Learning Models of Unknown Events

Matthew Molineaux MATTHEW.MOLINEAUX@KNEXUSRESEARCH.COM
Knexus Research Corporation, 9120 Beachway Lane, Springfield, VA 22153

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

Abstract

Agents with incomplete models of their environment are likely to be surprised by it. For agents in immense environments that defy complete modeling, this represents an opportunity to learn. We investigate approaches for situated agents to detect surprise, discriminate among different forms of surprise, and ultimately hypothesize new models for the unknown events that surprised them. We instantiate these approaches in a new goal reasoning agent, FOOLMETWICE, and investigate how that agent performs in a simulated environment. In this case study, we found that FOOLMETWICE learn models that substantially improve its performance.

1. Introduction

In most work on planning and reasoning, the world is assumed to be reasonably well-behaved, changing according to a known model (e.g., a policy, a fixed set of rules). In contrast to most other work, we relax the assumption that the agent has a complete and correct environment model. Thus, the environment can surprise our agent, meaning that an event can occur for which the agent lacks knowledge to predict or immediately recognize it.

For example, surprises can occur due to incomplete knowledge of events and their locations. In the fictional *Princess Bride* (Goldman, 1973), the main characters entered a fire swamp with three types of threats (i.e., flame spurts, lightning sand, and rodents of unusual size) for which they had no prior model. They learned models of each type of threat after encountering it, allowing them to predict and prevent each threat type. Surprising realistic events can likewise occur while an agent monitors an environment's changing dynamics: consider an autonomous underwater vehicle that detects an unexpected underwater oil plume for which it has no model. A typical default response might be to immediately surface (requiring hours) when surprised by a novel occurrence and report to an operator. However, if the vehicle first learned a model of the spreading plume, it could react to the projected effects, perhaps by identifying the plume's source.

Surprises are interesting because they occur frequently in real-world environments and cause failures in real-world robots. The ability to respond autonomously to failures would allow such robots to act for longer periods without oversight. While some surprises can be avoided by increased knowledge engineering, it's often impractical due to high environment variance, or events unknown to the knowledge engineers. Therefore, we instead focus on the task of learning from surprises. In this paper, we use FOIL (Quinlan, 1990) to learn environment models and demonstrate its utility to control a simulated mobile robot.

In this paper, we introduce an approach for learning models of unknown events in response to surprises as part of a goal reasoning agent. Surprise detection and response are critical to goal reasoning (Klenk, Molineaux, & Aha, 2013), which includes the study of agents that dynamically modify what goals they pursue in response to unexpected situations. Our approach to unknown event learning is developed as an agent named FOOLMETWICE, an extension to the Autonomous Response to Unexpected Events (ARTUE) agent. These agents implement the Goal-Driven Autonomy (GDA) model of goal reasoning (Molineaux, Klenk, & Aha, 2010a), which entails monitoring the environment for surprises, explaining the cause of surprises, and resolving them through dynamic modification of goals. We begin by discussing related work in Section 2. We review the GDA model in Section 3 and present a formal description of explanations and surprises. In Section 4, we review GDA's implementation in ARTUE and its extensions in the novel agent FOOLMETWICE, which extends ARTUE with the capability to learn event models using FOIL. Section 5 then describes its empirical evaluation. Our results support our research hypothesis, which states that by learning event models, FOOLMETWICE can outperform ablations that cannot learn these models as measured by the time required to perform navigation tasks. Finally, we conclude in Section 7.

2. Related Work

This paper extends our work on deriving explanations for surprises as detected by a GDA agent. Molineaux, Aha, and Kuter (2011) introduced DISCOVERHISTORY, an algorithm for discovering an explanation given a series of observations; it outputs an event history and a set of assumptions about the initial state. Rather than employing a set of enumerated assumptions, DISCOVERHISTORY enumerates which predicates are observable (when true). Assumptions can be made about the initial state value of any literal that is not observable. We showed that, given knowledge of event models, an agent that uses DISCOVERHISTORY could improve its prediction of future states. Molineaux, Kuter, and Klenk (2012) later described an extension of DISCOVERHISTORY that can increase an agent's accuracy for generating state expectations in the context of replanning tasks. They also reported that, for one task, it significantly increased its goal achievement rate versus an ablation that does not perform explanation. Our current work addresses further the question of explanation in the absence of a complete model. Recent related work on abductive diagnosis includes that by Sohrabi, Baier, and McIlraith (2010) concerning the diagnosis of discrete-event systems using planning algorithms. This does not consider the challenges of further reasoning and autonomous execution based on diagnoses, as does ours. Gspandl et al. (2011) also conduct history-based diagnosis in an execution environment. Our work differs in that we focus on diagnosis of exogenous events rather than failed actions, and we compute diagnoses iteratively when new problems are found.

Several other investigations have addressed the task of explaining surprises in the current state. Early work on SWALE (Leake, 1991) used surprises to drive a story understanding process that conducted goal-based explanation to achieve understanding goals. Weber, Mateas, and Jhala's (2012) GDA agent learns explanations from expert demonstrations when a surprise is detected, where an explanation is a prediction of a future state obtained from executing an adversary's actions. Hiatt, Khemlani, and Trafton (2012) introduce and instantiate a framework for *Explanatory Reasoning* to identify and explain surprises, where explanations are generated using a cognitively-plausible simulation process. Ramisinghe and Shen (2008) describe the

Surprise-Based Learning process, in which an agent learns and refines its action models. These models are represented by qualitative rules that can be used to predict state changes and identify when surprises occur (i.e., when the rules' predictions fail). Nguyen and Leong (2009) introduce the *Surprise Triggered Adaptive and Reactive* (STAR) framework to dynamically learn and revise an agent's models of its opponents' strategies in non-stationary game environments (i.e., opponent strategies can change over time). After an accumulated surprise threshold is exceeded, a STAR agent generates hypotheses to predict an opponent's strategy, and will adopt a strategy if its prediction accuracy exceeds a different threshold. While these studies, like our own, concern agents that can recognize and (in most cases) respond to surprises, our contribution here is unique: we describe an algorithm for learning (and applying) environment models of unknown *exogenous* events (i.e., rather than action models for the agent or other agents).

A substantial amount of research has focused on learning environment models such as action policies, opponent models, or task decomposition methods for planning (e.g., Zhuo et al., 2009). However, a variety of techniques have also been used to learn other types of models, and under different assumptions. For example, Bridewell et al. (2008) describe how *Inductive Process Modeling* techniques can be used to learn process models from time series data, and predict the trajectories of observable variables. Pang and Coghill (2010) instead survey methods for *Qualitative Differential Equation* (QDE) *Model Learning* (QML), which have been used to study real-world non-interactive dynamic systems. *Reverse Plan Monitoring* (Chernova, Crawford, & Veloso, 2005) can be used to automatically perform sensor calibration tasks by learning observation models during plan execution. In contrast to these prior investigations, we consider the problem of obtaining models for use by a deliberative agent in subsequent prediction and planning in an execution environment.

In model-free reinforcement learning (RL) (Sutton & Barto, 1998), agents are responsible for acquiring environment models for their immediate use. Our work diverges significantly from the RL framework in that it is goal-oriented rather than reward-driven, which allows frequent goal change without requiring significant re-learning of a policy.

3. Models

Goal Reasoning is a model for online planning and execution in autonomous agents (Klenk et al., 2013). As in our prior work, we focus on the *Goal-Driven Autonomy* (GDA) model of goal reasoning, which separates the planning process from procedures for goal formulation and management. Section 3.1 summarizes a minor extension of this model, Section 3.2 describes our formalism for plausible explanations, and Section 3.3 describes how to use these to explain anomalies. We describe GDA agent implementations in Section 4.

3.1 Modeling Goal-Driven Autonomy

Figure 1 illustrates how GDA extends Nau's (2007) model of online planning. The GDA model expands and details the *Controller*, which interacts with a *Planner* and a *State Transition System* Σ (an execution environment).

System Σ is a tuple (S,A,E,O,γ,ω) with states S, actions A, exogenous events E, observations O, state transition function γ: $S\times(A\cup E)\rightarrow S$, and observation function ω: $S\rightarrow O$. The transition function γ describes how an action's execution (or an event's occurrence) transforms the

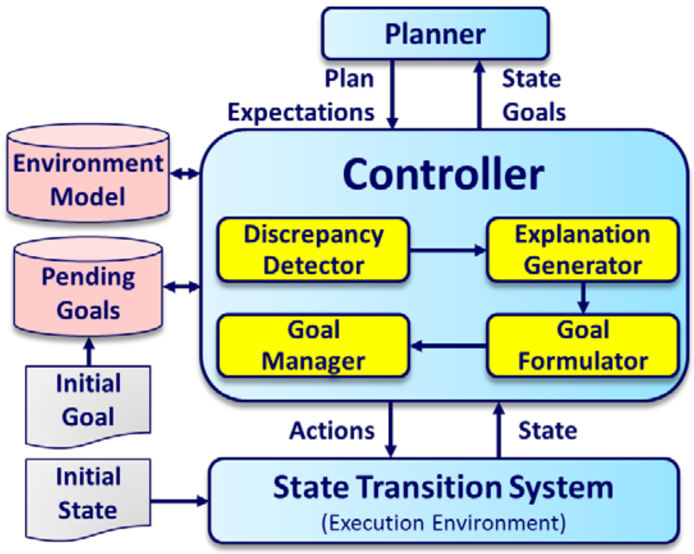

Figure 1: Conceptual Model for Goal-Driven Autonomy (GDA)

environment from one state to another. The observation function ω describes what observation an agent will receive in a given state. We will use the term "event" to refer to an exogenous event.

The Planner receives as input a planning problem (M_Σ, s_c, g_c), where M_Σ is a model of Σ, s_c is the current state, and g_c is the active goal, from the set of all possible goals G, that can be satisfied by some set of states $S_g \subseteq S$. The Planner outputs (1) a plan p_c, which is a sequence of actions $A_c=[a_{c+1},\ldots,a_{c+n}]$, and (2) a corresponding sequence of expectations $X_c=[x_{c+1},\ldots x_{c+n}]$, where each $x_i \in X_c$ is the state expected to result after executing a_i in A_c, and $x_{c+n} \in g_c$.

The Controller takes as input initial state s_0, initial goal g_0, and M_Σ, and sends them to the Planner to generate plan p_0 and expectations X_0. The Controller forwards p_0's actions to Σ for execution and processes the resulting observations, where Σ also processes exogenous events.

During plan execution, the Controller performs the following knowledge-intensive GDA tasks:

Discrepancy detection: GDA detects unexpected events by comparing the observation $\mathbf{obs}_c \in O$ (received after action a_c is executed) with expectation $x_c \in X$. If one or more discrepancies $d \in D$ (i.e., the set of possible discrepancies) are found, then explanation generation is performed.

Explanation generation: Given the history of past actions $[a_1,\ldots,a_n]$ and observations $[\mathbf{obs}_0,\ldots,\mathbf{obs}_c]$ and a discrepancy $d \in D$, this task hypothesizes one or more explanations of d's cause $\chi \in \mathbb{X}$, the set of possible explanations.

Goal formulation: Resolving a discrepancy may warrant a change in the current goal(s). If so, this task formulates a goal $g \in G$ in response to d, given also χ and $\mathbf{obs}_c$.

Goal management: The formulation of a new goal may warrant its immediate focus and/or edits to the set of Pending Goals $G_P \subseteq G$. Given G_P and new goal $g \in G$, this task may update G_P and then select the next goal $g' \in G_P$ to be given to the Planner. (It is possible that $g=g'$.)

GDA makes no commitments to specific types of algorithms for the highlighted tasks (e.g., goal management may involve comprehensive goal transformations (Cox & Veloso, 1998)), and treats the Planner as a black box.

3.2 Modeling Explanations

In this subsection, we present a detailed model of explanations useful for describing the explanation generation task. While this is not the only model of explanation compatible with explanation generation in GDA, it facilitates understanding of the DISCOVERHISTORY algorithm (Molineaux et al., 2012), which we use here, and, possibly, future algorithms as well. Under this model, explanations express statements about the occurrence and temporal ordering of a past sequence of observations, actions, and exogenous events.

This model represents exogenous environmental effects as deterministic exogenous events that must occur whenever their preconditions are met. In contrast to other representations for exogenous effects, such as contingent action effects (Peot & Smith, 1992; Pryor & Collins, 1996) or external actions (Sohrabi et al., 2010), this has three advantages. First, prediction or diagnosis of the exact time of an event's occurrence is possible, which reduces the set of potential explanations for a given sequence of observations. Second, exogenous events are a factored representation that allows effects to combine without an explosion in representation size. Finally, the multiplication of possible states is caused only by hidden information and never by a nondeterministic choice, which simplifies diagnosis.

3.2.1 Events

We assume several standard definitions from classical planning (Ghallab, Nau, & Traverso, 2004) for our model. Let $\mathcal{P}$ be the finite set of all propositions describing a planning environment, where a **state** assigns a value to each $p \in \mathcal{P}$. A planning environment is *partially* observable if an agent α has access to the environment only through **observations** that do not cover the complete state. Let $\mathcal{P}_{obs} \subset \mathcal{P}$ be the set of all propositions that α will observe, where an observation associates a truth value with each $p \in \mathcal{P}_{obs}$. Let $\mathcal{P}_{hidden} \subseteq \mathcal{P}$ be a set of *hidden* propositions that α cannot observe (e.g., the exact location of a robot that does not have a GPS contact).

An **event model** is syntactically identical to a classical planning operator, comprising a tuple (**`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 $\texttt{effects}^{-}$ and $\texttt{effects}^{+}$ to denote the negative and positive literals in **`effects`**, respectively. An **event** is a ground instance of an event model. We assume that an event always occurs immediately when all of its preconditions are met in the state. After each action, any events it triggers occur, followed by events they trigger, etc. When no more events occur, the agent receives a new observation.

3.2.2 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** $\mathcal{T}=\{t_0, t_1, t_2, \cdots, t_n\}$ and an **ordering relation** between two such points, denoted as $t_1 \prec t_2$, where $t_1, t_2 \in \mathcal{T}$.

Three types of occurrences exist. An **observation occurrence** is a pair $(\texttt{obs}, t)$, 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`** as a function such that $\texttt{occ}(o) \mapsto t$; 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 $\texttt{obs}_0; a_1; \texttt{obs}_1; a_2; \cdots; a_k; \texttt{obs}_{k+1}$. An agent's **explanation** of a state given an execution history is a tuple $\chi = (C, R)$ such that C is a finite set of occurrences that includes each $\mathbf{obs}_i$ for $i = 0, \cdots, k-1$ and each action a_j for $j = 1, \cdots, k$ for some number k. C also includes 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 $\mathbf{occ}(o_i) \prec \mathbf{occ}(o_j)$ such that $o_i, o_j \in C$. As a shorthand, we will sometimes write $o_i \prec o_j$ if and only if $\mathbf{occ}(o_i) \prec \mathbf{occ}(o_j)$.

We use the relations $\mathbf{knownbefore}(p, o)$ and $\mathbf{knownafter}(p, o)$ 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, $\mathbf{knownbefore}(p, o)$ is true iff $p \in \mathbf{preconds}(o)$. Similarly, $\mathbf{knownafter}(p, o)$ is true iff $p \in \mathbf{effects}(o)$. If o is an observation occurrence and $p \in \mathbf{obs}$, then both $\mathbf{knownbefore}(p, o)$ and $\mathbf{knownafter}(p, o)$ are true, and otherwise are false.

An occurrence o is **relevant** to a proposition p if the following holds:

$$\mathbf{relevant}(p, o) \equiv \begin{array}{c} \mathbf{knownafter}(p, o) \vee \mathbf{knownafter}(\neg p, o) \vee \\ \mathbf{knownbefore}(p, o) \vee \mathbf{knownbefore}(\neg p, o). \end{array}$$

We also use the predicates $\mathbf{prior}(o, p)$ and $\mathbf{next}(o, p)$ to refer to the prior and next occurrence relevant to a proposition p, where:

$$\mathbf{prior}(o, p) = \{o' \mid \mathbf{relevant}(p, o') \wedge \nexists o'' \ s.t. \ \mathbf{relevant}(p, o'') \wedge o' \prec o'' \prec o\}.$$
$$\mathbf{next}(o, p) = \{o' \mid \mathbf{relevant}(p, o') \wedge \nexists o'' \ s.t. \ \mathbf{relevant}(p, o'') \wedge o \prec o'' \prec o'\}.$$

3.2.3 Plausible Explanations

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. $p \in \mathbf{preconds}(e)$
2. $\mathbf{knownafter}(p, o)$
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 $\mathbf{knownafter}(\neg p, o)$, $\mathbf{knownbefore}(p, o')$, 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 $\boldsymbol{o}, \boldsymbol{o}' \in \boldsymbol{C}$ and $\mathbf{occ}(\boldsymbol{o}) = \mathbf{occ}(\boldsymbol{o}')$, there may be no conflicts before or after. That is, for all $\boldsymbol{p}$, $\mathbf{knownafter}(\boldsymbol{p}, \boldsymbol{o}) \Rightarrow \neg\mathbf{knownafter}(\neg\boldsymbol{p}, \boldsymbol{o}')$, and $\mathbf{knownbefore}(\boldsymbol{p}, \boldsymbol{o}) \Rightarrow \neg\mathbf{knownbefore}(\neg\boldsymbol{p}, \boldsymbol{o}')$.
4. If $\mathbf{preconds}(e)$ of an event e are all satisfied at an occurrence point t, e is in χ at t.

3.3 Modeling Surprise

We now give a precise definition of surprise as it affects various agents, in order to conscribe our task. Informally, we will say that surprise occurs when an observation contradicts an agent's *expectations*. In some cases, the observations also contradict an agent's *model of the environment.*

It follows from this that an agent which neither generates expectations nor models the environment, such as a random or a greedy agent, cannot be surprised. However, disparate agents such as those that instantiate a cognitive architecture or a reinforcement learning agent, can be surprised. Recognizing when a surprise is caused by an environment model contradiction is necessary to correctly detect and model unknown events.

3.3.1 Surprise as Contradiction of Expectations

Formally, we denote the *a priori* expectations of a logical agent α about the state of its environment at time t, before making an observation, as $\mathtt{expectations}(\alpha, t)$, and its observations at time t as $\mathtt{observations}(\alpha, t)$. Furthermore, if it has a model (or background theory) of its environment that relates expectations to observations, then we shall denote that theory as β. In simple cases, expectations and observations may be contradictory assertions about the state, and β may be empty. Given this, we define the condition of an agent being *surprised by a contradiction to its expectations* as follows:

$$\mathtt{surprise}_{\mathtt{x}}(\alpha, t) \equiv \mathtt{expectations}(\alpha, t) \cup \mathtt{observations}(\alpha, t) \cup \beta \vDash \bot. \quad (1)$$

In terms of a GDA agent, we can describe $\mathtt{expectations}(\alpha, t)$ as X_t and $\mathtt{observations}(\alpha, t)$ as $\mathtt{obs}_t$. While the GDA framework describes no semantics of logical entailment, the comparison process that takes place in discrepancy detection is sometimes equivalent to an entailment test. In particular, ARTUE's discrepancy detection process (Molineaux et al., 2010) detects discrepancies precisely when it is surprised under this definition; every time ARTUE detects a discrepancy, it is surprised by a contradiction to its expectations.

3.3.2 Surprise as Contradiction of an Environment Model

In many agents, all surprises result from a contradiction of the environment model. This is because the agent's expectations are a function of only prior observations and the environment model itself; since prior observations (by definition) cannot change, only the model can be wrong. This holds in particular for many agents that reason with uncertainty, such as those based on Partially Observable Markov Decision Processes. Because these agents' beliefs are so all-encompassing, they are rarely contradicted and cannot be surprised unless the model itself is wrong. On the opposite extreme, many agents assume that uncertainty is entirely absent. Their environment models do not accommodate external change, and therefore every (frequent) surprise contradicts their environment models.

However, in some cases expectations are a function of *assumptions* about the state. We define assumptions as properties of the environment that the agent reasons about despite not being able to observe them. For example, after inferring a model of fire spouts, Westley might assume, in the absence of information, that there is no fire spout behind a tree in front of him, even though he cannot yet observe the location. In algorithms that infer expectations based on assumptions, surprises often result from faulty assumptions rather than a faulty model, and contradiction of the model therefore has a special status. We define a set of possible assumptions Θ, and a function $\mathtt{derivedexpectations}(\alpha, \theta, t)$ that yields the expectations derived from the set of assumptions $\theta \subset \Theta$ taken by the agent as true. We define the condition of an agent being *surprised by a contradiction to its model* as follows:

$$\mathtt{surprise}_{\mathtt{m}}(\alpha, t) \equiv \forall \theta \subset \Theta \colon [\mathtt{derivedexpectations}(\alpha, \theta, t) \cup \mathtt{observations}(\alpha, t) \cup \beta \vDash \bot] \ (2)$$

From this, we can derive the fact that

$$[\exists\theta\subset\Theta\colon \mathbf{expectations}(\alpha,t) = \mathbf{derivedexpectations}(\alpha,\theta,t)] \\ \vDash \left(\mathbf{surprise_m}(\alpha,t) \vDash \mathbf{surprise_x}(\alpha,t)\right). \qquad (3)$$

This second type of surprise (Equation 2) requires that the expectations derived from all possible sets of assumptions be inconsistent with the observations. In this case, we say that β, the model itself, contradicts the observations. As derived in Equation 3, a model contradiction surprise will always result in an expectations surprise, if the agent's expectations at time t are derived from a set of possible assumptions.

In our model of explanations, the definition of a consistent explanation is based on logical entailment of observations from some set of assumptions and the model. Therefore, if a plausible explanation exists, an agent's surprise is not due to a contradiction with its environment model. Therefore, the occasions when a GDA agent using this explanation formalism is surprised due to a model contradiction (i.e., the set of all t such that $\mathbf{surprise_m}(\alpha,t)$) must be a subset of those occasions when a discrepancy is detected and no consistent explanation is found. Below, we describe an agent that uses this method as a means of identifying when model contradictions occur. By using this procedure to identify model contradictions, we avoid the computational complexity of testing every possible set of assumptions, instead incurring only the complexity of the search for consistent explanations.

3.3.3 Surprise example

If Westley has a model of a fire spout, he may still be surprised by one; if a fire spout exists at a location X, and Westley has not observed it, and assumes there is no fire spout at location X, then his expectations do not predict that a flame will spurt at location X. Once this spurt occurs, he is surprised; this surprise contradicts his expectations, but not his model. If Westley then adopts the assumption that a fire spout exists at location X, his expectations change and the contradiction disappears.

In contrast, without a model of fire spouts, Westley will be surprised by the flame spurt even with the assumption that a flame spout exists at location X, because his model fails to predict that the flame spurt occurs. In this case, Westley's expectations are contradicted as well as his model.

4. Learning Event Models

We perform our investigation of learning from surprise by creating FOOLMETWICE, an agent that extends ARTUE (Section 4.1) (Molineaux et al., 2010a) with the ability to learn models of unknown events whose observations caused model contradictions. Our process for learning these models has three steps: (1) recognizing unknown events (Section 4.2), (2) generalizing event preconditions (Section 4.3), and (3) hypothesizing an event model (Section 4.4).

4.1 ARTUE

ARTUE performs the four GDA tasks as follows: (1) *discrepancy detection* is performed by checking for element-wise contradictions between its observations and expectations, (2) *explanation generation* is performed by searching for consistent explanations using DISCOVERHISTORY (Molineaux et al., 2012), (3) *goal formulation* uses a rule-based system to generate new goals with associated priorities, and (4) *goal management* enacts the goal with the highest current priority. ARTUE uses a version of the hierarchical network (HTN) planner SHOP2 (Nau et al., 2003) to generate plans. To predict future events, Molineaux, Klenk, and Aha (2010b) extended SHOP2 to reason about planning models that include events in the PDDL+ representation. To work with an HTN planner, ARTUE uses a pre-defined mapping from each possible goal to an HTN task that accomplishes it.

4.2 Recognizing Unknown Events

As described in Section 3.3.2, in FOOLMETWICE a surprise due to model contradiction can occur only when a discrepancy is detected and no consistent explanation can be found. In our current work, we assume that (1) a model contradiction *has* occurred each time no consistent explanation can be found and (2) the surprise that triggered discrepancy detection was caused by some unknown event e. An explanation that contains all correct events other than unknown events must be inconsistent with regard to the effects of each unknown event e. However, this event need not be the proximate cause; e may have instead triggered another event or event sequence that was directly responsible for the contradictory observation. For this reason, the unknown event may have occurred in advance of the surprise. To find an explanation that is correct with respect to all known events, FOOLMETWICE searches for a **minimally inconsistent explanation** that is more plausible than any other inconsistent explanation that can be described based on the current model and observations. This inconsistent explanation does not fix the model contradiction, but does help to pinpoint the unknown events that caused it.

DISCOVERHISTORY searches through the space of possible explanations by iteratively refining an existing inconsistent explanation (Molineaux et al., 2012). These refinements can include event removal, event addition, and hypothesis of different initial conditions. At each successive iteration, a refinement can cause additional inconsistencies. Search ends when the entire explanation is consistent or a search depth bound is reached.

To search for minimally inconsistent explanations, we extend DISCOVERHISTORY with an additional refinement that ignores a single inconsistency by creating an *inconsistency patch*. Given an inconsistency (p, o, o'), it refines the explanation by adding a patch occurrence $o_h = (e_h, t')$. Here, e_h is a **patch event** that satisfies $\mathbf{effects}^{+}(e_u) = \{p\}$ and $\mathbf{precond}(e_u) = \{\neg p\}$, and t' is an occurrence point such that $\mathbf{occ}(o) \prec t' \prec \mathbf{occ}(o')$. This operation will not change any other literal, and thus will never cause an inconsistency. An explanation containing a patch event is not consistent and all patched inconsistencies are considered for purposes of determining whether the explanation is minimally inconsistent.

The extended DISCOVERHISTORY used by FOOLMETWICE conducts a breadth-first search, stopping only when all inconsistencies are resolved or patched. We define the *minimally inconsistent explanation* as the inconsistent explanation with the lowest cost, where cost is a measure of the explanation’s plausibility. In particular, we define the cost for patching an inconsistency to be much greater (10) than other refinements (1). Since we define lower cost explanations as more plausible than higher cost explanations, this cost differential reflects that a known and modeled event is a much more likely cause than an unknown event. As a result, the search process heavily favors explanations with fewer patches. If all correct events are described by the explanation, unknown events correspond directly to the inconsistency patches; the unknown effects are the same as those of the patch events.

The predominant computational cost of DISCOVERHISTORY and the extended version presented here is the breadth-first search for explanations. We bound this depth to a constant factor to ensure manageable execution times; a depth bound of 20 was used in this paper, resulting in a worst-case complexity of $O(n^{20})$, where n is the branching factor (i.e., the number of possible refinements available at each node). Typical values range between 2 and 10. Each explanation search conducted in these experiments took less than 60 seconds to perform.

4.3 Generalizing Event Preconditions

Once we have determined when unknown events occur using a minimally inconsistent explanation, we must generalize over the states that trigger that events to create a model of its preconditions. We chose FOIL (Quinlan, 1990) for our preliminary investigation on learning event models because it is well-known, operates on relational data, and generates logical hypotheses (called *concept definitions*). FOIL takes as input a set of positive and negative examples of a *target relation* (i.e., ground literals that are true and false in the domain), as well as an *extensional definition* of other relations (i.e., a set of all true ground literals for each relation). To find a target definition, FOIL recursively adds non-ground literals to a Horn clause until some positive examples but no negative examples are covered. FOIL greedily chooses a literal that produces the most information gain to add to the Horn clause at each step of this search. When a rule is discovered, the positive examples covered by that rule are removed and the process repeats until all positive examples are covered. The resulting clauses form a set of rules from which the concept can be inferred. To prefer shorter concept definitions, we employ an iterative deepening search through the FOIL target definition space.

For the purpose of event learning, the target concept we wish to learn is the state that triggers an unknown event. We create a relation `event-occurs` to represent this concept; the arguments of this literal include the name and arguments of the inconsistent literal *p* we believe to be caused by the event, as well as the occurrence point at which a patch was created, and a unique symbol referring to the scenario during which it occurred. For example, a minimally inconsistent explanation for the Princess Bride Fire Swamp environment might include a patch occurrence (e_u, t') where $\textbf{effects}(e_u) = \{(\texttt{sinking–rapidly Buttercup})\}$. Here, the target concept is the event that causes the effect literal. The positive example literal here would be $(\texttt{event–occurs sinking–rapidly Buttercup}\ t'\ \texttt{PrincessBride})$.

Along with these examples, we must provide an extensional definition of the environment that includes all ground literals explained by the current minimally inconsistent explanation χ. Like the `event-occurs` predicate, we extend each predicate in the domain to include an occurrence point at which it is known to be true and a unique id for the scenario in which it occurred, so that literals describing the same state can be grouped. For example, the extensional definition for the Fire Swamp could include the following literals:

```
[(friend-of Westley Buttercup $t_0$ PrincessBride)
 (friend-of Buttercup Westley $t_0$ PrincessBride)
 (location Buttercup house $t_0$ PrincessBride)
 (location Westley stable $t_0$ PrincessBride)
 (friend-of Westley Buttercup $t'$ PrincessBride)
 (friend-of Buttercup Westley $t'$ PrincessBride)
```

```
    (location Buttercup under-a-tree t′ PrincessBride)
    (location Westley on-the-path t′ PrincessBride)
    (sandy-location under-a-tree t′ PrincessBride)].
```

4.4 Hypothesizing an Event Model

After FOOLMETWICE completes a scenario, it searches for a minimally inconsistent explanation that uses none of the previously learned event models. These models are kept out of this explanation, which is input to the learning process, to avoid compounding errors between multiple learning iterations. All literals consistent with this explanation are added to a persistent extensional definition for their respective relations, as well as `event-occurs` literals corresponding to all inconsistency patches. Then, FOIL is called once for each ground literal covered by an inconsistency patch during the most recent scenario.

Each Horn clause output by FOIL is used to construct a new learned event model that has as its condition the Horn clause output, and as its effects, the single ground literal believed to be found to be inconsistent. While all original relation terms are ground in the target concept to learn, the occurrence point and scenario id are free variables. For example, the target concept for the Princess Bride Fire Swamp example would be `(event-occurs sinking-rapidly Buttercup ?t ?scn)`. If FOIL then output as its result the Horn clause `(event-occurs sinking-rapidly Buttercup ?t ?scn) ← (location Buttercup ?x ?t ?scn) (sandy-location ?x ?t ?scn)`, FOOLMETWICE would construct the event:

```
  (:event new-event51
   :conditions ((location Buttercup ?x) (sandy-location ?x))
   :effect (sinking-rapidly Buttercup)
  )
```

FOOLMETWICE adds formulated events to its environment model, which can be used for planning and explanation during future scenarios. With each scenario, the extensional definition knowledge is increased, which allows FOOLMETWICE to induce more accurate models, if necessary, after experiencing additional scenarios. Thus, models improve over time.

5. Experiment

While the learning task is to construct accurate event models, multiple models may accurately predict the same phenomena. Therefore, we evaluate FOOLMETWICE based on its capability to construct better plans with the learned models than without.

5.1 Environment and Hypothesis

For this study, we use a simple deterministic environment called *MudWorld*, which consists of a discrete navigational grid on which a simulated robot can move in the four cardinal directions. The robot is aware of its location and destination, and its only obstacle is mud. Each location can be muddy or not muddy, and the robot can see the mud when it enters an adjacent grid location. If the robot enters the mud, its movement speed is halved until it leaves. The robot's plan cost function attempts to minimize traversal execution time, so spending time in mud will decrease its performance. However, the initial model given to our robot does not describe this decrease in

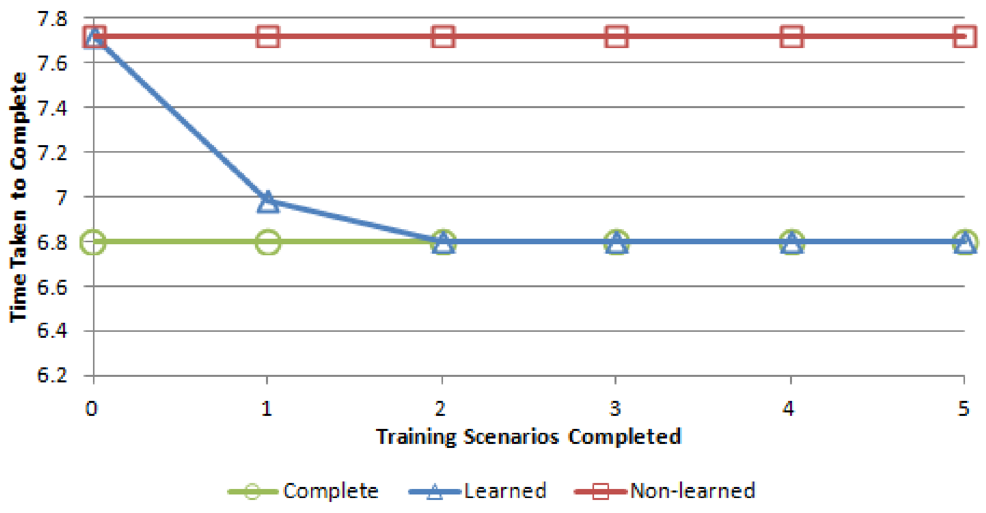

Figure 2: Time required by FOOLMETWICE to complete a scenario.

speed. It can observe its current speed, and it will therefore be surprised when it changes due to the mud. We hypothesize that, by learning, the robot can improve its performance, spending less time achieving its navigation goals than it would otherwise.

5.2 Experiment Description

We randomly generated 50 training and 25 test scenarios in MudWorld, where each scenario consists of a 6x6 grid with random start and destination locations, and a 40% chance of each grid location being muddy. Start and destination locations were constrained so that all routes between start and destination locations contain at least 4 steps, irrespective of mud. We conducted 10 replications. In each replication, we measured its performance on each of the 25 test scenarios before and after learning on each of 5 successive training scenarios (i.e., each of the 50 training scenarios was used once in our experiment).

5.3 Results

Figure 2 shows the results of our experimental evaluation. We depict the results of testing FOOLMETWICE in blue, and for comparison show results achieved on the same test scenarios for a non-learning version with a complete hand-engineered model (in green) and without (in red). The vertical axis depicts the simulated time required to complete the test scenarios, where lower numbers are better (faster completion times). The horizontal axis depicts the number of training scenarios provided. Each point on the red (square markers) and green (circle markers) curves is an average of performance on the 25 testing scenarios. Each point on the blue (triangle markers) curve is an average of performance on the 25 testing scenarios across all 10 replications of the experiment.

In our tests, FOOLMETWICE was always able to achieve the same maximal performance as an agent with a complete model after only two training scenarios. After even one scenario, we can state with high statistical confidence ($p < .001$) that average performance is improved over the prior model.

We expect that similar results could be obtained for similar domains in which unknown events are deterministic and based only on predicate literals. Our results do not currently generalize to nondeterministic events, willed actions, or events dependent on values of function literals. We discuss some future research topics in Section 6.

6. Conclusions

We described an initial investigation into the problem of learning from surprises in the context of Goal-Driven Autonomy. We provided a novel definition of surprise that distinguishes types of surprise (i.e., contradiction of expectations versus contradiction of the environment model) that has not been previously recognized. We described a novel agent, FOOLMETWICE, which uses a new technique for identifying contradictions present in a model based on surprise and explanation generation, and a method for using relational learning to update an environment model in such a context. Finally, we conducted an initial evaluation of FOOLMETWICE in an execution context. This evaluation showed that it is possible to learn a better environment model rapidly under some conditions.

FOOLMETWICE's mechanism for detecting unknown events is not infallible; while we have so far assumed that an expectation’s surprise which cannot be explained is the result of a model contradiction, it is possible that an existing explanation simply was not found, perhaps due to computational constraints. In such cases, FOOLMETWICE will incorrectly attempt to learn a new model to explain the contradiction. In other cases, unknown events do not cause a model contradiction, because an incorrect explanation can be found for a surprise. These false positives and false negatives are an important area for future investigation.

In addition to improving detection of unknown events, future work will focus on demonstrating the performance of FOOLMETWICE in domains with greater complexity. In particular, research into *opportunistic* domains, where surprises provide affordances rather than represent obstacles, is an important next step. In addition, after our agent reaches an acceptable level of performance at learning unknown event models for these more complex domains, we will investigate the problem of learning process models that represent continuous change, as well as models of the actions of other agents and their motivations.

We also intend to apply the algorithms described here to the problem of *active transfer learning*, in which an agent acting in a similar domain to one it understands quickly acquires environment models in that domain with minimal expert intervention. FoolMeTwice can theoretically perform transfers between similar domains by treating the environment model of a source domain as an incomplete model of its new domain. However, additional research into integrating expert feedback and removing prior incorrect models are necessary to fulfill this promise.

Acknowledgements

Thanks to OSD ASD (R&E) for sponsoring this research, and to the anonymous reviewers for their recommendations. The views and opinions contained in this paper are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of NRL or OSD.

References

Bridewell, W., Langley, P., Todorovski, L., & Džeroski, S. (2008). Inductive process modeling. *Machine learning, 71*(1), 1-32.

Chernova, S., Crawford, E., & Veloso, M. (2005). Acquiring observation models through reverse plan monitoring. *Proceedings of the Twelfth Portuguese Conference on Artificial Intelligence* (pp. 410-421). Covilhã, Portugal: Springer.

Cox, M.T., & Veloso, M.M. (1998). Goal transformations in continuous planning. In M. desJardins (Ed.), *Proceedings of the AAAI Fall Symposium on Distributed Continual Planning* (pp. 23-30). Menlo Park, CA: AAAI/MIT Press.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated planning: Theory & practice*. San Mateo, CA: Morgan Kaufmann.

Goldman, W. (1973). *The princess bride.* San Diego, CA: Harcourt Brace.

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

Hiatt, L.M., Khemlani, S.S., & Trafton, J.G. (2012). An explanatory reasoning framework for embodied agents. *Biologically Inspired Cognitive Architectures, 1*, 23-31.

Klenk, M., Molineaux, M., & Aha, D.W. (2013). Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence*, *29*(2), 187-206.

Leake, D. B. (1991). Goal-based explanation evaluation. *Cognitive Science*, *15*, 509–545.

Molineaux, M., Aha, D.W., & Kuter, U. (2011). Learning event models that explain anomalies. In T. Roth-Berghofer, N. Tintarev, & D.B. Leake (Eds.) *Explanation-Aware Computing: Papers from the IJCAI Workshop*. Barcelona, Spain.

Molineaux, M., Klenk, M., & Aha, D.W. (2010a). Goal-driven autonomy in a Navy strategy simulation. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. Atlanta, GA: AAAI Press.

Molineaux, M., Klenk, M., & Aha, D.W. (2010b). 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.

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 Multiagent Systems* (Volume 2) (pp. 989-996). Valencia, Spain: International Foundation for Autonomous Agents and Multiagent Systems.

Nau, D.S. (2007). Current trends in automated planning. *AI Magazine*, *28*(4), 43–58.

Nau, D., Au, T.-C., Ilghami, O, Kuter, U, Murdock, J.W., Wu, D., & Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, *20*, 379-404.

Nguyen, T.H.D., & Leong, T.Y. (2009). A surprise triggered adaptive and reactive (STAR) framework for online adaptation in non-stationary environments. In *Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference*. Stanford, CA: AAAI Press.

Pang, W., & Coghill, G.M. (2010). Learning qualitative differential equation models: A survey of algorithms and applications. *Knowledge Engineering Review*, *25*(1),~~:~~ 69-107.

Peot, M., & Smith, D.E. (1992). Conditional nonlinear planning. *Proceedings of the First International Conference on Artificial Intelligence Planning Systems* (pp. 189-197). College Park, MD.

Pryor, L., & Collins, G. (1996). Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research*, *4*, 287-339.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine learning*, *5*(3), 239-266.

Ramisinghe, N., & Shen, W.-M. (2008). Surprised-based learning for developmental robotics. *Proceedings of the ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems* (pp. 65-70). Edinburgh, Scotland: IEEE Press.

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

Sutton, R.S., & Barto, A.G. (1998). *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press.

Weber, B., Mateas, M., & Jhala, A. (2012). Learning from demonstration for goal-driven autonomy. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence.* Toronto, Canada: AAAI Press.

Zhuo, H.H., Hu, D.H., Hogg, C., Yang, Q., & Muñoz-Avila, H. (2009). Learning HTN method preconditions and action models from partial observations. *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence* (pp. 1804-1810). Pasadena, CA: AAAI Press.