Advances in Cognitive Systems X (20XX) 1–6 Submitted X/20XX; published X/20XX

Continuous Explanation Generation in a Multi-Agent Domain

Matthew Molineaux MATTHEW.MOLINEAUX@KNEXUSRESEARCH.COM
Knexus Research Corporation, 9120 Beachway Lane, Springfield, VA

David W. Aha DAVID.AHA@NRL.NAVY.MIL
Code 5514, Naval Research Laboratory, 4555 Overlook Ave. SW, Washington, DC 20375

Abstract

An agent operating in a dynamic, multi-agent environment with partial observability should continuously generate and maintain an explanation of its observations that describes what is occurring around it. We update our existing formal model of occurrence-based explanations to describe ambiguous explanations and the actions of other agents. We also introduce a new version of DiscoverHistory, an algorithm that continuously maintains such explanations as new observations are received. In our empirical study this version of DiscoverHistory outperformed a competitor in terms of efficiency while maintaining correctness (i.e., precision and recall).

1. Introduction

In many domains, it is desirable for a cognitive agent to collaborate or compete with other agents, especially humans. In general, this requires the agent to understand what other agents are doing. This task may be non-trivial, particularly in partially observable environments. Because many modern robotic sensors can only gather information at discrete intervals, a cognitive agent for real-time environments should be able to infer the occurrence of an action taken by another agent from observations that precede and follow it. This requires the agent to perform a diagnostic or explanatory task, inferring actions, events, and processes that explain its observations. For example, an agent performing alongside an army patrol would, when the patrol suddenly comes under fire, recognize that other team members are taking cover, and execute appropriate actions to help.

Prior research in diagnosis or explanation of action sequences has examined the problem in isolation, which removes the need to examine several important issues. First, an agent in a real environment must explain incrementally, so that other cognitive processes have continuous access to an up-to-the-minute inferred action sequence. Second, the agent should recognize the presence of exogenous events as well as actions, which may (or may not) be caused indirectly by other agents' actions. Finally, a situated cognitive agent should be able to infer states from partial observations, so as to make informed action selections. The number of consistent interpretations of a partially observable environment may be infinite when referring to hidden continuous quantities. Thus, it is desirable to represent ambiguity directly rather than attempt to represent all possibilities (or a sample of them). We refer to the combined task of inferring exogenous actions, events, processes, and the initial state of an agent's environment as *continuous explanation*

generation. Continuous explanation generation can be a useful component in a larger reasoning process: the explanation generator provides other components with an interpretation of events at a more manageable level of abstraction than the complete observations, thus simplifying their reasoning. For example, belief management can be performed by considering the consequences of the explanation (Molineaux et al., 2012), plan recognition can be performed by finding plans that include the inferred actions (Kautz & Allen, 1986), and domain model learning can be performed by inferring the conditions of unresolvable inconsistencies (Molineaux & Aha, 2014).

In this paper, we describe an extension of the DiscoverHistory algorithm that is appropriate for multi-agent domains; in addition to inferring explanations consisting of exogenous events that explain observations, this version searches a space of *ambiguous* explanations, maintains ambiguity for efficiency purposes and represents exogenous actions. We also present a new formal model to describe the explanations used by this extension. Finally, we present an empirical study showing DiscoverHistory to be efficient and accurate relative to a baseline continuous explanation generator in a multi-agent domain.

Section 2 describes related work on explanation and diagnosis. Section 3 describes how we formally model explanations with actions and ambiguity. Section 4 describes DiscoverHistory and how it searches the space of explanations. Section 5 describes DH-Agent, an agent that uses DiscoverHistory to maintain its explanation over time. Section 6 then presents an experimental comparison of DiscoverHistory with a baseline explanation generator to examine their relative correctness and efficiency.

2. Related Work

Work on the topic of explanation is not new (e.g., Josephson & Josephson 1996; Leake 1995). Explanation research focuses on finding a hypothesis or hypotheses that entail some observations and match some model. In this paper, the hypotheses constitute lists of occurrences with temporal and variable constraints, the model is a domain model that describes action and event types as used in automated planning, and the observations are partial states of a progressing environment. We focus on this specific case because it is highly important in many multi-agent contexts.

AbRA (Bridewell & Langley, 2011) also incrementally constructs explanations. It can perform online plan recognition, a sequential task, but does not infer specific occurrences based on a domain model. UMBRA (Meadows et al., 2013) performs plan understanding by incrementally constructing explanations, and infers an agent's beliefs, desires, and intentions to explain observed actions, rather than inferring occurrences to explain observed states. As, for example, Ram (1993) and Leake (1995) have noted, these tasks are likely to be highly important to a range of cognitive agents. However, their requirements differ significantly from the task of constructing continuous action sequences as we discuss here.

A related problem to explanation is diagnosis of discrete-event systems (Sampath et al., 1995), also referred to as history-based diagnosis (Gspandl et al., 2011), which finds action or event histories that account for a series of observations that sometimes include observed events. Aside from representations, our formal model of explanations differs from the standard discrete-event system diagnosis model in the following ways: (1) we distinguish actions from events (which are deterministic), to better understand which actors are responsible for which actions and to predict

unavoidable consequences; (2) we provide a formal model of ambiguous occurrences and a way to characterize their correctness, and (3) we assume that only partial states, and never actions or events, are observable. Diagnosis efficiency is considered a major issue in this community; recent efficient systems convert diagnosis problems to satisfiability or planning problems and adopt efficient techniques for solving the new problem (Grastien et al., 2011; Sohrabi et al., 2010).

Finally, earlier work on DiscoverHistory (Molineaux et al., 2012; Molineaux & Aha, 2014) established a formal model and method for inferring sequences of events and assumptions that explain a series of observations. The novel contributions of this paper are: (1) a formal model of the plausibility of ambiguous explanations with variables; (2) a formal model of exogenous actions; (3) extensions to DiscoverHistory for retention of ambiguity and efficient search; and (4) an empirical study comparing the correctness and efficiency of DiscoverHistory to a baseline in a multi-agent environment.

3. Modeling Explanations

In this section, we update our earlier formal model (Molineaux et al., 2012; Molineaux & Aha, 2014) to include exogenous actions and represent ambiguity. Below, we formally describe: *events* and *actions; event models* and *action models; explanations* that list, order, and constrain those events and actions; *plausibility* of an explanation; and inference of *projected states and events.*

3.1 Events and Actions

We use standard classical planning definitions (Ghallab, Nau, & Traverso, 2004) for our model. Let P be the finite set of all propositions describing a planning environment, where a **state** assigns a value to each $p \in \mathrm{P}$, denoted as value(p). A planning environment is *partially* observable if an agent α can access it only through **observations** that do not reveal the complete state. Let default(p) give the default value for a proposition p to be assumed in the absence of contradictory information. Then, $\mathrm{P}_{obs} \subset \mathrm{P}$ is the set of all propositions that α will observe when $\mathrm{value}(p) \neq \mathrm{default}(p)$. Let $\mathrm{P}_{hidden} \subseteq \mathrm{P}$ be a set of *hidden* propositions that α does not observe (e.g., the exact location of a robot that does not have a GPS contact).

A **domain model** is a description of what actions or events can occur in the environment to be explained. It includes 0 or more event models and 0 or more action models.

An **event model** is a description of what causes events of a specific type to occur and how they affect the world. It is described by a tuple (name; preconds; effects; constraints) denoting an event's name, sets of preconditions and effects (sets of variable literals), and a set of constraints over variables present in the model. An **event** is an occurrence described by an event model and bindings to one or more model variables. Events occur immediately when all their preconditions are met. After each action, any events it triggers occur, followed by events they trigger, etc.

Similar to an event model, an **action model** describes the circumstances under which an action can be performed and the effects of performing that action. It is described by a tuple (name; preconds; effects; constraints; performer). Aside from the performer, meanings of action model elements are identical to the corresponding event model elements. The performer of an action model is described by a variable with a range consisting of labels for all agents known. An **action**

is an occurrence described by an action model and bindings to one or more model variables. An action occurs only when its preconditions are met *and* its performer intends for it to occur.

If an event or action includes one or more unbound variables, it is an **ambiguous event** or **ambiguous action**.

3.2 Explanations

We formalize the agent's knowledge about the changes in its environment as an explanation of the environment's history. We define a finite set of **occurrence points** T=$\{t_0, t_1, t_2, \cdots, t_n\}$ and an ordering relation between pairs of points, denoted as $t_1 \prec t_2$, where $t_1, t_2 \in$ T. While these occurrence points do not have a metric relationship to time, an ordering between occurrence points implies an identical temporal ordering.

Three types of occurrences exist. An **observation occurrence** is a pair (obs, t), where obs is an observation and t is an occurrence point. An **action occurrence** is a pair (a, t), where a is an action. Finally, an **event occurrence** is a pair (e, t), where e is an event. Given an occurrence o, we define occ() such that $\text{occ}(o) \mapsto t$.

An **execution history** $\text{obs}_0; a_1; \text{obs}_1; a_2; \cdots; a_k; \text{obs}_{k+1}$ is a finite sequence of observations and actions. An **explanation** is a description of all occurrences that an agent infers have taken place that can be used for inferring believed world states. Formally, we describe an explanation as a tuple $\chi = (O, R, C)$, where *O* is a finite set of occurrences that includes each obs_i ($i \in [0, k+1]$) and each action a_j ($j \in [1, k]$) in the history, as well as 0 or more event occurrences and 0 or more assumed exogenous actions. *R* is a partial ordering over *O*, described by ordering relations $\text{occ}(o_i) \prec \text{occ}(o_j)$ where $o_i, o_j \in O$, which we will sometimes write as $o_i \prec o_j$. Finally, *C* is a set of inequalities and equations that describe relationships between variables in occurrences in *O*.

We use $\text{knownbefore}(l, o, v)$ and $\text{knownafter}(l, o, v)$ to refer to the value *v* of literal l immediately before or after occurrence $o \in O$. For action and event occurrences, $\text{knownbefore}(l, o, v)$ holds iff $\{\text{value}(l) = v\} \in \text{preconds}(o)$ and $\text{knownafter}(l, o, v)$ holds iff $\{\text{value}(l) = v\} \in \text{effects}(o)$. If o is an observation occurrence and $\{\text{value}(l) = v\} \in \text{obs}$, then $\text{knownbefore}(l, o, v)$ and $\text{knownafter}(l, o, v)$ hold, and otherwise are false. Note that a literal l may contain unbound, existentially quantified variables. Literals in these definitions may or may not be ground. We say that occurrence o is **relevant** to a literal l iff:

$$\text{relevant}(l, o) \equiv \exists \text{v: } \text{knownafter}(l, o, v) \vee \text{knownbefore}(l, o, v).$$

3.3 Plausible Explanations

The **proximate cause** of an event occurrence (e, t) is the preceding occurrence that triggers the event. It must be an occurrence o that satisfies three conditions with respect to an explanation $\chi = (O, R, C)$ and some literal $l : \{\text{value}(l) = v\} \in \text{preconds}(e)$, $\text{knownafter}(l, o, v)$, and there is no other occurrence $o' \in O$ *s.t.* $o \prec o' \prec (e, t) \in R$. Standard event occurrences must have at least one proximate cause, so by the third condition, they must occur immediately after their preconditions are satisfied. Because actions do not necessarily occur when their conditions are met, but rather when a performer intends them to, actions do not have a proximate cause. Similarly, observations and special events that assume information about the initial state have no proximate cause, as they are not triggered directly by an occurrence. For this reason, exogenous

actions may be assumed to occur at any point when their preconditions are met; we call these occurrences, and initial state occurrences, **assumptions**.

An **inconsistency** is a tuple $i = (l, o, o')$ where o and o' are two occurrences in χ such that $\text{knownafter}(l, o, v)$, $\text{knownbefore}(l, o', w)$, $v \neq w$, and there is no other occurrence $o'' \in O$ such that $o \prec o'' \prec o' \in R$ and l is relevant to o''. Roughly speaking, an inconsistency occurs when there *may be* a contradiction between an effect literal of an earlier occurrence and a precondition literal of a later occurrence. Note that any unordered pair of occurrences may be inconsistent. Furthermore, a literal l with a variable may contradict a legal interpretation of l (i.e., if some other legal interpretation of l contradicts it). As a special case, an inconsistency also exists when a hidden literal with no precedent contradicts a default assumption. Formally, the inconsistency $i = (l, none, o')$ exists where (1) $\exists v$: $\text{knownbefore}(l, o', v)$ is true, (2) there is no occurrence o such that $\text{relevant}(l, o) \wedge o \prec o' \in R$, and (3) $\text{default}(l) \neq v$.

Some inconsistencies are identified as **ambiguous**. This means that one (or both) occurrences in the inconsistency is ambiguous, and multiple legal bindings θ to unbound variables in those occurrences would remove the inconsistency without requiring additional ordering constraints.

An explanation $\chi = (O, R, C)$ is **strictly plausible** iff:

1. There are no inconsistencies in χ.
2. Every event occurrence $(e, t) \in O$ has a proximate cause in O.
3. Simultaneous occurrences do not contradict in preconditions or effects. For all $\boldsymbol{l, o, o', v, w}$:
   $\text{knownafter}(\boldsymbol{l, o, v}) \wedge \text{knownafter}(\boldsymbol{l, o', w}) \wedge \text{occ}(\boldsymbol{o}) = \text{occ}(\boldsymbol{o'}) \Longrightarrow \boldsymbol{v = w}$,
   $\text{knownbefore}(\boldsymbol{l, o, v}) \wedge \text{knownbefore}(\boldsymbol{l, o', w}) \wedge \text{occ}(\boldsymbol{o}) = \text{occ}(\boldsymbol{o'}) \Longrightarrow \boldsymbol{v = w}$.
4. If $\text{preconds}(e)$ of an event e are all satisfied at an occurrence point t, the event occurrence (e, t) is in O.
5. R forms a total preorder: each pair of occurrences is either ordered or simultaneous.
6. All actions $a \in O$ not in the execution history are exogenous (performed by another agent).
7. No occurrence in O has an unbound variable.

In summary, an explanation must describe an uninterrupted causal network that implies all observations to be considered plausible, and the identities of all participants in all occurrences must be known. This is an extension of the notion of plausibility given in earlier work; additional requirements (i.e., 5-7) are added to accommodate ambiguity and actions.

A relaxation of this notion is **ambiguous plausibility**. To be ambiguously plausible, explanation $\chi = (O, R, C)$ must meet all conditions for strict plausibility except for conditions 1 and 7, which are replaced by:

1. There are no *unambiguous* inconsistencies in χ.
7. There is at least one possible binding for each variable that meets all constraints in *C*.

3.4 Projected States and Events

A projected state gives all facts that would be true in the environment at time t, given that all assumptions in χ are correct and χ is consistent. The projected state proj(*t*) is given by:

$$\text{proj}(t) = \left\{ l \,\middle|\, \exists o, v \;\; \text{knownafter}(l, o, v) \wedge \text{occ}(o) \prec t \wedge \nexists o' \left(\begin{array}{c} \text{knownafter}(l, o', w) \;\wedge \\ \text{occ}(o) \prec \text{occ}(o') \prec t \;\wedge \\ v \neq w \end{array} \right) \right\}.$$

Projected events are the set of events that must happen at time *t* because their preconditions are met in the projected state $\text{proj}(t)$:

$$\text{projectedevents}(t) = \{(e,t)|\text{proj}(t) \vdash \text{preconds}(e)\}.$$

4. DiscoverHistory

The DiscoverHistory algorithm is intended to facilitate a search through explanation space; it takes a single explanation as input and returns a set of explanations that are closer to being ambiguously plausible. This process is detailed is pseudocode in Figure 1. The default approach, as shown in steps 3 and 4, is to resolve a single inconsistency i by applying all applicable refinement methods, and return a set of explanations that no longer have inconsistency i. For efficiency, DiscoverHistory adds all projected events to the explanation after a new assumption is found or the explanation appears otherwise complete, as shown in steps 1 and 2. Note that the explanations returned by DiscoverHistory may have *more* inconsistencies than the original, but include an event, binding, or other constraint that is needed to achieve consistency. DiscoverHistory may be used with many different types of search as an expansion function, and is not dependent on any specific commitment to the ordering of explanations that are searched.

The following three subsections describe the specific implementation in DiscoverHistory of event projection, inconsistency selection, and refinement methods respectively, and how they differ from prior versions of the algorithm. Following that, we present an extended example that describes how each of these affects the explanation generation process in a single iteration of explanation generation and maintenance. This extended example is comprehensive and is presented in lieu of an ongoing example; therefore, readers may find it useful to refer ahead to it for better understanding of sections 4.1-4.3 (next).

4.1 Event projection

Event projection is a fast means of extending an explanation with all events that it already causally implies. This procedure identifies what events have preconditions met by states succeeding an assumption. All such events are added to the explanation. Event projection

DiscoverHistory(*X*)

1. If the last assumption added to X:
   is ordered with respect to all observations
   and has no unambiguous inconsistencies,
   and event projection hasn't been performed,
   Add projected events to X and return it as a singleton list //See Section 4.1
2. If all occurrences are totally preordered by the precedence relation,
   and X has no unambiguous inconsistencies,
   Add projected events to X and return it as a singleton list //See Section 4.1
3. Select inconsistency $i \in X$. //See Section 4.2
4. Find and return a set of refined explanations [X'] by applying all refinements of i. //See Section 4.3

Figure 1. DiscoverHistory Pseudocode

proceeds by iterating over each successive occurrence point t, and adding to the explanation the set projectedevents(t) of events projected to occur at time t. Projection may find events that unify with events already in the explanation, which causes the less specific events to be replaced. Formally, if any existing event occurrence (e, t) in the explanation has the same occurrence point as a new projected event e', and some binding θ when applied to e results in an occurrence e'' that matches pairwise each literal in e', then e is removed from the occurrence set. Event projection is exemplified in Figures 3 and 4 below.

This procedure is not novel, but has been extended from earlier versions to accommodate ambiguous occurrences. It is used here to speed up search and ensure that no events whose preconditions are satisfied are left out.

4.2 Inconsistency selection

Selection of inconsistencies is an important heuristic process. As they can be resolved in any order, and sometimes create new inconsistencies, choice of inconsistency affects both the branching factor and depth of search. To intelligently select among these inconsistencies in the large explanation space caused by multiple agents, we have developed a new inconsistency selection method. DiscoverHistory considers inconsistencies in the following order, stopping with the first non-empty set:

1. Inconsistencies with a single resolution (causing no immediate branching) or none (causing termination of that search path).
2. Unambiguous inconsistencies involving the most recently added assumption (action or initial state), if any.
3. Unambiguous inconsistencies involving the most recently added event, if any.
4. Unambiguous inconsistencies between a pair of occurrences that includes an assumption that is unordered with respect to one or more other occurrences.
5. Unambiguous inconsistencies between a pair of occurrences that includes an occurrence that is unordered with respect to one or more other occurrences.
6. Ambiguous inconsistencies.

This ordering method for selecting inconsistencies is designed to drive refinement toward identifying a consistent assumption that is ordered with respect to all observations. Once such an assumption is found, event projection can be used to deduce all events that result from it, which quickly reduces the remaining inconsistencies.

Whenever multiple inconsistencies are considered, an inconsistency with the fewest refinements (i.e., which causes the least branching in search) is selected for expansion.

4.3 Refinement methods

There are several methods by which an explanation can be refined to produce an explanation that no longer has a given inconsistency. Each method below presents one manner in which certain inconsistencies can be refined to expand the next level of a search tree. Of these, the first is modified and the last two are new to this version of DiscoverHistory.

4.3.1 Hypothesize event or action

One method for refining an inconsistency (l, o, o') is to find an ambiguous exogenous action or event occurrence o'' that satisfies $o \prec o'' \prec o'$ and causes l. In this ambiguous occurrence, all parameters are left ambiguous that are not directly constrained by the need to cause l. This refinement will correctly resolve problems when a true occurrence has not yet been inferred. All constraints in an event model referring to variables that remain unbound in the hypothesized event are added to the refined explanation. As multiple events and actions may entail the same literal, multiple instances of this refinement may be applicable to the same inconsistency.

When a literal l contains unbound variables, constraints on variables in l may contradict constraints added by the new event. If so, no refinement that adds that new event is possible.

4.3.2 Remove event or action

In some cases, an inconsistency (l, o, o') may contain one or more occurrences that did not actually occur. The correct resolution in this case is to remove o or o' from the explanation. Observations and actions taken by the explaining agent are not removed in this fashion, as they are certain. Furthermore, DiscoverHistory prevents the removal of an event or action that was previously added during the same search, to prevent cycles. This refinement may be applied up to twice per inconsistency, for the preceding and succeeding events.

4.3.3 Add assumption to initial state

When an inconsistency $(l, none, o')$ exists that has no prior occurrence (i.e., it contradicts the default assumption), an occurrence may be added that describes a different assumption about the initial state. The added event has the single effect l and occurs at t_0, the same time as the initial observation. This refinement can be applied at most once per inconsistency.

4.3.4 Bind an unbound variable(s)

In two different conditions, the binding of one or more unbound variables throughout an explanation can resolve an inconsistency (l, o, o'). The first situation occurs when the inconsistent occurrences match except for a missing variable binding. If either o or o' is an ambiguous occurrence, and some legal binding set θ exists such that substitution of θ in o, o', and l results in $\text{effects}(o) \vdash l$, substituting θ throughout the explanation resolves the inconsistency.

The second situation occurs when some event in the explanation could cause p, but it is not yet ordered so as to come between the inconsistent occurrences. Formally, this occurs when a known ambiguous occurrence o'' exists such that o is not known to precede o'' and o'' is not known to precede o', and some legal binding set θ exists such that substitution of θ in o'', o', and p results in $\text{effects}(o'') \vdash p$. In this case, the explanation must be modified by substituting θ throughout and adding precedence constraints $o \prec o'' \prec o'$.

The requirement of legality applies to direct and indirect consequences of substitution. If a variable x becomes bound, and some variable y is constrained to have a functional relationship to x, y may take on a value that violates some constraint as a result; this binding would be illegal.

As part of the refinement process, when an occurrence o is bound, it is checked against each other occurrence o' to determine if o is identical to, or more specific than o'. If the substitution of some θ for variables in o' would result in an occurrence identical to o, then o' is discarded.

4.3.5 Constrain order of two occurrences

Suppose that for some occurrence (l, o, o'), it is not known that $o \prec o'$. Then the precedence $o' \prec o$ can be added to the explanation.

4.4 Extended Example

To illustrate DiscoverHistory's operation, the following example illustrates how it modifies the current explanation after a contradictory observation is received in the synthetic military domain *Autonomous Squad Member (ASM)*. In this environment, a robot assists an army patrol by following and carrying equipment, which requires the robot to monitor what patrol members are doing. We start near the beginning of a scenario, where one team member of the patrol has just started to walk away from the starting location. So far, the robot has started to follow the team leader, labelled as member1, and made three successive observations of the environment. When the fourth observation arrives, it yields two surprising observations: (1) a team member, labelled member2, is reported to be at an unnamed location (generic label unk-location) rather than the starting location (labelled locstart), as the robot expected; and (2) team member member2 is reported to be somewhere along the route route1. Using DiscoverHistory, the agent attempts to modify its explanation to explain the unexpected observations. Figure 2 lists parts of the robot's memory before DiscoverHistory begins, including the existing explanation that the robot has been maintaining, partial observations, and the discovered inconsistencies.

Initial Explanation

Occurrences

observation1: ... (object-location member2 locstart) ...
action1: (follow robot1 member1)
observation2: ...
observation3: ... (reported-location member2 locstart), (reported-route member2 no-route) ...
observation4: ... (reported-location member2 unk-location), (reported-route member2 route1) ...

Precedence

observation1 ≺ action1.
action1 ≺ observation2.
observation2 ≺ observation3.
observation3 ≺ observation4.

Constraints: None.

Inconsistencies

<(reported-location member2 unk-location), observation3, observation4>
<(reported-route member2 route1), observation3, observation4>

Figure 2. Robot's memory near the beginning of an ASM scenario

The first inconsistency is selected, because only one refinement applies to it. This refinement is an event hypothesis refinement; it adds a new event of type gps-observe-location, which represents an update received by the robot from the patrol member's GPS transponder. The event reports a label for a known location a patrol member is near, and a label for any named route the patrol member may be following. The event is partially bound to match the inconsistent literal as well as static literals. Figure 3 gives a complete representation of the general event model and event.

This event is ordered after observation3 and before observation4 by the hypothesis refinement (Section 4.3.1). In addition to member2 and unk-location, several other values can be bound to model variables using static predicates that unambiguously identify them. Several new inconsistencies are found in this new explanation (shown in Figure 4). Each corresponds to a literal from the preconditions or effects of the added event that does not match other occurrences:

1. The route being followed by member2 is referenced by the precondition of the gps-observe-location event, and represented as the literal (person-route member2 ?route56). The value of this literal has not been assigned by any previous occurrence, and the default value is no-route. Intuitively, this is an inconsistency because ?route56, being unbound, matches nothing. Formally, some interpretation of the inconsistent literal contradicts the default assumption.
2. Preconditions of the gps-observe-location event state that the value ?route56 must be of type route. However, it is as yet unassigned to any value, so the precondition is not met. This inconsistency also contradicts the default assumption.

Event Model

```
(:event gps-observe-location
:precondition
   (and (on-team ?teammate ?team)
      (is-robot ?self)
      (on-team ?self ?team)
      (eq (object-location ?teammate) ?loc)
      (eq (person-route ?teammate) ?route)
      (or (neq (reported-location ?teammate) ?loc)
          (neq (reported-route ?teammate) ?route)))
 :effect
   (and
     (set (reported-location ?teammate) ?loc)
     (set (reported-route ?teammate) ?route)))
```

Event

Type: gps-observe-location

Preconds:

(on-team member2 team1); (is-robot robot1); (on-team robot1 team1); (object-location member2 unk-location); (person-route member2 ?route56); (is-route ?route56); (not (reported-location member2 unk-location))

Effects: (reported-location member2 unk-location); (person-route member2 ?route56)

Constraints: None.

Signature Tuple: (gps-observe-location member2 team1 robot1 no-location ?route56)

Figure 3.Representation of gps-observe-location event and model

3. A precondition of the gps-observe-location event indicates that the location of member2 is unknown, but it is known at the time of the initial observation, observation1.
4. The occurrence observation4 indicates that member2 is reported to be following route1, which contradicts the effect of the gps-observe-location event which places it on the route ?route56. This information is represented by the literal (reported-route member2 route1).

<(person-route member2 ?route56), none, (gps-observe-location member2 team1 robot1 unk-location ?route56)>
<(is-route ?route56), none, (gps-observe-location member2 team1 robot1 unk-location ?route56)>
<(reported-route member2 route1), (gps-observe-location member2 team1 robot1 unk-location ?route56), observation4>
<(object-location member2 unk-location), observation1, (gps-observe-location member2 team1 robot1 unk-location ?route56)>

Figure 4. Possible inconsistencies after addition of gps-observe-location event

To select an inconsistency for resolution, DiscoverHistory first considers inconsistencies with only one possible refinement; there are none. Second, it considers unambiguous inconsistencies relating to the most recently added assumption; none exists. Third, it considers unambiguous inconsistencies relating to the most recently added event; this includes inconsistencies 1, 3, and 4. Among these, inconsistency 3 is selected because it has the fewest possible refinements.

DiscoverHistory applies refinements to this inconsistency to obtain refined explanations, as follows: hypothesis of a move action (not shown) succeeds, because it has a literal in its effects of type person-route. Removal is inapplicable because the prior event is an occurrence and the following event was added earlier in the same search. An initial assumption can be added because no prior occurrence is relevant to the predicate. Binding the variable ?route56 to the value no-route is an applicable refinement that assigns a legal value to ?route56. Constraining the precedence order is inapplicable because observation1 and gps-observe-location are already ordered. In total, the refinement methods result in three modified explanations. A representation of these, omitting the information carried over from the initial explanation, is shown in Figure 5.

After some further searching, an explanation is found that is close to being ambiguously plausible, as it includes a move assumption that has no unambiguous inconsistencies, and all occurrences are ordered. At this point, event projection must be performed, to ensure that all events that should be caused by changes to the explanation are added. This causes two events to occur, of the type human-sees. While these events have no observable effect on the state, they intuitively provide the information that the other patrol members must know what member2 is doing. Figure 6 shows the explanation before and after projection.

After projection, no unambiguous inconsistencies remain, and search returns the explanation produced by projection. After more observations are received, new information may cause the destination of the move action, ?dest57, seen in Figure 6, to be bound. However, the activity id, ?act60, is not present in any observable literals, so it will remain unbound indefinitely, with no consequence to the robot's understanding of what is happening.

Occurrences:
event1: (gps-observe-location member2 team1 robot1 unk-location ?route56)
action2: (move member2 ?dest57 ?route56 ?tm58 ?origin59 ?act60)
Precedence:
observation1 ≺ action2.
action2 ≺ event1.
observation3 ≺ event1.
event1 ≺ observation4.
Constraints: ?route56 != no-route, ?dest57 != locstart
Computed cost: 2 (precedence ambiguity) + 10 (assumptions) + 2 (depth) + 6 (event load) = 20

Occurrences:
event1: (gps-observe-location member2 team1 robot1 unk-location ?route56)
event2: (initial-assumption (person-route member2 ?route56))
Precedence:
event2 ≺ observation1.
observation3 ≺ event1.
event1 ≺ observation4.
Constraints: None
Computed cost: 3 (age) + 10 (assumptions) + 2 (depth) + 6 (event load) = 21

Occurrences:
event1: (gps-observe-location member2 team1 robot1 unk-location no-route)
Precedence:
observation3 ≺ event1.
event1 ≺ observation4.
Constraints: None
Computed cost: 2 (depth) + 6 (event load) = 8

Figure 5: Resulting explanations with computed explanation costs
(ASSUMPTION_COST = 10, EVENT_COST = 6)

5. DH-Agent

DH-Agent is an agent designed to interact with the world, make plans, replan, and maintain an understanding of its environment through use of an explanation generator. The hierarchical task network planner SHOP2-PDDL+ (Molineaux et al., 2010) is used for planning and replanning, and an external simulator executes actions and generates observations.

DH-Agent maintains a set of plausible explanations of the world. Initially, this consists of a single empty explanation, containing no occurrences or constraints. Each time it executes an action or receives an observation, DH-Agent adds the occurrence to its execution history and current explanation. After each observation, DH-Agent determines whether any of its current explanations are still ambiguously plausible. If not, it uses DiscoverHistory or a separate baseline explanation generator (see Section 6.1) to update its explanation set. If search fails, DH-Agent keeps its current explanations.

Occurrences:
event1: (gps-observe-location member2 team1 robot1 unk-location route1)
action2: (move member2 ?dest57 route1 600 locstart ?act60)
Precedence:
observation3 ≺ action2.
action2 ≺ event1.
event1 ≺ observation4.
Constraints: ?dest57 != locstart
Computed cost: 10 (assumptions) + 4 (depth) + 6 (event load) = 20

Occurrences:
action2: (move member2 ?dest57 route1 600 locstart ?act60)
event1: (gps-observe-location member2 team1 robot1 unk-location route1)
event2: (human-sees member1 ?act60)
event3: (human-sees member3 ?act60)
Precedence:
observation3 ≺ action2.
action2 ≺ event1.
event1 ≺ observation4.
occ(event1) = occ(event2) = occ(event3)
Constraints: ?dest57 != locstart
Computed cost: 10 (assumptions) + 5 (depth) + 6 (event load) = 21

Figure 6: Explanation before and after projection with computed explanation costs (ASSUMPTION_COST = 10, EVENT_COST = 6)

When using DiscoverHistory, DH-Agent finds successor explanations by performing a best-first-search through explanation space, using DiscoverHistory to expand the search tree at each selected node. This search prioritizes nodes that minimize a heuristic cost function. Search terminates when the first ambiguously plausible explanation is found, returning that single explanation. The cost of an explanation is calculated based on plausibility and efficiency. Plausibility is measured as the sum of three metrics:

- *Age* is calculated as the number of observations between the earliest occurrence added during the current search and the current time. It measures how long something must have gone unnoticed for this explanation to be correct. This reflects a belief that a more recent mistake is more likely than an older one, which might have been noticed earlier.
- *Precedence ambiguity* is calculated, for each event and action, as the number of observations that have no precedence relationship with that occurrence. Formally, $\text{precedence-ambiguity}(occ) = |\bigcup_{obs \in O} obs \not\prec occ \wedge occ \not\prec obs|$. This must be 0 in an ambiguously plausible explanation, and it measures the degree to which the precedence relation is still indeterminate.
- *Assumption cost* is counted as ASSUMPTION_COST for each exogenous action and each initial state assumption. It measures how many distinct factors were unknown prior to search. This component rewards parsimony.

In combination, these three metrics guide the search toward more plausible explanations, but there is no guarantee of optimality.

The efficiency component of the explanation cost function includes two factors:

- *Search depth* is equal to the depth of an explanation in a search tree. Use of search depth prevents recursive applications of refinement operators that do not change plausibility from dominating the search space. While incorporating search depth prevents an infinite recursion, it is not intended as a significant heuristic component.
- *Event load* starts at 0 and increases by EVENT_COST each time an event is added by the hypothesis refinement. This biases the search toward explanations with fewer abductively inferred events, which reduce search depth with earlier application of event projection.

See Figure 5 for examples of the explanation cost function and the metrics that support it.

6. Experiment

6.1 Design of a Baseline Generator

Efficiency results in the diagnosis of discrete-event systems indicate that use of automated planners to solve diagnosis problems is currently the most efficient solution (Grastien et al., 2011). To ensure that generated plans satisfy a sequence of observations rather than a distant goal, Sohrabi et al. (2010) demonstrated the addition of a special *advance* action to a planning problem; it forces a planner to generate plans that explain all observations. This approach is not directly applicable to the incremental problem, which requires modification of an existing action sequence rather than construction of a new one. However, we use this as inspiration to create a baseline continuous explanation generator, the Deductive Explanation Generator (DEG), which uses the same principle to maintain a set of strictly plausible explanations. In a single planning step, a forward state-space planner considers every possible action whose preconditions are met and projects its consequences. Analogously, DEG finds a set of possible actions that could have been performed after each new observation is received, and then adds each such action to each explanation it maintains. Then, consequences are projected for each explanation as in Section 4.2. DEG retains a subset of the resulting explanations that have no inconsistencies, and therefore explain the new observation, as the successor explanation set. The initial (empty) explanation is strictly plausible, and by induction, every explanation DEG maintains is strictly plausible.

Because the branching factor becomes large, the full set of explanations generated by DEG can exceed the available memory space. To avoid this, DEG retains only a subset XMAX of the possible explanations found. These are selected uniformly at random among the plausible explanations found after each observation, and the rest are discarded. Thus, there is no guarantee that DEG can find a consistent explanation indefinitely. If no plausible explanation can be found, DEG stops trying to explain and thereafter returns the last non-empty explanation set found. This is analogous to the memory space problems that plague typical planners in large domains.

DEG is designed to consider all reasonable assumptions, and unless it drops explanations to save memory space, it always finds an explanation with no false positives.

6.2 Experiment Description

As we are interested in incorporating DiscoverHistory as a component of a cognitive agent in a real-time environment, it's important to consider whether it can efficiently infer an explanation. We believe this to be a plausible goal despite the conventional assumption that abductive systems are slow, and therefore present an investigation of DiscoverHistory's efficiency. We hypothesize that continuous explanation generation using DiscoverHistory will outperform DEG in terms of efficiency, while maintaining a comparable level of correctness. To assess this, we examine the performance of DEG and DiscoverHistory on a series of random runs from scenarios defined in the ASM domain (Section 4.4). Variation across these runs primarily comes from the choices made by the human patrol members, who frequently replan based on a nondeterministic hierarchical task network, and act in a nondeterministic order when executing actions.

We measure correctness with respect to a ground truth explanation generated with knowledge of the team members' actions. Each event and exogenous action (i.e., not performed by the robot) in an inferred explanation is paired with a matching occurrence in the ground truth explanation, if any exists. A generated occurrence that matches no occurrence in the true explanation is a *false positive*. Conversely, an occurrence from the true explanation that matches no occurrence in the true explanation is a *false negative*. An event or action in the generated explanation that matches an event or action from the true explanation is a *true positive*. Two events or actions *match* iff some interpretation of each is identical in all preconditions, effects, and its performer, and each is ordered in the same way with respect to observations and other shared occurrences. A match is *partial* if some variables in the generated action or event must be bound to achieve equality.

Based on these definitions, we measure correctness using a modified version of precision and recall. Under this definition, each true positive resulting from a partial match is discounted by the ratio of (1) the number of variable substitutions necessary to achieve equality to (2) the number of variables in the original action or event model. We call this the *match ratio*, and define a *true positive ratio* that sums this for all matches. The true positive ratio is similar to the partial precision and recall scoring used in (Meadows et al., 2013). Using this definition, our correctness metrics are:

$$\text{partial precision} = \frac{\text{true positive ratio}}{\#\text{ true positives} + \#\text{ false positives}}$$

$$\text{partial recall} = \frac{\text{true positive ratio}}{\#\text{ true positives} + \#\text{ false negatives}}$$

We define efficiency as the number of seconds required to perform explanation on the test machine, which is a virtual machine using 4 Xeon X5650 CPUs and 24GB of physical memory. Each iteration was allocated 4GB of process space and 1 CPU.

6.3 Results

A typical state in the ASM domain is described by ~400 literals; 3 external agents (i.e., squad members) were present in the scenario, and by the end of a run, DiscoverHistory's explanation typically included more than 100 actions and 400 events. We ran each experimental condition 10 times with the same initial state in the ASM domain, with a different random seed causing distinct behaviors. We used parameter values of EVENT_COST = 6 and ASSUMPTION_COST = 10 in

our experiments, which for the ASM domain results in similar ranges for the three metrics of age, assumption cost, and event load. DEG is time and memory-intensive at some XMAX values and achieves lower precision and recall at others. With an infinite value for XMAX, DEG would necessarily achieve a higher precision and recall than DiscoverHistory due to its exhaustive strategy; however, testing has shown that high memory usage at this level inevitably causes failure. Therefore, we instead report on the following conditions:

- **baseline-10**, using DH-agent and DEG with XMAX set to 10
- **baseline-30**, using DH-agent and DEG with XMAX set to 30
- **DiscoverHistory**, using DH-agent and the revised DiscoverHistory explanation generator

Figure 7 plots the average partial precision and partial recall of the most accurate explanation found by each agent as explanations change over time. The x-axis of each plot describes the number of observations explained so far. DiscoverHistory, represented by the solid blue line, achieves similar partial precision and recall to baseline-30. However, baseline-10's correctness decreases over time (relative to the other conditions) due to its unreliability. In some runs, baseline-10 failed to maintain a plausible explanation; its random sample is too small to provide sufficient generality to always find a consistent explanation. Subsequently, it repeatedly reported previously found explanations, which decreased in correctness as more occurrences accumulated.

These graphs also show that no experimental condition achieves a partial recall value far above 0.8; this is because some of the true events and actions in the world occur away from the robot, where their effects cannot be directly observed. Although these events might be inferred based on later observations, achieving near-perfect recall is highly unrealistic.

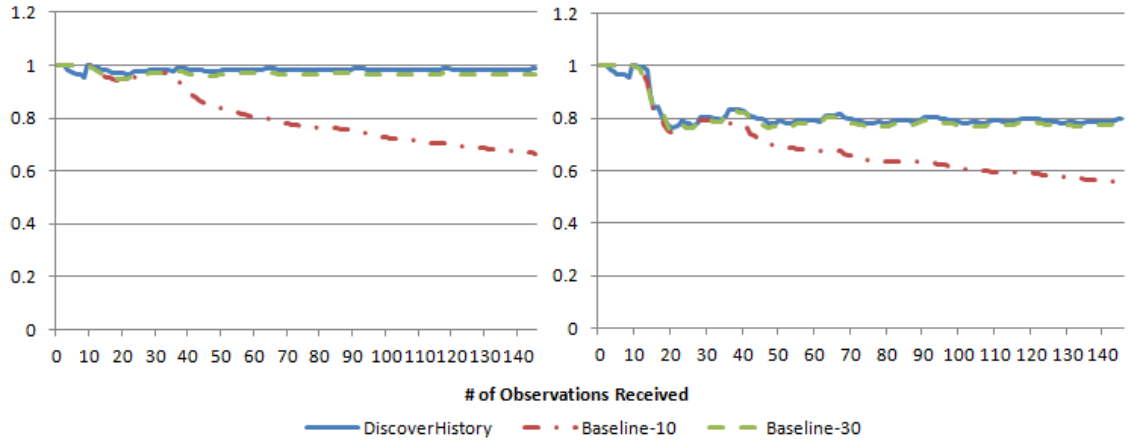

Figure 7: Partial Precision (left) and Partial Recall (right) vs. Observation Count (ASM Domain)

To perform a statistical comparison, we compared the ranges of the 95% confidence interval for mean precision and recall between conditions at each point on the curve. In each comparison, DiscoverHistory eventually outperforms, maintaining a lower bound for mean precision or recall greater than the upper bound of the other conditions for all later observations. For precision and baseline-30, that occurs at 36 observations; for precision and baseline-10, 12 observations; for recall and baseline-30, 63 observations; for recall and baseline-10, 42 observations.

Comparing the efficiency of these conditions (Table 1) highlights DiscoverHistory's major advantage. The differences shown are highly significant, with $p < .001$. While a large enough body of unambiguous explanations can maintain reasonable correctness, it is highly inefficient. In contrast, DiscoverHistory's intelligent search techniques reduce explanation time to a relatively short interval. However, even when maintaining relatively few explanations, DEG is too slow for realistic use. The average interval between novel observations in the ASM domain is 45 seconds. The baseline-10 condition would consume nearly all of that time, leaving no time for replanning and other activities. The baseline-30 condition takes even longer, meaning that several novel observations would be received while the agent was considering a previous observation.

Table 1. Efficiency results for the ASM Domain

Experimental Condition	**Ave. Time Spent Generating Explanations** (minutes)	**Ave. Time Spent per Observation** (seconds)	**Ave. Simulated Time Between Novel Observations**
Baseline-10	94.0*	45.2	45.5
Baseline-30	425.6 (> 7 hours)	176.1 (~3 min.)	
DiscoverHistory	5.4	2.2	

* Some runs stopped explaining early due to inability to maintain plausible explanations

In summation, our investigation has shown the effectiveness of DiscoverHistory in a somewhat complex domain. We have verified out hypothesis that DiscoverHistory is capable of outperforming a deductive explanation generator, while also maintaining a high level of correctness. We believe that the reason for higher performance here is the high number of actions, which increases the branching factor of a deductive search relative to an abductive search, which generally has a higher cost per search node to perform inferences.

7. Conclusions

We introduced a revised version of DiscoverHistory that can efficiently explain actions which occurred in the execution context we used in our study. Efficiency results indicate it may be fast enough for some real-world environments, and its correctness is competitive with other approaches. However, its performance is to some degree dependent on a heuristic function that requires more investigation. The space of possible metrics is not yet well explored and the impact of the weights in different domains may be important. In particular, we would like to conduct an examination of performance tuning and metric combination strategies, as well as look into further metrics.

We are currently integrating this version of DiscoverHistory into an architecture for a future ASM domain robot that acts on explanations of notable events using a goal reasoning process. In particular, we are interested in investigations that use explanation generation as a precursor to plan recognition and goal selection. In future work we will also examine the generality of DiscoverHistory.

Acknowledgements

Thanks to OSD ASD (R&E) for sponsoring this research. Thanks also to the reviewers for their helpful comments.

References

Bridewell, W., & Langley, P. (2011). A computational account of everyday abductive inference. *Proceedings of the Thirty-Third Annual Meeting of the Cognitive Science Society* (pp. 2289-2294). Boston, MA: Cognitive Science Society.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated planning: Theory & practice*. Cambridge, MA: Elsevier.

Grastien, A., Haslum, P., & Thiébaux, S. (2011). Exhaustive diagnosis of discrete event systems through exploration of the hypothesis space. *Proceedings of the Twenty-Second International Workshop on Principles of Diagnosis* (pp. 60-67). Murnau, Germany.

Gspandl, S., Pill, I., Reip, M., Steinbauer, G., & Ferrein, A. (2011). Belief management for high-level robot programs. *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence* (pp. 900-905). Barcelona, Spain: AAAI Press.

Josephson, J.R., & Josephson, S.G. (1996). *Abductive inference: Computation, philosophy, technology*. New York: Cambridge University Press.

Kautz, H.A., & Allen, J.F. (1986). Generalized plan recognition. In *Proceedings of the Fifth National Conference on Artificial Intelligence*. Philadelphia, PA: Morgan Kaufmann.

Leake, D. (1995). Toward goal-driven integration of explanation and action. In A. Ram & D.B. Leake (Eds.) *Goal-Driven Learning.* Cambridge, MA: MIT Press.

Meadows, B., Langley, P., & Emery, M. (2013). Seeing beyond shadows: Incremental abductive reasoning for plan understanding. In *Plan, Activity, and Intent Recognition: Papers from the AAAI Workshop (W9)*. Bellevue, WA: AAAI Press.

Molineaux, M., & Aha, D.W. (2014). Learning unknown event models. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. Quebec City, CA: AAAI Press.

Molineaux, M., Kuter, U., & Klenk, M. (2012). DiscoverHistory: Understanding the past in planning and execution. *Proceedings of the Eleventh International Conference on Autonomous Agents and Multi-Agent Systems* (pp. 989-996). Valencia, Spain: IFAAMAS.

Molineaux, M., Klenk, M., & Aha, D.W. (2010). Planning in Dynamic Environments: Extending HTNs with Nonlinear Continuous Effects. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. Atlanta, GA: AAAI Press.

Ram, A. (1993). Indexing, Elaboration and Refinement: Incremental Learning of Explanatory Cases. *Machine Learning* 10(3). 201-248.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., & Teneketzis, D. (1995). Diagnosability of discrete-event systems. In *IEEE Transactions on Automatic Control*.

Sohrabi, S., Baier, J., & McIlraith, S. (2010). Diagnosis as planning revisited. *Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning* (pp. 26-36). Toronto (Ontario), Canada: AAAI Press.