DISCOVERHISTORY: Understanding the Past in Planning and Execution

Matthew Molineaux
Knexus Research Corporation
9120 Beachway Lane
Springfield, VA 22153 USA
matthew.molineaux
@knexusresearch.com

Ugur Kuter
Smart Information Flow
Technologies
211 North 1st Street
Minneapolis, MN 55401 USA
ukuter@sift.net

Matthew Klenk
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304 USA
Matthew.Klenk@parc.com

ABSTRACT

We consider the problem of automated planning and control for an execution agent operating in environments that are partially-observable with deterministic exogenous events. We describe a new formalism and a new algorithm, DISCOVERHISTORY, that enables our agent, DHAgent, to proactively expand its knowledge of the environment during execution by forming explanations that reveal information about the world. We describe how DHAgent uses this information to improve the projections made during planning. Finally, we present an ablation study that examines the impact of explanation generation on execution performance. The results of this study demonstrate that our approach significantly increases the goal achievement success rate of DHAgent against an ablated version that does not perform explanation.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Plan execution, formation, and generation; I.2.3 [**Deduction and Theorem Proving**]: Abductive Reasoning; I.2.11 [**Distributed Artificial Intelligence**]: Intelligent Agents

General Terms

Algorithms

Keywords

Planning, execution, abductive reasoning, explanation generation

1. INTRODUCTION

In real-world tasks, perceptions are incomplete and the world is constantly changing due to exogenous events. Such events often cause plans to fail, and understanding why they occur is sometimes necessary to achieve strong performance. Real-world agents do not necessarily observe such events directly, so they must reason that changes in the world are explained by events.

Consider the following real-world example of the power of explanation. In May 2005, NASA's Opportunity rover was crossing a dune on the surface of Mars when its human operators noticed an *inconsistency* with their expectations: Opportunity was not moving as much as expected [23]. The operators were not able to observe the surroundings of the rover fully and precisely, but they nevertheless *explained* this inconsistency by *assuming* that the rover was stuck in loose soil. This explanation enabled the operators to formulate a new plan to escape from the unobserved loose soil and continue the mission.

The focus of this work is on algorthmically explaining the history of a partially-observable, dynamic environment in order to understand prediction failures and thereby improve future predictions. To accomplish this, we have devised an algorithm that models change in terms of deterministic exogenous events that can be predicted and reasoned about. This reasoning about prediction failures contrasts with typical work on replanning approaches, which focuses on resolving failures between individual causal links in the plan, and does not attempt to understand the causes of that failure (e.g., [18, 10, 24, 1]). In the diagnosis field, work on constructing plan diagnoses addresses similar issues of constructing histories during execution in planning domains (e.g., [6], [4]) and discrete-event systems (e.g., [20]), but this work is based on a traditional representation of exogenous events as *actions* conducted by another agent or nature. Our work represents events as natural consequences that occur automatically, rather than by choice, which is not supported by current replanning or diagnosis systems. Finally, the SDR system (e.g., [19]) uses regression to generate causal explanations of an agent's history based on a contingent planning domain. In this work, different possible outcomes of actions are possible, which is modeled using conditional effects. While this work is similar in aim to ours, the representation understood by the SDR system is strictly less expressive than the representation used in our work.

Our contributions are the following:

- We describe a formalism for reasoning about the causes of inconsistencies between observations and expectations that arise during plan execution. This formalism includes a novel model for deterministic exogenous events and their effects on the world that models partial observability using a distinction between observable and hidden facts.
- We describe DISCOVERHISTORY, an algorithm that generates abductive explanations of possible histories during plan execution.
- We describe DHAgent, a simple planning and execution agent that uses DISCOVERHISTORY to maintain its description of the world and reason about hidden state. This allows DHAgent to (1) discard plans which would fail due to the occurrence of predicted events and (2) take advantage of opportunities afforded by future events.
- We discuss experiments in two planning domains, modified versions of the Rovers and Satellite domains from past International

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

Planning Competitions, to show that DHAgent's performance is improved by a statistically significant margin when using DISCOVERHISTORY for explanation. These experiments do not compare DISCOVERHISTORY with existing algorithms, which, as we describe later, do not support these domains.

2. DEFINITIONS AND NOTATION

In this section, we describe a new formalism for explanation generation. Representations and reasoning in this formalism use observations, actions, and events as the basic building blocks, so that an agent can conduct planning and explanation using a single domain model. This reduces the knowledge engineering burden by making an additional use of existing knowledge; it works because explanation is an inversion of planning.

Unlike typical formalisms from planning and diagnosis, ours models events as deterministic. These are somewhat similar to conditional effects on actions, since they occur deterministically following the execution of an action. However, deterministic events can be triggered by *any* action, or another event; modeling the same information as effects of actions would require the domain author to consider the effects of all series of events that could ever happen following an action's execution. Therefore, the size of an equivalent domain model using conditional effects rather than events would be exponential in the number of events to be represented.

The advantages of representing deterministic exogenous events are twofold: (1) we can determine (predictively or after the fact) the exact time when events must occur, reducing the set of potential explanations for a given series of observations and (2) interacting effects can combine without causing an explosion in the number of actions or events considered. Unpredictability in environments under our representation arises only from hidden facts, not a "choice" made by an environment as to whether an event will occur.

2.1 Basics

We use the standard definitions from classical planning for variable and constant symbols, logical predicates and atoms, literals, groundings of literals, propositions, planning operators and actions [3, Chapter 2].

Let $\mathcal{P}$ be the finite set of all propositions describing a planning environment; the *state* of the environment is described by assigning a value to each proposition in P. A planning environment is partially observable if an agent only has access to the environment through *observations* which do not cover the complete state. We let $\mathcal{P}_{obs}$ be the set of all propositions that the agent will observe when true. An observation associates a truth value with each of these propositions. Let $\mathcal{P}_{hidden}$ be a set of *hidden* propositions representing aspects of the world an agent cannot observe; for example, the exact location may be hidden to a robot with no GPS contact.

An *event template* is defined syntactically the same as a classical planning operator: $(\mathsf{name}, \mathsf{preconds}, \mathsf{effects})$, where name, the *name* of the event, $\mathsf{preconds}$ and $\mathsf{effects}$, the *preconditions* and *effects* of the event, are sets of literals. We use $\mathsf{effects}^-$ and $\mathsf{effects}^+$ to denote the negative and positive literals in $\mathsf{effects}$, respectively. An *event* is a ground instance of an event template. We assume that an event always occurs immediately when all of its preconditions are met in the world. After each action, any events triggered by that action occur, followed by events triggered by those events, etc. When no more events occur, the agent receives a new observation.

2.2 Explanations

We formalize the planning agent's knowledge about the changes in its environment as an *explanation* of the world. We define a finite set of symbols $\mathcal{T} = \{t_0, t_1, t_2, \ldots, t_n\}$, called *occurrence points*. An *ordering relation* between two occurrence points is denoted as $t_i \prec t_j$, where $t_i, t_j \in \mathcal{T}$.

There are three types of *occurrences*. An *observation occurrence* is a pair of the form (obs, t) where obs is an observation. An *action occurrence* is a pair of the form (a, t) where a is an action. Finally, an *event occurrence* is a pair (e, t) where e is an event. In all of the occurrence forms, t is an occurrence point. Given an occurrence o, we define occ as a function such that $\mathsf{occ}(o) \mapsto t$; that is, occ refers to the occurrence point t of any observation, action, or event.

An *execution history* is a finite sequence of observations and actions $\mathsf{obs}_0, a_1, \mathsf{obs}_1, a_2, \ldots, a_k, \mathsf{obs}_{k+1}$. A planning agent's *explanation* of the world given an execution history is a tuple $\chi = (C, R)$ such that C is a finite set of occurrences that includes each obs_i for $i = 0, \ldots, k-1$ and each action a_j for $j = 1, \ldots, k$ for some number k. C may also include zero or more event occurrences that happened according to that explanation. R is a partial ordering over a subset of C, described by ordering relations $\mathsf{occ}(o_i) \prec \mathsf{occ}(o_j)$ such that $o_i, o_j \in C$. As a shorthand, we sometimes will say $o_i \prec o_j$ if and only if $\mathsf{occ}(o_i) \prec \mathsf{occ}(o_j)$.

We use the definitions $\mathsf{knownbefore}(p, o)$ and $\mathsf{knownafter}(p, o)$ to refer to the value of a proposition p before or after an occurrence $o \in C$ occurs. Let o be an action or event occurrence. Then, the relation $\mathsf{knownbefore}(p, o)$ is true iff $p \in \mathsf{preconds}(o)$. Similarly, the relation $\mathsf{knownafter}(p, o)$ is true iff $p \in \mathsf{effects}(o)$. If o is an observation occurrence and $p \in \mathsf{obs}$, then both $\mathsf{knownbefore}(p, o)$ and $\mathsf{knownafter}(p, o)$ are true, and otherwise are false.

We say that an occurrence o is *relevant* to a proposition p if the following holds:

$$\mathsf{relevant}(p, o) \equiv \mathsf{knownafter}(p, o) \vee \mathsf{knownafter}(\neg p, o) \vee \mathsf{knownbefore}(p, o) \vee \mathsf{knownbefore}(\neg p, o).$$

We use the predicates $\mathsf{prior}(o, p)$ and $\mathsf{next}(o, p)$ to refer to the prior and next occurrence relevant to a proposition p. That is to say, $\mathsf{prior}(o, p) = \{o' \mid \mathsf{relevant}(p, o') \wedge \neg\exists o'' s.t. \mathsf{relevant}(p, o'') \wedge o' \prec o'' \prec o\}$. Similarly, $\mathsf{next}(o, p) = \{o' \mid \mathsf{relevant}(p, o') \wedge \neg\exists o'' s.t. \mathsf{relevant}(p, o'') \wedge o \prec o'' \prec o'\}$.

2.3 Plausibility

The *proximate cause* of an event occurrence (e, t) is an occurrence o that satisfies the following three conditions with respect to some proposition p: (1) $p \in \mathsf{preconds}(e)$, (2) $\mathsf{knownafter}(p, o)$, and (3) there is no other occurrence o' such that $o \prec o' \prec (e, t)$. Every event occurrence (e, t), must have at least one proximate cause, so by condition 3, every event occurrence must occur immediately after its preconditions are satisfied.

An *inconsistency* is a tuple (p, o, o') where o and o' are two occurrences in χ such that $\mathsf{knownafter}(\neg p, o)$, $\mathsf{knownbefore}(p, o')$, and there is no other occurrence o'' such that $o \prec o'' \prec o' \in R$ and p is relevant to o''.

An explanation $\chi = (C, R)$ is *plausible* if and only if the following holds:

1. There are no inconsistencies in χ;
2. Every event occurrence $(e, t) \in \chi$ has a proximate cause in χ;
3. For every pair of simultaneous occurrences such that $o, o' \in C$ and $occ(o) = occ(o')$, there may be no conflicts before or after: for all p, $\mathsf{knownafter}(p, o) \implies \neg\mathsf{knownafter}(\neg p, o')$, and $\mathsf{knownbefore}(p, o) \implies \neg\mathsf{knownbefore}(\neg p, o')$.
4. If $\mathsf{preconds}(e)$ of an event e are all satisfied at an occurrence point t, e is in χ at t;

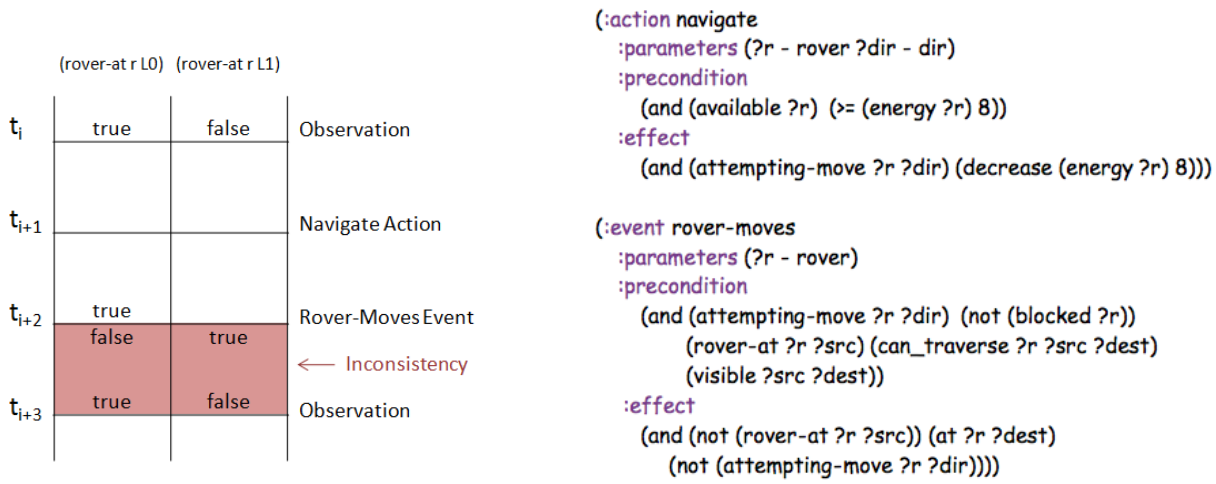

Figure 1: Example of an inconsistent explanation, with occurrence points ordered on the left hand side. Relevant action and event descriptions are given on the right. Boolean values are the knownbefore **and** knownafter **relations; for example, the value "false" at top right indicates that the relation** knownbefore((¬ **(rover-at r L1)),** o_i**) holds.**

Algorithm 1: A high-level description of DISCOVERHISTORY.

```
1  Procedure DiscoverHistory (χ)
2  begin
3      if Inconsistencies(χ) = ∅ then
4          χ ← FindExtraEvents (χ)
5          if χ = ∅ then return ∅
6          if Inconsistencies(χ) = ∅ then return {χ}
7      arbitrarily select an i ∈ Inconsistencies(χ)
8      X ← Refine (χ, i)
9      foreach χ_new ∈ X do
10         X ← X ∪ DiscoverHistory (χ_new)
11     return X
```

EXAMPLE 1. *Suppose that a rover r attempts to move after its wheel has, unobserved, become stuck. Figure 1 illustrates part of an inconsistent explanation, which includes the prior observation occurrence at* t_i*, in which the rover is observed at location* L0*; followed by a navigate action occurrence, illustrating the rover's attempt to move, at* t_{i+1} *directing the rover to location* L1*; followed by an event occurrence illustrating the predicted event* o_{i+2} *at* t_{i+2} *that changes the rover's location from* L0 *to* L1*; followed by the most recent observation occurrence* o_{i+3} *at* t_{i+3}*, which states that the rover is at* L0*.*

There are two inconsistencies in this explanation, between the event occurrence and observation occurrence: ⟨(rover-at r L0), o_{i+2}, o_{i+3}⟩ *and* ⟨¬(rover-at r L1), o_{i+2}, o_{i+3}⟩.

3. GENERATING ABDUCTIVE EXPLANATIONS

This section describes DISCOVERHISTORY, our search algorithm for generating plausible explanations by recursively applying refinements to implausible explanations. DISCOVERHISTORY is designed to find possible histories of a partially-observable dynamic environment. DISCOVERHISTORY generates successive explanations by attempting to resolve inconsistencies in the current explanation given its observations.

Algorithm 1 shows a high-level description of DISCOVERHISTORY. The base case occurs when the current explanation is plausible. In the recursive case, REFINE chooses an inconsistency and generates a set of explanations that resolve it. Each such explanation is then recursively considered by DISCOVERHISTORY to remove any remaining inconsistencies.

Let χ be a planner's current explanation of the world, which includes obs, the most recent observation received by the agent. DISCOVERHISTORY starts by finding all inconsistencies in the current explanation χ by calling Inconsistencies(χ). If none are present, then the first condition for plausibility is met, so the other conditions are checked to ensure plausibility (see Section 2.3). Plausibility condition 3 is assumed to be true of the initial explanation, and are maintained throughout the resolution process by removing events that contradict any new additions. Once no inconsistencies exist, therefore, FINDEXTRAEVENTS checks plausibility conditions 2 and 4. Condition 2 requires that any explanation containing events with no proximate cause be rejected; condition 4 requires that any new events caused by the changes to χ be added to χ as well. These new events may cause new inconsistencies. Otherwise, all four conditions for a plausible explanation are met and DISCOVERHISTORY returns the explanation χ in Line 6.

In Line 7, DISCOVERHISTORY selects an inconsistency i from Inconsistencies(χ) and attempts to resolve it.[1] The REFINE subroutine (Line 8 of Algorithm 1) finds all explanations that result from resolving that inconsistency and recursively calls itself on each. We further describe REFINE below.

We refer to the initial explanation constructed by the agent as χ_0. We use this explanation as the basis for our definition of goodness. An explanation χ_a is *better than* χ_b if DISCOVERHISTORY requires fewer changes to transform χ_0 into χ_a than χ_b. We use an iterative deepening search to find all explanations at a minimum depth. Since each invocation of REFINE adds a single change to its base explanation, these explanations are the *best* explanations by our goodness definition. To prevent searches from continuing indefinitely, we employ a maximum depth bound (not shown in pseudocode). The search conducted by DISCOVERHISTORY has a

[1]Any inconsistency may be selected; it does not affect the correctness of the algorithm. It may affect the algorithm's efficiency, but we do not discuss this topic further.

Algorithm 2: Subroutines REFINE and REMOVEOCC

```
1  Procedure REFINE (χ = (C, R), i =(p,o,o'))
2  begin
3      X ← ∅
       /* Adding a new event occurrence */
4      foreach (t, t') ∈ T × T : t' ⊀ t do
5          if occ(o) ≺ t ≺ occ(o') ∧ occ(o) ≺ t' ≺ occ(o')
           then
               /* (t, t') is a possible interval in which the */
               /* occurrence o could happen */
6              foreach e ∈ E : effects(e) ⊨ p do
7                  new symbol t''
8                  o'' ← (e, t'')          // New event occurrence
9                  χ_new ← (C + o'', R + t ≺ t'' ≺ t')
10                 X ← X + χ_new      // New explanation added
       /* Removing an occurrence */
11     if o ∉ χ0 ∧ o is an event occurrence then
12         X ← X ∪ REMOVEOCC (χ, o)
13     if o' ∉ χ0 ∧ o' is an event occurrence then
14         X ← X ∪ REMOVEOCC (χ, o')
       /* Hypothesizing an initial value */
15     if p ∈ P_hidden ∧ o = occ0 then
16         X ← X + (C + (e_p, t0), R)          // New initial
17     return X                                // value occurrence
18
19 Procedure REMOVEOCC (χ = (C, R), o)
20 begin
21     X ← ∅
22     foreach p ∈ preconds(o) do
23         X ← X + (C \ o + (e_r, occ(o)), R)  // New removal
24     return X                                // occurrence
```

maximum depth of $|C| * (2^{|E|})^2$ and a maximum branching factor of $|E|$, where E is the finite set of all possible event occurrences.

As shown in algorithm 2, there are three possible ways for REFINE to resolve an inconsistency in an explanation, each of which has different conditions for applicability. They are: (1) adding a new occurrence, (2) removing an occurrence, and (3) hypothesizing a different initial value for some proposition. All applicable methods must be tried, resulting in multiple explanations. Each resolution may create or resolve other inconsistencies, so the inconsistencies found in refined explanations are not necessarily a subset of those found in the parent. To save space, we omit the pseudo-code for REFINE. We detail the resolution strategies below.

3.1 The REFINE Subroutine

3.1.1 Adding a New Event Occurrence

Let $\chi = (C, R)$ be an explanation with an inconsistency $i = (p, o, o')$. One way to resolve an inconsistency is to show that some occurrence changed the value of a literal in between the preceding occurrence o and the following occurrence o'. This occurrence must be an event o'' relevant to p such that $o \prec o'' \prec o'$.

To find such occurrences REFINE considers every possible consecutive ordered pair of occurrence points (t, t') between $occ(o)$ and $occ(o')$, given the partial-ordering R (lines 3-4). For each such pair (t, t') and every e such that $\mathsf{effects}(e) \models p$ (line 5), REFINE creates a new occurrence point t'' (line 5) and a new event occurrence $o'' = (e, t'')$. Then the algorithm adds o'' into C and updates the partial ordering R with $t \prec t'' \prec t'$. This results in one new explanation that does not contain the inconsistency i for each event

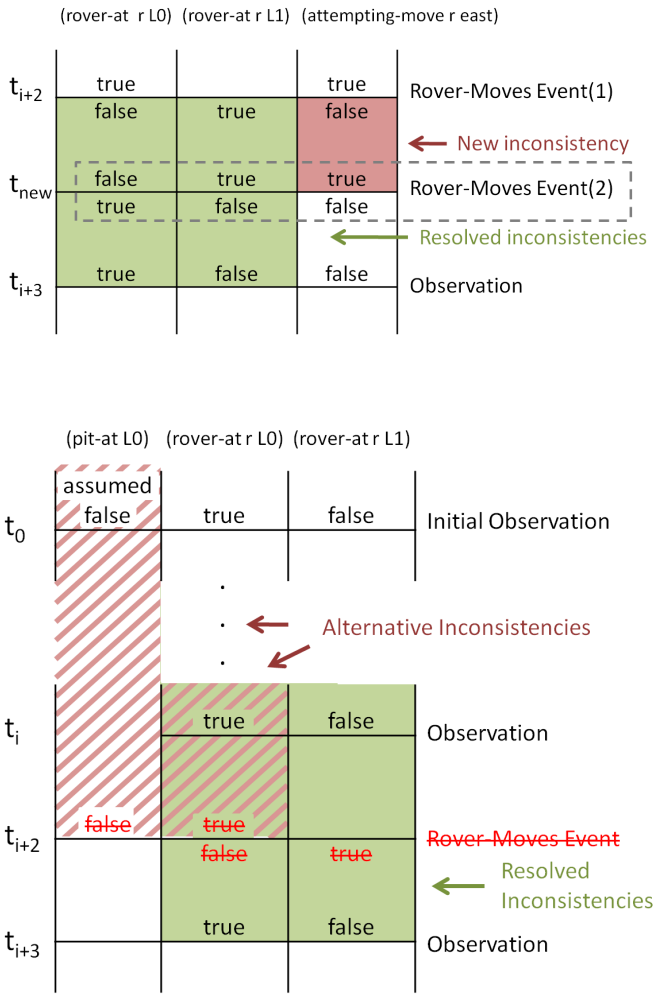

Figure 2: (a) Example of adding an occurrence (left). (b) Example of removing an occurrence (right).

$e : \mathsf{effects}(e) \models p$.

EXAMPLE 2. *Continuing Example 1, the event Rover-Moves causes the rover to enter a different location, so it could be added to resolve the inconsistency* (¬ (rover-at r L1), o_{i+2}, o_{i+3}). *In order for this to work, a new occurrence must be added between t_{i+2} and t_{i+3} (see Figure 2(a)).* REFINE *creates a new explanation with added occurrence $o_{new} = (e, t_{new})$ and new ordering relations $t_{i+2} \prec t_{new} \prec t_{i+3}$. A new inconsistency is also generated (see Figure 2(a)). The new inconsistency occurs because another precondition of the Rover-Moves event is the literal* (attempting-move r east)*, which is false after such an event occurs. The new inconsistency, (*(attempting-move r east)*, o_{i+2}, o_{new}), will need to be eliminated in a recursive call to* DISCOVERHISTORY.

3.1.2 Removing an occurrence

Another possible way to resolve (p, o, o'), where o and/or o' is an event, is to refine the current explanation to generate new explanations in which either o or o' is removed.

REFINE checks both o and o' for presence in χ_0; if either was, then removing it would cause a cycle. Otherwise, if it an event occurrence, each is eligible for removal. REMOVEOCC

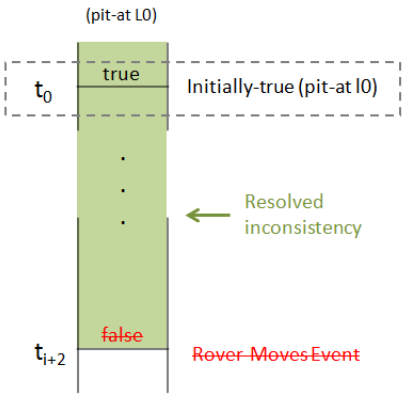

Figure 3: Example of hypothesizing an initial value.

first creates a new set of occurrences, C', by removing o from C. Then, in order to explain why the occurrence does not happen, one of the preconds(e) must be found not to hold at occurrence point t. Therefore, REMOVEOCC creates a new explanation for each precondition p' of e by creating a *removal occurrence* $o_r = (e_r, occ(o))$. The occurrence o_r occurs at the same time, and instead of o; and the new event e_r has no effects and satisfies preconds$(e_r) = \{\neg p'\}$. Due to the presence of the removal event, no new occurrence can be added to the resulting explanation which would cause o to recur.

EXAMPLE 3. *The above mechanism resolves the inconsistency found in example 2 by removing o_{i+2}. Note that no inconsistencies are generated by removing it (recall from example 1 that the rover really didn't move). Finally, the preconditions are examined to look for a precondition which could explain why o_{i+2} might not have occurred. Two preconditions on o_{i+2} are shown in Figure 2(b):* (rover-at r L0) *and* ¬ (pit-at L0). *A new explanation is created for each, with a removal occurrence corresponding to the negated precondition. Each dummy occurrence will cause a new inconsistency, as shown in Figure 2(b).*

3.1.3 Hypothesizing an initial value

Given an inconsistency (p, o, o'), where o refers to the initial observation in the execution history, and p is not observable, a different initial occurrence o may be hypothesized. When this is the case, REFINE generates a new explanation by adding to χ an event occurrence $o_p = (e_p, t_0)$. The event e_p in this occurrence is the reserved initially-true event e_p, which has no preconditions or negative effects, and satisfies effects$^+(e_p) = \{p\}$. This operation has no side effects to any other literal, and thus will never cause new inconsistencies.

EXAMPLE 4. *One of the alternate inconsistencies found in example 3,* (¬ (pit-at L0), occ_0, occ_{i+2})*, has the characteristics required for hypothesizing an initial occurrence. A* pit-at *literal is not observable, and the prior occurrence of the inconsistency is occ_0. Therefore, the discrepancy can be resolved by adding the event occurrence $o_{(pit-atL0)}$, as shown in Figure 3.*

3.2 The FINDEXTRAEVENTS Subroutine

When an explanation no longer has any inconsistencies, it must still be checked for missing events and missing causes. This is due to requirements 2 and 4 of a plausible explanation (see Section 2.3). Requirements 2 and 4 implement the notion of deterministic event execution, by requiring that all events fire immediately when their conditions are met. We now discuss these requirements in more detail.

Algorithm 3: The FINDEXTRAEVENTS Subroutine

```
1 Procedure FindExtraEvents (χ = (C, R))
2 begin
3     foreach occ_i ∈ C do
4         occ_j ← FindProximateCause(C)
5         if occ_j = ∅ then return ∅
6     foreach t_i ∈ T do
7         C ← C ∪ EnumerateCausedEvents(χ)
8     return χ
```

According to requirement 2, an explanation that includes an event for which no proximate cause is implausible (because the event should have happened earlier). To ensure that such an explanation is not returned, FINDEXTRAEVENTS iterates over each event occurrence occ_i in χ, and attempts to find its proximate cause. This is shown in Algorithm 3, lines 3-5.To do so, it iterates over all occurrences in the explanation and execution history to find the set $PO = [\text{occ}_0 \ldots \text{occ}_n]$ of occurrences where for every $\text{occ}_j \in PO$, $\text{occ}_j \prec \text{occ}_i$ and there is no occ_k such that $\text{occ}_j \prec \text{occ}_k \prec \text{occ}_i$. If for any $\text{occ}_j \in PO$ the set $post(\text{occ}_j) \cap pre(\text{occ}_i)$ is non-empty, occ_i has a proximate cause. If there is an occ_i that has no proximate cause in the explanation, then the explanation is faulty and a null explanation is returned.

According to requirement 4, all events that are possible must occur. FINDEXTRAEVENTS guarantees this, as shown in Algorithm 3, lines 6-7, by adding events that are not in χ but do have causes in χ. This is done by considering each occurrence point $t_i \in \mathcal{T}$, and enumerating all events e_j whose preconditions are met at time t_i. If an occurrence $occ_i j = (e_j, t_i)$ is not already present in the explanation χ, that occurrence is added to the explanation.

4. DHAGENT

DHAgent is a software agent that operates in planning domains using a PDDL+ [2] domain definition to represent actions, events, and predicates. It conducts planning using the SHOP2 PDDL+ planner [13] and automatically maintains a consistent explanation of the world, which it modifies using the DISCOVERHISTORY algorithm as necessary.

Algorithm 4 shows a high-level description of DHAgent. It takes as input a set of top-level tasks to be performed. It then receives the initial observation, which gives the truth value for the initial state for the observable literals $\mathcal{P}_{obs}$. Because some literals are not observable, many possible worlds may be consistent with this observation. DHAgent creates an initial state according to the closed world assumption, and creates a plan to accomplish the top-level task from this initial state. Because of the closed world assumption, it plans only for the possible world where all unobservable facts are false. DHAgent then loops over the actions in the plan, executing them one at a time, adding projected events to its explanation, and receiving new observations.

When a new observation is inconsistent with its current explanation, DHAgent refines its explanation of the past using DISCOVERHISTORY, as described in Section 3. Although it may sometimes choose an incorrect explanation, it can retract prior assumptions and explained occurrences during subsequent execution steps if they become untenable. Every time the explanation is refined in

Algorithm 4: DHAGENT

```
   input: A set T of tasks to perform
 1 Procedure DHAGENT(T)
 2 begin
 3     obs_0 ←RECEIVEOBSERVATION()
 4     π ←PLAN(T, preconds(obs_0))
 5     χ ← ({obs_0}, ∅)
 6     i ← 1
 7     while π ≠ ∅ do
 8         a_i ←POP(π)
 9         χ ← χ + a_i
10         χ ←FINDEXTRAEVENTS(χ)
11         EXECUTEACTION(a_i)
12         obs_i ←RECEIVEOBSERVATION()
13         χ ← χ + obs_i
14         if Inconsistencies(χ) ≠ ∅ then
15             X_new ←DISCOVERHISTORY (χ)
16             if X_new ≠ ∅ then χ ←FIRST(X_new)
17             else χ ← ({obs_i}, ∅)
18             Create a set S of all literals satisfied at occ(obs_i)
               according to χ
19             p ←PLAN(T, S)
20         i ← i + 1
```

this way, DHAgent replans, because its current plan may no longer accomplish its task. As part of the explanation refinement process, DHAgent considers the initial world to have been a different possible world consistent with the initial observation. However, it still makes a closed world assumption about the initial state with regard to all literals not hypothesized to have different initial values in the explanation; thus it considers only one possible world at any given time. When no explanations are found, the explanation cannot be maintained further, because there will always be inconsistencies in any explanation based on the current one. Therefore, a new explanation is started as if the most recent observation was the initial observation.

When replanning, DHAgent takes into account the state of the unobservable literals found by projecting the effects of explanation events on the current assumed initial world state. The loop exits only when all actions from a plan have been performed. This occurs either when each top-level task has either been accomplished or has become unachievable.

The way DHAgent makes assumptions about the initial possible world is consistent with the way belief revision is performed in many BDI agents. In the language of belief revision, DHAgent initially believes that all facts in the initial observation are true, and all others are false. Over time it changes its beliefs according to the actions executed and the events in the explanation. Thus DHAgent always believes the state to be the projection of some individual initial world state to the present.

We believe that DHAgent's policy of making strong assumptions about the initial state until observations contradict them is reasonable in domains with many possible worlds and automatic sensing. In such domains, the high number of hidden facts causes planning for all possible worlds to be intractable. Furthermore, DHAgent will be relatively successful in domains where certain literals are rarely true, and cannot be observed directly; the existence of such a true literal would be surprising. For example, although sand pits may be ubiquitous on Mars, a Mars rover could not reason about all possible such pits. Indeed, by assuming sand pits are everywhere it might remain stuck in one place, never moving for fear of falling in. Instead, it initially assumes that the ground is safe and be prepared to revise its assumptions when new evidence arrives. In the same way, DHAgent is an agent that is prepared for surprises, but does not consider every possibility that might occur.

5. EXPERIMENTAL EVALUATION

We examined the performance of DHAgent in the context of planning and execution in two partially-observable domains. These domains have been engineered to include events that are triggered by hidden facts. Thus, events will occur at execution time that can not be predicted due to lack of knowledge at planning time.

Rovers-With-Compass (RWC) is a navigation domain with hidden obstacles inspired by the difficulties encountered by the Mars Rovers. Specifically, individual locations may be windy, sandy, and/or contain sand pits, which the rover cannot observe directly. Sandy locations cause the rover to be covered in sand; while covered in sand, the rover cannot observe its location or perform the "recharge" action. Sand pits stop the rover from moving; the rover can dig itself out at a high energy cost. Windy locations clear the sand off of the rover, but due to a malfunction, may confuse the rover's compass, causing it to move in the wrong direction. When rovers run out of energy, they stop moving.

Satellite-With-Malfunctions (SWM) is based on the Satellites domain from the 2002 International Planning Competition. The objective in each scenario is to acquire images of various phenomena and transmit them to earth. Our additions to this domain include various causes of satellite malfunction: supernova explosions, which can damage sensitive instruments that are pointed toward them; fuel leaks, which cause fuel reserves to diminish rapidly; and motor malfunctions, which delay a satellite's turn to a new perspective. When fuel reserves are depleted, no further goals can be accomplished.

We compared the performance of DHAgent with an ablated version that does not perform explanation, called Non-DHAgent. Non-DHAgent differs in how it replans; replanning occurs whenever the planner incorrectly predicts the new observation. Non-DHAgent then plans based on a state consisting of the latest observation, plus those hidden facts consistent with the events predicted so far by SHOP2. Other systems capable of reasoning about partially observable worlds such as Contingent-FF [5] and SDR[19] provide no support for deterministic exogenous events, or domains where no plan or conditional plan is guaranteed success. To our knowledge, no existing systems can operate in these domains, so we do not compare with existing work.

We wrote a problem generator for each domain that randomly creates an initial state including both observable and hidden facts and goals. For the RWC domain, each starting state contained 3 rovers, and a goal for each rover that required it to move to a new destination. Goal destinations were generated randomly such that the rover must cross at least 3 distinct locations to accomplish its goal. Each scenario took place on a 6×6 grid of locations connected in the four compass directions. Hidden state was assigned independently for each location and condition with probability p.

Each randomly-generated SWM problem included 3 satellites and required the attainment of 8 image acquisition goals. Each image target was chosen randomly from a set of 20. Targets were associated with supernovae, fuel leaks, and motor malfunctions based on a probability p.

We randomly generated 25 problems in each domain; Table 1 shows a comparison of the performance of DHAgent and Non-DHAgent. Here, performance is defined as a percentage of goals completed. For the RWC domain, the probability of hidden difficulties was $p = 0.1$; in the SWM domain, the probability of

Table 1: Statistical t-test results, comparing our explanation-based and non-explanation based systems in the RWC and SWM domains.

Domain	Non-DHAgent	DHAgent	t-test
Rovers	65.3%	78.7%	0.001
Satellite	52.5%	76.0%	< 0.001

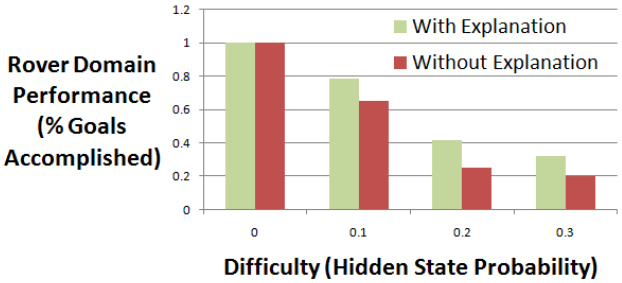

Figure 4: Comparison at various difficulty levels.

hidden state that induces malfunctions was $p = 0.3$. We used a depth bound of 7 in our experiments, i.e., the search for explanations could not include more than 7 recursive refinements.

We compared the mean performances of DHAgent and Non-DHAgent using a two-tailed t-test with paired samples, which showed that DHAgent statistically outperformed Non-DHAgent in both domains. As the only difference between the two agents is the use of explanation, it's clear that the use of DISCOVERHISTORY improved performance. This shows that abductive explanation of state events can improve performance over replanning alone in partially-observable dynamic environments.

To further examine the impact of hidden state on performance, we increased the difficulty of the RWC domain by varying the probability of hidden states. Figure 4 compares the performance of the two agents at 4 difficulty levels: $p = 0.0, 0.1, 0.2$, and 0.3. At $p = 0.0$, there is no hidden state; as expected, we see perfect performance from both agents since no explanation is necessary. As the probability of obstacles rises, both agents perform more poorly, but DHAgent continued to statistically outperform Non-DHAgent. At $p = 0.3$, DHAgent accomplished goals 50% more often than Non-DHAgent. Differences between them were statistically significant for all $p > 0.0$.

6. RELATED WORK

As with many topics in Artificial Intelligence, our work is related to research conducted in several subfields. We've tried to separate the related work by body of literature below, starting with the most closely related first.

Planning and Execution. Other work in planning and execution that involves reconsidering what happened in the past includes that by Molineaux, Klenk, and Aha [12] and Shani and Brafman [19]. Our work extends the work from [12] with a more principled formalism for exogenous events and the capability to reason over a longer history. In more recent work by Shani and Brafman [19], the SDR planner maintains beliefs by reconsidering facts from the past and initial state through a regression process. Unlike our DISCOVERHISTORY algorithm, a series of events that explain the observations is not constructed. Instead, SDR tries to determine what possible facts are consistent by searching paths through a branching and/or tree that covers possible histories tracing through executed sensing actions. SDR considers multiple possible initial states simultaneously to find a plan that works under a sample of possible worlds. The plan will not necessarily execute in the true world, however. When an action's preconditions are not known to be true, the agent replans. In contrast, DHAgent replans when any inconsistency is discovered. As a result of this, SDR will execute some number of actions in an incorrect plan before coming to one that is no longer possible. In contrast, DHAgent never executes actions toward a plan that is inconsistent with its knowledge of the world.

CASPER [10] uses a continuous planning approach to achieve a higher level of responsiveness in dynamic planning situations. The planner has goals, initial state, current state, and a model of the expected future state. The goals or current state may change at any time and invoke the planning process. The planner will then create a new plan based on the current information. This can happen repeatedly and the planner stands ready to continually modify the plan.

Continuous Planning and Execution Framework (CPEF) is introduced in [18]. CPEF assumes that plans are dynamic, that is, that they must be evolving in response to the changes in the environment. Over the years, there has been a large body of research on replanning and plan repair during execution in dynamic environments. These works focus only on execution-time failures as discrepancies; that is, an unexpected state observed during execution triggers a re-planning or plan repair process [24, 9, 1, 22, 21, 18, 17].

In all of the previous works mentioned above, a discrepancy is defined as based on causal links between the preconditions and effects of different actions in the plan. In particular, when the observed state of the world violates such causal-links in the plan, a discrepancy occurs and triggers the re-planning or plan-repair process. In contrast, our formalism and algorithms are designed to generate more expressive and informative explanations of the world, including causal-link failures as well as other changes that may occur due to exogenous events.

Real-Time Control and Planning. Existing research on *real-time control and execution* in Artificial Intelligence typically employs a reactive planning foundation, where the agent decides on an action and executes it immediately [14, 15, 16, **?**, 8]. Sometimes, the systems decide on the action to be executed by using planning heuristics. Sometimes, they generate a complete plan, off-line, that achieves the goal, and then execute the plan.

The Cooperative Intelligent Real-time Control Architecture (CIRCA) is an autonomous planning and control system that builds and executes safety-preserving plans in the face of unpredictable events [14]. CIRCA includes a Reaction Planner which devises a plan to accomplish mission goals while avoiding or preventing failures. The Reaction Planner takes in a problem description that species the initial state of the world, a set of goal states that the planner attempts to reach, a distinguished failure state that the planner must avoid, and a set of transitions that move the world from one state to another. Unlike most planning systems, CIRCA reasons about uncontrollable sources of changes, such as environmental disturbances, failures, and adversaries. The transition models also include timing characteristics that specify how fast the various transitions can occur. The Reaction Planner uses formal verification techniques to check its plans and ensure that failure is not reachable.

Diagnosis. In the diagnosis literature, some work (e.g., [11], [20], [7]) has focused on finding action histories that resolve contradictions by assuming the presence of faulty actions and/or missing assumptions about the initial state. This work differs from ours in

that it does not take place in the context of an execution framework, nor does it consider a model of deterministic exogenous events, which requires both simultaneous event occurrence and the elimination of explanations in which caused events fail to occur. Other work in diagnosis (e.g. [6], [4]) has supplemented the diagnosis capability with an execution component; however, when replanning in an execution context, re-explanation may also be necessary, and this earlier work does not support amending earlier explanations by removing events (or actions) previously believed to have happened. Finally, none of these authors has considered the capability of explaining directly based on common planning models as our system does, to reduce the effort required of experts in creating multiple domain models.

7. CONCLUSIONS AND FUTURE WORK

We have described a formalism and algorithm to abductively reason about unexpected event occurrences during planning and execution. The explanations generated using this approach can be used by a planning and execution system to proactively expand its knowledge of the exogenous events and hidden state in the world, and thereby improve the performance of its re-planning process. Our experiments in two planning domains showed that the percentage of goals achieved was significantly higher when using our abductive explanation generation algorithm, compared to an identical system that did not use them. We have shown that this algorithm can improve performance in environments with repeated exposure to hidden events and discrepancies.

There are several tasks that we would like to accomplish in the future based on our results. First, we'll investigate the theoretical properties of abductive reasoning in planning and execution in general. Based on this theory, we intend to generalize our work to investigate explanatory diagnosis in planning with incomplete action and event models, and in temporal planning, where actions and events may have durative effects.

Acknowledgments. This work was supported, in part, by DARPA and the U.S. Army Research Laboratory under contract W911NF-11-C-0037. The views expressed are those of the authors and do not reflect the official policy or position of the funders.

8. REFERENCES

[1] F. Ayan, U. Kuter, F. Yaman, and R. P. Goldman. HOTRiDE: Hierarchical Ordered Task Replanning in Dynamic Environments. In F. Ingrand and K. Rajan, editors, *ICAPS-07 Workshop on Planning and Plan Execution for Real-World Systems*, pages 31–36, 2007.

[2] M. Fox and D. Long. PDDL+: Modelling continuous time-dependent effects. In *Proc. 3rd International NASA Workshop on Planning and Scheduling for Space*, 2002.

[3] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, May 2004.

[4] S. Gspandl, I. Pill, M. Reip, G. Steinbauer, and A. Ferrein. Belief management for high-level robot programs. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[5] J. Hoffmann and R. Brafman. Contingent planning via heuristic forward search with implicit belief states. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, pages 71–80, 2005.

[6] G. Iwan. History-based diagnosis templates in the framework of the situation calculus. *KI 2001: Advances in Artificial Intelligence*, pages 244–259, 2001.

[7] G. Iwan and G. Lakemeyer. What observations really tell us. *KI 2003: Advances in Artificial Intelligence*, pages 194–208, 2003.

[8] F. Kabanza, M. Barbeau, and R. St-Denis. Planning control rules for reactive agents. *Artificial Intelligence*, 95(1):67–113, 1997.

[9] S. Kambhampati and J. A. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55:193–258, 1992.

[10] R. Knight, G. Rabideau, S. Chien, B. Engelhardt, and R. Sherwood. Casper: Space exploration through continuous planning. *IEEE Intelligent System*, pages 70–75, September/October 2001.

[11] S. McIlraith. Explanatory diagnosis: Conjecturing actions to explain observations. In *Proceedings of the Int'l Conference on Principles of Knowledge Representation and Reasoning*, pages 167–179. Morgan Kaufmann Publishers, 1998.

[12] M. Molineaux, M. Klenk, and D. Aha. Goal-driven autonomy in a Navy strategy simulation. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 1548–1554, 2010.

[13] M. Molineaux, M. Klenk, and D. Aha. Planning in dynamic environments: Extending htns with nonlinear continuous effects. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[14] D. Musliner, E. Durfee, and K. Shin. CIRCA: a cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6):1561–1574, 1993.

[15] D. J. Musliner and R. P. Goldman. CIRCA and the Cassini Saturn orbit insertion: Solving a prepositioning problem. In *Working Notes of the NASA Workshop on Planning and Scheduling for Space*, 1997.

[16] D. J. Musliner, M. J. S. Pelican, R. P. Goldman, K. D. Krebsbach, and E. H. Durfee. The evolution of CIRCA, a theory-based AI architecture with real-time performance guarantees, 2008.

[17] K. L. Myers. Advisable planning systems. In A. Tate, editor, *Advanced Planning Technology*. AAAI Press, 1996.

[18] K. L. Myers. A continuous planning and execution framework. *AI Magazine*, 20(4):63, 1999.

[19] G. Shani and R. Brafman. Replanning in domains with partial information and sensing actions. In *Twenty-Second Int'l Joint Conference on Artificial Intelligence*, 2011.

[20] S. Sohrabi, J. Baier, and S. McIlraith. Diagnosis as planning revisited. *Proceedings of the International Conference on the Principles of Knowledge Representation and Reasoning*, pages 26–36, 2010.

[21] X. Wang and S. Chien. Replanning using hierarchical task network and operator-based planning. *Recent Advances in AI Planning*, pages 427–439, 1997.

[22] I. Warfield, C. Hogg, S. Lee-Urban, and H. Munoz-Avila. Adaptation of hierarchical task network plans. In *Proceedings of the Twentieth International FLAIRS Conference (FLAIRS-07)*, 2007.

[23] G. Webster. Nasa's rovers continue martian missions. `http://marsrover.nasa.gov/newsroom/pressreleases/20050524a.html`, May 2005.

[24] S. Yoon, A. Fern, and R. Givan. FF-Replan: A baseline for probabilistic planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 352–359, 2007.