

What Just Happened? Explaining the Past in Planning and Execution

Matthew Molineaux¹, Ugur Kuter^{2,*}, and Matthew Klenk^{3,**}

¹Knexus Research Corporation; 9120 Beachway Lane; Springfield, VA 22153; USA

²Smart Information Flow Technologies, 211 North 1st Street Minneapolis, MN 55401, USA

³Naval Research Laboratory, Code 5514; Washington DC; USA

¹matthew.molineaux@kexusresearch.com

²ukuter@sift.net

³Matthew.Klenk@parc.com

Abstract. We consider the problem of automated planning in partially-observable dynamic environments, where exogenous events that cannot be directly observed affect the state of the world. In these environments, a planner’s knowledge of the world is limited, and state transitions can be both ambiguous and difficult to predict due to that lack of knowledge. We describe a new formalism and new algorithms that enable a planner to proactively expand its knowledge of the environment during planning and execution, by modeling the exogenous events that can occur and forming explanations that reveal information about the world. We have implemented our new algorithms in a variant of the well-known SHOP2 planner that can replan when a failure occurs during plan execution. We have conducted an ablation study in two planning domains to examine the effects of explanation on execution. The results demonstrate that our algorithm successfully increases the performance of an agent using it in two planning domains. This improvement results from the agent having increased knowledge of the environment, which allows it to more accurately predict future events and ultimately make better plans.

Keywords: planning and execution, abductive explanation, autonomous agents

1 Introduction

In real-world environments, unobserved events underlie causal link failures, and understanding these events can be necessary for strong performance. For example, during its mission in 2009, NASA’s Spirit rover was exploring the region known as “Home Plate” on the surface of Mars when its human operators noticed an *inconsistency* with their expectations: Spirit was not moving as much as expected [6]. 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.

* Author completed this work while at University of Maryland, Institute for Advanced Computer Studies, College Park, MD 20742; USA.

** Now at Palo Alto Research Center (PARC).

Existing research on planning and execution has typically focused on replanning when individual causal links in the plan fail, without reasoning explicitly about the causes of that failure [10]. Work in planning under uncertainty avoids the issue of failure causes by considering them as part of the uncertainty model of the planner, and does not reason explicitly about them [2].

Determining the causes of failures during planning and execution through abductive explanation is the focus of our paper. We describe an algorithm that reasons about a partially-observable, dynamic environment, and generates explanations that describe the cause of inconsistencies between expectations and outcomes. When those causes become known, a planning agent can use this causal knowledge in replanning to better predict future states. In particular:

- We describe a formalism for reasoning about the causes of unexpected observations during planning and execution. This formalism includes a model for exogenous events and their effects on the world.
- We describe an algorithm that generates abductive explanations for what might be happening in the world during planning and execution. This algorithm is designed to improve the performance of a planning agent by increasing its knowledge of hidden state. When replanning, a planning agent will therefore (1) discard plans which would be invalidated by events caused by future event and (2) take advantage of opportunities afforded by future events.
- We have implemented two planning and execution agents: one uses our explanation-generation algorithms and the other does not. Experiments in two planning domains, modified versions of the Rovers and Satellite domains from past International Planning Competitions, showed that the explanation-based agent outperformed the other by a statistically significant margin in these domains.

2 Definitions and Notation

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 [2, Chapter 2].

Let \mathcal{P} be the finite set of all possible propositions for describing a planning environment. 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 can observe in the world. An observation associates a truth value with each of these propositions. Further, let $\mathcal{P}_{\text{hidden}}$ 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: (name, preconds, effects), where name, the *name* of the event, preconds and effects, the *preconditions* and *effects* of the event, are sets of literals. We use effects^- and effects^+ to denote the negative and positive literals in 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.

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 $\mathcal{T} = \{t_0, t_1, t_2, \dots, t_n\}$, 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 $\text{occ}(o) \mapsto t$; 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 $\text{obs}_0, a_1, \text{obs}_1, a_2, \dots, a_k, \text{obs}_{k+1}$. 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 $i = 0, \dots, k-1$ and each action a_j for $j = 1, \dots, k$ 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 $\text{occ}(o_i) \prec \text{occ}(o_j)$ such that $o_i, o_j \in C$. As a shorthand, we sometimes will say $o_i \prec o_j$ if and only if $\text{occ}(o_i) \prec \text{occ}(o_j)$.

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

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

$$\begin{aligned} \text{relevant}(p, o) \equiv & \text{knownafter}(p, o) \vee \text{knownafter}(\neg p, o) \vee \\ & \text{knownbefore}(p, o) \vee \text{knownbefore}(\neg p, o). \end{aligned}$$

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

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) $p \in \text{preconds}(e)$, (2) $\text{knownafter}(p, o)$, 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 $\text{knownafter}(\neg p, o)$, $\text{knownbefore}(p, o')$, 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 $\text{occ}(o) = \text{occ}(o')$, there may be no conflicts before or after: for all p , $\text{knownafter}(p, o) \implies \neg \text{knownafter}(\neg p, o')$, and $\text{knownbefore}(p, o) \implies \neg \text{knownbefore}(\neg p, o')$.

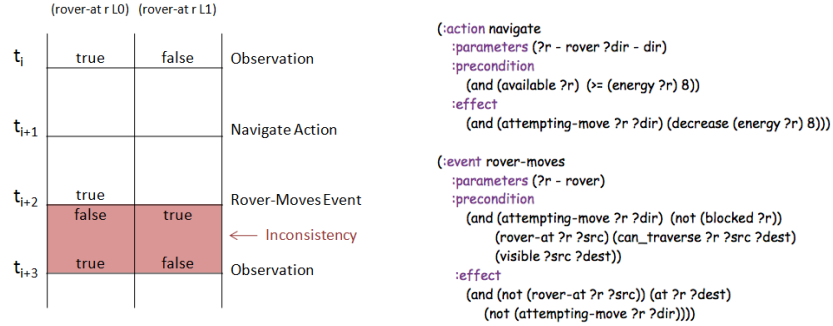


Fig. 1. Example of an inconsistent explanation, with occurrence points ordered on the left hand side. Relevant action and event descriptions are given on the right. Boolean values are the known-before and knownafter relations; for example, the value “false” at top right indicates that the relation knownbefore($\neg(\text{rover-at } r \text{ L1}), o_i$) holds.

4. If $\text{preconds}(e)$ of an event e are all satisfied at an occurrence point t , e is in χ at t ;

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 with the prior observation occurrence at t_i , in which the rover is observed at location L0, followed by a navigate action occurrence at t_{i+1} directing the rover to location L1, followed by an event occurrence o_{i+2} at t_{i+2} that changes the rover’s location from L0 to L1, followed by the succeeding observation occurrence o_{i+3} at t_{i+3} stating the rover is at L0.

Two inconsistencies then exist between the event occurrence and observation occurrence: $\langle(\text{rover-at } r \text{ L0}), o_{i+2}, o_{i+3}\rangle$ and $\langle\neg(\text{rover-at } r \text{ L1}), o_{i+2}, o_{i+3}\rangle$.

3 Generating Abductive Explanations

This section describes DISCOVERHISTORY, our search algorithm for generating plausible explanations by recursively applying refinements to implausible explanations, based on a series of observations and actions. DISCOVERHISTORY is designed to be used by a planning agent in a partially-observable dynamic environment. The planning agent takes actions, predicts events, and receives observations. Each time an observation is received, the agent creates an explanation consisting of all these occurrences and calls DISCOVERHISTORY to refine that explanation as necessary. DISCOVERHISTORY generates successive explanations by perceiving and 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 or no refinements can be made. In the recursive case, REFINE generates a set of explanations for each inconsistency, and each explanation is recursively considered by DISCOVERHISTORY. Cycles are prevented by maintaining a list of changes to the explanation (not shown) and cutting off branches which cause repetitions of the same inconsistency.

Algorithm 1: A high-level description DISCOVERHISTORY.

```
1 Procedure DISCOVERHISTORY( $\chi$ )
2 begin
3   if Inconsistencies( $\chi$ ) =  $\emptyset$  then
4      $\chi \leftarrow$  FINDEXTRAEVENTS( $\chi$ )
5     if Inconsistencies( $\chi$ ) =  $\emptyset$  then return  $\{\chi\}$ 
6   arbitrarily select an  $i \in$  Inconsistencies( $\chi$ )
7    $X \leftarrow$  REFINE( $\chi, i$ )
8   foreach  $\chi_{new} \in X$  do  $X \leftarrow X \cup$  DISCOVERHISTORY ( $\chi_{new}$ )
9   return  $X$ 
10 end
```

Let χ be a planner’s current explanation of the world which includes obs, the most recent observation received by the agent. If no inconsistencies are present in χ , then the first condition for plausibility (see Section 2) is met. Plausibility conditions 2 and 3 are assumed to be true of the initial explanation, and are maintained throughout the resolution process. Once no inconsistencies exist, therefore, DISCOVERHISTORY updates χ for plausibility condition 4 by adding any events that have been caused by the changes to the explanation in FINDEXTRAEVENTS. The new events that are added to the explanation may cause new inconsistencies. Otherwise, all four conditions for a plausible explanation are met and DISCOVERHISTORY returns the explanation χ in Line 5.

If there are inconsistencies between any pair of occurrences in χ , then the function Inconsistencies(χ) will return them. In Line 6, DISCOVERHISTORY selects an inconsistency i from Inconsistencies(χ) and attempts to resolve it.¹ The REFINE subroutine (Line 7 of Algorithm 1) finds all explanations that result from resolving each individual 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 maximum depth of $|C| * (2^{|E|})^2$ and a maximum branching factor of $|E|$, where E is the finite set of all possible events.

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 occurrence. All applicable methods must be applied, 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.

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.

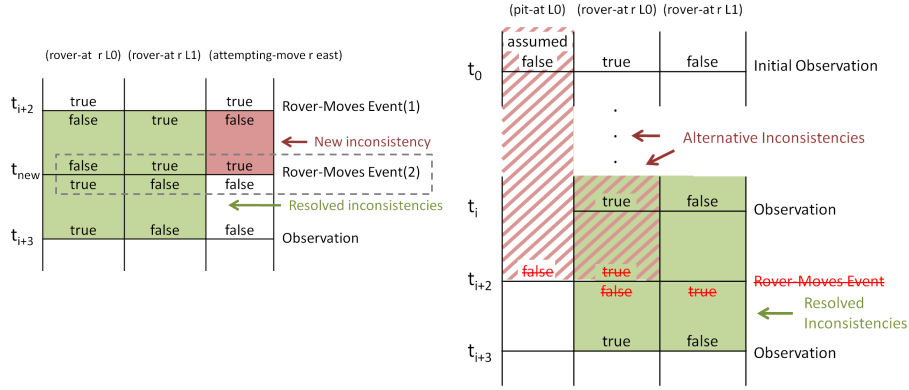


Fig. 2. (a) Example of adding an occurrence (left). (b) Example of removing an occurrence (right).

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 event o and the following event o' . An event o'' relevant to p must be found such that $o \prec o'' \prec o'$.

To add a new occurrence, REFINE starts by generating the set S of possible consecutive ordered pairs of occurrence points between $occ(o)$ and $occ(o')$, given the partial-ordering R . Then, for every e such that $effects(e) \models p$ and each $(t, t') \in S$, REFINE creates a new occurrence point t and a new event occurrence $o'' = (e, t)$. Then the algorithm adds o'' into C and updates the partial ordering R with $o \prec o'' \prec o'$. This results in a new explanation that does not contain the inconsistency i . Each resulting explanation is returned in a set to DISCOVERHISTORY for further refinement.

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 $(\neg(\text{rover-at r L1}), o_{i+2}, o_{i+3})$. 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, $((\text{attempting-move r east}), o_{i+2}, o_{new})$, will need to be eliminated in a recursive call to DISCOVERHISTORY.

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 accomplishes this as follows. The algorithm 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, REFINE 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 in this occurrence is the reserved **removal** event e_r , which has no effects and satisfies $\prec(e_r) = \{\neg p'\}$. Due to the presence of the removal event, no new occurrence can be added to the resulting explanation which would cause o to recur. REFINE returns the entire set of resulting explanations, each of which resolves the inconsistency.

Example 3. REFINE can use the above mechanism for removing occurrences to resolve the same inconsistency resolved 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 (\neg (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).

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, we generate 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 $\text{effects}^+(e_p) = \{p\}$. 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, (\neg (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 $o_{(\text{pit-at}L0)}$.

4 Experimental Evaluation

We examined the performance of DISCOVERHISTORY 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 and can not be predicted due to lack of knowledge during 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 perceive its location or recharge. 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

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

various causes of satellite malfunction: supernova explosions, which can damage sensitive instruments if 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 two agents, an explanation-based agent, called DHAgent (short for DISCOVERHISTORYAgent) and a non-explanation-based agent, called Non-DHAgent; each uses a modified version of the SHOP2 planner [9] to create plans that take into account events that are known to occur, and they use the same representation of the domain and the events that occur in it. During execution, when an observation contradicts the state predicted by SHOP2, each agent generates a new planning state and replans, then continues executing.

The two agents differ in how they construct the new planning state. Non-DHAgent removes all observable facts from the SHOP2 predictions, and adds in instead the facts in the environment observation. DHAgent first runs DISCOVERHISTORY. If DISCOVERHISTORY generates a plausible explanation, the explanation-based agent then generates a new planning state based on the most recent truth value for each predicate found in that explanation. If none is found, it falls back on the behavior of Non-DHAgent.

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 the 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 had 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 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 perfor-

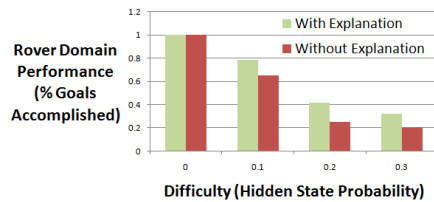


Fig. 3. Comparison at various difficulty levels.

mance. 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 3 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 $p > 0.0$.

5 Related Work

Planning and execution has been focus of several papers. A survey of existing replanning techniques is described in [10]. Continuous Planning and Execution Framework (CPEF) is introduced in [8]. 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 [13, 4, 1, 12, 11, 8, 7].

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.

McIlraith [5] focuses on an extension to the situation calculus theory for generation of *explanatory diagnoses* for aberrant behaviors given an observation, an initial state and a history of actions. McIlraith’s work has been extended to consider actions that occur during the history as well as actions that do not perform as expected [3]. Unlike ours, their formalization does not differentiate between the concepts of action and event; they only consider a single observation, rather than an execution history. This paper describes the first application of such ideas in a planning and execution system.

6 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 by our algorithm 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 algorithms, 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 (possibly uncertain) durative effects. Finally, we'd like to examine the impact of explanation in adversarial domains, and how it can be combined with plan recognition to achieve stronger results.

References

1. Ayan, F., Kuter, U., Yaman, F., Goldman, R.P.: HOTRiDE: Hierarchical Ordered Task Re-planning in Dynamic Environments. In: ICAPS-07 Workshop on Planning and Plan Execution for Real-World Systems (2007)
2. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann (May 2004), <http://www.laas.fr/planning>
3. Iwan, G., Lakemeyer, G.: What observations really tell us. In: KI-03 (2003)
4. Kambhampati, S., Hendler, J.A.: A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence* 55, 193–258 (1992)
5. Mcilraith, S.A.: Explanatory diagnosis: Conjecturing actions to explain observations. In: KR-98 (1998)
6. McKee, M.: Mars rover may not escape sand trap for weeks. <http://www.newscientist.com/article/dn17120-mars-rover-may-not-escape-sand-trap-for-weeks.html> (2009)
7. Myers, K.: Advisable planning systems. In: Tate, A. (ed.) *Advanced Planning Technology*. AAAI Press. (1996)
8. Myers, K.L.: A continuous planning and execution framework. *AI Magazine* pp. 63–69 (1999)
9. Nau, D., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: SHOP2: An HTN planning system. *JAIR* 20, 379–404 (Dec 2003), <http://www.jair.org/papers/paper1141.html>
10. Russell, S., Norvig, P.: *Artificial Intelligence, A Modern Approach* (Second Edition). Prentice-Hall, Upper Saddle River, NJ (2003)
11. Wang, X., Chien, S.: Replanning using hierarchical task network and operator-based planning. In: ECP (1997)
12. Warfield, I., Hogg, C., Lee-Urban, S., Munoz-Avila, H.: Adaptation of hierarchical task network plans. In: FLAIRS-2007 (2007)
13. Yoon, S., Fern, A., Givan, R.: FF-Replan: A baseline for probabilistic planning. In: ICAPS-07 (2007)