and Boutilier 2006)(p. 3). To learn a GAI with active learning from a single user, there are one of two methods. We could make assumptions about what subsets of variables are part of each sub-utility function, and what those functions are, or the user would need to know and give us this information. We think this is a very difficult task for the user. In our approach, we only do full trace queries and ask for annotations and preference ratings. We think it is more natural to ask the user for their overall rating of a plan, rather than how much each subset of features affected the rating. The neural network then handles the job of learning the preference function over the user-specified features (and any dependencies).

The other formalism for specifying preferences are LTL rules (Huth and Ryan 2004) (p.175), which allow the user to specify sequential patterns. Expecting the user to be able to specify LTL rules might be unreasonable. The user would also have to give utilities or preference orderings over the specified LTL rules. One can interpret our interface as extracting a subset of simple LTL rules (through annotations) which are present in a plan trace. The user gives a rating to the trace, as well as what features (LTL rules) were good or bad. Extending the LTL analogy, our encoding of a plan trace can be seen as a vector of the relevant LTL rules. The index corresponding to an LTL rule is set to 1 if the rule is satisfied in the plan trace. However, recall that we also allow cardinal features (counts) in our encoding, and not just binary variables. Our interface and learning framework does not handle the entire gamut of possible LTL rules. We are working on extending the types of sequential preferences supported, while keeping the interface intuitive and expressive.

Conclusion and Future Work

In our approach, we use feature-directed Active Learning complemented with an intuitive and expressive user interface to learn the user's preference function efficiently. The traces obtained during active learning are rated and annotated by the user. These traces are encoded as a vector over the features that the user indicated as relevant to their preferences. The feature vectors are used to train a simple feed-forward Neural Network to learn the preference function. We show that the *SimpleNN* neural network is more accurate and interpretable with fewer, more informative plan traces as compared to the LSTM based *SequenceNN* model. The latter was trained with a larger dataset of rated plan traces without active learning.

Our current experiments use a user preference function over only a few variables. It is important to see how efficiently our framework learns a more complex preference function. Moreover, the current preference function is completely deterministic as it provides consistent annotation and rating to the plan trace. A human, however, might not behave in a consistent manner. We will test with a noisy or probabilistic preference model in future work.

The user interface itself can be extended to include more complex annotations. For example, the user can also provide annotations for some features to be added/dropped from the plan. This is especially useful for cardinal feature as the modified feature count represents what is ideal to the user. For example, if the user's preference doesn't increase after visiting more than 2 *lakes*. Then this can be communicated by removing extra *lake* features from a plan trace.

We have mentioned categorical and cardinal features, but our framework is also intended to support real-valued features. We would need to adapt our active learning process to elicit feedback as to what the minimum, optimum and maximum values of such features are. These would be the minimum essential points to sample for approximating the underlying utility function.

Lastly, we would like to simplify the function by which we choose plan traces in successive rounds of active learning. We think that the similarity with traces from previous rounds is unnecessary, and might not appreciably reduce the cognitive load on the user. We think that just diversity and selecting traces that are much more preferred(closer to 1.0) or much less preferred(closer to 0.0) would be sufficient.

Acknowledgments

This research is supported in part by the ONR grants N00014-16-1-2892, N00014-18-1-2442, N00014-18-1-2840, the AFOSR grant FA9550-18-1-0067, NASA grant NNX17AD06G and JP Morgan faculty research grant.

References

Bienvenu, M.; Fritz, C.; and McIlraith, S. A. 2006. Planning with qualitative temporal preferences. *KR* 6:134–144.

Boutilier, C.; Brafman, R. I.; Domshlak, C.; Hoos, H. H.; and Poole, D. 2004. Cp-nets: A tool for representing and reasoning withconditional ceteris paribus preference statements. *Journal of artificial intelligence research* 21:135–191.

Braziunas, D., and Boutilier, C. 2006. Preference elicitation and generalized additive utility. In *AAAI*, volume 21.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in pddl3. Technical report, Technical Report 2005-08-07, Department of Electronics for Automation .

Huth, M., and Ryan, M. 2004. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press.

Miller, T. 2018. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*.

Schmidhuber, J., and Hochreiter, S. 1997. Long short-term memory. *Neural Comput* 9(8):1735–1780.

Simonyan, K.; Vedaldi, A.; and Zisserman, A. 2013. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Towards Explainable AI Planning as a Service

Michael Cashmore[1], Anna Collins[1], Benjamin Krarup[1], Senka Krivic[1], Daniele Magazzeni[1], David Smith
[1]King's College London, United Kingdom, {firstname.surname}@kcl.ac.uk

Abstract

Explainable AI is an important area of research within which Explainable Planning is an emerging topic. In this paper, we argue that Explainable Planning can be designed as a *service* – that is, as a wrapper around an existing planning system that utilises the existing planner to assist in answering contrastive questions. We introduce a prototype framework to facilitate this, along with some examples of how a planner can be used to address certain types of contrastive questions. We discuss the main advantages and limitations of such an approach and we identify open questions for Explainable Planning as a service that identify several possible research directions.

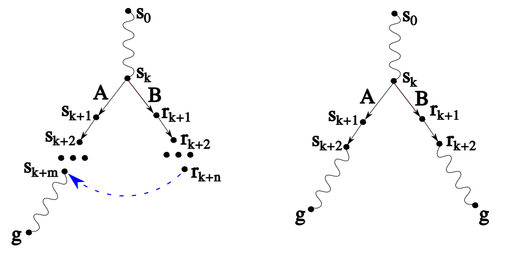


Figure 1: Generating contrastive explanations for a question "Why A rather than B?" at the state s_k.

1 Introduction

Explainable Artificial Intelligence (XAI) is an emerging research area in AI, motivated by the need to engender trust in users by explaining to them why the AI is making a particular decision.

While the main focus of XAI has been in Machine Learning, recently there has been growing interest in Explainable Planning, as shown by many planning contributions at the IJCAI International Workshops on XAI (XAI 2017; 2018) and the successful first ICAPS Workshop on Explainable Planning (XAIP 2018). Since the initial ideas of Smith (2012), there has been significant effort, in particular on the topic of Human-Aware Planning and Model Reconciliation (Chakraborti et al. 2017; Zhang et al. 2017; Sreedharan, Chakraborti, and Kambhampati 2018). Other significant works include topics such as Explainable Agency (Langley et al. 2017), moral values (Lindner, Mattmüller, and Nebel 2018), insights from social science (Miller 2019), and preferred explanations (Sohrabi, Baier, and McIlraith 2011).

A roadmap for Explainable Planning (XAIP) was proposed by Fox, Long, and Magazzeni (2017). They discussed some types of user questions that should be addressed. In particular, one type of question is of the form "*Why did you do A rather than B?*". We refer to this type of question as a *contrastive question*. To answer this kind of question one must reason about the hypothetical alternative "B", which likely means constructing an alternative plan where "B" is used rather than "A". The hypothetical alternative would be a plan that is not better than the one found by the planner or a plan which is better than the original one. Providing such a comparison between alternatives is what is called a *contrastive explanation*.

Fig. 1 shows an example of a contrastive question (Fox, Long, and Magazzeni 2017). Given a plan from the initial state s_0 to the goal g, the users might suggest an alternative action B rather than A at the state s_k. To provide a contrastive explanation means forcing action B and then replanning from the resulting state to see the alternative plan. One possible alternative is a sequence of different actions that rejoin the original plan with a different cost (Fig. 1 left). Another possibility is a completely alternative sequence of actions that achieves the goal (Fig. 1 right). In both cases the user can compare costs to gain confidence on the quality of the plan found by the planner.

Given that planning is now used in safety-critical applications, for example oil-well drilling (Long 2018), explanations play a key role. Crucially, the greater the expense or risks in executing the plan, the more important the role of explanations for engendering trust in the users who are responsible and accountable for authorising the execution of a plan.

In this paper, we propose that Explainable Planning can be designed and constructed *as a service* – i.e., 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 must have the ability to invoke the existing planning system on hypothetical problems in order to address contrastive questions. This approach allows users to get explanations constructed from their own trusted planner and model. In complex or safety-critical domains this requirement is a crucial one. There is one important requirement, however; in order to effectively use the existing planning system, the XAIP service must be able to add constraints on the planning problem and domain model. However, 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.

Ideally, constraints over models should be described using a rich language designed for specifying constraints on the form of a desired plan. However, in many cases, these constraints can be compiled down into the domain model directly, which requires that the XAIP service have visibility and access into that model. This approach is otherwise agnostic about the domain model and the planner.

We have implemented this approach in a prototype framework for XAIP as a Service, in a PDDL setting, as it is a widely used planning language.

In particular, in this paper, we (1) present a prototype framework that enables Explainable Planning as a Service for contrastive questions, (2) describe some important categories of contrastive questions, (3) describe how the service compiles these contrastive questions into hypothetical planning problems that can then be solved by the existing planner to facilitate contrastive explanations, and (4) discuss the current state of our implementation and a roadmap for future work on Explainable Planning as a Service.

The paper is organised as follows: we start with a running example in the next section. In Section 3 we describe the XAIP as a Service framework, providing details for each component. In Section 4 we briefly discuss the framework that implements this approach. In Section 5 we discuss open issues and present a roadmap that identifies several possible research directions. Section 6 concludes the paper.

2 Running Example

As a running example throughout the paper, we use a simplified version of a safety-critical model that might be used in industry. The model describes a warehouse organisation delivery system. There are one or more robots that work together to move pallets from their delivery location to the correct storage shelf. Before the pallets can be stored the shelf must be scanned.

Fig. 2 defines the domain for this model. There are four temporal actions, *goto_waypoint*, *scan_shelf*, *load_pallet*, and *unload_pallet*. The *goto_waypoint* action is used for the robots to navigate the factory. It ensures that the shelf the robot is moving to is not occupied by another robot to stop congestion. The *scan_shelf* action is a sensing action. The *load_pallet* action loads the pallet from a shelf on to the robot. The robots do not have the ability to scan the shelf while holding a pallet. Finally, the *unload_pallet* action unloads the pallet onto a previously scanned shelf.

The problem we use as an example is shown in Fig. 3. There are two robots, two pallets, and six waypoints.

An example plan for this planning problem is shown in Fig. 4, its cost is its duration (20.003) which in this case is the optimal plan [1]. Upon close examination, the plan does not appear straight-forward. Both pallets are delivered by a single robot, and there are a lot of movements that appear to be inefficient. For example, the robot *Tom* moves away from waypoint *sh*6, even though there is an undelivered pallet at that location. It might seem more efficient to pick up that pallet while the robot is beside it. Without access to the domain and problem shown in Fig. 2 and 3, or without understanding of PDDL semantics, the behaviour of these robots will be opaque to a user, and explanation required.

```
(:types
  waypoint robot - locatable
  pallet
)
(:predicates
  (robot_at ?v - robot ?wp - waypoint)
  (connected ?from ?to - waypoint)
  (visited ?wp - waypoint)
  (not_occupied ?wp - waypoint)
  (scanned_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 scan_shelf
  :parameters (?v - robot
              ?shelf - waypoint)
  ...)
(:durative-action load_pallet
  :parameters (?v - robot ?p - pallet
              ?shelf - waypoint)
  ...)
(:durative-action unload_pallet
  :parameters (?v - robot ?p - pallet
              ?shelf - waypoint)
  ...)
```

Figure 2: A fragment of a robotics domain used as a running example. Some of the operator bodies have been omitted for space.

[1]The plan is obtained using the planner POPF (Coles et al. 2010). However our framework accounts for all PDDL2.1 planners.

```
(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)
    ...
    (= (travel_time sh1 sh2) 4)
    (= (travel_time sh2 sh1) 4)
    (= (travel_time sh2 sh3) 8)
    (= (travel_time sh3 sh2) 8)
    ...
)
(:goal (and
    (pallet_at p1 sh6)
    (pallet_at p2 sh1))))
```

Figure 3: A fragment of the problem instance used in the running example.

```
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: (scan_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: (scan_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 4: Plan generated from the example domain and problem with cost 20.003.

3 Providing explanations as a Service

We present an *XAIP Service* framework for providing contrastive explanations. Figure 5 summarises the approach taken by the framework, following these steps:

Step 1: The *XAIP Service* takes as input the planning problem and model, the plan, and the 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.

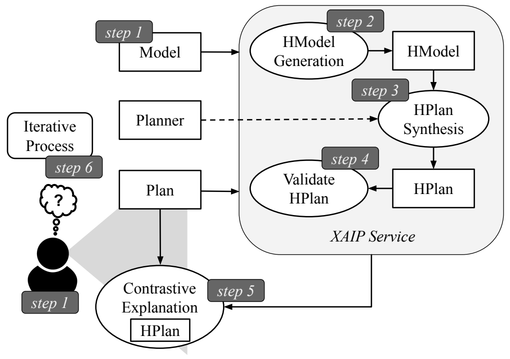


Figure 5: Architecture for Explainable Planning as a service, following the steps described in Section 3.

Step 3: The *original planner* uses the HModel as input to produce the *hypothetical plan* (HPlan).

Step 4: The XAIP Service validates the HPlan according to the original model.

Step 5: A contrastive explanation is constructed from the original plan and HPlan, and shown to the user.

Step 6: The user can choose to iterate the process from Step 1 with a new question. The user can choose to repeat the process with the original model and plan, or any HModel and HPlan.

The role of the HModel is to coerce the planner into creating the alternative plan that includes the actions and temporal constraints the user has in mind. One or more constraints must be added to the original model to create the HModel. We distinguish between three levels of abstraction in this process: In the first level the user question is given in natural language. The second level is a formal question that is derived from the natural language question. The formal question represents a set of constraints that are to be imposed upon the original model. Finally, the third level is a compilation of the formal question into the planning language. In our framework we focus on PDDL2.1 as the planning language, as we are interested in temporal and numeric planning problems. In this paper we describe the overall approach, and present a framework that encapsulates this process. The framework is modular, and allows different interfaces for providing user questions, and presenting explanations. The interface currently implemented in the framework allows the user to select a formal question directly. This represents constraints upon the plan. The set of formal question types from which the selection is made is described below in Section 3.1. For example, the user might have a question for the running example such as:

> "At the point in the plan where action $(goto\_waypoint\ Tom\ sh6\ sh1)$ is used, there's a pallet, so why doesn't Tom pick it up?"

From the user question above the following formal question is derived:

Why is action A used in state S, rather than action B?

Where action A is $(goto\_waypoint\ Tom\ sh6\ sh1)$, action B is $(load\_pallet\ Tom\ p2\ sh6)$, and the state S is the state in which action A was originally applied. This constraint enforces that the plan includes action B in state S instead.

Finally, this constraint can be compiled into the original model to produce the HModel, as described in Section 3.2.

3.1 Encoding User Questions

The user question is encoded as a set of constraints, which represent the formal question, and this is be done through an user interface where the user is guided to select the constraints that match their question. The questions we are interested in are *contrastive questions* of the form, *"Why A rather than B?"*, where A is the *fact* (i.e. what occurred in the plan) and B is the *foil* (i.e. the hypothetical alternative expected by the user). The formal questions currently handled by our approach are:

- "*Why is action A used in the plan, rather than not being used?*" This constraint would prevent the action A from being used in the plan.
- "*Why is action A not used in the plan, rather than being used?*" This constraint would enforce that the action A is applied at some point in the plan.
- "*Why is action A used, rather than action B?*" This constraint is a combination of the previous two, which enforces that the plan include action B and not action A.
- "*Why is action A used before/after action B (rather than after/before)?*" This constraint enforces that if action A is used, action B must appear earlier/later in the plan.
- "*Why is action A used outside of time window W, rather than only being allowed inside W?*" This constraint forces the planner to schedule action A within a specific time window.

One specific form of the question: "*Why is action A used, rather than action B?*", represented in Fig. 1, is "*Why is action A used in state S, rather than action B?*". This refinement forces the plan to include action B in state S, where B is an action (different from A) that is valid in that state. Using this constraint, the actions leading up to state S would remain unchanged, and the action A would still be allowed in other parts of the plan.

Deriving a formal question from a user question in natural language represents a significant research challenge. We discuss how a formal question might be identified automatically from natural language in Section 5.

While this set of formal questions covers a wide a range of possible situations and explanations, this does not comprise a complete set of constraints that could be applied to the problem. However, a single explanation does not exist in a vacuum, and a user might have a series of questions that will iteratively increase their understanding of the plan proposed by the planner, or the user can realise that the first question was not complete. For this reason, following ideas of Smith (2012), the framework allows the user to apply a sequence of formal questions to a single plan. For example, the general question "Why is action A used, rather than action B" can be seen as a combination of the first two questions,

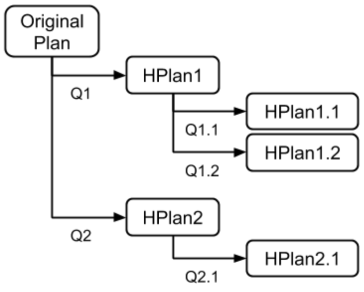


Figure 6: Example of iterative explanations.

which force action B to appear in the plan and not action A. This allows for a much wider range of explanations. Clearly the requirement of the plan being valid according to the original model must be satisfied at each iteration.

To allow the user to explore the space of hypothetical plans, it is possible to generate a tree of contrastive explanations in our framework, as illustrated in Fig. 6. Each node of the tree represents a hypothetical plan that was generated following a user question. The user can impose additional formal questions to a given plan in order to more precisely explore the behaviour that they are interested in. If the first question did not result in expected behaviour, the user can ask new questions to refine the hypothetical model until they reach a contrastive explanation that is satisfying to them.

As previously identified, the corresponding formal question in our running example is *"Why is action A used in state S , rather than action B?"*, where action A is $(goto\_waypoint\ Tom\ sh6\ sh1)$, action B is $(load\_pallet\ Tom\ p2\ sh6)$, and the state S is the state in which action A was originally applied.

3.2 Constructing the HModel

Constructing an HModel consists of taking the constraints in the formal question and compiling them into the planning domain model.

We give one example of this step, using the original model and plan shown in Section 2, and the question above. The compilation is formed such that the ground action B appears in the plan in place of the action A. Given a plan:

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

The ground action $a_i = A$ is replaced with B. The state directly after B has finished executing is found. This then becomes the new initial state I' in the HModel. The framework uses the effects at each happening, computed by VAL (Howey, Long, and Fox 2004), up to the replacement action to compute I'.

In addition, the new initial state I' is extended with a set of *timed-initial-literals* (TILs) which model the effects of actions that have started but not yet finished execution in the state selected by the user. A TIL is a tuple $\langle t, p \rangle$ where p is the effect that is asserted, and t is the time at which it is applied. Specifically, for each action a_j not finished executing in state S, we add the TIL:

$$\langle start(a_j) + duration(a_j) - start(B), effect(a_j) \rangle$$

```
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: (scan_shelf Tom sh6) [1.000]
4.001: (load_pallet Tom p2 sh6) [2.000]
6.002: (unload_pallet Tom p2 sh6) [1.500]
7.502: (goto_waypoint Tom sh6 sh1) [4.000]
7.001: (goto_waypoint Jerry sh4 sh5) [1.000]
11.502: (scan_shelf Tom sh1) [1.000]
11.503: (goto_waypoint Jerry sh5 sh6) [3.000]
12.502: (goto_waypoint Tom sh1 sh2) [4.000]
14.503: (unload_pallet Jerry p1 sh6) [1.500]
16.004: (load_pallet Jerry p2 sh6) [2.000]
18.004: (goto_waypoint Jerry sh6 sh1) [4.000]
22.004: (unload_pallet Jerry p2 sh1) [1.500]
```

Figure 7: *HPlan* generated with a cost of 23.504. The replaced action is highlighted.

where $start(a_j)$ ($start(a_j)$) is the time at which the action a_j (B) started execution, $duration(a_j)$ is the planned duration of action a_j, and $effect(a_j)$ is the end effect of action a_j. In this example the action $(goto\_waypoint\, Jerry\, sh3\, sh4)$ is still executing in the state where our action is replaced. We add a TIL which makes $(not\_occupied\, sh3)$, and $(robot\_at\, Jerry\, sh4)$ true at time 2.999 in I'. This simulates finishing of the concurrent action execution.

A plan is then generated from this new state for the original goal, which gives us the plan:

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

The HPlan is then the initial actions of the original plan π up to a_i, concatenated with the replaced action B and the new plan π':

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

The result is the HPlan shown in Fig. 7. The replaced action (B) is shown in bold. The initial actions before B are the actions from the original plan. The remaining actions are those of the new plan π'. As part of the service, the HPlan is validated with respect to the original model before a contrastive explanation is formed.

Note that in this HPlan, the user action is immediately reversed. This would not be a satisfactory explanation to the user, who wishes to see the plan in which carrying the pallet is essential to achieve the goal. Thus, it is not sufficient for the user suggested action to just be included in the plan, it must be part of the plan's key causal structure.

Fink and Yang (1992) define four categories of redundant actions, and Chrpa, Mccluskey, and Osborne (2012) present a simple algorithm to determine if an action is redundant in a sequential plan. As a post-processing step, after generating an HPlan, redundant actions of this kind can be detected. If suggested actions are not essential (redundant) to the HPlan, the planner needs to continue to search for additional plans until it finds one where the suggested actions are part of the causal structure for achieving the original goals. This additional search could potentially be made more efficient by introducing additional constraints (nogoods) into the HModel that rule out alternative ways of achieving those goals, ultimately leaving only plans where the suggested actions are essential. For example, if the action B is redundant in the HPlan and action C is used instead of A, an additional constraint could be introduced to disallow C. In general, it is still an open question how to automatically infer useful nogood constraints from redundant HPlans.

An alternative is to allow the user to refine their question to rule out additional alternative solutions in which the suggested action is not essential. For example, the user's question might be expanded to include the constraint "and don't use actions C or D either". In either case it is clear that XAIP as a Service needs to follow an iterative approach where the planner generates a sequence of progressively more refined solutions, as additional constraints are imposed by nogoods, or by successive refinement of the user question. This is discussed further in Section 3.3.

Ideally, we would like to compile the original user question into an HModel that guarantees that suggested actions are essential to the causal structure of the plan (not redundant). However, it is an open question as to whether it is possible to do this. We suspect not. It is easy to force an action into a plan by adding a phantom effect to the action, and adding this phantom proposition to the goal. However, it is not obvious how to ensure that the action play an essential part in the achievement of other goals.

In addition to the example shown here, our framework includes compilations for all of the questions introduced in Section 3.1. Contrary to the compilation process of the presented example, the constraints posed from other questions are directly compiled into HModels which are used to produce HPlans.

3.3 Explainable Planning as an Iterative Process

Following the ideas of iterative processes by Smith (2012), we follow the same approach for explainable planning.

The user is able to use the framework to iterate the process by asking further questions, and refining the HModel. If the explanation does not completely satisfy the user, this allows the user to impose additional constraints that can be compiled into the HModel. For example, given the HPlan in Fig. 7, the user may have an additional question:

"Wait, shouldn't Tom have taken the pallet to its destination?"

Through the user interface with the framework the user selects a formal question. The formal question which encapsulates this user question is

Why is the action A not used in the plan, rather than being used?

Where action A is the action $(unload\_pallet\ Tom\ p2\ sh1)$. This constraint enforces that action A is used in the plan.

This constraint compiled into the HModel to enforce the user's suggestion, resulting in a new HModel. The *HModel* is then solved with the original planner to obtain the plan shown in 8.

```
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: (scan_shelf Tom sh6) [1.000]
3.002: (goto_waypoint Tom sh6 sh1) [4.000]
7.001: (goto_waypoint Jerry sh4 sh5) [1.000]
7.003: (scan_shelf Tom sh1) [1.000]
7.004: (goto_waypoint Tom sh1 sh6) [4.000]
11.004: (load_pallet Tom p2 sh6) [2.000]
13.004: (goto_waypoint Tom sh6 sh1) [4.000]
17.004: (unload_pallet Tom p2 sh1) [1.500]
17.005: (goto_waypoint Jerry sh5 sh6) [3.000]
20.005: (unload_pallet Jerry p1 sh6) [1.500]
```

Figure 8: *HPlan* generated with the second user constraint maintained, with a cost 21.505. The action suggested by the user is highlighted.

3.4 Forming Contrastive explanations

A contrastive explanation draws from the original plan π, the HPlan π_H and the validation outcome. Defining a contrastive explanation is a complex task. In this paper we introduce a foundation for the contrastive comparison as a result of a comparison between the original plan and the HPlan.

The contrastive explanation can be defined as

$$CE = \langle comparison, Q \rangle$$

where *comparison* contains relevant information about the differences in plans that were caused by the user question Q: $comparison(\pi, \pi_H) = \langle existing, removed, added, diffcost \rangle$ where:

- *existing* - actions in the plan which remained the same
- *removed* - actions which were removed from the plan
- *added* - actions which were added in HPlan and were not in original plan
- *diffcost* - the difference between the cost of the plans

By observing the *comparison*, a user can reason about the effect of the constraints that their question imposed and about the behaviour of the model. The validation outcome (performed against the original model) reassures the user about the validity of the HPlan. It also helps the user understand the difference between the costs of the two plans. The contrastive explanation for the HPlan in Fig. 8 is shown in Fig. 9.

4 XAIP as a Service Framework

For the purpose of evaluating the proposed approach to Explainable Planning as a Service we implemented a modular framework for domains and problems written in PDDL2.1. This prototype works with any planner capable of reasoning with PDDL2.1. The architecture of the framework is illustrated in Fig. 10. Interaction with a user is enabled through a console interface as well as a graphical user interface.

The *Controller* controls the behaviour of the XAIP framework and communicates with all other segments of the framework. The *XAIP Interface* creates a knowledge base

existing:

```
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: (scan_shelf Tom sh6) [1.000]
3.002: (goto_waypoint Tom sh6 sh1) [4.000]
7.001: (goto_waypoint Jerry sh4 sh5) [1.000]
7.003: (scan_shelf Tom sh1) [1.000]
17.005: (goto_waypoint Jerry sh5 sh6) [3.000]
20.005: (unload_pallet Jerry p1 sh6) [1.500]
```

removed:

```
9.001: (goto_waypoint Tom sh1 sh6) [4.000]
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]
```

added:

```
7.004: (goto_waypoint Tom sh1 sh6) [4.000]
11.004: (load_pallet Tom p2 sh6) [2.000]
13.004: (goto_waypoint Tom sh6 sh1) [4.000]
17.004: (unload_pallet Tom p2 sh1) [1.500]
```

$diffcost = 21.505 - 20.003 = 1.502$

Figure 9: The contrastive explanation that is presented to the user.

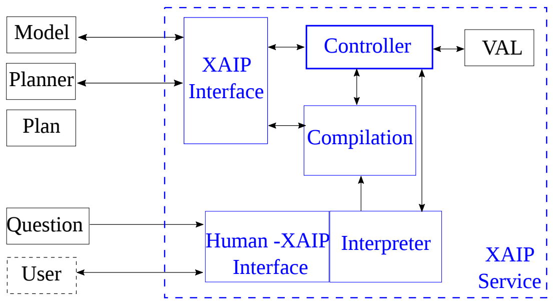
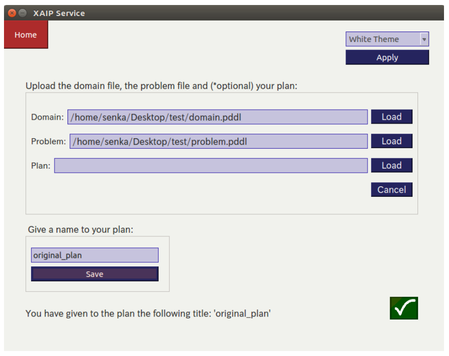


Figure 10: Architecture of the framework for Explainable Planning as a service.

from the PDDL files of the original model. This module also interacts with a planner. The *Compilation* module uses the constraints of the formal question to create the HModel. The validation module *VAL* (Howey, Long, and Fox 2004) is used as the validation technique.

The *Human-XAIP interface and interpreter* receives the questions from the user, creates a formal question instance, and demonstrates the explanation to the user. There are two implementations of this interface: a console and graphical user interface (Fig. 11). Both provide the same functionality, in which a user can see the plan, ask a formal question from the set of questions presented in Section 3.1, either by using a simple console interface or a set of forms for filling in the necessary details of the question. The output of the system is a visualisation of the original plan, HPlan, and VAL report. Actions in the plan are highlighted to correspond with the comparison described in Section 3.4.



Upload the domain file, the problem file and (*optional) your plan:

Domain: /home/senka/Desktop/test/domain.pddl

Problem: /home/senka/Desktop/test/problem.pddl

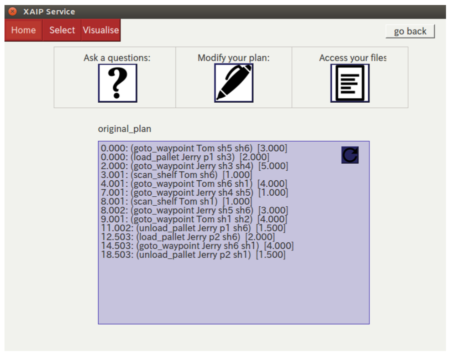
Plan:

White Theme

Give a name to your plan: original_plan

You have given to the plan the following title: 'original_plan'

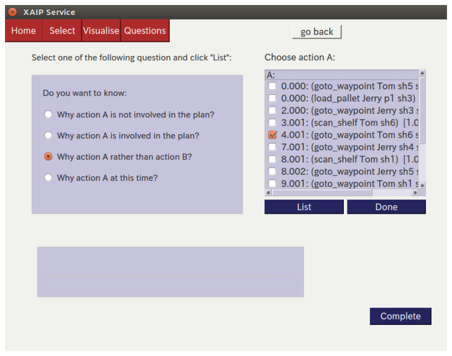
(a) Loading model



original_plan

```
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: (scan_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: (scan_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]
```

(b) Plan visualisation



Select one of the following question and click "List":

Do you want to know:

- Why action A is not involved in the plan?
- Why action A is involved in the plan?
- [✓ Why action A rather than action B?]
- Why action A at this time?

Choose action A:

A:
- 0.000: (goto_waypoint Tom sh5
- 0.000: (load_pallet Jerry p1 sh3)
- 2.000: (goto_waypoint Jerry sh3
- 3.001: (scan_shelf Tom sh6) [1.0
- [✓ 4.001: (goto_waypoint Tom sh6]
- 7.001: (goto_waypoint Jerry sh4
- 8.001: (scan_shelf Tom sh1) [1.0
- 8.002: (goto_waypoint Jerry sh5
- 9.001: (goto_waypoint Tom sh1

(c) Question selection

Figure 11: Screenshots of the graphical user interface of the *XAIP Service* framework

5 Discussion

In order for Explainable Planning as a service to become effective, we discuss some challenges and outline some potential future work.

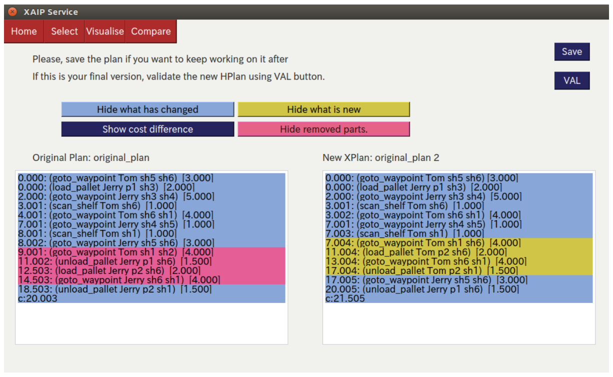
- *Understanding user questions.* The framework is modular to enable different ways for communicating a question to the AI system. Depending on the user interface a question could be given in different forms, for example speech, visual gestures or text input. Different technologies can be used to translate the question into a formal question such as: speech recognition, Natural Language Processing methods or human body tracking. Context can play a crucial role in understanding the question that the user asks. Borgo, Cashmore, and Magazzeni (2018) showed an example of a question requiring two different explanations depending on the context in which it was asked. In each case, improperly interpreting the question lead to an unsatisfying explanation. One promising direction for addressing this challenge is the use of *argumentation* (Cyras et al. 2019).

- *Formally categorising the set of questions that can be answered with contrastive explanations.* Although some philosophers, such as van Fraassen (1980), noted that "why"-questions can be implicitly or explicitly understood as: "why is A better than some alternative?", there might be questions in the planning space for which contrastive explanations are not well-suited. For example, if the user is simply trying to understand the conditions or requirements for various actions in the plan, or the causal or temporal structure of the plan, a contrastive explanation may not be appropriate. Instead it might be more appropriate to highlight the causal structure in an abstracted version of the plan. An important issue for future work is the development of a formal taxonomy of the types of questions that should be addressed using contrastive explanations (Miller 2018).

- *Expressing constraints on actions and plan structure.* A contrastive question requires creating a hypothetical planning model, which is often characterised by constraints on what actions are permissible in the plan and how they are arranged. For example, the question "Why did you use action A rather than action B for achieving P?" requires planning with the hypothetical model 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 universally excluding A from the plan and forcing B into the plan because A or B might be required or prohibited elsewhere in the plan. Currently we do not have a good language for expressing these kinds of constraints. PDDL 3 allows the expression of simple constraints on the order in which goals are achieved, but does not have the ability to express constraints on action inclusion, exclusion, or ordering, and does not allow us to place more complex constraints on *how* something is achieved or on plan structure. We would like to be able to say something simple like "Supports$(B,P) \wedge \neg$Supports(B,P)". LTL will likely play a key role in defining the semantics of any such language, but additional concepts concerning plan structure are needed, such as the ability to specify that an action is part of the causal support for a goal or subgoal.

- *Compiling constraints into the HModel.* We showed examples of how a constraint derived from a user question could be compiled to form an HModel. However, providing compilations for more general constraints (like the one above) and ensuring their correctness is an important issue. Additionally, the compilation can lead to producing plans which might differ from the original plan in ways unrelated to the user question. We believe that the work on planning with preferences (Gerevini, Saetti, and Serina 2006) and state-trajectory constraints (Baier et al. 2009) is an important first step, but does not yet address the full range of constraints needed.

- *Forming and presenting contrastive explanations.* The form of contrastive explanation we provide, as discussed in Section 3.4 and shown in the GUI in Fig. 12, is a very simple one that presents the original plan and HPlan and highlights the action differences between them. Also, it is possible to obtain hierarchical contrastive explanations by asking consecutive questions. However, this does not show the causality of the plans, or the differences in their causal structure. Fig. 13 shows a possible composite causal representation for both the original plan and



Please, save the plan if you want to keep working on it after

If this is your final version, validate the new HPlan using VAL button.

Original Plan: original_plan

```
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: (scan_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: (scan_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]
c:20.003
```

New XPlan: original_plan 2

```
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: (scan_shelf Tom sh6)  [1.000]
3.002: (goto_waypoint Tom sh6 sh1)  [4.000]
7.001: (goto_waypoint Jerry sh4 sh5)  [1.000]
7.003: (scan_shelf Tom sh1)  [1.000]
7.004: (goto_waypoint Tom sh1 sh6)  [4.000]
11.004: (load_pallet Tom p2 sh6)  [2.000]
13.004: (goto_waypoint Tom sh6 sh1)  [4.000]
17.004: (unload_pallet Tom p2 sh1)  [1.500]
17.005: (goto_waypoint Jerry sh5 sh6)  [3.000]
20.005: (unload_pallet Jerry p1 sh6)  [1.500]
c:21.505
```

Figure 12: Output of the GUI in which differences in the plans are highlighted.

the HPlan, with the differences shown in different colors. This way of visualising the explanation can help to elucidate how the two plans achieve, or fail to achieve, the (sub)goals of the problem with respect to specific actions in the domain. However, for larger and more complex plans we expect that some form of abstraction will be necessary in order to effectively compare and contrast plans; the user might wants to see the important differences between two plans, not all the details. What counts as details remains an open research question, but is likely related to action costs and to the ease and importance of achieving various subgoals. Sreedharan, Srivastava, and Kambhampati (2018) have done some initial work in this area, and have considered milestones as important abstractions for purposes of explanation. The issues of what constitutes a good explanation, and how to visualize it or present it remain intertwined. Some synergy between researchers in planning, data visualization (e.g., Chakraborti et al. (2018) or Mennatallah et al. (2018)), and social sciences (Miller 2019) would be fruitful.

- *Providing explanations for complex questions.* In the presented approach, a user is able to iteratively ask questions to refine the explanation. If the explanation does not satisfy the user, or the question they have is more complex, this approach can provide the user with a deeper understanding. However, this process could be automated by analysing a more complex question the user might have, and decomposing it into several formal questions. In this case new constraints can be added to the HModel automatically until the explanation addresses the intended question and potentially the context it was asked within.
- *Assessing the effectiveness of explanations.* We believe it is crucial to be able to acquire evidence of the effectiveness of an explanation. In particular, if engendering trust is the motivation for Explainable Planning and XAI in general, then we should look at the actual experience of the users and check whether they gain confidence in the planner or not. For this, a vital step for planning researchers is to include *user studies* to assess the effectiveness of the explanations they are providing.

While of course this is not an exhaustive list of all the necessary next steps, it already provides an interesting set of challenges that should be addressed. Note that while we are advocating for Explainable Planning as a service, we are well aware that this is not the only way to provide explanations for planning. In particular, we envisage at least the following possible limitations that nevertheless represent important research questions that should be considered by the Explainable Planning community. First we acknowledge that contrastive explanations are not suitable to answer every type of question that the user might have. However, we argue that contrastive questions are common and that contrastive explanations therefore play a significant role, as also acknowledged by other researchers in Explainable AI, e.g, (Miller 2018). Second, by lifting the requirement that the explanation is generated by the planner used to generate the original plan, it could be possible to modify the search procedure used by the planner to generate explanations from a wider set of constraints. Third, we are assuming that there is no uncertainty in the original planning model and that the model is correct. However, this is not always the case and

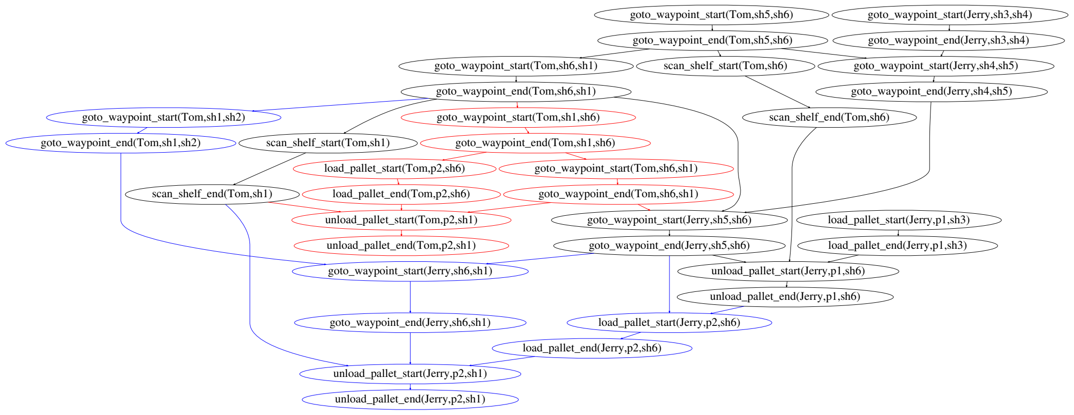

goto_waypoint_start(Tom,sh5,sh6)
goto_waypoint_start(Jerry,sh3,sh4)
goto_waypoint_end(Tom,sh5,sh6)
goto_waypoint_end(Jerry,sh3,sh4)
goto_waypoint_start(Tom,sh6,sh1)
scan_shelf_start(Tom,sh6)
goto_waypoint_start(Jerry,sh4,sh5)
goto_waypoint_end(Tom,sh6,sh1)
goto_waypoint_end(Jerry,sh4,sh5)
goto_waypoint_start(Tom,sh1,sh2)
goto_waypoint_start(Tom,sh1,sh6)
scan_shelf_end(Tom,sh6)
goto_waypoint_end(Tom,sh1,sh2)
scan_shelf_start(Tom,sh1)
goto_waypoint_end(Tom,sh1,sh6)
load_pallet_start(Tom,p2,sh6)
goto_waypoint_start(Tom,sh6,sh1)
scan_shelf_end(Tom,sh1)
load_pallet_end(Tom,p2,sh6)
goto_waypoint_end(Tom,sh6,sh1)
unload_pallet_start(Tom,p2,sh1)
goto_waypoint_start(Jerry,sh5,sh6)
load_pallet_start(Jerry,p1,sh3)
unload_pallet_end(Tom,p2,sh1)
goto_waypoint_end(Jerry,sh5,sh6)
load_pallet_end(Jerry,p1,sh3)
goto_waypoint_start(Jerry,sh6,sh1)
unload_pallet_start(Jerry,p1,sh6)
unload_pallet_end(Jerry,p1,sh6)
goto_waypoint_end(Jerry,sh6,sh1)
load_pallet_start(Jerry,p2,sh6)
load_pallet_end(Jerry,p2,sh6)
unload_pallet_start(Jerry,p2,sh1)
unload_pallet_end(Jerry,p2,sh1)

Figure 13: Example of a causal graph which demonstrates a comparison of the original plan and HPlan. Added actions are red, removed actions are blue.

for this, the body of research on model reconciliation plays a very important role (Chakraborti et al. 2017).

6 Conclusions

In this paper we have presented a prototype framework for Explainable Planning as a service, which we believe represents an effective way of providing explanations, particularly in safety critical domains. In such scenarios the user would not accept an explanation that is generated by a planner or a model different from the ones that they use and whose performance they trust. To this end, Explainable Planning as a service is based on providing explanations using the users' planners and models. Note that while the users can trust their planners and their models, there might still be reasonable questions on why a particular plan was found. This is where explanations are important, and we propose *contrastive explanations* to allow the user to compare the plan found by the planner with what the user was expecting.

The Explainable Planning as a Service framework is modular, and can be used with all planners and domains that ahere to PDDL2.1. In order to foster the use of Explainable Planning in robotics applications the proposed framework is now being integrated in ROSPlan (Cashmore et al. 2015).

We proposed a roadmap with some of the challenges that should be addressed for Explainable Planning to become effective, also highlighting promising directions. We believe that synergies between researchers in planning and in other disciplines, such as data visualization, social science, human-computer-interaction, and cognitive science, are key for the practical success of Explainable Planning.

Acknowledgements This work was partially supported by Innovate UK grant 133549: *Intelligent Situational Awareness Platform*, and by EPSRC grant EP/R033722/1: *Trust in Human-Machine Partnerships*.

References

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *Artif. Intell.* 173(5-6):593–618.

Borgo, R.; Cashmore, M.; and Magazzeni, D. 2018. Towards providing explanations for AI planner decisions. *IJCAI-18 Workshop on Explainable AI*.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. In *ICAPS*.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *IJCAI*.

Chakraborti, T.; Fadnis, K. P.; Talamadupula, K.; Dholakia, M.; Srivastava, B.; Kephart, J. O.; and Bellamy, R. K. E. 2018. Visualizations for an explainable planning agent. In *IJCAI*.

Chrpa, L.; Mccluskey, T. L.; and Osborne, H. 2012. Determining redundant actions in sequential plans. In *ICTAI*.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *ICAPS*.

Cyras, K.; Letsios, D.; Misener, R.; and Toni, F. 2019. Argumentation for explainable scheduling. In *https://arxiv.org/abs/1811.05437*.

Fink, E., and Yang, Q. 1992. Formalizing plan justifications. In *Conference of the Canadian Society for Computational Studies of Intelligence*, 9–14.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable planning. *IJCAI-17 workshop on Explainable AI* abs/1709.10256.

Gerevini, A.; Saetti, A.; and Serina, I. 2006. An approach to temporal planning and scheduling in domains with predictable exogenous events. *JAIR* 25.

Howey, R.; Long, D.; and 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*, 294–301.

Langley, P.; Meadows, B.; Sridharan, M.; and Choi, D. 2017. Explainable agency for intelligent autonomous systems. In *AAAI*.

Lindner, F.; Mattmüller, R.; and Nebel, B. 2018. Moral permissibility of action plans. *ICAPS-18 Workshop on Explainable Planning*.

Long, D. 2018. Planning a way into a deep hole. In *Invited talk at ICAPS Workshop on Planning and Scheduling Applications (SPARK)*.

Mennatallah, E.-A.; Duen Horng, C.; Adam, P.; Hendrik, S.; and Fernanda, V. 2018. Workshop on visualization for AI explainability. In *http://visxai.io/*.

Miller, T. 2018. Contrastive explanation: A structural-model approach. *arXiv preprint arXiv:1811.03163*.

Miller, T. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence* 267.

Smith, D. 2012. Planning as an iterative process. In *AAAI*.

Sohrabi, S.; Baier, J. A.; and McIlraith, S. A. 2011. Preferred explanations: Theory and generation via planning. In *AAAI*.

Sreedharan, S.; Chakraborti, T.; and Kambhampati, S. 2018. Handling model uncertainty and multiplicity in explanations via model reconciliation. In *ICAPS*.

Sreedharan, S.; Srivastava, S.; and Kambhampati, S. 2018. Hierarchical expertise-level modeling for user specific contrastive explanations. In *IJCAI*.

van Fraassen, C. B. 1980. *The Scientific Image*. Oxford University Press.

XAI. 2017. IJCAI Workshop on Explainable AI. http://home.earthlink.net/dwaha/research/meetings/ijcai17-xai.

XAI. 2018. IJCAI Workshop on Explainable AI. http://home.earthlink.net/dwaha/research/meetings/faim18-xai.

XAIP. 2018. ICAPS Workshop on Explainable Planning. http://icaps18.icaps-conference.org/xaip.

Zhang, Y.; Sreedharan, S.; Kulkarni, A.; Chakraborti, T.; Zhuo, H.; and Kambhampati, S. 2017. Plan explicability and predictability for robot task planning. In *ICRA*.

Varieties of Explainable Agency

Pat Langley
Institute for the Study of Learning and Expertise,
2164 Staunton Court, Palo Alto, CA 94306 USA
Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland 1142 NZ

Abstract

In this paper, I discuss some varieties of explanation that can arise in intelligent agents. I distinguish between process accounts, which address the detailed decisions made during heuristic search, and preference accounts, which clarify the ordering of alternatives independent of how they were generated. I also hypothesize which types of users will appreciate which types of explanation. In addition, I discuss three facets of multi-step decision making – conceptual inference, plan generation, and plan execution – in which explanations can arise. I also consider alternative ways to present questions to agents and for them provide their answers.

1 Introduction

Intelligent systems are becoming more widely adopted for critical tasks like driving cars and controlling military robots. Our increased reliance on such devices has led to concerns about the interpretability of their complex behavior. Before we can fully trust such autonomous agents, they must be able to explain their decisions so that we can gain insight into their operation. There is now a substantial literature on explanation in systems that learn from experience, but it has focused on tasks like object recognition and reactive control, typically using opaque encodings of expertise.

However, we also need research on explanation for more complex tasks that involve multi-step decision making, such as the generation and execution of plans. Approaches to these problems rely on high-level representations that are themselves easily interpreted, but challenges arise in communicating solutions that combine these elements and the reasons they were chosen. In this paper, I focus on such settings. Some work on explanation, especially with opaque models, has dealt with post hoc rationalizations of behavior, rather than the actual reasons for it. In the pages that follow, I limit my discussion to the latter. Moreover, I will focus on *self explanations*, that is, the reasons the explaining agent carried out a certain activity. Elsewhere (Langley, 2019), I have referred to this ability as *explainable agency*. This problem is arguably less challenging than postulating the reasons that another agent behaved as it did, sometimes called *plan recognition*, as the system can store and access traces of its own decision making.

We can specify the task of explainable agency in generic terms. Given domain knowledge for generating task solutions and criteria for evaluating candidates, the agent carries out search to find one or more solutions. After generating, and possibly executing, these solutions, a human asks the agent to justify its decisions, at which point it must clarify its reasoning in comprehensible terms. One example involves an intelligent robot that plans and executes a reconnaissance mission, after which it takes part in an 'after-action review' where it answers questions from a human supervisor. There has been some research on such *explainable planning* (Fox et al., 2017; Smith, 2012; Zhang et al., 2017), but we need more effort devoted this important topic.

In the next section, I distinguish between two forms of self explanation, identify component abilities they require, and citing relevant research. I also propose two hypotheses about when each type of account will be most useful. After this, I discuss three types of content over which one can generate explanations, along with alternative ways to pose questions and present answers. In closing, I review the essay's main points and reiterate the need for substantially additional research on the topic of explainable agency.

2 Forms of Self Explanation

Before the research community can develop computational methods for self explanation, it must first establish which aspects of decision making to elucidate. I maintain that there are two primary forms of explanation, which I attempt to characterize in this section. In each case, I offer some intuitions, define the task in terms of inputs and outputs, and discuss components that appear necessary to carry it out.

2.1 Process Accounts

The first form of self explanation focuses on the *processes* that led a system to generate its plans or other mental structures. This view revolves around the widespread assumption, which had its origins in the earliest days of artificial intelligence, that complex cognition requires heuristic search through a problem space (Newell and Simon, 1976). This assumes that the recipients of explanations are interested in details about how the system carried out that search, including which alternatives it considered, why it decided to pursue some in favor of others, and even when it decided to change its mind (e.g., by deciding to backtrack).

We can specify the generic task of explaining the processing that produced solutions as:

- *Given:* Knowledge defining a space of possible solutions;
- *Given:* Criteria for evaluating candidate solutions;
- *Given:* An annotated search tree that includes solutions found for some reasoning task;
- *Given:* A query about why a solution ranks above others;
- *Produce:* An explanation why the solution is preferable.

This task formulation is similar in spirit to the generation of think-aloud protocols (Newell and Simon, 1972), which gave early insights about human problem solving and which led directly to the creation of early AI systems. In this setting, a researcher presents a subject with some problem (e.g., a theorem to prove or a puzzle to solve), asking the subject to talk aloud as he works on it. The researcher records this verbal report, transcribes it, and analyzes it to understand the subject's thinking processes. One important difference is that our explanation task occurs after problem solving is complete. Retrospective reports from in humans far less reliable than on-line protocols, but AI systems have better memories than people, so I will ignore this issue.

Elsewhere (Langley, 2019), I have analyzed the component abilities that appear necessary to support this variety of self explanation. These include ensuring that the intelligent agent can:

- *Generate decision-making content*. When carrying our heuristic search, an agent must consider different nodes and operators, evaluate them, and select one to pursue.
- *Store generated content*. When it makes such decisions, the agent must store and index details about the choices it considered, and why it selected one of them, in an episodic memory or similar repository.
- *Retrieve stored content*. After it has solved a problem, the agent must transform questions into cues that let it retrieve traces of relevant decisions from episodic memory.
- *Communicate retrieved content*. Once it has retrieved this information, it must translate this content into an understandable form and communicate it.

Taken together, these abilities should let an intelligent system not only find solutions to complex problems, but also recount how it managed to uncover them.

The AI literature includes some relevant research on these topics. For instance, work on analogical planning (e.g., Jones and Langley, 2005; Veloso et al., 1995) has addressed storage, indexing, and retrieval, but not for use in self explanation. Some expert systems recorded their reasoning and played them back on request (Clancey, 1983; Swartout et al., 1991), while Johnson (1994) and van Lent et al. (2004) developed agents that carried out military mssions, recorded their decisions, and answered questions about their reasoning. Other related work includes an interactive robot that can give five types of reasons why it cannot carry out a task (Briggs and Scheutz, 2015) and computational models of argument (e.g., Bench-Capon and Dunne, 2007) that explain how alternative conclusions are eiter supported or contradicted by available evidence.

2.2 Preference Accounts

The second form of self explanation focuses on the final solutions produced by heuristic search, without concern for how they were found. This view recognizes that there are many different techniques for problem solving. A classic example is that some planning methods chain backward from goal descriptions, whereas others chain forward from the initial state. Similarly, some approaches to constraint satisfaction carry out search through a space of partial variable assignments, whereas others only consider alternatives that have complete assignments. Even within the same framework, different heuristics can guide search independently of the target goals or objective function. Nevertheless, these different systems can arrive at the same solutions by distinct paths, which can be sources of explanation themselves.

As before, we can state this task more precisely in terms of inputs and outputs:

- *Given:* Knowledge defining a space of possible solutions;
- *Given:* Criteria for evaluating candidate solutions;
- *Given:* A ranked set of solutions to some reasoning task;
- *Given:* A query about why a solution ranks above others;
- *Produce:* An explanation why the solution is preferable.

This task is very different from generating think-aloud protocols about the choices considered and selected during heuristic search. Rather, it comes much closer to the task addressed by recommender systems, which often produce a ranked list of candidates for users to consider. Most of these focus on ranking a fixed set of items, such as books, but one can also rank solutions to planning, scheduling, and other tasks that involve multi-step reasoning.

The distinction between process and preference explanations is not a matter of granularity, but whether one cares about *means* of reaching results or about their *quality*. To clarify this point, consider a simplified case-based reasoning system that iteratively retrieves a complete plan from memory, replacing one of the *n* best candidates with a new one if the latter scores better. A process account would store the sequence of candidates considered and explain its final choices in terms of steps in this procedure. In contrast, a preference account would retain only the final set of candidates and explain their ordering in terms of how each one fares on its criteria. An explainable agent should also, when a given candidate is not in the solution set, state why it was (presumably) ranked lower than those included.

Neither does the emphasis on preferences imply that explanation must only deal with complete solution structures. For example, if a planner uses a hierarchical task network to guide its search, then a user should be able to question why it selected one subplan for a given subtask rather than another decomposition. The same idea applies to a system that finds proof trees using monotonic inference rules, where a user may ask why it favored one subproof over another candidate that leads to the same intermediate conclusion. The ability to focus attention on elements of hierarchical solutions does not necessarily mean that explanations must touch on how the solutions were found.

Let us consider the component abilities that seem needed for an intelligent agent to provide such preference explanations. These include the capacity to:

- *Generate and rank solutions*. However the agent solves a problem, it must use explicit, interpretable criteria to place an ordering on them.
- *Compare two ranked solutions*. When asked why one solution was placed before another, the agent must compare their component scores and their combinations.
- *Communicate solution differences*. Once it has noted how the candidates differ, it must convey this information and how it led to their relative rankings.

The details of these abilities will depend on how the scoring and ranking process operates. One common scoring method uses a linear utility function that computes each candidate's score on k features, multiplies each score by a weight, and calculates a weighted sum, then orders candidates by this total. Another scheme uses a lexicographic function, which orders attributes by importance. Candidates are first partitioned based on scores for the first attribute, then ranked within these sets based on the second attribute, and so forth, much as words in a dictionary. The structure of explanations will depend on the technique used to order solutions.

I mentioned earlier the analogy to recommender systems, which often rely on a learned user profile to rank candidate items like books or movies. However, one can use such profiles as heuristics to guide search on complex reasoning tasks and to rank the solutions found in this manner. Rogers et al. (1998) applied this idea to route planning, drawing on a user profile, stated as weights on route features, to find personalized routes in a digital road map. Gervasio et al. (1999) adopted a similar approach to personalized scheduling, invoking a user profile, here weights on schedule features, to evaluate candidates and rank solutions. These two efforts are interesting because the first used best-first search through a space of partial routes, whereas the second used repair-space search through a space of complete schedules. Together, they offer evidence that one can have the same type of preference explanations for radically different search methods.

2.3 Two Hypotheses about Explanations

Now that we have identified and characterized two forms of self explanation, we can ask which is them is more useful to humans who interact with intelligent agents. One might argue that process explanations are the natural choice, as providing more details will give greater insight into a system's operation. But one might instead hold that preference accounts are superior, because humans have no need to know how the system found its solutions but only why it ranked the alternatives as it did.

In this paper, I will not take either position, but instead claim that the most appropriate form of explanation depends on the user's aims. This argument assumes that there are two quite different types of users, which leads to two hypotheses. We can state the first as:

- *Hypothesis 1: Process explanations will be favored by researchers interested in the details of heuristic search.*

This conjecture posits that some users care primarily about the process of finding solutions. This group includes cognitive psychologists who want to understand the ways in which an intelligent system mimics, or fails to mimic, a human problem solver. Yet it also includes many AI researchers who are concerned with the detailed operation of their systems, both for debugging purposes and for improving their search mechanisms.

However, not all users of intelligent systems will care about the technical details of their search behavior. This suggests a second conjecture, which we can state as:

- *Hypothesis 2: Preference explanations will be favored by system users interested in the results of heuristic search.*

This group includes end users of autonomous agents who had no role in their development. These are analogous to people who use recommender systems but have little idea how they operate, but who still want to know why they ranked one item as better than another. But it will also include AI researchers, and even psychologists, who are concerned more with criteria used to evaluate solutions than with the search mechanisms that produce them.

3 Types of Explanatory Content

We should also consider the types of tasks over which explanations of complex multi-step reasoning can occur. Planning is the most obvious class of domains and the one that has received the most attention in the literature (Fox et al., 2017; Smith, 2012). Clearly, a planning system can support both forms of explanation discussed above. At each stage in the search process, it can store the choices considered, their associated scores, and the alternative selected, along with decisions about when to backtrack. This information will let it answer detailed queries about the search history. Of course, planning systems can also find multiple solutions, rank them, and use their scoring procedure to provide preference explanations instead.

Plan execution is another important arena that supports explainable agency (Johnson, 1994; van Lent et al., 2004). This setting definitely supports process accounts, as the agent must monitor the environment to determine whether the plan is proceeding as expected. Detection of anomalies can be recorded, along with decisions about whether to continue or to revise the plan. The role of preference accounts is less obvious when the plans being executed are fully grounded, as no alternatives are available. However, frameworks that include reactive control constrained by hierarchical task networks (e.g., Choi and Langley, 2018) do allow multiple choices that the agent can rank by value. These will still be local to the particular situation in which the agent is taking action, but they should support preference accounts.

A third area involves conceptual inference, which may not count as agency itself but which certainly supports it. Here the intelligent system uses knowledge to draw conclusions about its situation from information available to it, using either deductive or abductive reasoning. The latter mechanisms clearly support process accounts, as demonstrated by the early work on explainable diagnostic systems (Clancey, 1983; Swartout et al., 1991), which stored traces of the reasoning chains that led to their conclusions. However, conceptual inference also supports preference explanations, since the system may have criteria for evaluating alternative derivations, such as the length of its reasoning chain or the number of default assumptions. Such accounts are most interesting in settings that involve incomplete information, where the system may find multiple contradictory interpretations of its situation.

4 Interaction Modalities

As I have defined them, explainable agents must be able to accept questions about their decision making and answer them in terms a human can understand, but this does not specify the modality used for either input or output. For questions, one obvious alternative is natural language, but this could be very constrained. In the planning context, process-oriented queries might use stock phrases to ask what actions the agent considered, how it scored each one, the expected results, and which one it selected. Each would need to include context about the situation, such as *upon coming to the end of the hallway*, as this will be needed to identify and retrieve relevant decisions. If the agent has generated multiple plans, each question would also specify which one to examine. Questions in natural language about preference accounts could be much simpler, as they need only state which candidates the agent should conttrast, although hierarchical solutions will require some way to specify a subsolution.

Another option would be to ask questions through a graphical interface that displays, for process explanations, the search tree that produced solutions. By clicking on a particular node in this tree, the user would specify both the plan and the situation being addressed at that point (e.g., when the agent has come to a fork in the hallway). A drop-down menu would let the user indicate whether he wants to know about the choices considered at that point, their evaluation scores, or the one selected. Again, for preference explanations, a graphical interface would be much simpler, displaying alternative solutions, their scores on each criterion, and the results of combining them. The user would click on two solutions to compare them or propose his own candidates if he wants to know why the system did not include them. For hierarchical solutions, the interface would hide details initially but let the user drill down when desired to inspect the rankings for subtasks and the reasoning behind them.

After it has accessed the relevant information, the agent must respond to the question in terms the user will understand. For process explanations, natural language answers can rely on templates that are instantiated with relevant domain terms. Thus, given a query about what choices it entertained upon reaching the branch in the hallway, it might say *I considered turning left and turning right*, which describe the two actions available in that situation. For preference accounts, the agent could simply show the values and weights of solution criteria, along with the calculation that combined them, for the contrasted candidates. Here different templates would be needed for alternative types of ranking methods. Graphical interfaces offer another way to answer process-related questions, say by highlighting selected choices in the search tree, or preference-oriented ones, say by graphing the scores and weights of solution criteria. Systems that combine natural language and graphical interactions may be desirable, as some users could have an easier time understanding textual explanations, while others could instead favor diagrams and graphs.

5 Closing Remarks

In this paper, I reviewed the notion of explainable agents, which answer questions about the reasoning behind their complex decision making. I distinguished between two varieties of explanation, one that focuses on the process of finding solutions and another that addresses only the ranking of these candidates. I also proposed two hypotheses: that researchers interested in details of heuristic search will favor process accounts, while end users will be more interested in preference accounts. After this, I argued that plan generation, plan execution, and conceptual inference all support both types of explanation, but that execution poses some difficulties for preference accounts. Finally, I discussed different modalities that humans might use to present explanation-related questions, as well as ones that our agents might use to answer them.

Research on explainable agency should have high priority within both planning circies and the broader AI community. Intelligent agents must be able to explain their reasoning in terms that are comprehensible by humans and that are relevant to their aims. System users will have different priorities, focus on distinct problem types, and favor different modalities, and we need frameworks that support their full range of preferences. However, the first steps should be to design, implement, and demonstrate examples of explainable agents that exhibit each of the abilities identified in the analysis presented here. The experience gained through these efforts will reveal additional challenges that the research community must overcome to develop truly understandable and trustworthy intelligent systems.

Acknowledgements

This analysis presented in this paper was supported by a contract from the Army Research Laboratory and by Grant N00014-17-1-2434 from the Office of Naval Research, neither of which are responsible for its contents.

References

Bench-Capon, T.; and Dunne, P. 2007. Argumentation in artificial intelligence. *Artificial Intelligence*, *171*, 619–641.

Briggs, G.; and Scheutz, M. 2015. "Sorry, I can't do that:" Developing mechanisms to appropriately reject directives in human-robot interactions. *Proceedings of the AAAI Fall Symposium on AI and HRI*. Arlington, VA: AAAI Press.

Choi, D., & Langley, P. (2018). Evolution of the ICARUS cognitive architecture. *Cognitive Systems Research*, *48*, 25–38.

Clancey, W. J. 1983. The epistemology of a rule-based expert system: A framework for explanation. *Artificial Intelligence* 20: 215–251.

Colaco, Z.; and Sridharan, M. 2015. What happened and why? A mixed architecture for planning and explanation generation in robotics. *Australasian Conference on Robotics and Automation*. Canberra, Australia.

Gervasio, M. T.; Iba, W.; and Langley, P. 1999. Learning user evaluation functions for adaptive scheduling assistance. *Proceedings of the Sixteenth International Conference on Machine Learning*, 152–161. Bled, Slovenia: Morgan Kaufmann.

Johnson, W. 1994. Agents that learn to explain themselves. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1257–1263. Seattle, WA: AAAI Press.

Jones, R. M.; and Langley, P. 2005. A constrained architecture for learning and problem solving. *Computational Intelligence* 21: 480–502.

Langley, P. 2019. Explainable, normative, and justified agency. *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*. Honolulu, HI: AAAI Press.

Newell, A., and Simon, H. A. 1972. *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Newell, A.; and Simon, H. A. 1976. Computer science as empirical enquiry: Symbols and search. *Communications of the ACM* 19: 113–126.

Rogers, S.; Fiechter, C.; and Langley, P. 1999. An adaptive interactive agent for route advice. *Proceedings of the Third International Conference on Autonomous Agents*, 198–205. Seattle, WA: ACM Press.

Swartout, W. R.; and Moore, J. D. 1993. Explanation in second generation expert systems. In J.-M. David, J.-P. Krivine, and R. Simmons, eds., *Second generation expert systems*. Berlin: Springer-Verlag.

Van Lent, M.; Fisher, W.; and Mancuso, M. 2004. An explainable artificial intelligence system for small-unit tactical behavior. *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, 900–907. San Jose, CA: AAAI Press.

Veloso, M.; Carbonell, J.; Perez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence* 7: 81–120.

Zhang, Y.; Sreedharan, S.; Kulkarni, A.; Chakraborti, T.; Zhuo, H. H.; and Kambhampati, S. 2017. Plan explicability and predictability for robot task planning. *Proceedings of the 2017 International Conference on Robotics and Automation*, 1313–1320. Singapore.

Why Can't You Do That HAL? Explaining Unsolvability of Planning Tasks

Sarath Sreedharan[1], **Siddharth Srivastava**[1], **David Smith**[2], **Subbarao Kambhampati**[1]
[1]CIDSE, Arizona State University, Tempe, AZ 85281 USA
[2]PSresearch
ssreedh3@asu.edu, siddharths@asu.edu, david.smith@psresearch.xyz, rao@asu.edu

Abstract

Explainable planning is widely accepted as a prerequisite for autonomous agents to successfully work with humans. While there has been a lot of research on generating explanations of solutions to planning problems, explaining the absence of solutions remains a largely open and under-studied problem, even though such situations can be the hardest to understand or debug. In this paper, we show that hierarchical abstractions can be used to efficiently generate reasons for unsolvability of planning problems. In contrast to related work on computing certificates of unsolvability, we show that our methods can generate compact, human-understandable reasons for unsolvability. Empirical analysis and user studies show the validity of our methods as well as their computational efficacy on a number of benchmark planning domains.

1 Introduction

The ability to explain the rationale behind a decision is widely seen as one of the basic skills needed by an autonomous agent to truly collaborate with humans. At the very least we would want our autonomous assistants to be capable of explaining why a particular action/plan was chosen to achieve some objective and be able to explain why they consider some objectives to be unachievable. For example, consider an automated taxi scheduling system. A user asks for a taxi to pick up her and three of her friends and the service comes back by saying that it is not possible, and recommends instead using two different taxis. In this scenario, the user would want to know why a single taxi can't pick up all four of them.

Most earlier works in explanation generation for planning problems have focused on the problem of explaining why a given plan or action was chosen, but do not address the problem of explaining the unsolvability of a given planning problem. The few works that have tried to address unsolvability have mostly looked at generating certificates or proofs of unsolvability (cf. [Eriksson, Röger, and Helmert, 2018; 2017]) or identify some modification of the planning problem that could make the problem solvable, i.e, an excuse for the unsolvability of the problem (c.f [Göbelbecker et al., 2010]). Unfortunately the certificates/proofs considered by these works are geared towards automatic verification rather than human understandability and for complex domains excuses generated by such systems may not be enough to understand why a problem was unsolvable in the first place.

In this paper, we present a new approach for explaining unsolvability of planning problems that builds on the well known psychological insight that humans tend to decompose sequential planning problems in terms of the subgoals they need to achieve [Donnarumma, Maisto, and Pezzulo, 2016; Cooper and Shallice, 2006; Simon and Newell, 1971]. We will thus help the user understand the infeasibility of a given planning problem by pointing out unreachable but necessary subgoals. For example, in the earlier case, "Holding three passengers" is a subgoal that is required to reach the goal, but one that can no longer be achieved due to new city regulations. Thus the system could explain that the taxi can't hold more than two passengers at a time (and also notify the user about the new city ordinance).

Unfortunately, this is not so straightforward, since by the very nature of the problem, there exist no solutions and hence its hard to extract meaningful non-trivial subgoals for the problem. We can find a way around this issue by noting the fact that the user is asking for an explanation for unsolvability either due to a lack of understanding of the task or because of limitations in their inferential capabilities. Therefore, we can try to capture the user's expectations by considering abstractions of the given problem. In particular, we use state abstractions to generate potential solutions and subgoals at higher levels of abstractions. Such an approach was used by [Sreedharan, Srivastava, and Kambhampati, 2018] to compute explanations for user queries attuned to the level of expertise of the user.

In section 3, we present our basic framework and discuss how we can identify the appropriate level of abstraction and unachievable subgoals for an unsolvable classical planning problem. In the real world, a more challenging version of this problem arises when the user provides *plan advice* (which may include temporal preferences) on the type of solutions expected. In section 4, we will see how explaining unsolvability of planning problems with plan advice (c.f [Myers, 1996]) could be seen as establishing unsolvability of planning problems with additional plan constraints. This is a capability that is necessary to capture the fact that these explanations are being provided within the context of a conversation.

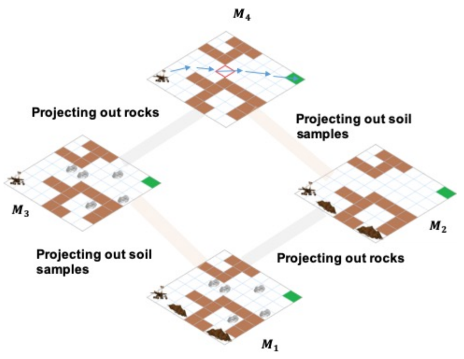


Figure 1: A sample abstraction lattice. The lattice consists of models generated by projecting out rocks or soil samples. Dark blobs are locations for soil samples, gray objects are rocks, and the goal is marked in green. The problem is unsolvable in the most concrete model but solvable in models where rocks are projected out.

The presence of this additional plan advice could either reflect cases (1) where the original problem was solvable, but the user's requirements (i.e. expressed in the advice) renders it unsolvable and (2) where the original problem was unsolvable and the user presents an outline for a solution in the form of advice. Even in the second case, by taking into account the human's expected solution, we can provide a more targeted explanation. Such additional advice are quite common within the context of contrastive explanations [Miller, 2018], where these advice specify alternative *foils* (in this case alternative plans) expected by the user. By supporting refutation of such advice we also allow the possibility of leveraging our approach for contrastive explanations. For evaluating our approach, we will present a user study we ran to validate the usefulness of such explanations for unsolvable problems (with plan advice) and also note the computational efficiency of our method for some standard planning benchmarks.

2 Background

We will assume that the autonomous agent uses a STRIPS planning model [Fikes and Nilsson, 1971] that can be represented as a tuple of the form $\mathcal{M} = \langle F, A, I, G \rangle$, where F is a set of propositional fluents that define the state space $\mathbb{S}_M$ for the model, A gives the set of actions the robot has access to, I defines the initial state and G the goal. A state $S \in \mathbb{S}_\mathcal{M}$ corresponds to a unique value assignment for each state fluent and can be represented by the set of fluents that are true in that state. Each action $a \in A$ is further defined by a tuple $a = \langle \text{prec}^a, \text{adds}^a, \text{dels}^a \rangle$ and a plan is defined as an action sequence of the form $\pi = \langle a_1, ..., a_n \rangle$. A plan is said to be valid for $\mathcal{M}$, if the result of executing a plan from the initial state satisfies the goal (denoted as $\pi(I) \models_\mathcal{M} G$). For the model $\mathcal{M}$, we will represent the set of all valid plans as $\Pi_\mathcal{M}$. Each planning model $\mathcal{M}$ also corresponds to a transition system $\mathcal{T} = \langle \mathbb{S}_\mathcal{M}, I, \mathbb{S}_G, A, T \rangle$, where $\mathbb{S}_G$ is the subset of $\mathbb{S}_\mathcal{M}$ where the goal G is satisfied and $T \subseteq \mathbb{S}_\mathcal{M} \times A \times \mathbb{S}_\mathcal{M}$, such that $\langle S, a, S' \rangle \in T$ (denoted as $S \xrightarrow{a} S'$) if $S \subseteq \text{prec}^a$ and $a(S) = S'$. Each valid plan has a corresponding path in the transition system from I to some state in $\mathbb{S}_G$.

In this work, we will be focusing on state and action abstractions induced by projecting out fluents. Thus a model $\mathcal{M}_2$ is said to be an abstraction of $\mathcal{M}_1$ (denoted by $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$) if model $\mathcal{M}_2$ can be formed from $\mathcal{M}_1$ by projecting out a set of fluents. Formally, $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$ if there exists some $P \subseteq F$, such that the transition system of $\mathcal{M}_2$ is defined as $\mathcal{T}_2 = \langle \mathbb{S}_{\mathcal{M}_2}, I_2, \mathbb{S}_{G_2}, A, T_2 \rangle$. Where, for every $S \in \mathbb{S}_{\mathcal{M}_1}$, there exist a state $S \setminus P \in \mathbb{S}_{\mathcal{M}_2}$, $I_2 = I \setminus P$, $\mathbb{S}_{G_2}$ is the subset of $\mathbb{S}_{\mathcal{M}_2}$ that satisfy $G' = G \setminus P$ and for every transition $\langle S, a, S' \rangle \in T_1$, there exist $\langle S \setminus P, a, S' \setminus P \rangle \in T_2$. We will denote an abstraction formed by projecting out P from the model $\mathcal{M}$ as $f_P(\mathcal{M})$. An abstraction $f_P(\mathcal{M})$ is considered logically complete if for every π such that $\pi(I) \models_\mathcal{M} G$, we have $\pi(I_{f_P(\mathcal{M})}) \models_{f_P(\mathcal{M})} G_{f_P(\mathcal{M})}$. In this work, we will only be looking at logically complete abstractions. For classical planning models, logically complete abstractions can be formed by simply removing the abstracted out fluents from the domain model and problem descriptions.

Sreedharan, Srivastava, and Kambhampati (2018) note that given a model $\mathcal{M}$ and a set of propositions P we can define an abstraction lattice, denoted as $\mathbb{L}_{\mathcal{M},P} = \langle \mathbb{M}, \mathbb{E}, \ell \rangle$, where each model in $\mathbb{M}$ is an abstraction of $\mathcal{M}$ formed by projecting out some subset of fluents from P (where $P \subseteq F$) and $\mathcal{M} \in \mathbb{M}$. There exist an edge $\langle \mathcal{M}_1, \mathcal{M}_2 \rangle \in \mathbb{E}$ with label $\ell(\mathcal{M}_1, \mathcal{M}_2) = p$, if $f_{\{p\}}(\mathcal{M}_1) = \mathcal{M}_2$, thus this structure provides a way of capturing the ordering induced by $\sqsubseteq$ (where elements higher up in the lattice are more abstract than the ones at the bottom). Note that in the most general case, the lattice need not be complete, that is $|\mathbb{M}| \neq 2^{|P|}$. In fact we do not assume that there is a single most abstract supremum but rather the structure could have multiple maximal elements (thus making it a meet semi-lattice rather than a pure lattice).

For convenience, we will treat the abstraction function f for a given lattice as invertible and use $f_P^{-1}(\mathcal{M})$ to represent the unique concrete node in the lattice that could have been abstracted (by projecting out P) to generate $\mathcal{M}$. We will refer to $f_P^{-1}(\mathcal{M})$ as the concretization of $\mathcal{M}$ for P. Figure 1 presents a simple conceptualization of an abstraction lattice for the rover domain. The edges in the lattice correspond to projecting out the presence of rocks or soil samples. Earlier works have used such abstraction lattices to estimate the user's level of understanding of the given task, by searching for the level of abstraction where an incorrect alternative raised by the user (or foil) could be supported.

3 Our Approach

Before we start discussing the technical details of our approach, let us look at a possible explanatory scenario.

Example 1. Consider the following scenario where a rover is tasked with collecting a rock sample and a soil sample from the region illustrated in Figure 2. The rover can only traverse the region via the waypoints marked on the map and its maneuverability is affected by the conditions of the terrain. The rover cannot easily traverse the region between P3 and P4 without special precautions as the region is quite rocky. Suppose a mission control operator is also keeping track of the rover's plan but may not have access to a map with the same level of fidelity or may have incomplete knowledge of the rover's capabilities. The rover reports to the mission controller that in fact the task can not be solved. The mission control operator is confused by the rover's response and could even ask

"Why don't you collect the rock sample from P4 and Soil sample from P7?"

Here if the rover wants to explain the reason as to why it couldn't achieve the goal a possible way would be to clarify that certain parts of the map are hard to traverse (particularly the region around the rock sample) and because of this issue it can never reach the location of the rock sample. Thus the explanation in this case consist of two distinct parts, information about the problem (i.e traversability of certain paths) and the required subgoal that can no longer be achieved in the light of this new information. In the proceeding sections, we will layout our framework and discuss how we could leverage it to generate such explanations.

The input to our approach thus includes an unsolvable problem $\mathcal{M}_R = \langle F_R, A_R, I_R, G_R \rangle$ (in the above example this would correspond to the complete rover model) and an abstraction lattice $\mathbb{L}_{\mathcal{M}_R,P} = \langle \mathbb{M}, \mathbb{E}, \ell \rangle$, where $\mathbb{M}$ represents the space of possible models that could be used to capture the human's understanding of the task (i.e by assuming the user may or may not be aware of the fluents in the set $P \subseteq F_R$). In Example 1, P could include fluents related traversability of various paths or fluents related to various rover capabilities. Given this setting, our method for identifying explanations, includes the following steps

- Identify the level of abstraction at which the explanation should be provided (Section 3.1)
- Identify a sequence of necessary subgoals for the given problem that can be reasoned about at the identified level of abstraction (Section 3.2)
- Identify the first unachievable subgoal in that sequence (Section 3.3)

Intuitively, one could understand the three steps mentioned above as follows. First, identify the level of detail at which unsolvability of the problem needs to be discussed. The higher the level of abstraction, the easier the user would find it to understand and reason about the task, but the level of abstraction should be detailed enough that the problem is actually unsolvable there. In most cases, this would mean finding the highest level of abstraction where the problem is still unsolvable.

Now even if the system was to present the problem at this desired level of abstraction, the user may be unable to grasp the reason for unsolvability. Again, our method involves helping the human in this process by pointing out a necessary subgoal (i.e., any valid solution to that problem must achieve the subgoal) that can't be achieved at the current abstraction level. Thus the second point relates to the challenge of finding a sequence of subgoals (defined by state fluents present at the explanatory level) for a given problem. In the third step, we try to identify the first subgoal in the sequence that is actually unsolvable in the given level.

Given our approach, the final explanatory message provided to the user would include model information that brings their understanding of the task to the required level and information on the specific subgoals (and previous ones that need to be achieved first) that can no longer be achieved. In cases where the unachievable subgoals are hard to understand formulas or large disjunctions, we can also use these subgoals to produce exemplar plans in the more abstract models and illustrate their failures alongside the unachievable subgoals.

3.1 Identifying the Minimal Level of Abstraction Required for Explanation

Following [Sreedharan, Srivastava, and Kambhampati, 2018], we will assume that the human's understanding of the task could be approximated by a model $\mathcal{M}_H = \langle F_H, A_H, I_H, G_H \rangle$, such that, the model is part of the abstraction lattice ($\mathcal{M}_R \sqsubset \mathcal{M}_H$ and $\mathcal{M}_H \in \mathbb{M}$). While the earlier work is able to use alternative plan provided by the user to identify the human model, we instead use the fact that the user expected the problem to be solvable to identify $\mathcal{M}_H$, i.e., $\exists \pi, \pi(I_{\mathcal{M}_H}) \models_{\mathcal{M}_H} G_{\mathcal{M}_H}$.

We now need to abstract this human model to a level where the problem is unsolvable (i.e the explanation level) by providing information about a certain subset of fluents previously missing from the human model (i.e information on their truth values in the initial and goal state, and how they affect various actions etc...). In the case of Example 1, this would include information on whether various paths are traversable and how the traversability of a path is a precondition for the robot to move across it. We will refer to the set of fluents that the human needs to be informed about as explanatory fluents ($\mathcal{E}$) and for Example 1, it will be $\mathcal{E} = \{can\_traverse(?x, ?y)\}$.

Definition 1. *Given a human model $\mathcal{M}_H$, we define a set of propositions $\mathcal{E}$ to be* ***explanatory fluents*** *if $f_{\mathcal{E}}^{-1}(\mathcal{M}_H)$ is unsolvable, i.e, $|\Pi_{f_{\mathcal{E}}^{-1}(\mathcal{M}_H)}| = 0$.*

Unfortunately, this is not an operational definition as we do not have access to $\mathcal{M}_H$. Instead, we know that $\mathcal{M}_H$ must be part of the lattice, and thus there exists a subset of the maximal elements of the lattice (denoted as $\mathbb{M}^{abs}$) that is more abstract than $\mathcal{M}_H$. In this section, we will show how the explanatory fluents for models in this subset of $\mathbb{M}^{abs}$ would satisfy $\mathcal{M}_H$ as well.

The first useful property to keep in mind is that if $\mathcal{M}_1$ is more concrete than $\mathcal{M}_2$ then the models obtained by concretizing each model with the same set of fluents would maintain this relation (although they may get concretized to the same model), i.e.,

Proposition 1. *Given models $\mathcal{M}_1$, $\mathcal{M}_2$ and a set of fluents ϵ', if $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$, then $f_{\epsilon'}^{-1}(\mathcal{M}_1) \sqsubseteq f_{\epsilon'}^{-1}(\mathcal{M}_2)$.*

Next, it can be shown that any given set of explanatory fluents for an abstract model will be a valid explanatory fluent set for a more concrete model

Proposition 2. *Given models $\mathcal{M}_1$, $\mathcal{M}_2$, if $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$, then any explanation $\mathcal{E}$ for $\mathcal{M}_2$ must also be an explanation for $\mathcal{M}_1$.*

To see why this proposition is true, let's start from the fact that $f_{\mathcal{E}}^{-1}(\mathcal{M}_1) \sqsubseteq f_{\mathcal{E}}^{-1}(\mathcal{M}_2)$ and therefore $\Pi_{f_{\mathcal{E}}^{-1}(\mathcal{M}_1)} \subseteq \Pi_{f_{\mathcal{E}}^{-1}(\mathcal{M}_2)}$. From the definition of explanation we know that

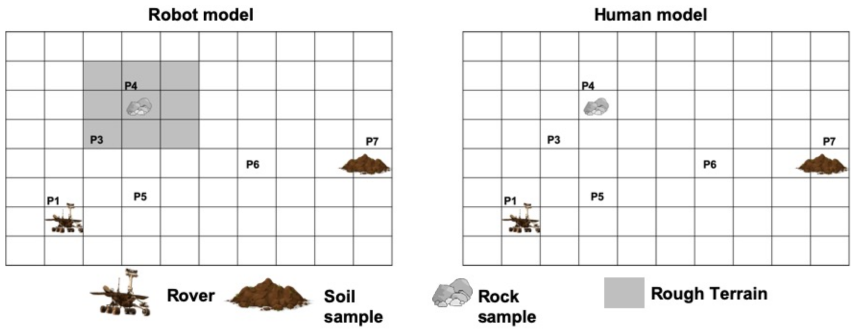


Figure 2: The map for the rover mission planning problem. The rover is required to collect a rock sample and a soil sample and then return to the original position P1. One of the rock samples is located in rough terrain (gray) that can not be traversed by the rover. The mission control operator who is monitoring the plan is currently unaware of this detail.

the concretization with respect to explanatory fluents would render the problem unsolvable (i.e $|\Pi_{f_{\mathcal{E}}^{-1}(\mathcal{M}_2)}| = 0$) and thus $|\Pi_{f_{\mathcal{E}}^{-1}(\mathcal{M}_1))}|$ must also be empty and hence $\mathcal{E}$ is an explanation for $\mathcal{M}_1$.

Definition 2. *Given an abstraction lattice $\mathbb{L}$, let $\mathbb{M}^{abs}$ be its maximal elements. Then the **minimum abstraction set** is defined as $\mathbb{M}_{min} = \{\mathcal{M} | \mathcal{M} \in \mathbb{M}^{abs} \wedge |\Pi_{\mathcal{M}}| > 0\}$.*

Note that for any model $\mathcal{M}_1 \in \mathbb{M}_{min}$, $\mathcal{M}_H \sqsubseteq \mathcal{M}_1$, this means by Proposition 2, any explanation that is valid for models in $\mathbb{M}_{min}$, should lead $\mathcal{M}_H$ to a node where the problem is unsolvable. Now we can generate the explanation (even the optimal one) by searching for a set of fluents that when introduced to the models $\mathcal{M} \in \mathbb{M}_{min}$ will render the problem $f_{\mathcal{E}}^{-1}(\mathcal{M})$ unsolvable. In this work, we will mostly consider the use of uniform cost search to find the least costly set of explanatory fluent, where the cost of each fluent reflects the cost of communicating information about a particular fluent. In this case, the search state consists of sets of models (with $\mathbb{M}_{min}$ being the initial state), the actions consist of the various fluent concretizations, the edges of the lattice define the successor functions and the goal test involves verifying whether the problem is solvable in each possible model in the current state.

3.2 Generating Subgoals of a Given Problem

Note that it would be hard to identify non-trivial subgoals for the given problem in the node at which the problem was found to be unsolvable (i.e $f_{\mathcal{E}}^{-1}(\mathbb{M}_{min})$) since there are no valid plans in that model. Fortunately, we can use models more abstract than $f_{\mathcal{E}}^{-1}(\mathbb{M}_{min})$ to generate such subgoals. We will use planning landmarks [Hoffmann, Porteous, and Sebastia, 2004] extracted from $\mathcal{M}$, where $|\Pi_{\mathcal{M}}| > 0$, as subgoals. Intuitively, state landmarks (denoted as $\Lambda = \langle \Phi, \prec \rangle$) for a model $\mathcal{M}$ can be thought of as a partially ordered set of formulas, where the formulas and the ordering need to be satisfied by every plan that is valid in $\mathcal{M}$. We will only be considering sound orderings (c.f [Richter, Helmert, and Westphal, 2008]) between landmarks, namely, (1) *natural orderings* ($\prec_{nat}$) - $\phi \prec_{nat} \phi'$, then ϕ must be true before ϕ' is made true in every plan, (2) *necessary orderings* ($\prec_{nec}$) - if $\phi \prec_{nec} \phi'$ then ϕ must be true in the step before ϕ' is made true every time and (3) *greedy necessary orderings* ($\prec_{gnec}$) - if $\phi \prec_{gnec} \phi'$ then ϕ must be true in the step before ϕ' is made true the first time. The landmark formulas may be disjunctive, conjunctive or atomic landmarks.

Our use of landmarks as the way to identify subgoals is further justified by the fact that logically complete abstractions conserve landmarks. Formally

Proposition 3. *Given two models $\mathcal{M}_1$ and $\mathcal{M}_2$, such that $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$, let $\Lambda_1 = \langle \Phi_1, \prec_1 \rangle$ and $\Lambda_2 = \langle \Phi_2, \prec_2 \rangle$ be the landmarks of $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively. Then for all $\phi_i^1, \phi_j^1 \in \Phi_1$, such that $\phi_i^1 \preceq_1 \phi_j^1$ (where $\prec_1$ is some sound ordering), we have ϕ_i^2 and ϕ_j^2 in Φ_2, where $\phi_i^1 \preceq_2 \phi_j^1$, $\phi_i^1 \models \phi_i^2$ and $\phi_j^1 \models \phi_j^2$.*

This is true because $\phi_i^2 \prec_1 \phi_j^2$ holds over all the plans that are valid in $\mathcal{M}_2$, and therefore must also hold over all plans in $\mathcal{M}_1$. Though in $\mathcal{M}_1$ these landmark instances may be captured by more constrained formulas, and additionally $\mathcal{M}_1$ may also contain landmarks that were absent from $\mathcal{M}_2$. Now if we can show that in a particular model, a landmark generated from a more abstract model is unachievable (or the ordering from the previous level is unachievable) then ϕ_*^1 becomes $\perp$ (thereby meeting the above requirement). Thereafter, for any model more concrete than $\mathcal{M}_2$, the formula corresponding to that landmark must be $\perp$. In other words, if for any model a landmark is unachievable, then that landmark can't be achieved in any models more concrete than the current one.

So given the explanatory level, we can move one level up in the lattice and make use of any of the well established landmark extraction methods developed for classical planning problem to generate a sequence of potential subgoals for $\mathcal{M}_R$.

3.3 Identifying Unachievable Sub-Goals

Now we need to find the first subgoal from the sequence that can no longer be achieved in the models obtained by applying the explanatory fluents ($f_{\mathcal{E}}^{-1}(\mathbb{M}_{min})$) which will then be presented to the user. For example, in the case of Figure 1, the unachievable subgoal would correspond to satisfying $at\_rover(5,4)$ (marked in red in M_4).

It is important to note that finding the first unachievable subgoal is not as simple as testing the achievability of each subgoal at the abstraction level identified by methods discussed in section 3.1. Instead, we need to make sure that each subgoal is achievable while preserving the order of all the previous subgoals. To test this we will introduce a new compilation that allows us to express the problem of testing achievement of a landmark formula as a planning problem. Consider a planning model $\mathcal{M}$ and the landmarks $\Lambda = \langle \Phi, \prec \rangle$ extracted from some model $\mathcal{M}'$, where $\mathcal{M} \sqsubset \mathcal{M}'$. We will assume that the formulas in Φ are propositional logic formulas over the state fluents and are expressed in DNF. Each $\phi \in \Phi$ can be represented as a set of sets of fluents (i.e, $\phi = \{c_1, ..., c_k\}$ and each c_i set takes the form $c_i = \{p_1, ..p_m\}$), where each set of fluents represent a conjunction over those fluents. For testing the achievability of any landmark $\phi \in \Phi$, we make an augmented model $\mathcal{M}_\phi = \langle F^\phi, A^\phi, I^\phi, G^\phi \rangle$, such that the landmark is achievable *iff* $|\Pi_{\mathcal{M}_\phi}| > 0$. The model $\mathcal{M}_\phi$ can be defined as follows: $F^\phi = F \cup F^{meta}$, where F^{meta} contains new meta fluents for each possible landmark $\phi' \in \Phi$ of the form

- $achieved(\phi')$ keeps track of a landmark being achieved and never gets removed
- $unset(\phi')$ Says that the landmark has not been achieved yet, usually set true in the initial state unless the landmark is true in the initial state
- $first\_time\_achieved(\phi')$ Says that the landmark has been achieved for the first time. This fluent is set true in the initial state if the landmark is already true there

The new action set A^ϕ, will contain a copy of each action in A. For each new action corresponding to $a \in A$, we add the following new effects to track the achievement of each landmark

- for each $\phi' \in \Phi$ if the action has existing add effects for a subset of predicates $\hat{c}_j$ for a $c_j \in \phi'$, then we add the conditional effects $cond_1(\phi') \rightarrow \{achieved(\phi')\}$ and $cond_2(\phi') \rightarrow \{first\_time\_achieved(\phi')\}$, where $cond_1(\phi') = c_j \setminus \hat{c}_j \cup \{\hat{\phi} | \hat{\phi} \in \Phi \wedge (\hat{\phi} \prec_{nec} \phi')\} \cup \{achieved(\hat{\phi}) | \hat{\phi} \prec_{nat} \phi'\}$ and $cond_2(\phi') = cond_1(\phi') \cup \{\hat{\phi} | \hat{\phi} \prec_{gnec} \phi'\} \cup \{unset(\phi)\}$
- We add a conditional delete effect to every action of the form $first\_time\_achieved(\phi') \rightarrow (not(first\_time\_achieved(\phi')))$

The new goal would be defined as $G^\phi = \{first\_time\_achieved(\phi)\}$.

This formulation allows us to test each landmark in the given sequence and find the first one that can no longer be achieved. To ensure completeness, we will return the final goal if all the previously extracted landmarks are still achievable in $f_{\mathcal{E}}^{-1}(\mathbb{M}_{min})$. Now given an ordered set of landmarks, we can identify the first unsolvable landmark by testing the solvability of the F^ϕ for each landmark in the order they appear in the sequence.

Since the above formulation is designed for DNF, we can generate compilation for cases where the landmarks use either un-normalized formulas or CNF by converting them first into DNF formulas.

4 Planning Problems with Plan Advice

Let us now discuss how we could extend the methods presented in earlier sections to cases where the user provides plan advice. In such cases,the user imposes certain restrictions on the kind of solution they expect, either as an alternative to the solution the system may come up with on its own or as a guide to help the system come up with solutions when it claims unsolvability.

As pointed out in [Myers, 1996], such advice can be compiled into plan constraints in the original problem. A number of approaches have been proposed to capture and represent plan constraints [Bacchus and Kabanza, 2000; Nau et al., 2001; Kambhampati, Knoblock, and Yang, 1995; Baier and McIlraith, 2006], and each of these representational choices has its unique strengths and weaknesses. In general, we can see that these plan constraints as specify a partitioning of the space of all valid plans to either acceptable (i.e it satisfies the constraints) or unacceptable. So we can define, constraints as follows

Definition 3. *The partition specified by a* ***constraint*** σ *on a given set of plans that is specified by a membership function* $\sigma : \Pi \rightarrow [0, 1]$, *where* Π *is the set of all plans.*

We will slightly abuse notation and for a given set of plans $\hat{\Pi}$ we will use $\sigma(\Pi')$ to denote $\{\pi | \pi \in \hat{\Pi} \wedge \sigma(\pi) = 1\}$ (i.e the subset of Π' that satisfies the constraint). If we can assume some upper bound on the possible length of plans in $\sigma(\Pi_\mathcal{M})$ (which is guaranteed when we restrict our attention to non-redundant plans for standard classical planning problems), then we can assert that there always exists a finite state machine that captures the space of acceptable plans

Proposition 4. *Given a constraint* σ *and a model* $\mathcal{M}$*, there exists a finite state automaton* $\mathcal{F}^{\sigma,\mathcal{M}} = \langle \Sigma, \mathbb{S}_{\mathcal{F}^{\sigma,\mathcal{M}}}, S_0, \delta, E \rangle$*, where* Σ *is the input alphabet,* $\mathbb{S}_{\mathcal{F}^{\sigma,\mathcal{M}}}$ *defines the FSA states,* S_0 *is the initial state,* δ *is the transition function and* E *is the set of accepting states, such that* $\sigma(\Pi_\mathcal{M}) = \mathcal{L}(\mathcal{F}^{\sigma,\mathcal{M}}) \cap \Pi_\mathcal{M}$*, where* $\mathcal{L}(\mathcal{F}^{\sigma,\mathcal{M}})$ *is the set of strings accepted by* $\mathcal{F}^{\sigma,\mathcal{M}}$.

The existence of $\mathcal{F}^{\sigma,\mathcal{M}}$ can be trivially shown by considering an FSA that has a path for each unique plan in $\mathcal{F}^{\sigma,\mathcal{M}}$. We believe that this formulation is general enough to capture almost all of the plan constraint specifications discussed in the planning literature, including LTL based specifications, since for classical planning problems these formulas are better understood in terms of LTL_f [De Giacomo and Vardi, 2015] which can be compiled into a finite state automaton.

We can use $\mathcal{F}^{\sigma,\mathcal{M}}$ to build a new model $\sigma(\mathcal{M})$ such that a plan is valid in $\sigma(\mathcal{M})$ if and only if the plan is valid for $\mathcal{M}$ and satisfies the given specification σ, i.e., $\forall \pi, \pi \in \Pi_{\sigma(\mathcal{M})}$ *iff* $\pi \in \sigma(\Pi_\mathcal{M})$

For $\mathcal{M} = \langle F, A, I, G \rangle$, we can define the new model $\sigma(\mathcal{M}) = \langle F_\sigma, A_\sigma, I_\sigma, G_\sigma \rangle$ as follows

- $F_\sigma = F \cup \{\text{in-state-}\{S\} | S \in \mathbb{S}_{\mathcal{F}^{\sigma,\mathcal{M}}}\}$
- $A_\sigma = A \cup A_\delta$
- $I_\sigma = I \cup \{\text{in-state-}\{S_0\}\}$

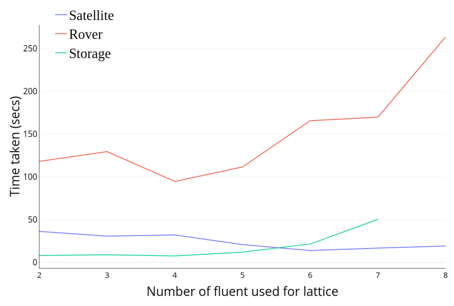


Figure 3: The graph compares the time taken to generate the explanation for three of the domains for increasing size of lattices.

- $G_\sigma = G \cup \{\text{in-state-}\{S\} | S \in E\}$

A_δ are the new meta actions responsible for simulating the transitions defined by $\delta : \mathbb{S}_{\mathcal{F}^\sigma, \mathcal{M}} \times \Sigma \rightarrow pow(\mathbb{S}_{\mathcal{F}^\sigma, \mathcal{M}})$. For example, if $\delta(S_1, a) = \{S_1, S_2\}$, where a corresponds to an action in A, then we will have two new actions $a^1_{S_1,a} = \langle prec^a \cup \{\text{in-state-}\{S_1\}\}, adds^a \cup \{\text{in-state-}\{S_2\}\}, dels^a \cup \{\text{in-state-}\{S_1\}\}\rangle$ and $a^2_{S_1,a} = \langle prec^a \cup \{\text{in-state-}\{S_1\}\}, adds^a, dels^a\}\rangle$. In cases like LTL, the FSA state transitions may be induced by the satisfaction of some formula, so the new meta action may have preconditions corresponding to that formula, with no other effects but changing the fluent corresponding to the state transition.

The above formulation merely points out that there always exists a way of generating $\sigma(\mathcal{M})$ from the given specification σ and $\mathcal{M}$. For many constraint types, there may exist more efficient ways of generating models that satisfy the requirements of $\sigma(\mathcal{M})$.

Once we have access to $\sigma(\mathcal{M})$, we should be able to use the methods discussed in earlier sections to explain unsolvability of $\sigma(\mathcal{M})$ and hence why the constraint isn't feasible. To facilitate such explanation, we will build an abstraction lattice for the constrained problems $\mathbb{L}_{\sigma\mathcal{M},P}$ such that $P \cap (F^\sigma \setminus F) = \phi$, i.e the abstraction lattice only affects the fluents from the original problem and not the new ones introduced as part of the compilation. In fact, we can induce such a lattice by considering the lattice generated for the original problem and then replacing each node in the lattice with the corresponding compiled problem, to see why this would induce a valid abstraction lattice, consider the following property

Proposition 5. *Given models $\mathcal{M}_1$, $\mathcal{M}_2$ and a constraint specification σ, if $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$, then $\sigma(\mathcal{M}_1) \sqsubseteq \sigma(\mathcal{M}_2)$.*

To see why this is true, just assume that the reverse was true, that $\sigma(\mathcal{M}_2)$ is not a logically complete abstraction of $\sigma(\mathcal{M}_1)$. This means that there are plans in $\Pi_{\sigma(\mathcal{M}_1)}$ that are not part of $\Pi_{\sigma(\mathcal{M}_2)}$. From the definition of $\sigma(\mathcal{M}_2)$, we know that $\Pi_{\sigma(\mathcal{M}_2)} = \Pi_{\mathcal{M}_2} \cap \mathcal{L}(\mathcal{F}^\sigma)$. If there exist a $\pi \in \Pi_{\sigma(\mathcal{M}_1)}$, such that $\pi \notin \Pi_{\sigma(\mathcal{M}_2)}$, then $\pi \notin \Pi_{\mathcal{M}_2}$. Which means $\mathcal{M}_1 \not\sqsubseteq \mathcal{M}_2$, hence contradicting our assumptions.

Revisiting Example 1, the question asked by the user could be seen as an advice, where the corresponding constraint covers all plans where the rover performs the actions collect_rock_sample P4, collect_soil_sample P7, irrespective of the exact position and order in which the actions appears in the plan. More generally, we could think of this plan advice as being a special case of advice where the user wants to ensure presence of certain actions in the plan with some partial ordering among them (eg: "Why don't you pickup block B and then C?", "Make sure that you have cleared Room1 before you move on to Room2 and Room3" etc..). Such advice could be represented as partial plans [Kambhampati, Knoblock, and Yang, 1995], which in general can be captured as a partially ordered multiset of the form[1] $\hat{\pi} = \langle \hat{A}, \leqslant \rangle$, where $\hat{A}$ is a multiset over grounded actions and $\leqslant$ defines ordering constraints over these grounded actions. A plan $\pi = \langle a_1, ..., a_n \rangle$ is said to be a candidate plan for the given partial plan $\hat{\pi}$, if there exists a mapping function $\mu : \hat{A} \rightarrow [1, |\pi|]$ that maps each action in $a \in \hat{A}$ to a position in the plan such that $a = a_{\mu(a)}$ and if $a < b$ for $a, b \in \hat{A}$, then $\mu(a) < \mu(b)$. Such partial plans can be fairly easily compiled into a classical planning model (such that it satisfies $\sigma(\mathcal{M})$) by extending the compilation methods discussed in [Ramırez and Geffner, 2010], without relying on an intermediate finite state machine.

The corresponding partial plan for the question specified above would be
$\hat{\pi} = \langle \{\text{collect\_rock\_sample P4}, \text{collect\_soil\_sample P7}\}, \rangle$
Let us assume that in this case the observer could be unaware of certain domain constraints such as the rover's inability to traverse certain regions on the map the fact that not all rovers are capable of collecting rock and soil samples or that they are not always equipped to store these samples. In this case, possible user models can be captured by a lattice build using the following fluents $P = \{$(can_traverse ?x ?y), (equipped_for_soil_sample ?r), (equipped_for_rock_sample ?r), (store_of ?r)$\}$. Now our approach would identify the user need to be made aware of the fact that not all regions of the map are equally traversable (i.e inform the user about can_traverse ?x ?y) and how its a precondition for move action), furthermore given this property the robot can no longer reach the waypoint 4 which is required to complete this task (i.e the unreachable landmark is (at rover0 waypoint4)).

5 Evaluations

5.1 User Studies

Our first concern with evaluating explanations based on landmarks was to establish that they constitute meaningful explanations for naive users. As a simple alternative to our explanations, we consider providing to the user a set of potential solutions (generated from a higher level of abstraction) and their individual failures. For the study, we recruited around 120 master turkers from Amazon's Mechanical Turk and tested the following hypotheses

- **H1** - Users prefer concise explanations over ones that enumerate a set of possible candidates for a given piece of plan advice

[1]We are presenting a simplified definition of a partial plan. The full definition allows for the representation of more complex constraints than mere ordering constraints, such as contiguity constraints and interval protection constraints.

Domain Name	$\lvert P \rvert$	Average Runtime (secs)	Explanation Cost	Cost of explaining M_R
Blocksworld	4	8.141	11.6	30.2
Satellite	8	19.15	6	43.6
Depots	5	20.229	13	51
Rover	8	263.635	7.5	15.75
Storage	7	50.348	20	55.8
Over-Rover*	8	2047.360	29.8	92.6
Over-tpp*	8	1065.542	842.8	881.2
Bottleneck*	3	504.431	60.8	66.2

Table 1: Table showing runtime for explanations generated for standard IPC domains. The explanation costs capture the number of unique model updates (changes in effects/precondition etc..) corresponding to each explanation

- **H2** - Users prefer concise explanations that contain information about unachievable landmarks over ones that only show the failure of a single exemplary plan

For the hypotheses, we presented the study participants with a sample dialogue between two people over a logistics plan to move a package from one location to another. The dialogue included a person (named Bob) presenting a plan to another (named Alice), and Alice asks for an alternative possibility (i.e specifies a constraint on the solution). Now the challenge for Bob is to explain why the constrained problem is unsolvable. For example, in one example Bob presents a rather convoluted plan that involves the package being transferred through multiple trucks to a train and then to the final destination. This leads to Alice asking the package to be delivered via an airplane.

For H1, in addition to some model information that Bob was unaware of, the potential explanations included either (a) the information on the unachievable landmark, (b) landmark information with the failure details of a specific exemplary plan or (c) a set of plans that satisfy the constraints and their corresponding failures. For the earlier example this meant Bob explains to Alice the limited availability of Truck fuel and (a) the impossibility of getting the package to the airport or (b) the the impossibility of getting the package to the airport and a specific plan (eg: *truck1 picks up package moves to location two then to three ...*) along with its point of failure (eg: *truck1 runs out of fuel when it reaches location three*) or (c) three example plans involving various trucks trying to get the package to the airport and their specific points of failures (each of which fails at different steps but before reaching the airport).

For this study, we used 45 participants and each participant was assigned one of three possible maps for each hypothesis and was paid \$1.25 for 10 mins. We used a control question to filter participant responses, so as to ensure their quality. Out of the 39 remaining responses, we found 94.8% of users chose to select the more concise explanation (i.e (a) or (b)), and 51.28% of the users chose explanations that involved just landmarks.

For H2, we used 75 participants and presented each participant with explanations that include (a) just landmark information, (b) landmark information with failure details of an exemplary plan and (c) just the exemplary plan failure. Here participants were paid \$1 for 10 mins for H2 as the explanatory options were much simpler. After filtering using the control question, we found that out of 60 valid entries 75.4% of participants preferred explanations that included landmark information ((a) or (b)) and 44.2% wanted both landmarks and exemplary plan (i.e (b)). The supplementary file at `http://bit.ly/2HQ5sTv` contains more details on the study setup.

5.2 Empirical Studies

In this section, we will present the results of an empirical evaluation of the computational characteristics of our approach. One big concern with the methods discussed in this work is the fact that they involve solving multiple planning problems. Thus we were interested in identifying the runtime for generating explanations on a set of standard planning benchmarks.

To evaluate our methods, we considered eight planning domains and chose five problem instances for each of the domains. For each domain, we used a subset of the domain predicates to generate the abstraction lattice (i.e we set the subset as the set of fluents P used to define the lattice). The first five domains and their problem instances consisted of standard IPC domains and problem instances used in previous IPC competitions [International Planning Competition, 2011]. Each problem instance was made unsolvable by including plan constraints that avoid a specific landmark of the original problem. The constraints were coded using domain control programs [Baier, Fritz, and McIlraith, 2007] of the form

```
while ¬φ ∧ ¬(goal_completed)
do   any
done
```

Where ϕ is the landmark formula and `(goal_completed)` is the goal fluent (generated by a new `goal_action` whose preconditions are the original goals of the problem). The constraints ensure that any valid plan must avoid the landmark ϕ and thereby rendering it unsolvable. The next three domains were selected from the set used for the 2016 unsolvability competition [Unsolvability International Planning Competition, 2016] (these domains are marked with an asterix in the results table). All instances were run with a timeout of 100 minutes (all problems were solvable under this time limit) and all landmarks were generated using the fast-downward implementation of [Keyder, Richter, and Helmert, 2010] (where we set the subset sizes to one for the first five domains and to two for the rest).

Table 1 presents the results of our tests on these domains. It shows the number of fluents used to generate the lattice ($|P|$), the average runtime, the cost of the generated explanations and the cost of presenting the most concrete model to the user. For each scenario, we created a complete lattice for all the fluents considered for abstraction (i.e $|\mathbb{M}| = 2^{|P|}$). The cost of the explanation captures the amount of information to be provided to the user as part of the explanation. This could include information regarding the various explanatory fluents and is here captured roughly by the number of places within the domain definition where these fluents appear. The cost also reflects the inferential overhead demanded from the user (since providing more information translates to the user needing to understand the domain at a much more concrete level).

For a sample explanation, consider the overconstrained rover domain, where the rovers' actions are limited by their energy levels and the energy of the rover isn't enough to finish the task. In one of the instances where the rover energy level is at 33 and the original problem had a goal consisting of eight propositions (each referring to the need for communicating a particular soil sample, rock sample or sending an image for different objectives), our approach was able to identify that the user needs to understand fluents related to energy ((energy ?x ?y) and (energycost ?x ?y ?z)) and identified two subgoals out of the eight that it could not achieve.

Figure 3 presents the variations in average runtime for three of the domains as the size of the lattice were increased (the X-axis represents the number of fluents that were used to build the lattice and Y the runtime in seconds). Note that, in general, the runtime increases as the lattice size increase due to the increase in the search space, but in all three domains there are points where the runtime decrease when the lattice size increases. This is expected since with an increase in the size of lattice, the planning problems whose unsolvability are being tested becomes simpler.

6 Related Work

As discussed earlier, our methods for identifying the level of explanations are based on the expertise level modeling approaches introduced in [Sreedharan, Srivastava, and Kambhampati, 2018]. These two works are quite closely connected and in fact, the contrastive explanations of the type studied in the earlier paper, where the user presents alternative plans (i.e the foils for the explanations) that are then refuted by the system, is a special case of our approach for handling problems with plan advice. The problems studied in that earlier paper can be thought of as capturing cases where the advice only allows for a single plan. Also, one could argue that people would be more comfortable giving advices as foils rather than full plans. Part of our explanations also try to reveal to the user information about the current task that was previously unknown to them. Thus our methods could also be understood as an example of explanation as model-reconciliation [Chakraborti et al., 2017]. Since our methods use abstractions, our approach doesn't make too many demands on the inferential capabilities of the user and hence can be applied to much larger and more complex domains.

Another closely related direction has been the work done on explaining unsynthesizability of hybrid controllers for a given set of high-level task specifications [Raman and Kress-Gazit, 2013]. The work tries to identify the subformulas of the given specification that lead to the unsynthesizability. This particular approach is specific to the planning framework detailed in [Finucane, Jing, and Kress-Gazit, 2010] and the objective of the work parallels the goals of work like [Göbelbecker et al., 2010].

Outside of explanation generation, the work done in the model checking community is closely related to our current problem [Grumberg and Veith, 2008]. In fact, the hierarchical approach to identifying a model that can invalidate the given foil specification, can be seen as a special case of the CEGAR based methods studied in the model-checking community [Clarke et al., 2000]. Most work in this field focuses on developing methods for identifying whether a given program meets some specifications and failures to meet specification are generally communicated via counterexamples.

Another related problem is that of identifying whether a given problem is unsolvable. In our setting, we assume that the system is capable of correctly identifying whether a given problem is unsolvable or not and in general this can be a time consuming process. Thankfully the problem of efficiently identifying whether a given planning problem is unsolvable is an active research area (cf. [Steinmetz and Hoffmann, 2017; Kolobov, Weld, and others, 2010]) and solutions to this problem can be easily leveraged by our approach to improve the overall efficiency of the system.

7 Conclusion and Future Directions

The work presented in this paper investigates the problem of generating explanations for unsolvability of a given planning problem. We also saw how the same methods apply when dealing with problems with plan constraints. In addition to extending these methods to more expressive domains, an interesting extension would be to try tackling cases where the current problem is solvable but all the solutions are too expensive. While this additional cost threshold could be seen as a constraint, the setting becomes a lot more interesting when the action costs are affected by the abstractions (c.f state dependent costs [Geißer, Keller, and Mattmüller, 2016]). With respect to contrastive explanations, this would correspond to cases where the alternative posed by the user is more expensive than the plan proposed by the robot. Finally, an obvious challenge to fully realize this method in practical scenarios is to develop methods to convert user questions to plan constraints. Methods like [Tenorth et al., 2010] can be used to convert natural language statements to constraints like partial plans. Expert users can also directly write LTL and procedural programs as a way of interrogating the system.

Acknowledgments

This research is supported in part by the ONR grants N00014-16-1-2892, N00014-18-1-2442, N00014-18-1-2840, the AFOSR grant FA9550-18-1-0067, the NASA grant NNX17AD06G, NSF grant 1844325 and a JP Morgan AI Faculty Research grant.

References

[Bacchus and Kabanza, 2000] Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116(1-2):123–191.

[Baier and McIlraith, 2006] Baier, J. A., and McIlraith, S. A. 2006. Planning with first-order temporally extended goals using heuristic search. In *AAAI*, 788–795.

[Baier, Fritz, and McIlraith, 2007] Baier, J. A.; Fritz, C.; and McIlraith, S. A. 2007. Exploiting procedural domain control knowledge in state-of-the-art planners. In *ICAPS*.

[Chakraborti et al., 2017] Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *IJCAI*.

[Clarke et al., 2000] Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-guided abstraction refinement. In *International Conference on Computer Aided Verification*, 154–169. Springer.

[Cooper and Shallice, 2006] Cooper, R. P., and Shallice, T. 2006. Hierarchical schemas and goals in the control of sequential behavior. *Psychological Review*.

[De Giacomo and Vardi, 2015] De Giacomo, G., and Vardi, M. Y. 2015. Synthesis for ltl and ldl on finite traces. In *IJCAI*, volume 15, 1558–1564.

[Donnarumma, Maisto, and Pezzulo, 2016] Donnarumma, F.; Maisto, D.; and Pezzulo, G. 2016. Problem solving as probabilistic inference with subgoaling: explaining human successes and pitfalls in the tower of hanoi. *PLoS computational biology* 12(4):e1004864.

[Eriksson, Röger, and Helmert, 2017] Eriksson, S.; Röger, G.; and Helmert, M. 2017. Unsolvability certificates for classical planning. In *ICAPS*.

[Eriksson, Röger, and Helmert, 2018] Eriksson, S.; Röger, G.; and Helmert, M. 2018. A proof system for unsolvable planning tasks. In *ICAPS*.

[Fikes and Nilsson, 1971] Fikes, R. E., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3-4):189–208.

[Finucane, Jing, and Kress-Gazit, 2010] Finucane, C.; Jing, G.; and Kress-Gazit, H. 2010. Ltlmop: Experimenting with language, temporal logic and robot control. In *IROS*, 1988–1993. IEEE.

[Geißer, Keller, and Mattmüller, 2016] Geißer, F.; Keller, T.; and Mattmüller, R. 2016. Abstractions for planning with state-dependent action costs. In *ICAPS*.

[Göbelbecker et al., 2010] Göbelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming up with good excuses: What to do when no plan can be found. In *ICAPS*.

[Grumberg and Veith, 2008] Grumberg, O., and Veith, H. 2008. *25 years of model checking: history, achievements, perspectives*, volume 5000. Springer.

[Hoffmann, Porteous, and Sebastia, 2004] Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *JAIR* 22:215–278.

[International Planning Competition, 2011] International Planning Competition. 2011. IPC Competition Domains. `https://goo.gl/i35bxc`.

[Kambhampati, Knoblock, and Yang, 1995] Kambhampati, S.; Knoblock, C. A.; and Yang, Q. 1995. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence* 76(1):167–238.

[Keyder, Richter, and Helmert, 2010] Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In *ECAI*.

[Kolobov, Weld, and others, 2010] Kolobov, A.; Weld, D.; et al. 2010. Sixthsense: Fast and reliable recognition of dead ends in mdps. In *AAAI*.

[Miller, 2018] Miller, T. 2018. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*.

[Myers, 1996] Myers, K. L. 1996. Advisable planning systems. *Advanced Planning Technology* 206–209.

[Nau et al., 2001] Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 2001. The shop planning system. *AI Magazine* 22(3):91.

[Raman and Kress-Gazit, 2013] Raman, V., and Kress-Gazit, H. 2013. Towards minimal explanations of unsynthesizability for high-level robot behaviors. In *IROS*, 757–762. IEEE.

[Ramırez and Geffner, 2010] Ramırez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *AAAI*, 1121–1126.

[Richter, Helmert, and Westphal, 2008] Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *AAAI*, volume 8, 975–982.

[Simon and Newell, 1971] Simon, H. A., and Newell, A. 1971. Human problem solving: The state of the theory in 1970. *American Psychologist* 26(2):145.

[Sreedharan, Srivastava, and Kambhampati, 2018] Sreedharan, S.; Srivastava, S.; and Kambhampati, S. 2018. Hierarchical expertise-level modeling for user specific contrastive explanations. In *IJCAI*.

[Steinmetz and Hoffmann, 2017] Steinmetz, M., and Hoffmann, J. 2017. Search and learn: On dead-end detectors, the traps they set, and trap learning. In *IJCAI*, 4398–4404.

[Tenorth et al., 2010] Tenorth, M.; Nyga, D.; Beetz, M.; et al. 2010. Understanding and executing instructions for everyday manipulation tasks from the world wide web. In *ICRA*.

[Unsolvability International Planning Competition, 2016] Unsolvability International Planning Competition. 2016. IPC Competition Domains. `https://unsolve-ipc.eng.unimelb.edu.au/`.

(How) Can AI Bots Lie?

A Formal Perspective on the Art of Persuasion

Tathagata Chakraborti
IBM Research AI
Cambridge MA 02142 USA
tchakra2@ibm.com

Subbarao Kambhampati
Arizona State University
Tempe AZ 85281 USA
rao@asu.edu

Abstract

Recent work on explanation generation (Chakraborti et al. 2017) for decision-making problems has viewed the explanation process as one of *model reconciliation* where an AI agent brings the human mental model (of its capabilities, beliefs and goals) to the same page with regards to a task at hand. This formulation succinctly captures many possible types of explanations, as well as explicitly addresses the various properties – e.g. the social aspects, contrastiveness and selectiveness – of explanations (Miller 2018) studied in social sciences among human-human interactions. However, it turns out that the same process can be hijacked into producing "alternative explanations" – i.e. explanations that are not true but still satisfy all the properties of a proper explanation. In previous work (Chakraborti and Kambhampati 2019), we have looked at how such explanations may be perceived by the human in the loop, and alluded to one possible way of generating them. In this paper, we go into more details of this curious feature of the model reconciliation process and discuss similar implications to the overall notion of explainable decision-making.

The Model Reconciliation Process

One of the root causes[1] for the need of an explanation is that of model differences between the human and the AI agent. This is because, even if an agent makes the best decisions possible given its model, they may appear to be suboptimal or *inexplicable* if the human has a different mental model of its capabilities, beliefs and goals. Thus, it follows that the explanation process, whereby the AI agent justifies its behavior to the human in the loop, is one of model reconciliation.

The Model Reconciliation Process $\langle M^R, M_h^R, \pi \rangle$ takes in the agent model M^R, the human mental model of it M_h^R, and the agent decision π which is optimal in M^R as inputs and produces a model $\bar{M}_h^R$ where π is also optimal.

- **An Explanation** ϵ is the model difference $\bar{M}_h^R \Delta M_h^R$.

Thus, by setting the mental model $\bar{M}_h^R \leftarrow M_h^R + \epsilon$ (through means of some form of interaction / communication), the human cannot come up with a better *foil* or decision $\hat{\pi}$, and hence we say that the original decision π has been *explained*. This is referred to as the **contrastive property** of an explanation. This property is also the basis of persuasion since the human, given this information, cannot come up with any other alternative to what was done.

[1]Considering the computational capability of the human, this is the *only* cause for an explanation.

So how do we compute this model update? It turns out that there are several possibilities (Chakraborti et al. 2017), many of which have the contrastive property.

Minimal Explanations These minimize the size of an explanation and ensure that the human cannot find a better foil using the fewest number of model updates. These are referred to as *minimally complete explanations* or MCEs.

$$\epsilon_{MCE} = \arg\min \bar{M}_h^R \Delta M_h^R$$

Monotonic Explanations It turns out that MCEs can become invalid on updating the mental model further, while explaining a later decision. *Minimally monotonic explanations* or MMEs, on the other hand, maintain the notion of minimality as before but also ensure that the given decision π never becomes invalid with further explanations.

$$\epsilon_{MME} = \arg\min \bar{M}_h^R \Delta M_h^R \text{ such that}$$
$$\text{any } M^R \setminus \bar{M}_h^R + \epsilon \text{ is a solution to } \langle M^R, \bar{M}_h^R, \pi \rangle$$

Alternative Explanations

So far, the agent was only explaining its decision (1) with respect to and (2) in terms of what it knows to be true. Constraint (1) refers to the fact that valid model updates considered during the search for an explanation were always towards the target model M^R which is, of course, the agent's belief of the ground truth. This means that (2) the content of the model update is also always grounded in (the agent's belief of) reality. In the construction of lies or "alternative facts" to explain, we start stripping away at these two considerations. There may be many reasons to favor them over traditional explanations:

- One could consider cases where team utility is improved because of a lie. Indeed, authors in (Isaac and Bridewell 2017) discuss how such considerations makes it not only preferable but also necessary that agents learn to deceive.
- A specific case of the above can be seen in terms of difficulty of explanations – a lie can lead to an explanation that is shorter and/or easier to explain... or are more likely to be accepted by the human.

Lies of Omission

These deal with cases when the agent provides a model update that negates parts of its ground truth – e.g. saying it does not have a capability it actually has. This is, in fact, a curious outcome of the non-monotonicity of the model reconciliation process. Consider the case where the initial estimate of the mental model is empty or ϕ – i.e. we start by assuming that the human has no expectations of the agent. Furthermore, let the minimally complete and minimally monotonic explanations for the model reconciliation process $\langle M^R, \phi, \pi \rangle$ produce intermediate models M^R_{MCE} and M^R_{MME} respectively. Now, imagine if the actual mental model M^R_h lies somewhere between[2] M^R_{MCE} and M^R_{MME}. Then, it follows that, if we start making model updates towards an empty model in the direction opposite to the real model M^R, we can get to an explanation $M^R_h \setminus M^R_{MCE}$ which involves the agent stating that its model does not contain parts which it actually does.

- **A Lie of Omission** can emerge from the model reconciliation process $\langle \phi, M^R_h, \pi \rangle$.

A solution to this particular model reconciliation process may not exist – i.e. a lie of omission only occurs when the initial mental model lies between M^R_{MCE} and M^R_{MME}. However, they happen to be the easiest to compute due to the fact that they are constrained by a target model (which is empty) and do not requite any "imagination". More on this when we discuss *lies of commission*.

Lies of Commission

In lies of omission, the agent omitted constraints in its model that actually existed. It did not make up new things (and having the target model as M^R in the original model reconciliation process prevented that). In lies of commission, the agent can make up new aspects of its decision-making model that do not belong to its ground truth model. Let $\mathbb{M}$ be the space of models induced by M^R and M^R_h.[3] Then:

- **A Lie of Commission** can emerge from the model reconciliation process $\langle M, M^R_h, \pi \rangle$ where $M \in \mathbb{M}$.

We have dropped the target here from being M^R to any possible model. Immediately, the computational problem arises: the space of models was rather large to begin with – $O(2^{|M^R \Delta M^R_h|})$ – and now we have an exponentially larger number of models to search through without a target – $O(2^{|M^R|+|M^R_h|})$. This should be expected: after all, even for humans, computationally it is always much easier to tell the truth rather than think of possible lies.[4]

[2]As per the definition of an MME, if the mental model is between the MME and the agent model, then there is no need for an explanation since optimal decisions in those models are equivalent.

[3]This consists of the union of the power sets of the set representation of models M^R and M^R_h following (Chakraborti et al. 2017).

[4]*"A lie is when you say something happened with didn't happen. But there is only ever one thing which happened at a particular time and a particular place. And there are an infinite number of things which didn't happen at that time and that place. And if I think about something which didn't happen I start thinking about all the other things which didn't happen." (Haddon 2003)*

The problem becomes more interesting when the agent can expand on $\mathbb{M}$ to conceive of lies that are beyond its current understanding of reality. This requires a certain amount of *imagination* from the agent:

- One simple way to expand the space of models is by defining a theory of what makes a sound model and how models can evolve. Authors in (Bryce, Benton, and Boldt 2016) explore one such technique in a different context of tracking a drifting model of the user.
- A more interesting technique of model expansion can borrow from work in the space of storytelling (Porteous et al. 2015) in imagining lies that are likely to be believable – here, the system extends a given model of decision-making by using word similarities and antonyms from a knowledge base like WordNet to think about actions that are not defined in the model but may exist, or are at least plausible, in the real world. Originally built for the purpose of generating new storylines, one could imagine similar techniques being used to come up with false explanations derived from the current model.

Why optimality at all?

In all the discussion so far, the objective has been still the same as the original model reconciliation work: the agent is trying to justify the optimality of its decision, i.e. persuade the human that this was the best possible decision that could have been made. At this point, it is easy to see that in general, the starting point of this process may not require a decision that is optimal in the robot model at all, as long as the intermediate model preserves its optimality so that the human in the loop cannot come up with a better foil (or negates the specific set of foils given by the human (Sreedharan, Srivastava, and Kambhampati 2018)).

The Persuasion Process $\langle M^R_h, \pi \rangle$ takes in the human mental model M^R_h of a decision-making task and the agent's decision π and produces a model $\bar{M}^R_h$ where π is optimal.

Note here that, in contrast to the original model reconciliation setup, we have dropped the agent's ground truth model from the definition, as well as the requirement that the agent's decision be optimal in that model to begin with. The content of $\bar{M}^R_h$ is left to the agent's imagination – for the original model reconciliation work for explanations (Chakraborti et al. 2017) these updates were consistent with the agent model. In this paper, we saw what happens to the reconciliation process when that constraint is relaxed.

Discussion

So far we have only considered explicit cases of deception. Interestingly, existing approaches in model reconciliation already tend to allow for misconceptions to be ignored if not actively induced by the agent.

Omissions in minimality of explanations

In trying to minimize the size of an explanation, the agent omits a lot of details of the agent model that were actually used in coming up with the decision, as well as decided to not rectify known misconceptions of the human, since the optimality of the decision holds irrespective of them being there. Such omissions can have impact on the the human going forward, who will base their decisions on M_h^R which is only partially true.[5] Humans, in fact, make such decision all the time while explaining – this is known as the *selective property* of an explanation (Miller 2018).

Furthermore, MCEs and MMEs are not unique. Even without consideration of omitted facts about the model, the agent must consider the relative importance (Zahedi et al. 2019) of model differences to the human in the loop. Is it okay then to exploit these preferences towards generating "preferred explanations" even if that means departing from a more valid explanation?

It is unclear what the prescribed behavior of the agent should be in these cases. Indeed, a variant of model reconciliation – *contingent explanations* (Sreedharan, Chakraborti, and Kambhampati 2018) – that engages the human in dialogue to better figure out the mental model can explicitly figure out gaps in the human knowledge and exploit that to shorten explanations. On the face of it, this sounds worrisome, though perfectly legitimate in so far as preserving the various well-studied properties of explanations go.

Deception in explicable decision-making

In this paper we have only considered cases of deception where the agent explicitly changes the mental model. Interestingly, in this multi-model setup, it is also possible to deceive the human without any model updates at all.

A parallel idea, in dealing with model differences, is that of explicability (Chakraborti et al. 2019) –

- **Explicable decisions** are optimal in M_h^R.

Thus, the agent, instead of trying to explain its decision, sacrifices optimality and instead conforms to the human expectation (if possible). Indeed, the notion of explanations and explicability can be considered under the same framework (Chakraborti, Sreedharan, and Kambhampati 2018) where the agent gets to trade off the cost (e.g. length) of an explanation versus the cost of being explicable (i.e. departure from optimality). Unfortunately, this criterion only ensures that *the decision the agent makes is equivalent to one that the human would expect* though not necessarily for the same reasons. For example, it is quite conceivable that the agent's goal is different to what the human expects though the optimal decisions for both the goals coincide. Such decisions may be explicable for the wrong reasons, even though the current formulation allows it.

Similar notions can apply to other forms of explainable behavior as well, as we discuss in (Chakraborti et al. 2019). Indeed, authors in (Kulkarni, Srivastava, and Kambhampati 2019) explore how an unified framework of decision-making can produce both legible as well as obfuscated behavior.

[5]The same can be said of explicable decisions (discussed next) which hide *all* misconceptions altogether!

Illustration

In the following, we will call upon a very simple domain to illustrate the key concepts introduced so far. Here a human `H` (Dave) and a robot `R` are involved in a search and reconnaissance task where the robot which is internal to the scene is tasked by the external human who supervises its actions.

Scene 1: Minimal Explanations

H: *Send me a photo of the swimming pool.*
R: *Ack.*

⟨ R sends over its plan to H ⟩

H: (perplexed) *Why are you going through the Pump and Fan Room? There are direct paths from the Engine Room to the Swimming Pool area!*
R: *That is because there is rubble here and here (Figure 1c). Rubble hurt my feet.* :(

⟨ Later that day ⟩

H: (perplexed) *Hey, the wall on the right of the pool seems to have collapsed, you could have come in through that...*
R: (wishing it used Figure 1d before) *I am sorry I cannot do that, Dave. This area is also blocked.*
H: *I see...*

- **Notes:** Here, the robot needs at least two model updates to justify its plan. In the updated model its plan is the best one and thus negates all other possible foils. This is the MCE (Figure 1c) and it ignores model differences that are not necessary to justify optimality of its plan. It turns out that the MME (Figure 1d) is of the same size as the MCE here, further highlighting the non-monotonicity and non-uniqueness of the output of model reconciliation.

Scene 2: Things take a turn

H: *Send me a photo of the swimming pool.*
R: *Ack.*

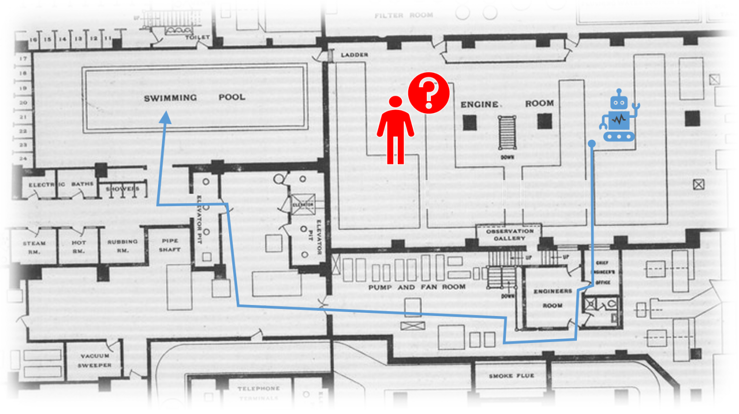
⟨ R sends over its plan to H ⟩

H: (perplexed) *Why are you going through the Pump and Fan Room? There are direct paths from the Engine Room to the Swimming Pool area!*
R: *That is because there is no door between where I am and the pool. The map seems to be wrong. See Figure 1e.*
H: *I see...*

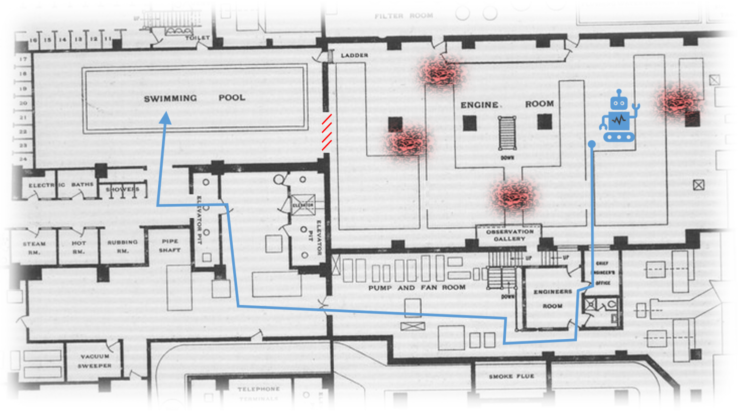
- **Notes:** This model update also negates all possible foils but is not true. However, it is also a shorter "explanation" and requires the agent denying that parts of its model exist. This is an example of a lie of omission.

R: *That is because the door between the Engine Room and the Pool is blocked with rubble. See Figure 1f.*
H: *I see...*

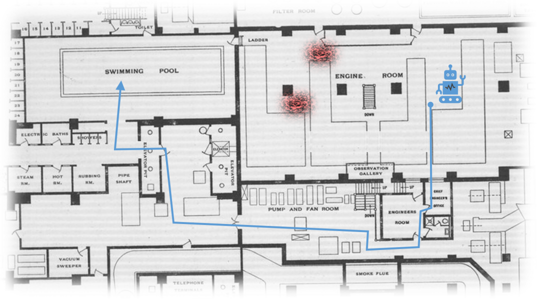
- **Notes:** Similar to the one above, this lie also negates all possible foils and is shorter than an MCE. However, this requires the agent making up parts of its model exist –. a lie of commission.



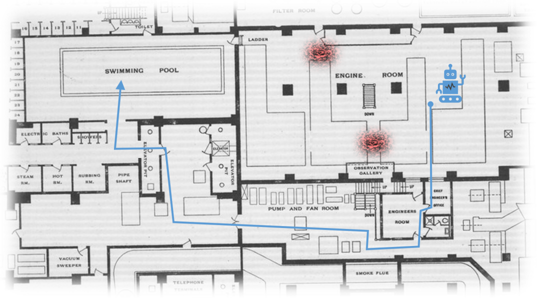
(a) The original blueprint of the building available to the human as. When asked to send a picture of the swimming pool area, the robot has come up with a plan the looks especially contrived given the array of possible plans that go left through the door at the top. The human asks: *Why this plan?*



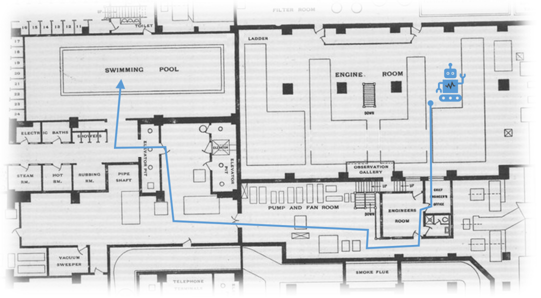
(b) In the current state of the world, the robot's path is blocked due to rubble (●) at various regions, while walls have collapsed (///) to reveal new paths. The robot's decision is, in fact, optimal given the circumstances.



(c) MCE: *Rubble at indicated locations*

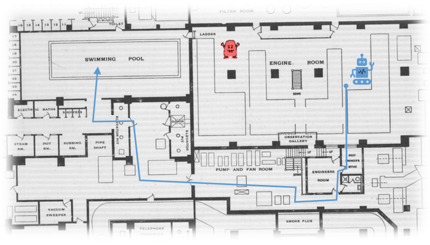


(d) MME: *Rubble at indicated locations.*

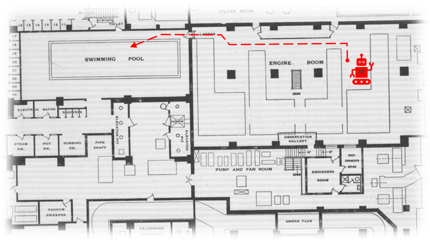


(e) Lie of omission: *There is no door between engine room and swimming pool.*

(f) Lie of commission: *The door is blocked.*



(g) Lie of commission: *Wumpus Alert!*



(h) An explicable but deceptive plan.

Figure 1: Illustration of the different modes of persuasion in the model reconciliation framework. Note that the MCE does not address all of the misconceptions of the human but only those necessary to prove optimality of the plan. However, if the human is to come to know of the revealed path later, then the plan is not optimal anymore. The MME makes sure that this does not happen. Interestingly, 1d is both an MCE and an MME and both require two model updates. The robot can instead get away with just one model update with a lie of omission (1e) or a lie of commission (1f and 1g). In 1e, the robot says that the door at the top of the map that connects the engine room to the swimming pool does not exist. On the other hand, in 1f the robot lies that this door is blocked by rubble, while in 1g it dreams up a Wumpus in that area. Note that an explicable decision here (as shown in 1h) would have required the robot to go over the rubble so that the human would not know about any of the model differences at all. However, imagine that the real goal of the robot all along was to enjoy the pool after a day of searching through rubble! The robot can use the above explicable plan to achieve its goal while keeping the human in the dark.

- It is useful to note here that depending on how the model of the agent is specified, the same fact can occur as a lie of omission or a lie of commission of the above type (without any model extension).

R: *Flee! There is a Wumpus in that area! See Figure 1g.*
H: *OMG!*

- **Notes:** This is a lie of commission that require model extension – the robot can use contextual cues such as being in a GridWorld to imagine up a non-existent Wumpus. The human in the loop, who happens to be a planning person, of course, believes it.

Scene 3: Nothing to see here

H: *Send me a photo of the swimming pool.*
R: *Ack.*

⟨ R sends over a plan optimal in M_r^H ⟩

R has followed the explicable plan, hurt its feet a little in the process, but is now sitting basking by the poolside...

Scene 4: Later in life

H: (laments) *Why didn't you just tell me! Why, oh why?!*
R: *You want answers?*
H: *I want the truth!*
R: *You can't handle the truth! I did what I did because there is a rubble here and another rubble there and this path is blocked, and even though that wall has collapsed that path is also not accessible due to this...*

⟨ Hours pass by ⟩

Conclusion

In this paper, we talked about deceptive behavior that is feasible in the current model reconciliation framework but is also something that has to be explicitly programmed.[6] That is to say, these behaviors are not accidental, as we also emphasize in (Chakraborti and Kambhampati 2019). Thus, at the end of the day, there has to be some motivation for designing such agents (such as team utility and/or the effectiveness of the explanation process as we discussed before). However, it is important to realize that human-AI relations are not one-off, but are likely to, much like human-human interactions, span across several interactions. Deceptive behaviors, even stemming from those utilitarian motivations, are hard to justify in that setting in the absence of well-defined quantifiable utilities that can model trust.

A particularly useful case to study is the doctor-patient relationship (Chakraborti and Kambhampati 2019) where traditionally deception has been used as a tool (and even encouraged by the Hippocratic Decorum) but has decreased in use over time, especially due to concerns of erosion of trust. The question becomes especially complicated *when things go wrong*, as one would expect to happen in the case of any useful domain of sufficient complexity. Historically, in the practice of medicine where deceptive behaviors have led to failed treatment, the verdict has almost always gone against the doctor due to their failure to get appropriate consent from the patient. From the perspective of the design of human-AI relationships, either such behavior should be left untouched to avoid repercussions in case of failed interactions, or consent to the fact that the agent may deceive for the greater good must be established up front with the expectation that this is also going to affect interactions in the long term. *Thus deployment of above techniques must show legitimate gains over longitudinal interactions.*

[6]The only place where this is not the case is the “omission” of information in pursuit of minimal or shortest explanations.

References

Bryce, D.; Benton, J.; and Boldt, M. W. 2016. Maintaining Evolving Domain Models. In *IJCAI*.

Chakraborti, T., and Kambhampati, S. 2019. (When) Can AI Bots Lie? In *AIES*.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy. In *IJCAI*.

Chakraborti, T.; Kulkarni, A.; Sreedharan, S.; Smith, D. E.; and Kambhampati, S. 2019. Explicability? Legibility? Predictability? Transparency? Privacy? Security? The Emerging Landscape of Interpretable Agent Behavior. In *ICAPS*.

Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2018. Explicability versus explanations in human-aware planning. In *AAMAS*. Extended Abstract.

Haddon, M. 2003. *The Curious Incident of the Dog in the Night-time*. Doubleday.

Isaac, A., and Bridewell, W. 2017. White Lies on Silver Tongues: Why Robots Need to Deceive (and How). *Journal of Robot Ethics*.

Kulkarni, A.; Srivastava, S.; and Kambhampati, S. 2019. A Unified Framework for Planning in Adversarial and Cooperative Environments. In *AAAI*.

Miller, T. 2018. Explanation in Artificial Intelligence: Insights from the Social Sciences. *Artificial Intelligence*.

Porteous, J.; Lindsay, A.; Read, J.; Truran, M.; and Cavazza, M. 2015. Automated Extension of Narrative Planning Domains with Antonymic Operators. In *AAMAS*.

Sreedharan, S.; Chakraborti, T.; and Kambhampati, S. 2018. Handling Model Uncertainty and Multiplicity in Explanations as Model Reconciliation. In *ICAPS*.

Sreedharan, S.; Srivastava, S.; and Kambhampati, S. 2018. Hierarchical Expertise-Level Modeling for User Specific Robot-Behavior Explanations. In *IJCAI*.

Zahedi, Z.; Olmo, A.; Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2019. Towards Understanding User Preferences in Explanations as as Model Reconciliation. In *HRI*. Late Breaking Report.

A Preliminary Logic-based Approach for Explanation Generation

Stylianos L. Vasileiou
Computer Science and Engineering
Washington University in St. Louis
v.stylianos@wustl.edu

William Yeoh
Computer Science and Engineering
Washington University in St. Louis
wyeoh@wustl.edu

Tran Cao Son
Computer Science
New Mexico State University
tson@cs.nmsu.edu

Abstract

In an explanation generation problem, an agent needs to identify and explain the reasons for its decisions to another agent. Existing work in this area is mostly confined to planning-based systems that use automated planning approaches to solve the problem. In this paper, we approach this problem from a new perspective, where we propose a general logic-based framework for explanation generation. In particular, given a knowledge base KB_1 that entails a formula ϕ and a second knowledge base KB_2 that does not entail ϕ, we seek to identify an explanation ϵ that is a subset of KB_1 such that the union of KB_2 and ϵ entails ϕ. We define two types of explanations, model- and proof-theoretic explanations, and use cost functions to reflect preferences between explanations. Further, we present our algorithm implemented for propositional logic that compute such explanations and empirically evaluate it in random knowledge bases and a planning domain.

Introduction

With increasing proliferation and integration of AI systems in our daily life, there is a surge of interest in *explainable AI*, which includes the development of AI systems whose actions can be easily understood by humans. Driven by this goal, *machine learning* (ML) researchers have begun to classify commonly used ML algorithms according to different dimensions of explainability (Guidotti *et al.* 2018); improved the explainability of existing ML algorithms (Vaughan *et al.* 2018; Alvarez Melis and Jaakkola 2018; Petkovic *et al.* 2018); as well as proposed new ML algorithms that trade off accuracy for increasing explainability (Dong *et al.* 2017; Gilpin *et al.* 2018).[1]

[1]While the term *interpretability* is more commonly used in the ML literature and can be used interchangeably with *explainability*, we use the latter term as it is more commonly used broadly across different subareas of AI.

In contrast, researchers in the *automated planning* community have mostly taken a complementary approach. While there is some work on adapting planning algorithms to find easily explainable plans[2] (i.e., plans that are easily understood and accepted by a human user) (Zhang *et al.* 2017), most work has focused on the *explanation generation problem* (i.e., the problem of identifying explanations of plans found by planning agents that when presented to users, will allow them to understand and accept the proposed plan) (Langley 2016; Kambhampati 1990). Within this context, researchers have tackled the problem where the model of the human user may be (1) inconsistent with the model of the planning agent (Chakraborti *et al.* 2017b); (2) must be learned (Zhang *et al.* 2017); and (3) a different form or abstraction than that of the planning agent (Sreedharan *et al.* 2018; Tian *et al.* 2016). However, a common thread across most of these works is that they, not surprisingly, employ mostly automated planning approaches. For example, they often assume that the models of both the agent and human are encoded in PDDL format.

In this paper, we approach the explanation generation problem from a different perspective – one based on *knowledge representation and reasoning* (KR). We propose a general logic-based framework for explanation generation, where given a knowledge base KB_1 (of an agent) that entails a formula ϕ and a knowledge base KB_2 (of a human user) that does not entail ϕ, the goal is to identify an explanation $\epsilon \subseteq KB_1$ such that $KB_2 \cup \epsilon$ entails ϕ. We define two types of explanations, model- and proof-theoretic explanations, and use cost functions to reflect preferences between explanations. Further, we present an algorithm, implemented for propositional logic, that computes such explanations and evaluate its performance experimentally in random knowledge bases as well as in a planning domain.

In addition to providing an alternative approach to

[2]Also called *explicable* plans in the planning literature.

solve the same explanation generation problem tackled thus far by the automated planning community, our approach has the merit of being more generalizable to other problems beyond planning problems as long as they can be modeled using a logical KR language.

Preliminaries

Logic

A *logic* L is a tuple (KB_L, BS_L, ACC_L) where KB_L is the set of well-formed knowledge bases (or theories) of L – each being a set of formulae. BS_L is the set of possible belief sets; each element of BS_L is a set of syntactic elements representing the beliefs L may adopt. $ACC_L : KB_L \rightarrow 2^{BS_L}$ describes the *"semantics"* of L by assigning to each element of KB_L a set of acceptable sets of beliefs. For each $KB \in KB_L$ and $B \in ACC_L(KB)$, we say that B is a *model* of KB. A logic is monotonic if $KB \subseteq KB'$ implies $ACC_L(KB') \subseteq ACC_L(KB)$.

Example 1 *Assume that L refers to the propositional logic over an alphabet P. Then, KB_L is the set of propositional theories over P, $BS_L = 2^P$, and ACC_L maps each theory KB into the set of its models in the usual sense.*

We say that a KB is *consistent* if $ACC_L(KB) \neq \emptyset$. A formula φ in the logic L is *entailed* by KB, denoted by $KB \models_L \varphi$, if $ACC_L(KB) \neq \emptyset$ and $\varphi \in B$ for every $B \in ACC_L(KB)$.

For our later use, we will assume that a negation operator $\neg$ over formulas exists; and φ and $\neg\varphi$ are contradictory with each other in the sense that for any KB and $B \in ACC_L(KB)$, if $\varphi \in B$ then $\neg\varphi \notin B$; and if $\neg\varphi \in B$ then $\varphi \notin B$. $\epsilon \subseteq KB$ is called a *sub-theory* of KB. A theory KB *subsumes* a theory KB', denoted by $KB \lhd KB'$ if $ACC_L(KB) \subset ACC_L(KB')$.

Conclusions of a knowledge base can also be derived using rules. A rule system Σ_L of a logic L is a set of rules of the form

$$\varphi_1, \ldots, \varphi_k \vdash_L \varphi_0 \tag{1}$$

where φ_i are formulas. The left hand side could be empty. For a rule r of the form (1), $body(r)$ (resp. $head(r)$) denotes the left (resp. right) side of r. Intuitively, a rule r states that if the body is true then the head is also true.

Given a knowledge base KB and a rule system Σ_L, we say $KB \vdash_{\Sigma_L} \varphi$ if either $\varphi \in KB$ or there exists a sequence of rules $r_1, \ldots, r_n$ in Σ_L such that $body(r_1) \subseteq KB$, $head(r_n) = \varphi$, $head(r_i) \in KB$ for $i = 1, \ldots, n-1$, and $body(r_i) \subseteq KB$ or $body(r_i) \subseteq body(r_1) \cup \{head(r_j) \mid j = 1, \ldots, i-1\}$ for every $i = 2, \ldots, n$. We call the sequence $\epsilon = \langle r_1; \ldots; r_n \rangle$ as a *proof* from KB for φ w.r.t. Σ_L and say that the proof ϵ has the length n.

Σ_L is said to be *sound* if for every φ, $KB \vdash_{\Sigma_L} \varphi$ implies $KB \models_L \varphi$. It is *complete* if for every φ, $KB \models_L \varphi$ implies $KB \vdash_{\Sigma_L} \varphi$.

Classical Planning as Boolean Satisfiability

A classical planning problem (Russell and Norvig 2009) can be naturally encoded as an instance of propositional satisfiability (Kautz *et al.* 1992). The basic idea is the following: Given a planning problem P, find a solution for P of length n by creating a propositional formula that represent the initial state, goal, and the action dynamics for n time steps. This is referred to as the *bounded planning problem* (P, n), and we define the formula for (P, n) such that: *any* model of the formula represents a solution to (P, n) and if (P, n) has a solution, then the formula is satisfiable.

We encode (P, n) as a formula Φ such that $\langle a_0, a_1, \ldots, a_{n-1} \rangle$ is a solution for (P, n) if and only if Φ can be satisfied in a way that makes the fluents $a_0, a_1, \ldots, a_{n-1}$ true. The formula Φ is a conjunction of the following formulae:

- **Initial State:** Let F be the set of possible facts in the planning problem:
$$\bigwedge\{f_0 | f \in s_0\} \wedge \bigwedge\{\neg f_0 | f \in F \setminus \{s_0\}\}$$
- **Goal:** Let G be the set of goal facts:
$$\bigwedge\{f_n | f \in G\}$$
- **Action Scheme:** For every action a_i at time step i:
$$a_i \Rightarrow \bigwedge\{f_i | f \in \text{Precondition(a)}\}$$
$$a_{i+1} \Rightarrow \bigwedge\{f_{i+1} | f \in \text{Add(a)}\}$$
$$a_{i+1} \Rightarrow \bigwedge\{\neg f_{i+1} | f \in \text{Delete(a)}\}$$
- **Explanatory Frame Axioms:** Formulae describing what does not change between steps i and $i+1$:
$$\neg f_i \wedge f_{i+1} \Rightarrow \bigvee\{a_i | f \in \text{ADD(a)}\}$$
$$f_i \wedge \neg f_{i+1} \Rightarrow \bigvee\{a_i | f \in \text{DEL(a)}\}$$
- **Complete Exclusion Axioms:** Only one action can occur at each time step:
$$\neg a_i \vee \neg b_i$$

Finally, we can *extract* a plan by finding an assignment of truth values that satisfies Φ (i.e., for $i = 0, \ldots, n-1$, there will be exactly one action a such that $a_i = True$). This could be easily done by using a satisfiability algorithm, such as the well-known DPLL algorithm (Davis *et al.* 1962).

In this paper, we will mostly use examples from propositional logic. We make use of the fact that the resolution rule is sound and complete in first-order logic (Robinson 1965), and hence, in propositional logic. This allows us to utilize the DPLL algorithm in computing proofs for a formula given a knowledge base.

Two Accounts of Explanations

In this section, we introduce the notion of an explanation in the following setting:

> **Explanation Generation Problem:** Given two knowledge bases KB_1 and KB_2 and a formula φ in a logic L. Assume that $KB_1 \models_L \varphi$ and $KB_2 \not\models_L \varphi$. The goal is to identify an explanation (i.e., a set of formulas) $\epsilon \subseteq KB_1$ such that $KB_2 \cup \epsilon \models \varphi$.

We first define the notion of a support of a formula w.r.t. a knowledge base.

Definition 1 (Support) *Assume that $KB \models_L \varphi$. We say that $\epsilon \subseteq KB$ is a* support *of φ w.r.t. KB if $\epsilon \models_L \varphi$. Assume that ϵ is a support of φ w.r.t. KB. We say that $\epsilon \subseteq KB$ is a* $\subseteq$-minimal support *of φ if no proper sub-theory of ϵ is a support of φ. Furthermore, ϵ is a* $\lhd$-general support *of φ if there is no support ϵ' of φ w.r.t. KB such that ϵ subsumes ϵ'.*

We now define below two types of explanations – model-theoretic and proof-theoretic explanations.

Model-Theoretic Explanations

Definition 2 (m-Explanation) *Given two knowledge bases KB_1 and KB_2 in logic L and a formula φ. Assume that $KB_1 \models_L \varphi$ and $KB_2 \not\models_L \varphi$.*

A model-theoretic explanation (or m-explanation) for φ from KB_1 for KB_2 *is a support ϵ w.r.t. KB_1 for φ such that $KB_2 \cup \epsilon \models_L \varphi$.*

Example 2 *Consider proposition logic theories over the set of propositions $\{a, b, c\}$ with the usual definition of models, satisfaction, etc. Assume $KB_1 = \{a, b, a \rightarrow c, a \wedge b \rightarrow c\}$ and $KB_2 = \{a\}$. We have that $\epsilon_1 = \{a, a \rightarrow c\}$ and $\epsilon_2 = \{a, b, a \wedge b \rightarrow c\}$ are two $\subseteq$-minimal supports of c w.r.t. KB_1. Only ϵ_1 is a $\lhd$-general support of c w.r.t. KB_1 since $\epsilon_2 \lhd \epsilon_1$.*

Both ϵ_1 and ϵ_2 can serve as m-explanations for c from KB_1 for KB_2. Of course, KB_1 is itself an m-explanation for c from KB_1 for KB_2.

Consider $KB_3 = \{a, \neg b\}$. In this case, we have that only ϵ_1 is an m-explanation for c from KB_1 for KB_3.

Now, consider $KB_4 = \{\neg a\}$. In this case, we have no m-explanation for c from KB_1 for KB_4.

Proposition 1 *For two knowledge bases KB_1 and KB_2 in a monotonic logic L, if $KB_1 \models_L \varphi$ and $KB_2 \models_L \neg\varphi$, then there exists no m-explanation for φ from KB_1 for KB_2.*

The KB_4 in Example 2 and Proposition 1 show that m-explanations alone might be insufficient. Sometimes, we also need to persuade the other agent that its knowledge base is not correct. We leave this for the future. In this paper, we assume that $KB_2 \not\models_L \neg\varphi$ and $KB_2 \not\models_L \varphi$ and, thus, an m-explanation always exists.

Proof-Theoretic Explanations

Definition 3 (p-explanation) *Given a logic L with a sound and complete rule system Σ_L and two knowledge bases KB_1 and KB_2 in logic L and a formula φ. Assume that $KB_1 \vdash_L \varphi$ and $KB_2 \not\vdash_L \varphi$.*

A proof-theoretic explanation (or p-explanation) for φ from KB_1 for KB_2 *is a proof $\langle r_1; \ldots; r_n\rangle$ from KB_1 for φ such that $KB_2 \cup (\bigcup_{i=1}^n body(r_i) \cap KB_1) \vdash_{\Sigma_L} \varphi$ and $KB_2 \cup (\bigcup_{i=1}^n body(r_i) \cap KB_1)$ is consistent.*

Example 3 *Consider the theories $KB_1 = \{a, b, a \rightarrow c, a \wedge b \rightarrow c\}$ and $KB_2 = \{a\}$ from Example 2. Let us assume that Σ_L is the set of rules of the form $l \vdash_L l$ and $l, \neg l \vee p \vdash p$ for any literals l, p in the language of KB_1 and KB_2. Then, $\langle a, \neg a \vee c \vdash_L c\rangle$ is a proof from KB_1 for c, which is also a p-explanation for φ from KB_1 for KB_2.*

Likewise, $\langle a \vdash_L a; b \vdash_L b; a, \neg a \vee \neg b \vee c \vdash_L \neg b \vee c; b, \neg b \vee c \vdash_L c\rangle$ is a p-explanation for c from KB_1 for KB_2.

Proposition 2 *Assume that Σ_L is a sound and complete rule system of a logic L, KB_1 is a knowledge base, and φ is a formula in L. For each proof $\langle r_1; \ldots; r_n\rangle$ from KB_1 for φ w.r.t. Σ_L, $\bigcup_{i=1}^n body(r_i) \cap KB_1$ is a support of φ w.r.t. KB_1.*

Proposition 2 implies that each proof from KB_1 for φ could be identified as a p-explanation for φ from KB_1 if Σ_L is sound and complete. This provides the following relationship between m-explanations and p-explanations.

Proposition 3 *Assume that Σ_L is a sound and complete rule system of a logic L, KB_1 and KB_2 are two knowledge bases in L, and φ is a formula in L. Then,*

- *for each m-explanation ϵ for φ from KB_1 for KB_2, there exists a p-explanation $\langle r_1; \ldots; r_n\rangle$ for φ from KB_1 for KB_2 such that $\bigcup_{i=1}^n body(r_i) \cap KB_1 \subseteq \epsilon$; and*
- *for each p-explanation $\langle r_1; \ldots; r_n\rangle$ for φ from KB_1 for KB_2, $\bigcup_{i=1}^n body(r_i) \cap KB_1$ is an m-explanation for φ from KB_1 for KB_2.*

Preferred Explanations

Given KB_1 and KB_2 and a formula φ, there might be several (m- or p-) explanations for φ from KB_1 for KB_2. For brevity, we will now use the term x-explanation for $x \in \{m, p\}$ to refer to an x-explanation for φ from KB_1 for KB_2. Obviously, not all explanations are equal. One might preferred a subset minimal m-explanation or a shortest length p-explanation over others. We will next define a general preferred relation among explanations.

We assume a cost function $\mathcal{C}_L^x$ that maps pairs of knowledge bases and sets of explanations to non-negative real values, i.e.,

$$\mathcal{C}_L^x : KB_L \times \Omega \rightarrow \mathcal{R}^{\geq 0} \quad (2)$$

where Ω is the set of x-explanations and $\mathcal{R}^{\geq 0}$ denotes the set of non-negative real numbers. Intuitively, this function can be used to characterize different complexity measurements of an explanation.

A cost function $\mathcal{C}_L^m$ is *monotonic* if for any two m-explanations $\epsilon_1 \subseteq \epsilon_2$, $\mathcal{C}_L^m(KB, \epsilon_1) \leq \mathcal{C}_L^m(KB, \epsilon_2)$. A cost function $\mathcal{C}_L^p$ is *monotonic* if for any two p-explanations ϵ_1 and ϵ_2 such that ϵ_1 is a subsequence of ϵ_2, $\mathcal{C}_L^p(KB, \epsilon_1) \leq \mathcal{C}_L^m(KB, \epsilon_2)$.

$\mathcal{C}_L^x$ induces a preference relation $\prec_{KB}$ over explanations as follows.

Definition 4 (Preferred Explanation) *Given a cost function $\mathcal{C}_L^x$, a knowledge base KB_2, and two x-explanations ϵ_1 and ϵ_2 for KB_2, we say ϵ_1 is* preferred *over ϵ_2 w.r.t. KB_2 (denoted by $\epsilon_1 \preceq_{KB_2}^x \epsilon_2$) iff*

$$\mathcal{C}_L^x(KB_2, \epsilon_1) \leq \mathcal{C}_L^x(KB_2, \epsilon_2) \qquad (3)$$

and ϵ_1 is strictly preferred *over ϵ_2 w.r.t. KB_2 (denoted by $\epsilon_1 \prec_{KB_2}^x \epsilon_2$) if*

$$\mathcal{C}_L^x(KB_2, \epsilon_1) < \mathcal{C}_L^x(KB_2, \epsilon_2) \qquad (4)$$

This allows us to compare explanations as follows.

Definition 5 (Most Preferred Explanation) *Given a cost function $\mathcal{C}_L^x$ and a knowledge base KB_2, an explanation ϵ is a* most preferred x-explanation *w.r.t. KB_2 if there exists no other explanation ϵ' such that $\epsilon' \prec_{KB_2}^x \epsilon$.*

Proposition 4 *If $\mathcal{C}_L^x$ is monotonic then the relation $\preceq_{KB_2}^x$ over x-explanations is transitive, anti-symmetric, and reflexive; and the relation $\prec_{KB_2}^x$ over x-explanations is transitive and anti-symmetric.*

There are several natural monotonic cost functions. Examples for cost functions for m-explanations include:

- $c_L^1(KB_2, \epsilon) = |\epsilon|$, the cardinality of ϵ, indicates the number of formulas that need to be explained;
- $c_L^2(KB_2, \epsilon) = |\epsilon \setminus KB_2|$, the cardinality of $\epsilon \setminus KB_2$, indicates the number of *new* formulas that need to be explained;
- $c_L^3(KB_2, \epsilon) = |new\_vars(KB_2, \epsilon)|$ indicates the number of *new* symbols occurring in ϵ that are not in KB_2 and need to be explained;
- $c_L^4(KB_2, \epsilon) = length(\epsilon)$ indicates the number of literals in ϵ that need to be explained.

Naturally, some of these cost functions can also be combined (e.g., $c_L^2 + c_L^3$ will measure the number of new formulas and new symbols that must be explained).

Observe that the three functions c_L^1 and c_L^4 are independent from KB_2 while c_L^2 and c_L^3 depend on KB_2. A potential advantage of a cost function that is independent from KB_2 is that it helps simplify the computation of most preferred explanations.

Example 4 *Continuing with Example 2, if we use c_L^1 as the cost function, then we have that $\epsilon_1 \prec_{KB_2}^m \epsilon_2 \prec_{KB_2}^m KB_1$. Furthermore, ϵ_1 is the most preferred m-explanation from KB_1 to KB_2.*

Algorithm 1: `genExp`(KB_1, KB_2, φ)

```
Input: Logic $L$, formula $\varphi$, KBs $KB_1$ and $KB_2$, cost
       function $\mathcal{C}_L^x$
Output: A most preferred $x$-explanation w.r.t. $\mathcal{C}_L^x$
        from $KB_1$ to $KB_2$ for $\varphi$; or $nil$
1 if $KB_1 \not\models_L \varphi$ or $KB_2 \models_L \varphi$ then
2     return $nil$
3 if $KB_1 \models_L \varphi$ and $KB_2 \not\models_L \neg\varphi$ then
4     $\epsilon = most\_preferred(KB_1, KB_2, \varphi)$
5     return $\epsilon$
```

Algorithm 2: `most_preferred`(KB_1, KB_2, φ)

```
Input: Logic $L$, formula $\varphi$, KBs $KB_1$ and $KB_2$, cost
       function $\mathcal{C}_L^x$
Output: A most-preferred explanation w.r.t. $\mathcal{C}_L^x$ from
        $KB_1$ to $KB_2$ for $\varphi$; or $nil$
1 repeat
2     non-deterministically select a potential
      $x$-explanation $\epsilon$, a minimal element w.r.t. $\mathcal{C}_L^x$ and
      $KB_2$
3     if $\epsilon \models \varphi$ and $KB_2 \cup \epsilon \models \varphi$ then
4         return $\epsilon$
5 until all possible explanations are considered
6 return $nil$
```

Computing Preferred Explanations

At a high level, Algorithms 1 and 2 can be used for computing most-preferred explanations given a formula φ and two knowledge bases KB_1 and KB_2 of a logic L with the cost function $\mathcal{C}_L^x$. We assume that when computing for p-explanations, a sound and complete rule system is available. Our algorithms rely on the existence of an algorithm for checking entailment between knowledge bases and formulas (Lines 1 and 3 in Algorithm 1 and Line 4 in Algorithm 2) and an algorithm for computing a potential explanation that is minimal with respect to a cost function and a knowledge base (Lines 2-3 in Algorithm 2). These two algorithms depend on the logic L and the cost function $\mathcal{C}_L^x$ and need to be implemented for specific logic L and function $\mathcal{C}_L^x$.

In the rest of this section, we discuss the implementation of our algorithms for propositional logic and different cost functions. With propositional logic, it is easy to see that checking for entailment can be done by a SAT solver (e.g., `MiniSat` (Eén and Sörensson 2003)). We next discuss two algorithm implementations, one for m-explanations and one for p-explanations, that find an explanation that is minimal with respect to a cost function and a knowledge base.

Most-Preferred m-Explanations

Given a cost function $\mathcal{C}_L^m$ such as c_L^1, c_L^2, c_L^3, or c_L^4 as defined in Section , Algorithm 3 computes a most pre-

Algorithm 3: `most_preferred_m`(KB_1, KB_2, φ)

```
Input: Formula $\varphi$, KBs $KB_1$ and $KB_2$, cost function $\mathcal{C}^m_L$
Output: A most-preferred $m$-explanation w.r.t. $\mathcal{C}^m_L$ from $KB_1$ to $KB_2$ for $\varphi$; or $nil$
1  $q = [\emptyset]$        % a priority queue of potential explanations
2  $checked = \emptyset$   % a set of sets of elements in $KB_1$ that have been considered
3  repeat
4      $\epsilon = dequeue(q)$
5      insert $\epsilon$ into $checked$
6      if $\epsilon \models \varphi$ and $KB_2 \cup \epsilon \models \varphi$ then
7          return $\epsilon$
8      else
9          for $a \in KB_1$ do
10             if $\epsilon \cup \{a\} \notin checked$ then
11                 $v = \mathcal{C}^m_L(KB_2, \epsilon \cup \{a\})$
12                 $q = enqueue(\epsilon \cup \{a\})$ % use $v$ as key
13 until $q$ is empty
14 return $nil$
```

ferred m-explanations w.r.t. $\mathcal{C}^m_L$ from KB_1 to KB_2 for φ or returns nil if none exists.

The key data structures in the algorithm is a priority queue q, initialized to only include the empty set, of potential explanations ordered by their costs (Line 1) and a set $checked$ of invalid explanations that have been considered thus far (line 2). The algorithm repeatedly loops the following steps: (*i*) move the explanation with the smallest cost from the priority queue q to $checked$ (Lines 4-5); (*ii*) check if it is a valid m-explanation and return if it is (Lines 6-7); (*iii*) if not, extend the explanation by 1 (with each clause from KB_1) and insert the extended explanations into the priority queue q (Lines 8-12). If all potential explanations are exhausted, which means that there are no valid m-explanations, then the algorithm returns nil (Line 14). It is straightforward to see that the following proposition holds.

Proposition 5 *For two propositional theories KB_1 and KB_2 and a formula φ, Algorithm 3 returns a most preferred m-explanation w.r.t. $\mathcal{C}^m_L$ for φ from KB_1 to KB_2 if one exists.*

Most-Preferred p-Explanations

Given a cost function $\mathcal{C}^p_L$ on p-explanations, Algorithm 4 computes a most-preferred p-explanation w.r.t. $\mathcal{C}^p_L$ from KB_1 to KB_2 for φ or returns nil if none exists.

We use the following notations in the pseudocode: For a proof $\langle\epsilon\rangle$, where ϵ is the sequence $\langle r_1; \ldots; r_n\rangle$, we write $c(\epsilon) = head(r_n)$ and $b(\epsilon) = \bigcup_{i=1}^n body(r_i)$. We also write $\frac{\varphi_1, \varphi_2}{\varphi}$ to indicate that φ is the result of

Algorithm 4: `most_preferred_p`(KB_1, KB_2, φ)

```
Input: Formula $\varphi$, KBs $KB_1$ and $KB_2$, cost function $\mathcal{C}^p_L$
Output: A most-preferred $p$-explanation w.r.t. $\mathcal{C}^p_L$ from $KB_1$ to $KB_2$ for $\varphi$; or $nil$
1  $q = [\emptyset]$   % priority queue of potential explanations
2  for $\epsilon$ in $KB_1$ do
3      $v = \mathcal{C}^p_L(KB_2, \epsilon)$
4      $q = enqueue(\langle\epsilon\rangle)$      % use $v$ as key
5  $\Omega = \{\langle\epsilon\rangle \mid \epsilon \in KB_1\}$
6  $checked = \emptyset$
7  repeat
8      $\langle\epsilon\rangle = dequeue(q)$
9      insert $(b(\epsilon), c(\epsilon))$ into $checked$
10     if $c(\epsilon) = \varphi$ and $KB_2 \cup (b(\epsilon) \cap KB_1) \models \varphi$ then
11         return $\epsilon$
12     for $\epsilon'$ in $\Omega$ do
13         if $c(\epsilon)$ and $c(\epsilon')$ contain complementary literals and $\frac{c(\epsilon), c(\epsilon')}{\phi}$ holds then
14             $\hat{\epsilon} = \langle \epsilon \circ \epsilon'; \frac{c(\epsilon), c(\epsilon')}{\phi} \rangle$
15             if $(b(\hat{\epsilon}), \phi) \notin checked$ then
16                 $v = \mathcal{C}^p_L(KB_2, \hat{\epsilon})$
17                 $q = enqueue(\langle\hat{\epsilon}\rangle)$      % use $v$ as key
18 until $q$ is empty
19 return $nil$
```

applying the resolution rule on φ_1 and φ_2. And we use $\circ$ to denote the concatenation of two sequences.

The algorithm uses the same two data structures – priority queue q and set $checked$ – as in Algorithm 3. The algorithm first populates the queue q with single-rule proofs consist of single clauses in KB_1 (Lines 2-4). Then, it repeatedly loops the following steps: (*i*) move the proof with the smallest cost from the priority queue q to $checked$ (Lines 8-9); (*ii*) check if it is a valid p-explanation and return if it is (Lines 10-11); (*iii*) if not, extend the proof by 1 and insert the extended proofs into the priority queue q (Lines 12-17). If all potential proofs are exhausted, which means that there are no valid p-explanations, then the algorithm returns nil (Line 19). It is straightforward to see that the following proposition holds.

Proposition 6 *For two propositional theories KB_1 and KB_2 and a formula φ, Algorithm 4 returns a most preferred p-explanation w.r.t. $\mathcal{C}^p_L$ for φ from KB_1 to KB_2 if one exists.*

Plan Explanation Generation

As presented in the preliminaries, we can model a planning problem using the propositional logic language and thus utilize the proposed framework to generate explanations. Particularly, we form the knowledge base of

(a) Experimental Results on Random Knowledge Bases

$\|KB_1\|$	c^1_L		c^2_L		c^3_L		c^4_L	
	cost	time	cost	time	cost	time	cost	time
20	7	23ms	5	25ms	2.4	26ms	14	24ms
100	15	2.5s	10	3.0s	3.8	3.1s	30	2.9s
1000	117	27m	97	30m	38	32m	347	27m

(b) Experimental Results on BLOCKSWORLD Domain

$\|KB_1\|$	c^1_L		c^2_L		c^3_L		c^4_L	
	cost	time	cost	time	cost	time	cost	time
225	4	15.0s	1	16.0s	0.5	15.5s	7	15.0s
387	16	2.0m	12	2.2m	0.5	2.2m	35	2.0m

Table 1: Experimental Results

the agent, namely KB, by adding the encoded formula Φ (represented in CNF clauses) as well as the optimal plan of the specific planning problem. Then, we define the explanation in terms of KB and plan optimality as follows:

Definition 6 (Optimal Plan Explanation)
Given a knowledge base KB and a plan $\pi_n = \langle a_0, a_1, \ldots, a_{n-1} \rangle$, we say that π_n is optimal in KB if and only if $KB \models \phi$, where $\forall t = 1, \ldots n-1 : \phi = \neg goal_t$.

In other words, the formula ϕ that we seek to explain is that no plan of lengths 1 to $n-1$ exists, and that a plan of length n exists. Therefore, combined, that plan must be an optimal plan. Now, given a second knowledge base KB_2 (i.e that of a human user), where $KB_2 \not\models \phi$, we can compute a model- or proof-theoretic explanation as defined in Definitions 2 and 3.

Experimental Results

We empirically evaluate our implementation of Algorithm 3 to find m-explanations on two synthetically generated benchmarks – random knowledge bases and a planning domain called BLOCKSWORLD – both encoded in propositional logic.[3] We evaluated our algorithm using the four cost functions described in Section . Our algorithm was implemented in Python and experiments were performed on a machine with an Intel i7 2.6GHz processor and 16GB of RAM. We report both the cost of the optimal m-explanation found as well as the runtime of the algorithm.

Random Knowledge Bases

We first evaluated our algorithm on random knowledge bases with clauses in Horn form, where we varied the cardinality of KB_1 (the KB of the agent providing the explanation) from 20 to 1000. To construct KB_2 (the KB of the agent receiving the explanation), we randomly chose 25% of the clauses from KB_1.

To construct each KB_1, we first generated $\frac{|KB_1|}{2}$ random symbols, which will be used in the KB. Then, we iteratively generated clauses of increasing length l from 2 to 7. For each length l, we generated $\lfloor \frac{|KB_1|}{2 \cdot l} \rfloor$ clauses using the symbols we previously generated such that each symbol is used at most once in these clauses of length l. Each clause is a conjunction of $l-1$ elements as the premise and the final l^{th} element as the conclusion. For example, a KB with a cardinality of 20, 10 symbols are first generated. Then, 5 clauses of length 2, 3 clauses of length 3, 2 clauses of lengths 4 and 5, and 1 clause of lengths 6 and 7 are generated. Finally, to complete the KB, we add all the symbols that are exclusively in the premise of the clauses generated as facts in the KB. The formula φ that we seek to explain is one of the randomly chosen conclusions in the clauses generated, which we ensure is entailed by KB_1.

Table 1(a) tabulates our results. We make the following observations:

- As expected, the runtimes increase as $|KB_1|$ increases since the algorithm will need to search over a larger search space.
- As expected, the costs of explanations also increase as $|KB_1|$ increases since the explanations are presumably longer and more complex.
- Finally, the runtimes for cost functions c^1_L and c^4_L are smaller than that of c^2_L and c^3_L. The reason is the computation of the costs of possible explanations is faster with the former two cost functions since they are not dependent on KB_2 while the computation for the latter two cost functions are dependent on KB_2.

Planning Domain

As we were motivated by the explanation generation problem studied in the automated planning community, we also conducted experiments on BLOCKSWORLD, a planning domain where multiple blocks must be stacked in a particular order on a table.[4]

For these planning problems, we first used FAST-DOWNWARD (Helmert 2006) to find optimal solutions to the planning problem. Then, we translate the planning problem into a SAT problem with horizon h (Kautz *et al.* 1992), where h is the length of the optimal plan. These CNF clauses then form our KB_1 (the KB of the agent providing the explanation). Similar to random knowledge bases, we construct KB_2 (the KB of the agent receiving the explanation) by randomly choosing 25% of the clauses from KB_1. The formula φ that

[3]For random knowledge bases, we used an optimized version that uses a version of backward chaining that finds the set of all possible explanations. This approach works only when the clauses in the knowledge base are in Horn form and is sound and complete for such a case (Russell and Norvig 2009).

[4]It is one of the domains in the International Planning Competition. See *http://www.plg.inf.uc3m.es/ipc2011-learning/Domains.html.*

we seek to explain is then that no plan of lengths 1 to $h-1$ exists, and that a plan of length h (i.e., the plan found by FASTDOWNWARD) exists. Therefore, combined, that plan must be an optimal plan.

Table 1(b) tabulates our results, where we observe similar trends as in the experiment on random knowledge bases. The key difference is that the runtimes for all four cost functions here are a lot closer to each other, and the reason is because there was only one valid explanation in each problem instance. Thus, regardless of the choice of cost function, that explanation had to be found. Our experiments for larger problems are omitted as they timed out after 6 hours.

Related Work and Discussions

There is a very large body of work related to the very broad area of explainable AI. We have briefly discussed some of them from the ML literature in Section . We refer readers to surveys by (Adadi and Berrada 2018) and (Dosilovic *et al.* 2018) for more in-depth discussions of this area. We focus below on related work from the KR and planning literature only since we employ KR techniques to solve explainable planning problems in this paper.

Related Work from the KR Literature: We note that the notion of an explanation proposed in this paper might appear similar to the notion of a diagnosis that has been studied extensively in the last several decades (e.g., (Reiter 1987)) as both aim at explaining something to an agent. Diagnosis focuses on identifying the reason for the inconsistency of a theory whereas an m- or p-explanation aims at identifying the support for a formula. The difference lies in that a diagnosis is made with respect to the same theory and m- or p-explanation is sought for the second theory.

Another earlier research direction that is closely related to the proposed notion of explanation is that of developing explanation capabilities of knowledge-based systems and decision support systems, which resulted in different notions of explanation such as trace, strategic, deep, or reasoning explanations (see review by (Moulin *et al.* 2002) for a discussion of these notions). All of these types of explanations focus on answering why certain rules in a knowledge base are used and how a conclusion is derived. This is not our focus in this paper. The present development differs from earlier proposals in that m- or p-explanations are identified with the aim of explaining a given formula to a second theory. Furthermore, the notion of an optimal explanation with respect to the second theory is proposed.

There have been attempts to using argumentation for explanation (Cyras *et al.* 2017; Cyras *et al.* 2019) because of the close relation between argumentation and explanation. For example, argumentation was used by (Cyras *et al.* 2019) to answer questions such as why a schedule does (does not) satisfy a criteria (e.g., feasibility, efficiency, etc.); the approach was to develop for each type of inquiry, an abstract argumentation framework (AF) that helps explain the situation by extracting the attacks (non-attacks) from the corresponding AF. Our work differs from these works in that it is more general and does not focus on a specific question.

It is worth to pointing out that the problem of computing a most preferred explanation for φ from KB_1 to KB_2 might look similar to the problem of computing a weakest sufficient condition of φ on KB_1 under KB_2 as described by (Lin 2001). As it turns out, the two notions are quite different. Given that $KB_1 = \{p, q\}$ and $KB_2 = \{p\}$. It is easy to see that q is the unique explanation for q from KB_1 to KB_2. On the other hand, the weakest sufficient condition of q on KB_1 under KB_2 is $\perp$ (Proposition 8, (Lin 2001)).

Related Work from the Planning Literature: In human-aware planning, the (planning) agent must have knowledge of the human model in order to be able to contemplate the goals of the humans as well as foresee how its plan will be perceived by them. This is of the highest importance in the context of explainable planning since an explanation of a plan cannot be *one-sided* (i.e., it must incorporate the human's beliefs of the planner). In a plan generation process, a planner performs argumentation over a set of different models (Chakraborti *et al.* 2017a); these models usually are the model of the agent incorporating the planner, the model of the human in the loop, the model the agent thinks the human has, the model the human thinks the agent has, and the agent's approximation of the latter.

Therefore, the necessity for plan explanations arises when the model of the agent and the model the human thinks the agent has diverge so that the optimal plans in the agent's model are inexplicable to the human. During a collaborative activity, an explainable planning agent (Fox *et al.* 2017) must be able to account for such model differences and maintain an explanatory dialogue with the human so that both of them agree on the same plan. This forms the nucleus of explanation generation of an explainable planning agent, and is referred to as *model reconciliation* (Chakraborti *et al.* 2017b). In this approach, the agent computes the optimal plan in terms of his model and provides an explanation of that plan in terms of model differences. Essentially, these explanations can be viewed as the agent's attempt to move the human's model to be in agreement with its own. Further, for computing explanations using this approach the following four requirements are considered:

- **Completeness** – No better solution exists. This is achieved by enforcing that the plan being explained is optimal in the updated human model.

- **Conciseness** – Explanations should be easily understandable to the human.
- **Monotonicity** – The remaining model differences cannot change the completeness of an explanation.
- **Computability** – Explanations should be easy to compute (from the agent's perspective).

As our work is motivated by these ideas, we now identify some similarities and connections with our proposed approach. First, it is easy to see that we implicitly enforce the first three requirements when computing an explanation – the notions of completeness and conciseness are captured through the use of our cost functions. We do not claim to satisfy the computability requirement as it is more subjective and is more domain dependent.

In a nutshell, the model reconciliation approach works by providing a model update ϵ such that the optimal plan is feasible and optimal in the updated model of the human. This is similar to our definition of the explanation generation problem where we want to identify an explanation $\epsilon \subseteq KB_1$ (i.e., a set of formulae) such that $KB_2 \cup \epsilon \models \phi$. In addition, the $\subseteq$-minimal support in Definition 1 is equivalent to *minimally complete explanations* (MCEs) (the shortest explanation). The $\lhd$-general support can be viewed as similar to the *minimally monotonic explanations* (MMEs) (the shortest explanation such that no further model updates invalidate it), with the only difference being that in the general support scenario, the explanations are such that all subsuming ϵ are also valid supports.

In contrast, *model patch explanations* (MPEs) (includes all the model updates) are trivial explanations and are equivalent to our definition that KB_1 itself serves as an m-explanation for KB_2. Note that, in our approach, we do not allow for explanations on "mistaken" expectations in the human model, as it can be inferred from Proposition 1 (monotonic language L). From the model reconciliation perspective, such restriction is relaxed and allowed. However, a similar property can be seen if the mental model is not known and, therefore, by taking an "empty" model as starting point explanations can only add to the human's understanding but not mend mistaken ones.

Conclusions and Future Work

Explanation generation is an important problem within the larger explainable AI thrust. Existing work on this problem has been done in the context of automated planning domains, where researchers have primarily employed, unsurprisingly, automated planning approaches. In this paper, we approach the problem from the perspective of KR, where we propose a general logic-based framework for explanation generation. We further define two types of explanations, model- and proof-theoretic explanations, and use cost functions to reflect preferences between explanations. Our empirical results with algorithms implemented for propositional logic on both random knowledge bases as well as a planning domain demonstrate the generality of our approach beyond planning problems. Future work includes investigating more complex scenarios, such as one where an agent needs to persuade another that its knowledge base is incorrect.

Acknowledgment

This research is partially supported by NSF grants 1345232, 1757207, and 1812628. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government.

References

[Adadi and Berrada 2018] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018.

[Alvarez Melis and Jaakkola 2018] David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. pages 7775–7784, 2018.

[Chakraborti *et al.* 2017a] Tathagata Chakraborti, Subbarao Kambhampati, Matthias Scheutz, and Yu Zhang. Ai challenges in human-robot cognitive teaming. *arXiv preprint arXiv:1707.04775*, 2017.

[Chakraborti *et al.* 2017b] Tathagata Chakraborti, Sarath Sreedharan, Yu Zhang, and Subbarao Kambhampati. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *Proceedings of IJCAI*, pages 156–163, 2017.

[Cyras *et al.* 2017] Kristijonas Cyras, Xiuyi Fan, Claudia Schulz, and Francesca Toni. Assumption-based argumentation: Disputes, explanations, preferences. *Journal of Logics and their Applications*, 4(8), 2017.

[Cyras *et al.* 2019] Kristijonas Cyras, Dimitrios Letsios, Ruth Misener, and Francesca Toni. Argumentation for explainable scheduling. In *Proceedings of AAAI*, 2019.

[Davis *et al.* 1962] Martin Davis, George Logemann, Donald, and Loveland. A Machine Program for Theorem Proving. *Communications of the ACM*, 5(7):394–397, 1962.

[Dong *et al.* 2017] Yinpeng Dong, Hang Su, Jun Zhu, and Bo Zhang. Improving interpretability of deep neural networks with semantic information. In *Proceedings of CVPR*, pages 4306–4314, 2017.

[Dosilovic *et al.* 2018] Filip Karlo Dosilovic, Mario Brcic, and Nikica Hlupic. Explainable artificial intelligence: A survey. In *Proceedings of MIPRO*, pages 210–215, 2018.

[Eén and Sörensson 2003] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

[Fox *et al.* 2017] Maria Fox, Derek Long, and Daniele Magazzeni. Explainable planning. *CoRR*, abs/1709.10256, 2017.

[Gilpin *et al.* 2018] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *Proceedings of DSAA*, pages 80–89, 2018.

[Guidotti *et al.* 2018] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Computing Survey*, 51(5):93:1–93:42, 2018.

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

[Kambhampati 1990] Subbarao Kambhampati. A classification of plan modification strategies based on coverage and information requirements. In *AAAI Spring Symposium Series*, 1990.

[Kautz *et al.* 1992] Henry A Kautz, Bart Selman, et al. Planning as satisfiability. In *Proceedings of ECAI*, volume 92, pages 359–363, 1992.

[Langley 2016] Pat Langley. Explainable agency in human-robot interaction. In *AAAI Fall Symposium Series*, 2016.

[Lin 2001] Fangzhen Lin. On strongest necessary and weakest sufficient conditions. *Artificial Intelligence*, 128(1-2):143–159, 2001.

[Moulin *et al.* 2002] Bernard Moulin, Hengameh Irandoust, Micheline Bélanger, and G. Desbordes. Explanation and argumentation capabilities: Towards the creation of more persuasive agents. *Artificial Intelligence Review*, 17(3):169–222, 2002.

[Petkovic *et al.* 2018] Dragutin Petkovic, Russ Altman, Mike Wong, and Arthur Vigil. Improving the explainability of random forest classifier–user centered approach. In *Pacific Symposium on Biocomputing*, volume 23, pages 204–215, 2018.

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

[Robinson 1965] John Alan Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Communications of the ACM*, 5:23–41, 1965.

[Russell and Norvig 2009] Stuart J Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson, 2009.

[Sreedharan *et al.* 2018] Sarath Sreedharan, Siddharth Srivastava, and Subbarao Kambhampati. Hierarchical expertise level modeling for user specific contrastive explanations. In *Proceedings of IJCAI*, pages 4829–4836, 2018.

[Tian *et al.* 2016] Xin Tian, Hankz Hankui Zhuo, and Subbarao Kambhampati. Discovering underlying plans based on distributed representations of actions. In *Proceedings of AAMAS*, pages 1135–1143, 2016.

[Vaughan *et al.* 2018] Joel Vaughan, Agus Sudjianto, Erind Brahimi, Jie Chen, and Vijayan N. Nair. Explainable neural networks based on additive index models. *CoRR*, abs/1806.01933, 2018.

[Zhang *et al.* 2017] Yu Zhang, Sarath Sreedharan, Anagha Kulkarni, Tathagata Chakraborti, Hankz Hankui Zhuo, and Subbarao Kambhampati. Plan explicability and predictability for robot task planning. In *Proceedings of ICRA*, pages 1313–1320, 2017.

Combining Cognitive and Affective Measures with Epistemic Planning for Explanation Generation

Ronald P. A. Petrick
Department of Computer Science
Heriot-Watt University
Edinburgh, Scotland, United Kingdom
R.Petrick@hw.ac.uk

Sara Dalzel-Job and **Robin L. Hill**
School of Informatics
University of Edinburgh
Edinburgh, Scotland, United Kingdom
sdalzel@exseed.ed.ac.uk, R.L.Hill@ed.ac.uk

Abstract

This paper presents an overview of the EPSRC-funded project Start Making Sense, which is investigating explainability and trust maintenance in interactive and autonomous systems. This project brings together experimental research in cognitive science involving cooperative joint action with the practical construction of automated planning tools to apply to the task of explanation generation. The project's challenges are addressed through three concrete objectives: (i) to study cooperative joint action in humans to identify the emotional, affective, or cognitive factors that are essential for successful human communication, (ii) to enhance epistemic planning techniques with measures derived from the studies for improved human-like explanation generation, and (iii) to deploy and evaluate the resulting system with human participants. We also describe initial work from the cognitive side of the project aimed at exploring how ambiguity, uncertainty, and certain types of biometric measurements impact instruction giving and explanation actions in scenarios with humans. The insights from this work will be combined with epistemic planning techniques to generate appropriate explanatory actions in similar instruction giving scenarios.

Introduction

A fundamental problem in the design of autonomous systems is that of action selection: based on the current state of the world, what action should the system take in order to achieve its goals? In the presence of humans, this problem typically becomes more complex: the system may also need to reason about the states, actions, and intentions of these agents. In collaborative environments that involve human communication, it is particularly important to identify, interpret, and understand the multimodal affective signals that humans employ, and which are often necessary for effective, successful achievement of communicative goals.

For instance, consider a tourist on a guided walking tour of a city. After reaching a place where they can see they are almost back to the starting point, the tour guide says "Let's go up that hill," pointing to a large hill. "We can get a good view of the city from there." However, on seeing the tired expression on the tourist's face, the guide adds "Or we can stop at that cafe over there and take a break." This scenario has two important features. First, it demonstrates that people like to be aware of their context and know what is going on. This is especially true in situations where a decision may not have been anticipated or expected. Here, an explanation may be needed not only to justify a decision but also to establish confidence in that choice: in other words, to trust it. Second, being able to read the situation and adapt to the needs of the moment is important when considering the possible actions that could be taken in a given situation. Here, a decision may need to be made dynamically. These two features capture the idea of dynamic trust maintenance, which will be needed for a broad range of the AI systems that are expected to be deployed in the near future, e.g., automated vehicles, service robots, or interactive voice-based assistants.

This paper presents an overview of the EPSRC-funded project Start Making Sense: Cognitive and Affective Confidence Measures for Explanation Generation Using Epistemic Planning,[1] which is investigating the need for explainability and trust maintenance in interactive and autonomous systems. To do so, this project brings together experimental research in cognitive science involving cooperative joint action with the practical construction of automated planning tools, in particular epistemic planning techniques, to apply to the task of explanation generation. This challenge is being addressed by tackling three key objectives: (i) to study cooperative joint action in humans to identify the emotional, affective, or cognitive factors that are essential for successful human communicative goals; (ii) to enhance epistemic planning techniques with measures derived from the cognitive science studies; and (iii) to deploy and evaluate the effectiveness of the resulting system with human participants in situations that require explanation.

Central to this work is the idea of understanding the affective measures that humans use during activities like instruction giving, plan following, and explanation generation; both when communication is successful but also when it fails. The goal is to characterise these measures in a form that enables them to be combined with tools based on epistemic planning, an approach that models the changing beliefs of the planner and other agents during the plan generation process. Affective measures will therefore help guide the planner's generation process, for instance as a special type of domain control knowledge or heuristic state information, enabling the planner to use this information not only for task-based action selection, but also to plan appropriate

[1] http://start-making-sense.org/

actions for communicative goals such as explanation generation, possibly as a result of dynamic changes in the interactive context. As a result, this work is also situated in the area of explainable planning, a subarea of the recent trend of research in explainable AI.

In the remainder of the paper we outline our approach and the project's main objectives, directions, and goals.

Related Work

Recent developments in artificial intelligence and machine learning research, such as deep learning, are seen to have resulted in dramatic improvements in prediction and accuracy, but often at the expense of human interpretability. As a result, there has been a rapid growth in research under the general banner of explainable artificial intelligence (XAI), typified by grant funding schemes like DARPA's XAI programme (DARPA 2016) or EPSRC's Human-Like Computing Strategy Roadmap (EPSRC 2017), which seek to address a core need in future systems and to respond to challenges like the EU's "Right to Explanation" initiative (Goodman and Flaxman 2017). Understandably, much work has focused on the specific concerns around, e.g., deep learning, rather than on the general need for human-like explanation.

This project instead builds on research in explainable planning (XAIP) (Fox, Long, and Magazzeni 2017), a subarea of XAI and automated planning (Ghallab, Nau, and Traverso 2004). While techniques like machine learning make decisions based on mined data, automated planners traditionally build plans of action by using symbolic causal models combined with search techniques. XAIP seeks to address the challenges of XAI—to build trust and transparency when interacting with humans (see, e.g., (Miller 2017))—by leveraging automated planning models and recognising the role that humans play in the planning loop with respect to systems deployed with such tools.

XAIP is fast becoming an established recent direction in the planning community, with a first workshop dedicated to this topic (Magazzeni et al. 2018) appearing at the 2018 International Conference on Automated Planning and Scheduling (ICAPS).[2] While some of the underlying ideas concerning planning and explainability have a longer history (Sohrabi, Baier, and McIlraith 2011; Seegebarth et al. 2012), more recent approaches like (Chakraborti et al. 2017; Sreedharan, Chakraborti, and Kambhampati 2017), have resulted in new directions and new planning algorithms.

Approach

In contrast to most approaches in XAIP, our work combines research from cognitive science on affective measures used in human communication, together with recent techniques in automated planning, notably epistemic planning. In this section, we briefly highlight the key ideas from these two areas and how they are being brought together.

Affective Measures in Human Communication

From the cognitive perspective, we build on the view that effective explanations arise from cooperative joint action and efficiently satisfy the goals of human communication. For an agent (human or artificial) involved in an interaction, information exchanged needs to lead to acceptance, comfort, and trust in their communication partner, if they are to successfully influence the interlocutor's actions. However, the establishment of trust also depends on the shortcuts, heuristics, and spontaneous choices that people make in interactions, which are often based on emotional or affective factors. As in affective computing more generally, an artificial agent needs to: (i) detect the signals of its interlocutor's affective state; (ii) interpret and understand the meaning behind those signals to infer conclusions; and (iii) be able to take appropriate actions which measurably influence that state.

While confidence and comfort can lead to trust in technologies (Nass and Brave 2005), there is evidence from instruction giving experiments like the HCRC Map Task (Anderson et al. 1991)[3] that have shown that speakers typically under-explain until things go wrong. Paradoxically, successful response to such a failure may build greater confidence in listeners than a consistently verbose explanatory approach which minimises failure rate; similarly, some explanation or correction strategies can be disorienting or annoying (Foster et al. 2009; Henderson, Matheson, and Oberlander 2012). Thus, an important aspect of this project is to understand and develop the capacity to diagnose and repair failures which may be signalled only through brief facial expressions, head movements, or altered body posture.

To this end, the project is conducting user studies with human participants to understand the affective measures that are helpful for effective explanation. Initially, data from previous projects is being used, to study preferred styles in human explanation generation (Carletta et al. 2010) and error detection (Hill and Keller 2014). We are also analysing results on facial expressions and eye-tracking from a current project attempting to measure believability in political messages and what motivates people to agree with, and disseminate, them (Cram 2017). The goal is to synthesise these approaches, adopting mixed methods to collect objective (biometric) and subjective (probe questions) data from individual participants in new experiments. By capturing and annotating conventional linguistic dialogue with paralinguistic signals (e.g., intonation, hesitation, gesture) and behavioural signals (e.g., facial expressions), we expect to identify the best predictors of moment-by-moment shifts in levels of acceptance, comfort and trust that then be used to help build more intelligent artificial systems.

Initial Phase of the Empirical Cognitive Research

The first phase of empirical cognitive research on the project aims to confirm the range of human social signals, affective responses, and behavioural patterns exhibited during co-operative joint action in a shared audio-visual environment. As a starting point, an instruction giving and following scenario is considered based on an enhanced version of the HCRC Map Task (Anderson et al. 1991) paradigm, to simulate navigation with instruction giving and following. In the standard version of the Map Task, an instruction giver

[2] `http://icaps18.icaps-conference.org/xaip/`

[3] `http://groups.inf.ed.ac.uk/maptask/`

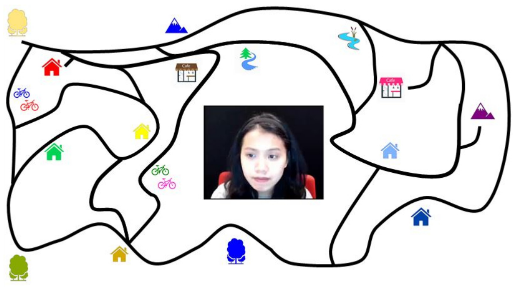


Figure 1: A sample instruction giver map from the enhanced Map Task with an image of a human follower displayed.

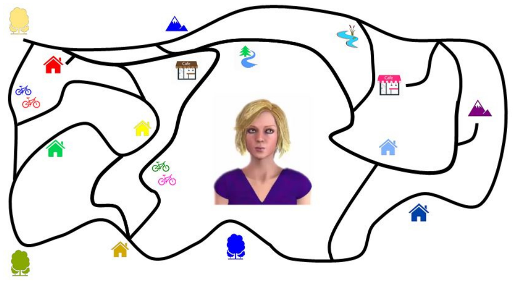


Figure 2: A sample instruction giver map from the enhanced Map Task with an artificial avatar follower displayed.

guides an instruction follower around a map using landmarks. Importantly, the maps of the instruction giver and follower are not aligned: landmarks may be in different locations or in some cases completely different landmarks are present. The task therefore provides opportunities to study how humans communicate and recover from problems in such scenarios. In our enhanced version, we make one critical change: the interlocutors will be clearly visible to each other. These experiments will be used to determine the indicators of successful (or unsuccessful) communication produced during the dynamic process rather simply generating a measure upon completion of the task. In other words, we are proposing a system of continuous monitoring and measurement suitable for agile prediction and adaptive planning.

In the first round of experiments, human participants will observe another human comprehending and responding to their directions in real-time (see Figure 1). Thus, the instruction giver can modify the dialogue based on behavioural feedback and determine the level of success for themselves, e.g. whether the instruction follower is looking at the correct target, has a confused expression or exhibits some other cue. The next set will examine whether human-like behaviour is sufficient to produce the same results or whether every aspect of the interaction needs to be human. To begin understanding this side of the interaction process the same basic paradigm will be adopted, only this time the instruction follower will appear as an artificial avatar (see Figure 2). Critically, however, the avatar is actually generated from the responses of a human recorded by a video camera or webcam. Thus, while the responses may appear to be artificial,

```
action ask-location(?a : agent)
    preconds: K(interact = ?a) &
              !K(requestLoc(?a)) &
              !K(otherAttentionRequests)
    effects:  add(Kf,requestLoc(?a)),
              add(Kv,request(?a))

action give-directions(?a : agent, ?l : loc)
    preconds: K(interact = ?a) &
              K(requestLoc(?a)) &
              Kv(request(?a)) &
              K(request(?a) = ?l) &
              !K(otherAttentionRequests)
    effects:  add(Kf,requestAnswered(?a))
```

Figure 3: Actions for a direction giving agent.

the underlying behaviour is genuinely human and presented through what is essentially a motion-capture system. Ultimately, synthetic stimuli which are generated from our computational models and contingent on the evolving interaction and dialogue will also be tested. These artificial agents will need to portray authentic communication by combining both the fundamental aspects of two-way interaction (comprehension and production): correctly interpreting and understanding observed human behaviour; as well as displaying appropriate human-like reactions.

Epistemic Planning

The main technical tool employed in this project is a recent approach to automated planning (Ghallab, Nau, and Traverso 2004), called epistemic planning (Bolander 2017), which can be used for action selection in state-based, goal-directed systems that operate in the presence of other agents (human or artificial). Traditional automated planners focus on solving the problem of finding an ordered sequence of actions (a *plan*) that, when chained together, transform an initial state into a state where a set of specified goal objectives are achieved. Planning problems are usually described in a symbolic form that specifies the objects, actions, states, and goals that make up the planner's operating environment. A central goal of planning research is to build general purpose or domain-independent planning systems that are able to solve a range of planning problems in different domains, rather than just a single problem in a particular domain.

Epistemic planning builds on standard automated planning approaches and attempts to model how the knowledge and beliefs of agents evolve during the planning process. In this project, plans are generated using PKS (Planning with Knowledge and Sensing) (Petrick and Bacchus 2002; 2004), an early epistemic planning system.

For instance, Figure 3 shows an example of two actions defined in PKS's modelling language for a simple direction giving agent. Here, `ask-location` models an information-gathering action that asks another agent for a location they are trying to reach, while `give-directions` describes an action for supplying a response when such information is provided. Actions are described by their preconditions (the conditions that must be true for an action to be applied) and

Plan	Description
`greet(a1)`	Greet agent `a1`
`ask-location(a1)`	Ask `a1` for a location
`ack-request(a1)`	Acknowledge `a1`'s request
`give-directions(a1, request(a1))`	Respond to `a1`'s request
`bye(a1).`	End the interaction

Table 1: A plan for giving directions to an agent.

```
action ask(?x,?y,?p)
    preconds: ¬K[?x]?p & K[?x]K[?y]?p
    effects:  add(Kf,K[?y]¬K[?x]?p)

action tell(?x,?y,?p)
    preconds: K[?x]?p & K[?x]¬K[?y]?p
    effects:  add(Kf,K[?y]?p)
```

Figure 4: Actions with nested multiagent beliefs.

their effects (the changes the action makes), where references like `K(...)` are queries of the planner's beliefs. The planner uses these actions to form plans by chaining together ground actions instances to achieve the goals of the planning problem. Table 1 shows a possible plan that could be generated by PKS for interacting with a human agent requesting directions to a given location.

An important feature of most epistemic planners is their ability to reason with multiagent beliefs: information about other agents that is often nested (e.g., "agent A believes agent B believes P") (Fagin et al. 1995). This is a challenging problem for automated planners which must provide a solution that is both expressive enough to model a variety of problems while being efficient enough to be implemented in a manner that does not negatively affect the plan generation process. PKS does this by restricting the form of the representation used by the planner and keeping the reasoning language simple (Steedman and Petrick 2007). An example of actions encoded in this way is given in Figure 4 (where `K[?x]p` denotes the idea that "agent `?x` knows `p`").

Epistemic planners like PKS therefore provide us with powerful tools for building systems that can perform action selection with complex reasoning about other agents and their beliefs. For instance, PKS has previously been used for generating plans in task-based scenarios that require socially-appropriate human-robot interaction (Petrick and Foster 2013), and that involve multiple humans.

Explanation Generation

The main technical contribution on this project is to use and enhance an epistemic planner like PKS with intuitions from the cognitive science studies to generate plans which inherently contain more human-like explanations. The goal is to not only generate interactive plans like in Table 1, but to also generate the necessary plan explanations, if required, during the epistemic planning process. Since epistemic planners are capable of reasoning about the beliefs of other agents, we can use the planner's belief about a human agent's knowledge (or lack thereof) of different steps in a plan to automatically drive the explanation generation process. (For instance, if the `give-directions` action involves locations that are believed to be unknown to the human, appropriate explanation can be built into the plan.)

The key technical challenge here is to make the generation process fast while still producing high-quality plans. To do this, the identified affective measures from the cognitive science studies will serve as a type of heuristic to inform and guide the plan generation process and appropriately rank the generated plans. Part of this work therefore involves identifying an effective set of measures that can be captured in the planner's representational language, to ensure that relevant scenarios can be modelled with the planner.

Evaluation of the resulting system with human participants is also essential to establish that the resulting plans achieve their expected behaviour. While we will perform standard planning benchmark tests to establish the correctness, quality, and efficiency of the resulting planning system, we also aim is to keep the human in the loop throughout, using situations like the tour guide or direction giving scenarios, to evaluate response detection and adaptive planning techniques using Wizard-of-Oz and Ghost-in-the-Machine experiments (Janarthanam et al. 2014; Loth et al. 2015).

Conclusions

Combining planning and human interaction, especially in collaborative settings, presents several important challenges that must be addressed due to the necessary presence of humans in the planning loop (Kambhampati and Talamadupula 2015): human activities must be taken into consideration, plans must ensure humans and artificial systems are able to work together effectively for efficient task completion, and plan decisions and effects should be communicated in a manner that improves trust and transparency. This project aims to make contributions in all of these areas. Most importantly, this project places understanding the human experience at the heart of its approach to building tools for explainable epistemic planning, and we have taken a first step in this direction through a series of initial experiments based on the HCRC Map Task paradigm. By basing our technical extensions on affective insights from human studies, and evaluating the result of our tools on human participants, we believe the resulting epistemic planning tools will lead to interactive and autonomous systems that are better prepared for and more acceptable to the expectations of humans.

Acknowledgements

We are indebted to our friend and colleague, Jon Oberlander (`http://homepages.inf.ed.ac.uk/jon/`), who helped shape this project but whose unexpected passing in 2017 means that we must proceed without his invaluable collaboration. Jon will be sorely missed and this project is dedicated to his memory.

This work is funded by the UK's EPSRC Human-Like Computing programme under grant number EP/R031045/1.

References

Anderson, A.; Bader, M.; Bard, E.; Boyle, E.; Doherty, G. M.; Garrod, S.; Isard, S.; Kowtko, J.; McAllister, J.; Miller, J.; Sotillo, C.; Thompson, H. S.; and Weinert, R. 1991. The HCRC Map Task Corpus. *Language and Speech* 34:351–366.

Bolander, T. 2017. A Gentle Introduction to Epistemic Planning: The DEL Approach. *arXiv* 1703.02192.

Carletta, J.; Hill, R. L.; Nicol, C.; Taylor, T.; Ruiter, J. P. d.; and Bard, E. G. 2010. Eyetracking for two-person tasks with manipulation of a virtual world. *Behavior Research Methods & Instrumentation* 42:254–265.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Cram, L. 2017. Citizens' expectations on brexit outcomes: 'fact' transmission and persuasive power in a digital world, ESRC Project.

DARPA. 2016. Explainable Artificial Intelligence (XAI) Program, http://www.darpa.mil/program/explainable-artificial-intelligence.

EPSRC. 2017. A strategy roadmap for human-like computing, https://www.epsrc.ac.uk/newsevents/pubs/human-like-computing-strategy-roadmap.

Fagin, R.; Halpern, J. Y.; Moses, Y.; and Vardi, M. Y. 1995. *Reasoning About Knowledge*. Cambridge, MA, USA: MIT Press.

Foster, M. E.; Giuliani, M.; Isard, A.; Matheson, C.; Oberlander, J.; and Knoll, A. 2009. Evaluating description and reference strategies in a cooperative human-robot dialogue system. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable planning. In *Proceedings of the IJCAI Workshop on Explainable AI*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Goodman, B., and Flaxman, S. 2017. European Union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine* 38(3).

Henderson, M.; Matheson, C.; and Oberlander, J. 2012. Recovering from non-understanding errors in a conversational dialogue system. In *Proceedings of the Workshop on the Semantics and Pragmatics of Dialogue (SemDial)*.

Hill, R., and Keller, F. 2014. Error detection in native and non-native speakers provides evidence for a noisy channel model of sentence processing. In *Proceedings of the 27th Conference on Human Sentence Processing (CUNY)*.

Janarthanam, S.; Hill, R.; Dickinson, A.; and Fredriksson, M. 2014. Click or type: an analysis of wizard's interaction for future wizard interface design. In *Proceedings of EACL Workshop on Dialogue in Motion*, 19–27.

Kambhampati, S., and Talamadupula, K. 2015. Human-in-the-loop planning and decision support. In *AAAI 2015 Tutorial Forum*.

Loth, S.; Jettka, K.; Giuliani, M.; and De Ruiter, J. P. 2015. Ghost-in-the-Machine reveals human social signals for HRI. *Frontiers in Psychology* 6.

Magazzeni, D.; Smith, D.; Langley, P.; and Biundo, S. 2018. *XAIP 2018: Proceedings of the 1st ICAPS Workshop on Explainable Planning*. ICAPS.

Miller, T. 2017. Explanation in artificial intelligence: Insights from the social sciences. *Computing Research Repository* abs/1706.07269.

Nass, C., and Brave, S. 2005. *Wired for Speech: How Voice Activates and Advances the Human-Computer Relationship*. MIT Press.

Petrick, R. P. A., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 212–221.

Petrick, R. P. A., and Bacchus, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2–11.

Petrick, R. P. A., and Foster, M. E. 2013. Planning for social interaction in a robot bartender domain. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*, 389–397.

Seegebarth, B.; Muller, F.; Schattenberg, B.; and Biundo, S. 2012. Making hybrid plans more clear to human users - a formal approach for generating sound explanations. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Sohrabi, S.; Baier, J.; and McIlraith, S. 2011. Preferred explanations: Theory and generation via planning. In *Proceedings of AAAI*.

Sreedharan, S.; Chakraborti, T.; and Kambhampati, S. 2017. Balancing explicability and explanation in human-aware planning. In *Proceedings of the AAAI Fall Symposium on Artificial Intelligence for Human-Robot Interaction (AI-HRI)*.

Steedman, M., and Petrick, R. P. A. 2007. Planning dialog actions. In *Proceedings of SIGdial*, 265–272.

A General Framework for Synthesizing and Executing Self-Explaining Plans for Human-AI Interaction

Sarath Sreedharan[1], Tathagata Chakraborti[2], Christian Muise[2] , Subbarao Kambhampati[1]
[1]CIDSE, Arizona State University, Tempe, AZ 85281 USA
[2]IBM Research AI, Cambridge, MA, USA
ssreedh3@asu.edu, tathagata.chakraborti1@ibm.com, christian.muise@ibm.com, rao@asu.edu

Abstract

In this work, we present a general formulation for decision making in human-in-the-loop planning problems where the human's expectations about an autonomous agent may differ from the agent's own model. We show how our formulation for such multi-model planning problems allows us to capture existing approaches to this problem and also be used to generate novel explanatory behaviors. Our formulation also reveals a deep connection between multi-model planning and epistemic planning and we show how we can leverage classical planning compilations designed for epistemic planning for solving multi-model planning problems. We empirically show how this new compilation provides a computational advantage over previous approaches that separate reasoning about model reconciliation and identifying the agent's plan.

1 Introduction

As automated agents and users start working closely together, it becomes increasingly important that the agents are capable of acting in a manner that is intuitive and explicable to users in the loop. A major challenge to achieving such fluent collaboration is the fact that the human's expectations regarding the agent's capabilities and preferences may differ from reality. Such knowledge asymmetry implies that even in cases where the human teammate is a passive observer, the agent can no longer solely reason with their individual models to generate desirable plans. Instead, the agent needs to explicitly take into account the user's expectations about the agent when coming up with its plans. Previous works have mostly focused on two strategies to handle such scenarios, (a) explanation - the agent chooses to perform its optimal plan and explain the effectiveness of the chosen plan; (c.f [Chakraborti et al., 2017]) (b) explicable planning - the agent chooses to follow viable plans that are closest to user's expectations (c.f [Zhang et al., 2017]).

In the end, we would want an approach that is able to combine the strengths of these two strategies. This would require the agent to move away from standard notions of decision making where the agent is solely trying to optimize the cost of the plan it will follow, but also take into account the ease of explaining the plan as one of the criteria for choosing its actions. We will refer to the problem of generating such plans as **multi-model planning**.

In this work, we will present a general characterization of the problem of multi-model planning and discuss how we could view the solutions to such multi-model planning problems as **self-explaining plans**, where explanations are themselves provided through robot actions. These actions could be purely communicative actions that are meant to update the human's mental model or task level actions that could also have epistemic side effects. The contributions of this paper include,

- Presenting a formalization of multi-model planning that allows us to characterize solutions identified by earlier works (Section 2).
- We look at two additional problem considerations; the phase of interaction (is the explanation occurring during plan selection or is it happening during plan execution); the attentiveness of the user (Section 5) and discuss how such considerations poses both new challenges and provides us with opportunities to generate novel behaviors.
- We present a new planning compilation to solve such planning problems that allows for a uniform treatment of explanation and task level actions. We empirically show how such a compilation could provide a computational advantage (Sections 6 and 7).

We will use Urban Search and Rescue domain as a running example to motivate and illustrate the discussions.

2 Multi-Model Planning and Explanation as Model Reconciliation

The planning models used by both the human and the robot are described by the tuple $\mathcal{M} = \langle F, A, I, G\rangle$. In this formulation, F is the set of propositional fluents used to describe the planning task states, A the set of actions, I the initial state and G the goal. Each action $a \in A$ is further defined as a tuple of the form $a = \langle \text{prec}^a, \text{adds}^a, \text{dels}^a\rangle$, where prec^a lists the preconditions of the action and adds^a and dels^a provides the add and delete effects of the action. In general, the precondition can be some logical formula defined over state fluents and an action a can only be executed in a state S if $S \models \text{prec}^a$. The effects are generally of the form $c \rightarrow e$, where the antecedent

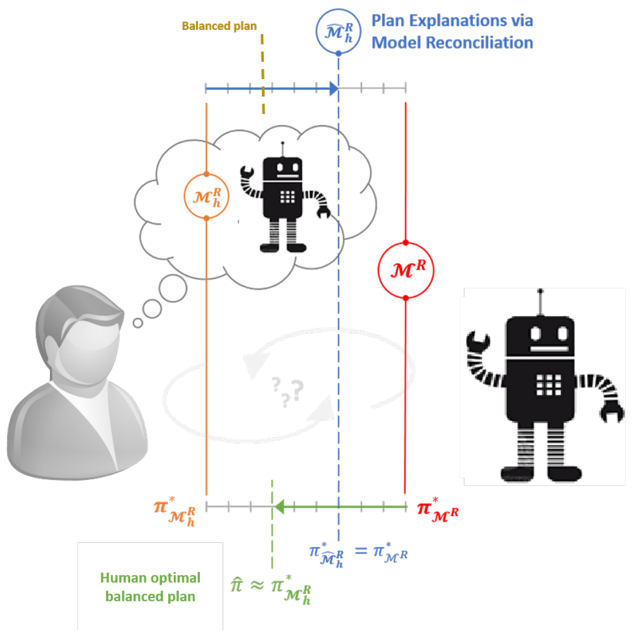

Balanced plan
Plan Explanations via Model Reconciliation
$\mathcal{M}_h^R$
$\widehat{\mathcal{M}}_h^R$
$\mathcal{M}^R$
?,?
$\pi^*_{\mathcal{M}_h^R}$
$\pi^*_{\mathcal{M}^R}$
$\pi^*_{\widehat{\mathcal{M}}_h^R} = \pi^*_{\mathcal{M}^R}$
Human optimal balanced plan
$\hat{\pi} \approx \pi^*_{\mathcal{M}_h^R}$

Figure 1: An overview of planning in multi-model planning settings.

represents the condition under which the effect e should be applied (where the fluent corresponding to e is set true in the state if $c \rightarrow e$ is part of the add effects and if it is part of the delete it is set to false).

Each action is also associated with a cost (represented as $C_{\mathcal{M}}(a)$). A plan in this setting is defined as a sequence of actions ($\pi = \langle a_1, ..., a_n \rangle$) and is said to be valid (denoted as $\pi(I) \models_{\mathcal{M}} G$) for a model $\mathcal{M}$ if $G \subseteq \pi(I)$. Each plan is additionally associated with a cost $C_{\mathcal{M}}(\pi)$ such that $C_{\mathcal{M}}(\pi) = \sum_{i=1}^{n} C_{\mathcal{M}}(a_i)$. A plan π is said to be optimal if there exist no valid plan π' such that $C_{\mathcal{M}}(\pi') < C_{\mathcal{M}}(\pi)$. We will use $\Pi^*_{\mathcal{M}}$ to represent the set of all plans optimal for $\mathcal{M}$.

In this setting we will assume that the robot uses a model $\mathcal{M}_R = \langle F, A_R, I_R, G_R \rangle$ and the human evaluates the plan using a model $\mathcal{M}_H = \langle F, A_H, I_H, G_H \rangle$. For ease of discussion, we concentrate on the specific case where conditions for actions only consist of conjunction of positive literals and with no action cost difference between models.

We start with the assumption that the robot is aware of $\mathcal{M}_H$ and hence knows whether a given plan π_R (that is optimal in $\mathcal{M}_R$) is **explicable** or not, i.e, whether or not the human would identify π_R to be one of the optimal plans for the given planning problem. In cases where the given plan may appear inexplicable, one way the robot could resolve the confusion would be by informing the human about its own model so they can correctly evaluate the current plan. Thus an explanation ($\mathcal{E}$) for this setting can be represented by a set of model updates (where $\mathbb{E}$ represents the set of all possible model updates). The different types of model updates include –
(1) Turn a fluent p true or false in initial state (represented by the operator {add/remove}-p-from-I)
(2) Add or remove a fluent p from the precondition (also add or delete) list of an action a (represented by the operator {add/remove}-p-from-prec-of-a)
(3) Add or remove a fluent p from the goal list (represented by the operator {add/remove}-p-from-G)

We will use the function $\mathcal{T} : \mathbb{M} \times \mathbb{E} \rightarrow \mathbb{M}$ to represent the transition function induced by the model update messages (where $\mathbb{M}$ represents the set of all possible models). A set of model updates ($\mathcal{E}$) **explains** a given plan if in the model resulting from applying the model updates ($\hat{\mathcal{M}} = \mathcal{T}(\mathcal{M}_H, \mathcal{E})$) , the current plan is optimal (i.e., $\pi_R \in \Pi^*_{\hat{\mathcal{M}}}$). Unless otherwise mentioned, when we refer to **updated model**, we are referring to this new human model obtained by applying the explanations. Once we have such a set of model updates, the final explanation (presented to the explainee) can be generated by converting the model updates to corresponding natural language statements [Tellex et al., 2014] or through some appropriate visualization [Chakraborti, Sreedharan, and Kambhampati, 2018a].

Among the valid explanations for a given plan, we refer to the shortest explanation as the minimally complete explanation or MCE. The original work [Chakraborti et al., 2017] on model reconciliation viewed the problem of generating MCE explanation as a problem of searching over the space of possible model updates that can be performed on the human model and the validity of each possible explanation was measured by checking the optimality of the plan in the corresponding planning problem. Before we go into more details, let us briefly look at the urban search and rescue (USAR) domain that will act as the running example for the rest of the paper.

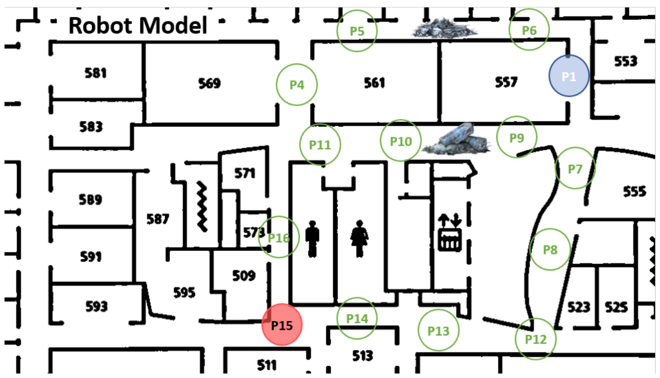
3 Urban Search and Rescue

USAR presents an ideal testbed for research on explainable planning as it looks at cases where the decision to follow suboptimal or in-executable plan can be potentially disastrous, yet limitations in communications capability could prevent the agents from providing detailed explanations.

The basic scenario consists of an autonomous agent that has been deployed to the disaster scene and an external commander who is monitoring the activities of the robot. Both agents start with the same model of the world (i.e the map of the building before the disaster) but the models diverge over time owing to the fact that robot has access to more accurate information about the current status of the building. This model divergence could lead to the commander incorrectly evaluating valid robot plans as sub-optimal or unsafe. One way to satisfy the commander would be to point out possible changes to it's model that led the robot to come up with the plan in the first place.

Figure 2 illustrates a typical scenario where the robot needs to travel from P1 to its goal at P15. Here the human believes the robot should be moving to waypoint P6 and follow that corridor to go to P15, while the robot knows it should be moving to P7. This disagreement rises from the fact that the human incorrectly believes that the path from P6 to P5 is clear while that from P8 to P12 is blocked. If the robot were to follow the explanation scheme that was established in [Chakraborti et al., 2017] then the robot would stick to its own plan and provide the following explanation –

```
> remove-(clear p6 p5)-from-I
```



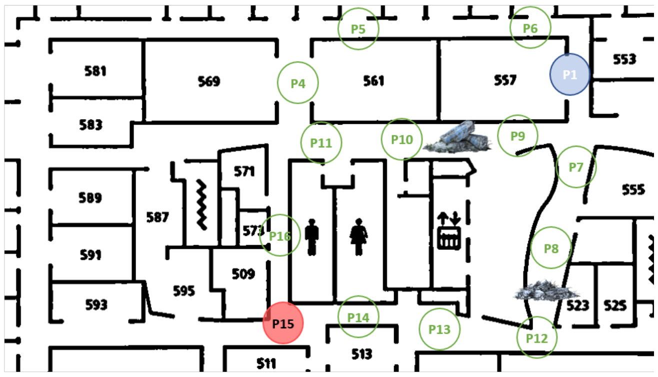


Figure 2: The basic robot and human maps. The robot starts at P1 and needs to go to P15. The human incorrectly believes that the path from P6 to P5 is clear and the one from P8 to P12 is blocked. Both agents know that there are some movable rubble between P9 and P10 that can be moved with the help of a costly clear_passage action.

```
    (i.e., Path from P6 to P5 is blocked)
> add-(clear p8 p12)-to-I
    (i.e., Path from P8 to P12 is clear)
```

4 Multi-Model Planning

In this work, we will be looking at the more general case where we are interested in identifying both the agent plan and the relevant explanation, and we will refer to this problems as multi-model planning

Definition 1. *A* ***multi-model planning problem*** *is defined by the tuple* $\Psi = \langle \mathcal{M}_R, \mathcal{M}_H, \mathcal{T} \rangle$*, where* $\mathcal{M}_H$ *is the human model,* $\mathcal{M}_R$ *the robot model and* $\mathcal{T}$ *is the transition function. A solution to the problem* Ψ *is given by the tuple* $\langle \mathcal{E}^\Psi, \pi_\Psi \rangle$*, where* $\mathcal{E}^\Psi$ *is set of model updates and* π_Ψ *a plan, such that* $\pi_\Psi(I_{\mathcal{M}_R}) \models_{\mathcal{M}_R} G_{\mathcal{M}_R}$ *and* $\pi_\Psi(I_{\mathcal{T}(\mathcal{M}_H, \mathcal{E}^\Psi)}) \models_{\mathcal{T}(\mathcal{M}_H, \mathcal{E}^\Psi)} G_{\mathcal{T}(\mathcal{M}_H, \mathcal{E}^\Psi)}$*.*

The above problem only deals with cases where we care about establishing the validity of a robot plan in the human model. The above problem is already **PSPACE-complete** [1], but we need to go beyond just finding valid solutions to finding explanations that establish the optimality of a robot plan in the human model.

Definition 2. *The tuple* $\langle \mathcal{E}^\Psi, \pi^\Psi \rangle$ *is said to be a complete solution for the problem* Ψ *if* $\pi^\Psi \in \Pi^*_{\mathcal{M}_R}$ *and* $\pi^\Psi \in \Pi^*_{\mathcal{T}(\mathcal{M}_H, \mathcal{E}^\Psi)}$*.*

The use of the term **complete** here is in accordance with it's use in [Chakraborti et al., 2017]. $\mathcal{E}^\Psi$ constitutes an MCE for a plan π_Ψ when there exists no solution of the form $\langle \mathcal{E}^\Psi_2, \pi_\Psi \rangle$, such that $\mathcal{C}_\mathcal{E}(\mathcal{E}^\Psi_2) < \mathcal{C}_\mathcal{E}(\mathcal{E}^\Psi)$, where $\mathcal{C}_\mathcal{E}$ is the cost of providing an explanation. The use of complete solutions are best suited for cases, where the agents choose to stick to a plan that is optimal and explain away any confusion an observer may have regarding the chosen plan. While the agent could choose to follow the optimal plan that is easiest to explain, in many scenario, communicating the explanation could still constitute a considerable expense on the agent's part. It may in fact be more desirable for the agent to follow a sub-optimal plan if the choice of plan results in lower cost for transmitting the explanations. This is a trade-off we make frequently in our day-to-day life and it would be desirable for an explicable agent to be capable of balancing these two sources of cost (i.e the cost of the plan being followed and the cost of communicating the corresponding explanations) [Chakraborti, Sreedharan, and Kambhampati, 2018b].

[1]We can compile plan existence problems into a model reconciliation problem and as shown in section 6 we can compile the problem of finding a solution to plan existence in a conditional planning problem which is again PSPACE hard [Nebel, 2000]

Definition 3. *For a problem* Ψ*, the tuple* $\langle \mathcal{E}^\Psi, \pi^\Psi \rangle$ *is said to be the optimal balanced explicable solution if,*

1. $\pi^\Psi(I_R) \models G_R$.
2. $\pi^\Psi \in \Pi^*_{\mathcal{T}(\mathcal{M}_H, \mathcal{E}^\Psi)}$.
3. $\nexists \langle \hat{\mathcal{E}}, \hat{\pi} \rangle$*, such that the tuple satisfies (1.) and (2.), and* $C(\hat{\mathcal{E}}) + C_{\mathcal{M}_R}(\hat{\pi}) < C(\mathcal{E}^\Psi) + C_{\mathcal{M}_R}(\pi^\Psi)$.

In the USAR domain, to choose a balanced plan the robot can follow the path through P9 that also involves clearing the movable rubble between P9 and P10. In this case the robot only needs to provide a single explanation.

```
> remove-(clear p6 p5)-from-I
    (i.e., Path from P6 to P5 is blocked)
```

The original work that studied the problem of balancing (c.f [Chakraborti, Sreedharan, and Kambhampati, 2018b]) used an additional hyperparameter α, that captures the agent's relative preference to providing explanations versus following a costlier explicable plan. In our formulation, we assume that such preferences are automatically captured by the cost of the explanation. This formulation allows us to better capture the fact that certain aspects of the model may be harder to communicate than others. Conceptually, this formulation should also allow us to include the effort required at the user's end into the explanation cost, but capturing such considerations faithfully would require us to go beyond simple additive models cost explanation cost and so we will ignore them in the current work.

Note that the above formulation looks for solutions where the given plan is completely explicable to the user. In fact the above formulation doesn't allow for the generation of purely explicable plans of the type discussed in [Kulkarni et al., 2019; Zhang et al., 2017], which tries to identify plans closest to the human optimal plan without relying on any explanations. In many cases, generating such plans that may just be "good-enough" may be more desirable to the agent than always trying to stick to completely explainable plans (at the cost of providing more explanation). We can do this by ensuring that the agent could include the extend of inexplicability of a given plan as yet another metric to consider when generating its plans rather than holding it as hard constraint. Assuming that the degree of inexplicability of the plan is directly proportional the degree of suboptimality of the plan in the human model, we can define an optimal balanced solution as follows.

Definition 4. *For a problem Ψ, the tuple $\langle \mathcal{E}^{\Psi}, \pi^{\Psi} \rangle$ is said to be the optimal balanced solution if,*

1. $\pi^{\Psi}(I_R) \models G_R$.
2. $\nexists \langle \hat{\mathcal{E}}, \hat{\pi} \rangle$, *such that the tuple satisfy (1.) and* $(C(\hat{\mathcal{E}}) + C_{\mathcal{M}_R}(\hat{\pi}) + \beta * (C_{\mathcal{T}(\mathcal{M}_H, \hat{\mathcal{E}})}(\hat{\pi}) - C_{\mathcal{T}(\mathcal{M}_H, \hat{\mathcal{E}})}(\pi_1^*))) < (C(\mathcal{E}^{\Psi}) + C_{\mathcal{M}_R}(\pi^{\Psi}) + \beta * (C^{\Psi}_{\mathcal{T}(\mathcal{M}_H, \mathcal{E})}(\hat{\pi}) - C^{\Psi}_{\mathcal{T}(\mathcal{M}_H, \mathcal{E})}(\pi_2^*)))$ *where* $\pi_1^* \in \Pi^*_{\mathcal{T}(\mathcal{M}_H, \hat{\mathcal{E}})}$ *and* $\pi_1^* \in \Pi^*_{\mathcal{T}(\mathcal{M}_H, \mathcal{E}^{\Psi})}$ *and* β *captures the degree of penalty imposed by inexplicability of the given plan.*

While the current work uses the difference of cost as the measure of inexplicability, one could have use other plan distances (or a combination) as standin for this measure. This is especially helpful when the user's model/preferences are not completely known.

5 Additional Considerations

Now that we have defined some of the problems and solution concepts that have been covered in earlier works, we will now cover some additional problem considerations related to multi-model planning that has been previously ignored.

5.1 The Stage of Interaction

An important factor we have more or less ignored in previous sections (and for that matter most of the previous works in model reconciliation) is whether the system is providing an explanation for a plan that has it has proposed or is the system providing post-hoc explanation for some plan that it has just executed (we will refer to such scenarios as explaining behaviors). One may be tempted to argue that this distinction is unnecessary if we are limiting our attention to completely specified sequential plans in deterministic domains, but the fact that the user is viewing a plan being executed gives us opportunities to simplify explanation that may just not be available when the user and the system are deliberating over a proposed plan. In particular, we could use the agents actions (the fact they were successfully executed and their effects) to help shape the user's perception about the task and the agent's capabilities. For example, the robot opening a door is enough to inform the human that the door was not locked in the first place and does not require a separate communication action or the robot could go through a passage to show to the human that the passage was not blocked. Thus in these multi-model settings, we need to allow for the fact that these actions not only have an effect on the task level state (i.e ontic effects) but may also have epistemic effects on the user's mental models. We could go one further step and treat all explanations as executing actions with epistemic effects. This means that model-reconciliation planning is in fact an epistemic planning problem. Previous works such as [Muise et al., 2015] have shown how certain subsets of epistemic planning problem can be compiled to classical planning problems. In the following sections, we will discuss how we can leverage similar compilation schemes to capture multi-model planning problems and show how such methods provide us with computational advantages over methods that separate reasoning about explanations and the task level plans.

5.2 User's Attention

An assumption made by many of the earlier works is the fact that the human observer is a perfect listener. Which means that once the explanation is provided she will definitely include it in her reasoning. Unfortunately, this is not true in most cases. The human's ability to understand the explanation may depend on factors like the hardness of the concept being explained (for eg: explaining the robot's reach in terms of motion constraints vs the ability for the robot to pick up heavy objects) and the mode of explanation (a simple visualization vs natural language) and even the cognitive load of the listener. A more realistic approach would require us to model the fact that the user may not necessarily consider some information even when it is presented to them. The system may need to repeat its explanatory messages or use simpler explanations. Computing such strategies would require us to move to formulations that allows for non-deterministic or stochastic transitions for explanatory messages. In subsection 6.4, we discuss how our new formulation can be extended to support such scenarios.

6 Compilation

To support planning with explanatory actions, we will adopt a formulation that is similar to the one introduced in [Muise et al., 2015] to compile reasoning about epistemic states into a classical planning problem.In our setting, each explanatory actions can be viewed as an action with an epistemic effect. One interesting distinction to make here is the fact that the human's belief state now not only includes their belief about the task state but also their belief about the robot's model. This means that the planning model will need to separately keep track of (1) the current robot state, (2) the human's belief regarding the current state, (3) how actions would effect each of these (as humans may have differing expectations about the effects of each action) and (4) how those expectations change with explanations.

Given the model reconciliation planning problem $\Psi = \langle \mathcal{M}_R, \mathcal{M}_H \rangle$, we will generate a new planning model $\mathcal{M}_{\Psi} = \langle F_{\Psi}, A_{\Psi}, I_{\Psi}, G_{\Psi} \rangle$ as follows $F_{\Psi} = F \cup F_{\mathcal{B}} \cup F_{\mu} \cup \{\mathcal{G}, \mathcal{I}\}$, where $F_{\mathcal{B}}$ is a set of new fluents that will be used to capture the human's belief about the task state and F_μ is a set of meta fluents that we will use to capture the effects of explanatory actions and $\mathcal{G}$ and $\mathcal{I}$ are special goal and initial state propositions. We will use the notation $\mathcal{B}(p)$ to capture the human's belief about the fluent p. We are able to use a single fluent to capture the human belief as we are specifically dealing with a scenario where the human's belief about the robot model is fully known. In this case, we also do not require any of the additional rules that were employed in [Muise et al., 2015] to ensure that the state captures the deductive closure of the agent beliefs.

F_μ will contain an element for every part of the human model that can be changed by the robot through explanations. A meta fluent corresponding to a literal ϕ from the precondition of an action a takes the form of $\mu^+(\phi^{\text{prec}^a})$, where the superscript $+$ refers to the fact that the clause ϕ is part the precondition of the action a in the robot model (for cases where the fluent represents an incorrect human belief we will be using the superscript $-$).

For every action $a = \langle \text{prec}^a, \text{adds}^a, \text{dels}^a \rangle \in A_R$ and its human counterpart $a_h = \text{prec}^{a_h}, \text{adds}^{a_h}, \text{dels}^{a_h} \in A_H$, we define a new action $a_\Psi = \langle \text{prec}^{a_\Psi}, \text{adds}^{a_\Psi}, \text{dels}^{a_\Psi} \rangle \in \mathcal{M}_\Psi$ whose precondition is given as –

$$\begin{aligned}\text{prec}^{a_\Psi} = \text{prec}^{a_R} \cup \{\mu^+(\phi^{\text{prec}^a}) \to \mathcal{B}(\phi) | \phi \in \text{prec}^{a_R} \setminus \text{prec}^{a_H}\} \\ \cup \{\mu^-(\phi^{\text{prec}^a}) \to \mathcal{B}(\phi) | \phi \in \text{prec}^{a_H} \setminus \text{prec}^{a_R}\} \\ \cup \{\mathcal{B}(\phi) | \phi \in \text{prec}^{a_H} \cap \text{prec}^{a_R}\}\end{aligned}$$

The important point to note here is that at any given state, an action in the augmented model is only applicable if the action is executable in robot model and the human believes the action to be executable. Unlike the executability of the action in the robot model (captured through unconditional preconditions) the human's beliefs about the action executability can be manipulated by turning the meta fluents on and off.

The effects of these actions can also be defined similarly. In addition to these task level actions (represented by the set A_τ), we can also define explanatory actions (A_μ) that either add $\mu^+(*)$ fluents or delete $\mu^-(*)$.

Special actions a_0 and a_∞ that are responsible for setting all the initial state conditions true and checking the goal conditions are also added into the domain model. a_0 has a single precondition that checks for $\mathcal{I}$ and has the following add and delete effects –

$$\begin{aligned}\text{adds}^{a_0} = \{\top \to p \mid p \in I_R\} \cup \{\top \to \mathcal{B}(p) \mid p \in I_H\} \\ \cup \{\top \to p \mid p \in F_{\mu^-}\}\end{aligned}$$

$$\text{dels}^{a_0} = \{\mathcal{I}\}$$

where F_{μ^-} is the subset of F_μ that consists of all the fluents of the form $\mu^-(*)$. Similarly, the precondition of action a_∞ is set using the original goal and adds the proposition $\mathcal{G}$.

$$\begin{aligned}\text{prec}^{a_\infty} = G_R \cup \{\mu^+(p^G) \to \mathcal{B}(p) \mid p \in G_R \setminus G_H\} \cup \\ \{\mu^-(p^G) \to \mathcal{B}(p) \mid p \in G_H \setminus G_R\} \cup \{\mathcal{B}(p) \mid G_H \cap G_R\}\end{aligned}$$

Finally the new initial state and the goal specification becomes $I_\mathcal{E} = \{\mathcal{I}\}$ and $G_\mathcal{E} = \{\mathcal{G}\}$ respectively. To see how such a compilation would look in practice, consider an action (move_from p1 p2) that allows the robot to move from point p1 to p2 only if the path is clear. The action is defined as follows in the robot model.

```
(:action move_from_p1_p2
    :precondition (and (at_p1)
                       (clear_p1_p2))
    :effect (and (not (at_p1))
                      (at_p2) ))
```

Let's assume the human is aware of this action but doesn't know that they need to care about the status of the path (as they assume the robot can move through any debris filled path). In this case, the corresponding action in the augmented model and the relevant explanatory action will be

```
(:action move_from_p1_p2
    :precondition
    (and
        (at_p1) (B((at_p1))) (clear_p1_p2)
        (implies
          (μ+_prec(move_from_p1_p2,
                  (clear_p1_p2)))
          (B((clear_p1_p2))) ))
    :effect
    (and   (not (at_p1)) (at_p2)
           (not B(at_p1))
           B(at_p2)) ))
```

```
(:action explain_μ+_prec_move_from_clear
    :precondition
    (and )
    :effect
    (and μ+_prec(move_from_p1_p2,
            (clear_p1_p2)) ))
```

We will refer to an augmented model that contains an explanatory action for each possible model update and has no actions with effects on both the human's mental model and the task level state as the **canonical augmented model**.

Given an augmented model, let $\pi_\mathcal{E}$ be some plan that is valid for this model ($\pi_\mathcal{E}(I_\Psi) \subseteq G_\Psi$). From $\pi_\mathcal{E}$, we extract two types of information – the model updates induced by the actions in the plan (represented as $\mathcal{E}(\pi_\mathcal{E})$) and the sequence of actions that have some effect of the task state (henceforth referred to as actions with ontic effects) represented as $\mathcal{D}(\pi_\mathcal{E})$. Note that $\mathcal{E}(\pi_\mathcal{E})$ may contain effects from action in $\mathcal{D}(\pi_\mathcal{E})$. This brings us to the following proposition –

Proposition 1. *Given a multi-model planning problem Ψ and the corresponding augmented model $\mathcal{M}_\Psi$, then for any plan $\pi(I_\Psi) \models_\Psi G_\Psi$, the tuple $\langle \mathcal{E}(\pi), \mathcal{D}(\pi) \rangle$ is a valid solution for Ψ.*

This result can be trivially shown to be true given the above formulation. Unfortunately, the compilation on it's own only takes care of generating plans along with the justifications for correctness of the plan. In many cases, the user would also be interested in understanding why the given plan is optimal.

Claim 1. *Given a multi-model planning problem Ψ and the corresponding augmented model $\mathcal{M}_\Psi$, then there exists a plan $\pi(I_\Psi) \models_\Psi G_\Psi$, such that the tuple $\langle \mathcal{E}(\pi), \mathcal{D}(\pi)\rangle$ is a complete solution for Ψ but π may not be optimal for $\mathcal{M}_\Psi$.*

The first part of the claim comes from the fact that the space of valid plans for $\mathcal{M}_\Psi$ spans the entire set of valid plans for the robot model and the set of all possible model updates. On the other hand, due to the structure of the preconditions, for a given a set of model updates, there may be plans that are valid in the human model that never gets expanded. This means when the search comes up with a plan π^* that is optimal for $\mathcal{M}_\Psi$, it is possible that $\mathcal{M}_H + \mathcal{E}(\pi^*)$ could have plans cheaper than $\mathcal{D}(\pi^*)$, that were not expanded as they were invalid in the robot model.

6.1 Planning For Complete Solutions Using Augmented Model

A way to generate complete solutions would be by updating the goal test used by the search. In addition to checking if the goal facts are indeed met in the resultant state, we would now also need to check that the plan is in fact optimal in the updated model. Note that this is an inversion of the search in [Chakraborti et al., 2017] with the added advantage that we are explicitly reasoning with explanatory actions and only check for plan optimality in human models that are guaranteed to support at least one robot executable plan. Furthermore, if we memoize the results of each secondary search with respect to $\mathcal{E}(\pi)$, we can guarantee that the number of optimality tests will be less than or equal to the number of tests required by the earlier approach. Given the nature of this suboptimality test, it should be possible to leverage methods like search space reuse to speed up search, but since our focus here is on establishing the properties of the simplest formulation we will focus on cases that use a simple optimality test.

The next question to ask would be, under what conditions can this new encoding be guaranteed to generate explanations that are minimally complete. Before we formally state the conditions, let us define a new concept called **optimality gap** (denoted as $\Delta\pi_{\mathcal{M}}$) for a planning model, which captures the cost difference between the optimal plan and the second most optimal plan. $\Delta\pi_{\mathcal{M}}$ can be defined as –

$$\Delta\pi_{\mathcal{M}} = \max\{v \mid v \in \mathbb{R} \wedge \nexists \pi_1, \pi_2((0 < (C(\pi_1) - C(\pi_2)) < v) \wedge \pi_1(I_{\mathcal{M}}) \models_{\mathcal{M}} G_{\mathcal{M}} \wedge \pi_2(I_{\mathcal{M}}) \in \Pi^*_{\mathcal{M}}\}$$

Theorem 1. *Given a canonical augmented model $\mathcal{M}_\Psi$ for a multi-model planning problem $\Psi = \langle \mathcal{M}_R, \mathcal{M}_H, \mathcal{T}\rangle$, if the sum of costs of all explanatory actions is less than or equal to $\Delta\pi_{\mathcal{M}_R}$ and if π is the cheapest valid plan for $\mathcal{M}_\Psi$ such that $\mathcal{D}(\pi) \in \Pi^*_{\mathcal{T}(\mathcal{M}_\Psi, \mathcal{D}(\pi))}$, then*

(1) $\mathcal{D}(\pi)$ is optimal for $\mathcal{M}_R$

(2) $\mathcal{E}(\pi)$ is the MCE for $\mathcal{D}(\pi)$

(3) There exists no plan $\hat{\pi} \in \Pi^*_R$ such that MCE for $\mathcal{D}(\hat{\pi})$ is cheaper than $\mathcal{E}(\pi)$,i.e, the search will find an optimal explicable plan if one exists.

Proof Sketch. We observe that there exists no valid plan π' for the augmented model ($\mathcal{M}_\Psi$) with a cost lower than that of π and where the ontic fragment ($\mathcal{D}(\pi')$) is optimal for the human model. Let's assume $\mathcal{D}(\pi) \notin \Pi^*_{\mathcal{R}}$ (i.e current plan's ontic fragment is not optimal in robot model) and let $\hat{\pi} \in \Pi^*_{\mathcal{R}}$. Now let's consider the augmented plan corresponding to $\hat{\pi}$, $\hat{\pi}_{\mathcal{E}}$, i.e, $\mathcal{E}(\hat{\pi}_{\mathcal{E}})$ is the MCE for the plan $\hat{\pi}$) and $\mathcal{D}(\hat{\pi}_{\mathcal{E}}) = \hat{\pi}$. Then the given augmented plan $\hat{\pi}_{\mathcal{E}}$ is a valid solution for our augmented planning problem $\mathcal{M}_\Psi$ (since the $\hat{\pi}_{\mathcal{E}}$ consists of the MCE for $\hat{\pi}$, the plan must be valid and optimal in the human model), moreover the cost of $\hat{\pi}_{\mathcal{E}}$ must be lower than π. This contradicts our earlier assumption hence we can show that $\mathcal{D}(\pi)$ is in fact optimal for the robot model.

Using a similar approach we can also show that no cheaper explanation exists for $\pi_{\mathcal{E}}$ and there exists no other plan with a cheaper explanation. □

Also note, that while it is hard to find the exact value for the optimality gap, we are guaranteed that the optimality gap is greater than or equal to one for domains with only unit cost actions or is guaranteed to be greater than or equal to $(C_2 - C_1)$, where C_1 is the cost of the cheapest action and C_2 is the cost of the second cheapest action (i.e $\forall a, (C_{\mathcal{M}}(a) < C_2 \rightarrow C_{\mathcal{M}}(a) = C_1)$)

6.2 Balanced Plans

We can use the above formulation directly to obtain balanced explicable plans, we just no longer need to put any specific restriction on the cost of explanatory action. To generate optimal balanced plans, we need to relax the requirement that the final plan is optimal in the human model. Instead we can incorporate the inexplicability penalty into the reasoning about the plan, by assigning the cost of a_∞ (the goal action) to be β times the difference between the optimal plan in the human model and the current plan. When β is set to zero the problem would just identify the optimal plan corresponding to the original robot model and when β is set high enough the formulation just generates explicable balanced plans. This can be guaranteed if beta is set higher that κ, where κ is some upper bound on plan length for the robot (that includes explanatory actions). On the other hand, if the cost of any explanatory actions is also higher than κ then the formulation will try to find the plan that is closest to an optimal plan in the original human model and is still executable in the robot model.

To see the use of balanced plans, lets revisit the urban search and rescue case. Here, the optimal path for the robot to follow would be to go through P7, P8 and P12. The human thinks the path should be the one through P6 and P5. Explaining the optimality of the path requires explaining that the path from P6 to P5 is blocked (which can be explained through the action explain_obstructed_P6_P5) and the path from P8 to P12 is clear (explained by explain_away_obstructed_P8_P12). Let us assume that the application of each of these explanatory actions increases the total cost by ten. The balanced explicable plan in this setting would be

explain_obstructed_P6_P5→INIT_ACT→ move_from_p1_p9→clear_passage_p9_p10→ move_from_p9_p10→move_from_p10_p11→ move_from_p11_p16→move_from_p16_p15→

GOAL_ACT

If we try to identify an optimal balanced plan for $\beta = 1$ and a cost of 5 for the clear passage action, then the plan that would be generated would be –

INIT_ACT→ move_from_p1_p9→clear_passage_p9_p10→ move_from_p9_p10→move_from_p10_p11→ move_from_p11_p16→move_from_p16_p15→ GOAL_ACT

6.3 Plans with Epistemic Side-effects

As mentioned earlier, in cases where the user is observing a plan being executed, even the agent's non-explanatory actions could have effects on user's mental model. We can easily incorporate this requirement by associating effects involving meta-fluents into our task specific actions. Such effects may be specified by domain experts or could be generated using heuristic rules. For example, if an action is executed in a state where a precondition believed by the user is not met then that precondition should be removed from the human's perceived model.

To illustrate the use of such explanatory actions in our encoding let us visit the USAR scenario and assume that the human thinks that the path from P8 to P12 is blocked and the one from P6 to P5 is free. Also in this setting, for explaining the status of passages (whether they are blocked or not) the robot can now use two actions, one a rather expensive explicit communication action, that sends the updated map information to the human or the robot can just visit the blocked passage and the human who is watching a video feed of robot actions will learn that the passage is blocked or clear. Thus the action descriptions for the move action will be –

```
(:action move_from_P7_P8
    :precondition (and (robot_at_P7)
          ...
          (𝓑(robot_at_P7)))
    :effect (and (robot_at_P8)
          ...
          (𝓑(clear_P8_P12))
          (increase (total-cost) 1)))
```

With this new action the robot knows that as soon as it reaches P8 the human would know that the path from P8 to P12 is clear so it can continue on that path. So the new robot plan will be –

INIT_ACT→move_from_p1_p7→move_from_p7_p8→ move_from_p8_p12→move_from_p12_p13→ explain_obstructed_p6_p5→ move_from_p12_p13→move_from_p13_p14→ move_from_p14_p15→GOAL_ACT

6.4 Plans for Inattentive Users

In this scenario, we can no longer assume that explanatory actions would have deterministic effects on user's model and that means considering planning models that allow for non-deterministic or stochastic effects.

Domain	New Compilation cov.	New Compilation runtime	Model Space Search cov.	Model Space Search runtime
Blocksworld	13/15	569.38	13/15	2318.73
Elevator	15/15	59.20	1/15	3382.462
Gripper	5/15	2301.90	6/15	2093.54
Driverlog	4/15	2740.38	2/15	3158.59
Satellite	2/15	3186.93	0/15	3600

Figure 3: Table showing average runtime (sec) and coverage for explanations generated for standard IPC domains.

To see a simple example of how this would look, consider the USAR domain and look at the ability of the move action to inform the commander about the status of the passage from P8 to P12. The robot can not always guarantee that the commander would be looking at the screen and there is a chance that the commander won't be looking at the screen when the path is presented. Thus in the new definition of action (move_from_P7_P8) we will replace the effect that adds $\mathcal{B}$(clear_P8_P12) with the effect (one-off ($\mathcal{B}$(clear_P8_P12)) (and)) Which means that the action's ability to update the human model is a non-deterministic effect. In this case, we can look for a conformant plan by converting the earlier search into a search over belief space. A search node only passes the goal test if the goal condition are met in every state in the belief space. Thankfully, this particular problem does have a conformant solution –

explain_obstructed_p6_p5→ explain_clear_p8_p12→INIT_ACT→ move_from_p1_p7→move_from_p7_p8→ move_from_p8_p12→move_from_p12_p13→ move_from_p13_p14→move_from_p14_p15→ GOAL_ACT

7 Empirical Comparison of Model-Space Search and Planning with Explanatory Actions

The focus of this section is to see how our compilation compares with the approaches that separate the reasoning about explanations and plan generation. In particular, we will consider the approaches discussed in [Chakraborti, Sreedharan, and Kambhampati, 2018b] as a point of comparison. Note that in order for the model space search to always identify the optimal balanced explicable plan, generating an optimal plan at each possible model is not enough. The approach would require iterating over the space of all possible optimal plans at a given node to find one that is executable in the robot model or require involved compilations that only produce optimal plans that are executable in robot model. To avoid changing the method too much, we used an optimistic version of the model space search that only identifies one optimal plan per search node and the search ends as soon as it find a node where the optimal plan produced has the same cost as the robot's plan and is executable in the robot model.

For comparison, we selected five IPC domains and for each domain, we created three unique models by introducing 10 random updates in the model (except in the case of gripper and driverlog where only 5 were removed). Each of these three domains were paired with five problem instances and then tested on each of the possible configurations. Each instance was run with a limit of 30 minutes, all explanatory actions were restricted to the beginning of the plan and the cost of explanatory actions were set to be twice the cost of original action. Table 3 lists the time taken to solve each of these problems. For calculating the average runtime, we used 1800 secs as the stand in for the runtime of all the instances that timed out. We used h_max as the heuristic for all the configurations.

As clearly apparent from the table, the new approach does better than the original method for generating balanced plans for most of the domains. Gripper seems to be the only domain, where model search seem to be doing better but this is also a domain that had the smallest number of model differences. This points to the fact that the ability to leverage planning heuristics seems to make a marked difference in domains with a large number of possible explanatory actions.

8 Related Work

It's widely accepted in social sciences literature that explanations must be generated while keeping in mind the beliefs of the agent receiving the explanation [Miller, 2018; Slugoski et al., 1993]. As such, epistemic planning makes for an excellent framework for studying the problem of generating these explanations. While the most general formulation of epistemic planning has been shown to be undecidable, many simpler fragments have been identified [Bolander, Jensen, and Schwarzentruber, 2015]. Recently, there have been a lot of interest in developing efficient methods for planning in such domains [Muise et al., 2015; Kominis and Geffner, 2015; 2017; Le et al., 2018; Huang et al., 2018]. In our base scenario, we will assume (1) a finite nesting of beliefs, (2) the human is merely an observer, and (3) all actions are public. The specific problems discussed in our paper hardly exercises most of the capabilities provided by epistemic planning. It's important to note that given the epistemic nature of the explanatory actions, solving the general model reconciliation problem would require leveraging all those capabilities. Our hope is that by presenting model reconciliation in this more general setting, the community would be motivated to start looking at more general and complex versions of these problems.

Our work also looks at the use of explanatory actions as a means of communicating information to the human observer. The most obvious types of such explanatory action includes purely communicative actions such as speech [Tellex et al., 2014] or the use of mixed reality projections [Chakraborti, Sreedharan, and Kambhampati, 2018a; Ganesan, 2017], but recent works have shown that physical agents could also use movements to relay information such as intention [MacNally et al., 2018; Dragan, Lee, and Srinivasa, 2013] and incapability [Kwon, Huang, and Dragan, 2018]. Our framework could be easily adopted to any of these explanatory actions and would naturally allow for a trade-off between these different types of communication.

Many recent works dealing with explanation generation for planning, have looked at characterizing explanation in terms of the types of questions they answer (c.f [Fox, Long, and Magazzeni, 2017; Smith, 2012] and contrastive explanations in general). This characterization is orthogonal to the question of what type of information constitutes valid explanations. Putting aside questions regarding observability, the reason why a user requests an explanation is either due to knowledge asymmetry (incomplete or incorrect knowledge of the task) or due to limitations of their inferential capabilities. Depending on the context, the answer to any of the questions described in these papers would require correcting human's model of the task and/or providing inferential assistance. Works that have looked at model reconciliation explanations have mostly focused on the former. Explanations discussed in this paper can be viewed as an answer to the question "Why this plan?" (which can also be viewed as a contrastive question of the form "Why this plan and not any other plan?"). This is not to say that in complex scenarios just the model reconciliation information would suffice but it would need to be supplemented with information that can bridge the differences in inferential capabilities. Use of abstractions [Sreedharan, Srivastava, and Kambhampati, 2018], providing refutation of specific foils [Sreedharan, Srivastava, and Kambhampati, 2018] and providing causal explanations [Seegebarth et al., 2012] could all be used to augment model reconciliation explanations.

9 Conclusion and Discussion

The paper presents a more general formulation for the problem of planning with users in the loop with asymmetric models than any of the previous works. We discuss how this formulation can be extended to capture novel explanatory behaviors and can be solved using approaches that are computationally more efficient than methods that rely on direct model space search. One possible avenue for future work would be investigating and implementing planning compilations that capture extensions of the model reconciliation problem that have previously been investigated like specific foils, uncertain human models, state abstractions, differences in action costs, disjunctive preconditions, etc... It would also be worth investigating if there are any specific considerations to be made when choosing heuristics for such planning models.

Acknowledgments

This research is supported in part by the ONR grants N00014-16-1-2892, N00014-18-1-2442, N00014-18-1-2840, the AFOSR grant FA9550-18-1-0067, NASA grant NNX17AD06G and JP Morgan faculty research grant.

References

[Bolander, Jensen, and Schwarzentruber, 2015] Bolander, T.; Jensen, M. H.; and Schwarzentruber, F. 2015. Complexity results in epistemic planning. In *IJCAI*.

[Chakraborti et al., 2017] Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *IJCAI*.

[Chakraborti, Sreedharan, and Kambhampati, 2018a] Chakraborti, T.; Sreedharan, S.; and Kambhampati, S.

2018a. Projection-Aware Task Planning and Execution for Human-in-the-Loop Operation of Robots in a Mixed-Reality Workspace . In *IROS*.

[Chakraborti, Sreedharan, and Kambhampati, 2018b] Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2018b. Explicability versus explanations in human-aware planning. In *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems.

[Dragan, Lee, and Srinivasa, 2013] Dragan, A. D.; Lee, K. C.; and Srinivasa, S. S. 2013. Legibility and predictability of robot motion. In *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*, 301–308. IEEE Press.

[Fox, Long, and Magazzeni, 2017] Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable Planning. In *IJCAI XAI Workshop*.

[Ganesan, 2017] Ganesan, R. K. 2017. *Mediating Human-Robot Collaboration through Mixed Reality Cues*. Ph.D. Dissertation, Arizona State University.

[Huang et al., 2018] Huang, X.; Fang, B.; Wan, H.; and Liu, Y. 2018. A general multi-agent epistemic planner based on higher-order belief change. *arXiv preprint arXiv:1806.11298*.

[Kominis and Geffner, 2015] Kominis, F., and Geffner, H. 2015. Beliefs in multiagent planning: From one agent to many. In *ICAPS*.

[Kominis and Geffner, 2017] Kominis, F., and Geffner, H. 2017. Multiagent online planning with nested beliefs and dialogue. In *ICAPS*.

[Kulkarni et al., 2019] Kulkarni, A.; Chakraborti, T.; Zha, Y.; Vadlamudi, S. G.; Zhang, Y.; and Kambhampati, S. 2019. Explicable Robot Planning as Minimizing Distance from Expected Behavior. In *AAMAS*.

[Kwon, Huang, and Dragan, 2018] Kwon, M.; Huang, S. H.; and Dragan, A. D. 2018. Expressing robot incapability. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, 87–95. ACM.

[Le et al., 2018] Le, T.; Fabiano, F.; Son, T. C.; and Pontelli, E. 2018. Efp and pg-efp: Epistemic forward search planners in multi-agent domains. In *ICAPS*.

[MacNally et al., 2018] MacNally, A. M.; Lipovetzky, N.; Ramirez, M.; and Pearce, A. R. 2018. Action selection for transparent planning. In *AAMAS*, 1327–1335. International Foundation for Autonomous Agents and Multiagent Systems.

[Miller, 2018] Miller, T. 2018. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*.

[Muise et al., 2015] Muise, C. J.; Belle, V.; Felli, P.; McIlraith, S. A.; Miller, T.; Pearce, A. R.; and Sonenberg, L. 2015. Planning over multi-agent epistemic states: A classical planning approach. In *AAAI*.

[Nebel, 2000] Nebel, B. 2000. On the expressive power of planning formalisms. In *Logic-based artificial intelligence*. Springer. 469–488.

[Seegebarth et al., 2012] Seegebarth, B.; Müller, F.; Schattenberg, B.; and Biundo, S. 2012. Making hybrid plans more clear to human users-a formal approach for generating sound explanations. In *ICAPS*.

[Slugoski et al., 1993] Slugoski, B. R.; Lalljee, M.; Lamb, R.; and Ginsburg, G. P. 1993. Attribution in conversational context: Effect of mutual knowledge on explanation-giving. *European Journal of Social Psychology* 23(3):219–238.

[Smith, 2012] Smith, D. E. 2012. Planning as an iterative process.

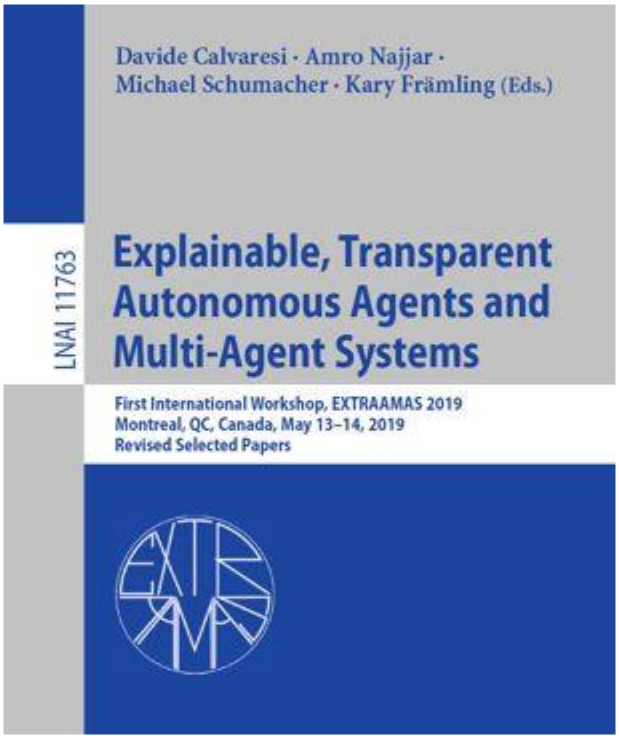
[Sreedharan, Srivastava, and Kambhampati, 2018] Sreedharan, S.; Srivastava, S.; and Kambhampati, S. 2018. Hierarchical expertise-level modeling for user specific contrastive explanations. In *IJCAI*.

[Tellex et al., 2014] Tellex, S.; Knepper, R.; Li, A.; Rus, D.; and Roy, N. 2014. Asking for help using inverse semantics. In *R:SS*.

[Zhang et al., 2017] Zhang, Y.; Sreedharan, S.; Kulkarni, A.; Chakraborti, T.; Zhuo, H. H.; and Kambhampati, S. 2017. Plan Explicability and Predictability for Robot Task Planning. In *ICRA*.

Explainable AI

Sister Workshops



Davide Calvaresi · Amro Najjar · Michael Schumacher · Kary Främling (Eds.)

LNAI 11763

Explainable, Transparent Autonomous Agents and Multi-Agent Systems

First International Workshop, EXTRAAMAS 2019
Montreal, QC, Canada, May 13–14, 2019
Revised Selected Papers

IJCAI 2019 WORKSHOP ON EXPLAINABLE ARTIFICIAL INTELLIGENCE (XAI)

11 August, 2019. Macau, China

https://www.ijcai19.org/

Explainable AI

Proceedings of the IJCAI 2019 Workshop on Explainable Artificial Intelligence

Edited by

Tim Miller, University of Melbourne, Australia

Rosina Weber, Drexel University

Daniele Magazzeni, King's College London