Learning Unknown Event Models

Matthew Molineaux[1] and David W. Aha[2]

[1]Knexus Research Corporation, Springfield, VA; [2]Naval Research Laboratory, Code 5514; Washington, DC

[1]matthew.molineaux@knexusresearch.com; [2]david.aha@nrl.navy.mil

Abstract

Agents with incomplete environment models are likely to be surprised, and this represents an opportunity to learn. We investigate approaches for situated agents to detect surprises, discriminate among different forms of surprise, and hypothesize new models for the unknown events that surprised them. We instantiate these approaches in a new goal reasoning agent (named FOOLMETWICE), investigate its performance in simulation studies, and report that it produces plans with significantly reduced execution cost in comparison to not learning models for surprising events.

1. Introduction

Most studies on planning and reasoning assume the availability of a complete and correct domain model, which describes how the environment changes. We relax the completeness assumption; events occur in our environments that the agent cannot predict or recognize because its model does not describe them. 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 from experience, which they used to predict and defeat future examples. Surprising realistic events can also occur while an agent monitors an environment's changing dynamics: consider an autonomous underwater vehicle (AUV) that detects an unexpected underwater oil plume for which it has no model. A default response might be to immediately surface (requiring hours) and report it. However, if the AUV first learns a model of the spreading plume, it could react to the projected effects (e.g., by identifying the plume's source).

Surprises occur frequently in real-world environments and cause failures in robots. Autonomous response to failures would allow them to act for longer periods without oversight. Some surprises can be avoided by increased knowledge engineering, but it is often impractical due to high environment variance, or unknown events. Therefore, we instead focus on the task of learning from surprises. In particular, we employ a variant of FOIL (Quinlan, 1990) to learn models of unknown exogenous events in partially observable, deterministic environments and demonstrate how they can be used by a *goal reasoning* agent (Klenk, Molineaux, & Aha, 2013). Surprise detection and response are critical to these agents, which dynamically determine what goals to pursue in response to notable situations. We implement this learning method in FOOLMETWICE, an extension of ARTUE (Molineaux, Klenk, & Aha, 2010a). Both agents implement a Goal-Driven Autonomy (GDA) model for goal reasoning, which entails planning, monitoring a partially observable environment for surprises, explaining their cause, and resolving them through dynamic selection of goals.

We discuss related work in §2, review the GDA model in §3, and present a formal description of explanations. In §4, we review GDA's implementation in ARTUE and FOOLMETWICE. §5 describes its empirical evaluation; our results support our hypothesis that, by learning event models, FOOLMETWICE can outperform ablations that do not learn (as measured by the time required to perform navigation tasks). Finally, we conclude in §6.

2. Related Work

We focus on deriving explanations for surprises as detected by a GDA agent. This extends work on DISCOVERHISTORY (Molineaux, Aha, and Kuter, 2011), which discovers an explanation given a series of observations. It outputs an event history and a set of assumptions about the initial state. DISCOVERHISTORY enumerates which predicates are observable (when true) and assumes initial state values for unobservable literals (as needed). We earlier showed that, given knowledge of event models, DISCOVERHISTORY can improve an agent's prediction of future states. In this paper, we instead focus on *learning* these event models.

Related work on learning environment models focuses on models of an agent's action. Pasula et al. (2007) describe how an agent can learn models of a world with

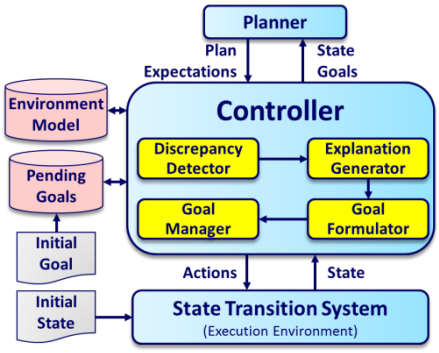

Figure 1: Conceptual Model for Goal-Driven Autonomy (GDA)

realistic physics, modelled stochastically and with noise, but fully observable. Zhuo et al. (2010) describe LAMP, which infers a sophisticated deterministic action model representation including conditionals and universal quantifiers, from executed plan traces, to reduce software engineering effort. They report it was accurate with ablated information. Mourao et al. (2012) employ a two-step learning process to boost the accuracy of models learned from noisy plan traces. Our work differs from these prior studies in its focus on exogenous events.

Several studies address the task of explaining surprises in the current state. SWALE (Leake, 1991) uses surprises to guide story understanding and goal-based explanation to achieve understanding goals. Weber, Mateas, and Jhala's (2012) GDA agent learns explanations from expert demonstrations when it detects a surprise, where an explanation predicts a future state obtained by executing an adversary's expected actions. Hiatt, Khemlani, and Trafton (2012) describe an *Explanatory Reasoning* framework that identifies and explains surprises, where explanations are generated using a cognitively-plausible simulation process. In Ranasinghe and Shen's (2008) *Surprise-Based Learning* process an agent learns and refines its action models, which can be used to predict state changes and identify when surprises occur. Nguyen and Leong's (2009) *Surprise Triggered Adaptive and Reactive* (STAR) framework dynamically learns models of its opponents' strategies in response to surprises. In contrast, we describe an algorithm for learning (and applying) environment models of unknown *exogenous* events.

Several methods exist for learning environment models such as action policies, opponent models, or task decomposition methods for planning (e.g., Zhuo *et al.*, 2009). Techniques also exist for learning other types of models under different assumptions. *Inductive Process Modeling* (Bridewell et al., 2008) can learn process models from time series data, and predict the trajectories of observable variables. *Qualitative Differential Equation Model Learning* (Pang & Coghill, 2010) methods can be used to study real-world non-interactive dynamic systems. *Reverse Plan Monitoring* (Chernova, Crawford, & Veloso, 2005) can automatically perform sensor calibration tasks by learning observation models during plan execution. In contrast, we address 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 (Sutton & Barto, 1998), agents learn environment models. Our work differs in that it is goal-oriented rather than reward-driven, and thus it allows frequent goal change without requiring substantial re-learning of a policy.

3. Models

Goal Reasoning is a model for online planning and execution in autonomous agents (Klenk *et al.*, 2013). We focus on the GDA model of goal reasoning; it separates planning from processes for discrepancy detection, explanation, goal formulation, and management. In §3.1 we summarize a GDA extension and in §3.2 describe a formalism for explanations. We describe GDA agent implementations in §4.

3.1 Modeling Goal-Driven Autonomy

GDA (Figure 1) extends Nau's (2007) model of online planning. It details the *Controller*, which interacts with a *Planner* and a *State Transition System* Σ, which is a tuple $(S, A, E, O, \gamma, \omega)$ with states S, actions A, exogenous events E, observations O, state transition function $\gamma : S \times (A \cup E) \rightarrow S$, and observation function $\omega : S \rightarrow O$ (i.e., it describes what observation an agent will receive in a given state). We will use "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 $g_c \subseteq S$ is the active goal, from the set of possible goals G. The Planner outputs (1) a plan π_c, which is an action sequence $A_c = [a_{c+1}, \cdots, a_{c+n}]$ and (2) a sequence of **expectations** $X_c = [x_{c+1}, \cdots, x_{c+n}]$, where $x_i \in X_C$ is the state expected after executing a_i in A_C, and $x_{c+n} \in g_c$.

The Controller takes as input initial state s_o, initial goal g_0, and M_Σ, and sends them to the Planner to generate plan π_0 and expectations X_0. The Controller forwards p_0's actions to Σ for execution and processes the resulting observations. 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 observation $\text{obs}_c \in O$ (received after action a_c is executed) with expectation $x_c \in X$. Mismatches are collected in the set of discrepancies D. If this set is non-empty, explanation generation is performed.

Explanation generation: Given the history of past actions $[a_1, \cdots, a_n]$ and observations $[obs_0, \cdots, obs_c]$ and a discrepancy $d \in D$, this task hypothesizes one or more possible explanations of d's cause $\chi \in \mathbb{X}$.

Most GDA models also perform *goal formulation* and *goal management* tasks. However, they are not central to our focus in this paper, and we do not discuss them here.

3.2 Modeling Explanations

FOOLMETWICE generates explanations to (1) infer that specific, known events have occurred, and (2) recognize when unknown events have occurred (through explanation failure). We briefly present a model of the explanations as used by DISCOVERHISTORY, where explanations express statements about the occurrence and temporal ordering of a sequence of observations, actions, and exogenous events.

Events

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 $p \in$ 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 $P_{obs} \subset$ P be the set of all propositions that α will observe when true. Let $P_{hidden} \subseteq$ 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).

An **event model** is a tuple (name; preconds; effects) denoting the event's name and sets of preconditions and effects (sets of literals). An **event** is a ground instance of an event model. Events occur immediately when all their preconditions are met. After each action, any events it triggers occur, followed by events they trigger, etc.

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 two such points, denoted as $t_1 \prec t_2$, where $t_1, t_2 \in$ T .

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** is a finite sequence of observations and actions $\text{obs}_0; a_1; \text{obs}_1; a_2; \cdots; a_k; \text{obs}_{k+1}$. An **explanation** of a state given an execution history is a tuple $\chi = (C, R)$, where C 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. R is a partial ordering over a subset of C, described by ordering relations $\text{occ}(o_i) \prec \text{occ}(o_j)$ where $o_i, o_j \in C$, which we will sometimes write as $o_i \prec o_j$.

We use $\text{knownbefore}(p, o)$ and $\text{knownafter}(p, o)$ to refer to the value of literal $p \in$ P immediately before or after occurrence $o \in C$. For action and event occurrences, $\text{knownbefore}(p, o)$ holds iff $p \in \text{preconds}(o)$ and $\text{knownafter}(p, o)$ holds iff $p \in \text{effects}(o)$. If o is an observation occurrence and $p \in \text{obs}$, then $\text{knownbefore}(p, o)$ and $\text{knownafter}(p, o)$ hold, and otherwise are false.

Occurrence o is **relevant** to a proposition p iff:

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

Plausible Explanations

The **proximate cause** of an event occurrence (e, t) is an occurrence o that satisfies three conditions with respect to some proposition p: $p \in \text{preconds}(e)$, $\text{knownafter}(p, o)$, and there is no other occurrence $o' \; s.t. \; o \prec o' \prec (e, t)$. Event occurrences must have at least one proximate cause, so by condition 3, they must occur immediately after their preconditions are satisfied. An **inconsistency** is a tuple (p, o, o') where o and o' are two occurrences in χ such that $\text{knownafter}(\neg p, o)$, $\text{knownbefore}(p, o')$, and there is no other occurrence o'' such that $o \prec o'' \prec o' \in R$ and p is relevant to o''. Discrepancies can be interpreted as inconsistencies between the most recent observation and a prior occurrence in the current explanation.

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

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

Projected States

A projected state proj(t) for occurrence point t is given by:

$$\text{proj}(t) = \left\{ p \middle| \exists o \left(\begin{matrix} \text{knownafter}(p, o) \wedge \\ \nexists o' \; \text{knownafter}(\neg p, o') \wedge \\ \text{occ}(o) \prec \text{occ}(o') \prec t \end{matrix} \right) \right\}.$$

A projected state gives all facts that would be true in the environment, given that all assumptions in χ are correct and χ is consistent.

4. Learning Event Models

We investigate our method for learning explanations in response to surprising exogeneous events by comparing the performance of FOOLMETWICE with ARTUE (Molineaux *et al.*, 2010a). FOOLMETWICE learns these models by detecting unknown events, generalizing event preconditions, and hypothesizing an event model.

4.1 ARTUE

ARTUE performs the four GDA tasks as follows: (1) it detects discrepancies by checking for mismatches between

its observations and expectations, (2) generates explanations by searching for consistent explanations with DISCOVERHISTORY, (3) formulates goals using a rule-based system that associates priorities with goals, and (4) manages goals by enacting a goal with the highest current priority. ARTUE uses a SHOP2 (Nau *et al.*, 2003) variant to generate plans; to predict future events, Molineaux, Klenk, and Aha (2010b) extended SHOP2 to reason about planning models that include events in PDDL+. To work with this variant, ARTUE uses a pre-defined mapping from each possible goal to a task that accomplishes it.

DISCOVERHISTORY searches the space of possible explanations by recursively refining an inconsistent explanation (Molineaux *et al*., 2012). Refinements include event removal, event addition, and hypothesis of different initial conditions. Each recursion may cause new inconsistencies. Search ends when an explanation is consistent or a search depth bound is reached.

4.2 Recognizing Unknown Events

FOOLMETWICE tries to find inaccuracies in environment model M_Σ by attempting to explain all observations received. When a consistent explanation cannot be found, it infers that some **unknown event** e_u happened that is not represented in M_Σ. However, it operates in a partially observable environment; events and their effects are not always immediately observed. Before inferring a model for e_u, it must determine when e_u most likely occurred. It does this by finding a **minimally inconsistent explanation** χ_{mi} that is more plausible than any other such explanation that can be described based on the current model and observations. χ_{mi} does not include any specific unknown event, but does help to pinpoint when it occurred.

To search for minimally inconsistent explanations, we extended DISCOVERHISTORY to ignore an inconsistency by creating an **inconsistency patch**. For an explanation with inconsistency (p, o, o'), it adds a patch occurrence $o_p = (e_p, t')$, where **patch event** e_p satisfies $\text{effects}(e_p) = \{p\}$ and $\text{precond}(e_p) = \{\neg p\}$, and t' is an occurrence point such that $\text{occ}(o) \prec t' \prec \text{occ}(o')$. This will not change any other literal, so it will never *cause* an inconsistency. However, the resulting explanation is always inconsistent because the patched inconsistency is not removed.

The extended DISCOVERHISTORY conducts a breadth-first search, stopping when all inconsistencies are resolved or patched. We define the minimally inconsistent explanation χ_{mi} as the lowest cost explanation, where cost is a measure of plausibility. That is, we set the cost for patching an inconsistency (10) to be much greater than other refinements (1). We define lower cost explanations as more plausible than higher cost explanations; a known and modeled event is considered more plausible than an unknown event, if considered independently from other events. Thus, search favors explanations with fewer patches. If an explanation describes all correct events, unknown events correspond to inconsistency patches; the unknown effects are the same as those of the patch events.

DISCOVERHISTORY's highest computational cost is its breadth-first search for explanations. We bound its depth to a constant (50) to ensure manageable execution times, yielding a worst-case complexity of $O(n^{50})$ for branching factor n, which is the number of possible refinements available per node (typically in $[2,10]$). Each search in our experiments took less than 600 seconds to perform.

4.3 Generalizing Event Preconditions

After determining when unknown events occur, creating a model of their preconditions requires generalizing over the states that trigger them. We adapted FOIL (Quinlan, 1990) to perform this task. Instead of inferring rules from a relational database, our adaptation must infer rules from a set of projected states, each of which contains all facts believed to be true at a specific prior time. States are found by projecting the effects at each occurrence point of a minimally inconsistent explanation χ_{mi}. The extension, FOIL-PS (Projected States), infers conditions by separating negative states, known not to trigger an unknown event e_u, from *bags* of positive states that may trigger it.

FOIL-PS maintains a set of minimally inconsistent explanations $\mathbb{X}$, one for each completed training scenario. To infer events that cause p, where $\text{effects}(e_p) = \{p\}$ for at least one event patch, it finds a set of bags of events that occur during each such e_p, because each of those states may have triggered e_u. Formally, a positive example bag for an inconsistency (p, o, o') is the set of states $\text{peb}(p, o, o') = \{\text{proj}(t) \mid \text{occ}(o) \prec t \prec \text{occ}(o')\}$.

To learn a model for events that cause p, we give FOIL-PS a set of positive example bags corresponding to each inconsistency relating to p in all explanations in $\mathbb{X}$. The negative examples are all the remaining projected states. Thus the positive example bags for some set of training explanations $\mathbb{X}$ and surprising literal p are:

$$\text{peb}(p, \mathbb{X}) = \left\{ \text{peb}(p, o, o') \;\middle|\; \begin{array}{c} (p, o, o') \text{ is an} \\ \text{inconsistency for } \chi \in \mathbb{X} \end{array} \right\}.$$

The corresponding negative examples are:

$$\text{ne}(p, \mathbb{X}) = \left\{ \text{proj}(t) \;\middle|\; \begin{array}{c} \text{proj}(t) \notin \text{peb}(p, o, o') \text{ for any} \\ \text{inconsistency for any } \chi \in \mathbb{X} \end{array} \right\}.$$

To find the triggering conditions for an event causing p, FOIL-PS searches the space of possible clauses that satisfy zero states in ne(p, $\mathbb{X}$) and at least one in each bag from peb(p, $\mathbb{X}$). The initial clause used is $\{\neg p\}$, and each node in the search tree adds one literal from its parent node.

As this search is costly, FOIL-PS does not consider literals that produce negative information gain according to FOIL's definition. Also, we restrict the number of zero information-gain literals to be added to a clause: FOIL-PS defines the search cost of a clause as the number of zero information gain additions made in the nodes leading to it, and conducts an iterative deepening search to find only

clauses with the minimal search cost. The first clause returned by each level of the search is one that covers the maximum number of positive example bags of any clause found within the search cost. Search repeats with the same cost until sufficient clauses are found to cover all positive example bags. If sufficient clauses cannot be found, search is repeated with an incremented cost.

Some projected states from a Princess Bride explanation are as follows:

proj(t_0) =
[(friend Westley Buttercup) (friend Buttercup Westley) (location Buttercup house) (location Westley stable)]

proj(t') = [(friend Westley Buttercup) (friend Buttercup Westley) (location Buttercup under-tree) (location Westley on-path) (sandy-location under-tree)].

4.4 Modifying the Environment Model

Each clause output by FOIL-PS is used to construct a learned event model whose condition is the clause output, and whose effect is the single ground literal believed to be inconsistent. If FOIL-PS then outputs the clause:

(and (not (sinking-rapidly Buttercup))
(location Buttercup ?loc)
(sandy-location ?loc)),

then FOOLMETWICE would construct the event:

(:event new-event51
:conditions (and (not (sinking-rapidly Buttercup))
(location Buttercup ?loc)
(sandy-location ?loc))
:effect (sinking-rapidly Buttercup))

FOOLMETWICE adds constructed events to its environment model, which can be used for planning and explanation in future scenarios. Ideally, the set of events that are inferred to cause a literal *p* will match the actual events that cause the condition modeled by *p*. However, FOIL-PS will not always initially find a correct set of models, so FOOLMETWICE updates the model periodically, after each scenario is completed.

If the learned event models fail to cover all environment events causing *p*, then *p* may be found to be inconsistent in a future χ_{mi}. When an inconsistent literal is found in the χ_{mi} of the most recent scenario, all previously learned events that cause it are removed from the model and new models are learned from scratch. Conversely, if the learned event models cover situations that do *not* trigger any event causing *p*, an event will be erroneously predicted, likely resulting in an inconsistent explanation. Thus, we designed DISCOVERHISTORY to resolve an inconsistency by *abandoning* a previously learned event model, removing it from the inconsistent explanation and marking it as invalid. This incurs less cost (5) than an inconsistency patch, preventing FOOLMETWICE from adding a second event that cancels the effects of the first, but costs more than other refinements, so that event models will not be abandoned often. After an event model is abandoned, the causes of that model's effect are re-learned.

In each of these situations, peb(*p*, $\mathbb{X}$) or ne(*p*, $\mathbb{X}$) contains a counterexample for the conditions of a prior learned event with incorrect conditions. The new models learned will therefore likely improve over time.

FOOLMETWICE cannot acquire exogenous event models with continuous conditions and effects, and cannot model inequalities or numeric relationships between cause and effect, which we will address as future work. Also, PDDL+ processes, which model continuous change over time, cannot be acquired.

5. Empirical Study

The learning task is to construct accurate event models, but multiple models may accurately predict the same phenomena. Thus, we evaluate FOOLMETWICE for its ability to achieve goals at lower execution cost.

5.1 Environments and Hypothesis

We tested FOOLMETWICE in two new deterministic, partially observable, single actor domains. While we intend to use it in more complex domains (e.g., where events can be triggered by other actor's actions or the environment), these suffice to test our agent. Each domain contains one event that is not part of the agent's model, and is based on a world state that is not directly observed. While no explicit learning goals exist, execution cost in each domain is lower when planning with knowledge of the unknown event.

The first domain, *Satellites*, is based on an IPC 2003 competition domain in which a set of satellites have instruments that can obtain images in many spectra, and goals consist of taking various images. Performance is judged based on the time required to achieve all goals. 1 time unit is used to turn a satellite to a new position, and 10 to repair a satellite lens. The unknown event causes a satellite's lens to break when taking an image of an excessively bright object. The fact that the object is too bright for the camera lens is hidden to the agent, but bright objects cause an observable lens flare during calibration.

The second domain, *MudWorld*, employs a discrete grid on which a simulated robot moves in the four cardinal directions. The robot observes its location and destination, and its only obstacle is mud. Each location can be muddy or not; the robot cannot observe mud directly, but it deterministically receives a related observation when entering a location adjacent to one that is muddy. If it enters mud, its movement speed is halved until it leaves. However, its initial model does not describe this decrease in speed, so it is surprised when its speed decreases.

In both domains, execution cost is based on time taken to achieve a goal. We hypothesize that, after learning models of unknown events, FOOLMETWICE will create plans that require less time.

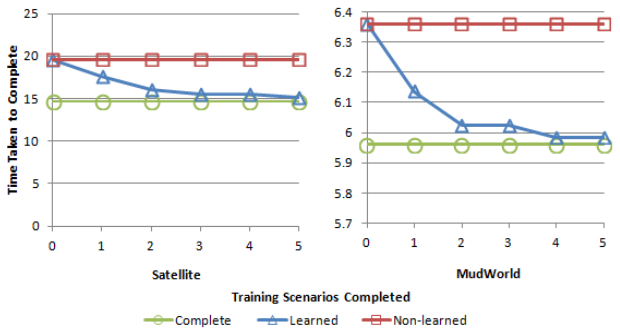

Figure 2: Average Execution Cost of FOOLMETWICE

5.2 Experiment Description

For each domain, we randomly generated 50 training and 25 test scenarios. In Satellites, the initial state of each scenario has 3 satellites with 12 instruments randomly apportioned among them. Each scenario has 5 goals requiring that an image of a random target be obtained in a random spectrum. MudWorld scenarios consist of a 6x6 grid with random start and destination locations, each of which may contain mud with 40% probability. We selected start and destination locations so that all routes between them contain at least 4 steps, irrespective of mud.

In each of 10 replications per domain, we measured FOOLMETWICE's performance on all test scenarios before and after training on 5 scenarios.

5.3 Results

Figure 2 displays the average execution cost incurred in each domain by FOOLMETWICE (blue with triangle markers), an "optimal" version with a complete model (green with circle markers), and a non-learning baseline (red with square markers). The vertical axes depict the simulated time required to complete the test scenarios, while the horizontal axes depict the number of training scenarios provided.

In each domain, FOOLMETWICE achieved a performance gain of more than 90% with respect to the optimal performance within 5 trials. After training on only one scenario in each domain, its average performance is significantly higher than when using the initial environment model ($p < .05$).

Similar results might be obtained for other domains in which unknown events are deterministic and based only on predicate literals. However, our results do not currently generalize to nondeterministic events, willed actions, or events dependent on values of function literals.

Table 3 shows average wall clock time spent during execution and learning per domain, and the number of explanation failures. The number of explanation failures trends downward, and learning time appears to decrease in MudWorld. However, execution time initially increases in Satellites. Review of individual trials indicates that this is caused by learning initial models that are inefficient to compute. We conclude that while explanation and planning clearly improve performance with learning, wall clock time can suffer and is an interesting subject for future study.

Table 3: Additional Performance Metrics for FOOLMETWICE

Learning Trials	Execution Time (Seconds)		Learning Time (Seconds)		Explanation Failures	
	Sat	Mud	Sat	Mud	Sat	Mud
0	43.9	3.5	N/A	N/A	1.8	2.2
1	183.3	3.8	0.16	51.8	0.93	0.78
2	186.5	3.5	0.28	40.5	0.57	0.23
3	171.8	3.5	0.24	2.3	0.41	0.14
4	171.5	3.3	0.18	13.9	0.41	0.03
5	160.1	3.3	0.21	0.7	0.36	0.03

6. Conclusions

We described FOOLMETWICE, a novel GDA agent that uses a new technique to identify unknown events in a model based on surprise and explanation generation, and a relational learning method to update environment models. We described its initial study on the task of learning from surprises, and found that it rapidly learned better environment models (i.e., reasoning with them results in lower execution costs; even inaccurate models may help to improve an agent's plans).

We did not compare FOOLMETWICE against other agents, mainly because we focus on learning models of exogenous events, which other agents do not target. However, it is possible that algorithms which learn the conditional effects of actions (e.g., LAMP (Zhuo *et al.*, 2010) may perform well on the tasks we used in our experiments, which could be expressed using actions with conditional effects rather than exogenous events.

FOOLMETWICE assumes that an explanation failure is due to an unknown event(s); as a result, it may incorrectly infer that its model is incomplete when explanation search fails (e.g., due to computational constraints). Conversely, search may sometimes find an incorrect explanation when ambiguous observations are received, causing it to ignore an opportunity for learning. Reducing these false positives and false negatives is a future research topic.

We will also assess performance on more complex domains, and in particular *opportunistic* domains, where surprises provide affordances rather than represent obstacles. We will also investigate the problem of learning process models that represent continuous change, and models of the actions and motivations of other agents.

Finally, we will apply FOOLMETWICE to *active transfer learning* contexts, in which an agent acting in a similar domain to one it understands may quickly learn in that domain with minimal expert intervention. FOOLMETWICE can theoretically transfer environment models among similar domains by treating a source domain model as an incomplete model of its new domain. This will require further research on integrating expert feedback and removing prior incorrect models.

References

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

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.

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.

Mourao, K., Zettlemoyer, L. S., Petrick, R., & Steedman, M. (2012). Learning strips operators from noisy and incomplete observations. *arXiv preprint arXiv:1210.4889*.

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.

Pasula, H. M., Zettlemoyer, L. S., & Kaelbling, L. P. (2007). Learning Symbolic Models of Stochastic Domains. *J. Artif. Intell. Res.(JAIR), 29*, 309-352.

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

Ranasinghe, 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.

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.

Zhuo, H. H., Yang, Q., Hu, D. H., & Li, L. (2010). Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, *174*(18), 1540-1569.