Advances in Cognitive Systems X (2019) 1-6 Submitted X/2019; published X/2019

Explanation-based Goal Monitors for Autonomous Agents

Zohreh Dannenhauer ZOHREH.DANNENHAUER@KNEXUSRESEARCH.COM
Matthew Molineaux MATTHEW.MOLINEAUX@WRIGHT.EDU
Michael Cox MICHAEL.COX@WRIGHT.EDU
Wright State Research Institue, Dayton, OH USA

Abstract

Goal reasoning agents operating in dynamic, partially observable worlds select between competing objectives which change over time. In this paper, we introduce goal monitors that interleave plan execution, planning and interpretation. These monitors assume that the goal is formulated in response to a inconsistency between the agent's expectations and its observations. We use DiscoverHistory to generate explanations to find possible events that caused these anomalies. In response to the anomalies, new goals are formulated to remove the cause of each anomaly as identified by the explanations. We introduce goal monitors as a way to determine when an agent should switch between its current goal and pending goals. Goal monitors link the causal information of the explanations to both (1) the newly generated goals and (2) monitoring capabilities for identifying if the goals remain relevant. We describe the process of creating goal monitors in the MIDCA cognitive architecture and explain how they lead to goal change among competing goals when anomalies arise. Finally, we demonstrate our system in the Minecraft domain.

1. Introduction

Real world autonomous agents will be expected to manage their behavior across many complex situations and to solve severe problems that arise while pursuing their goals. This work considers agents that are operating in partially observable worlds which are changing due to external events. Events are not always observable to agents and may cause them to fail in achieving their goals and plans. Agents should be able to explain the cause of these changes and adjust their goals and plans to perform competently. Specifically, this work addresses the research problem of how best to monitor the world for those changes that affect the agent's goals.

This paper focuses on *goal-driven autonomous (GDA)* agents [Aha *et al.*, 2010; Cox, 2007; Klenk *et al.*, 2013; Munoz-Avila *et al.*, 2010]. Goal-driven autonomy involves recognizing unexpected or possibly new problems, explaining the causal factors underlying the problems and generating goals to remove the cause of the problems in order to achieve the given task. The GDA agent itself is expected to identify situations in which new goals are to be formulated or current goals changed and abandoned [Cox *et al.*, 2017; Klenk *et al.*, 2013]. Identification of these situations is where plan monitors and goal monitors are needed.

Plan monitors can enable a planner to respond to changes in the world during plan generation without having to restart planning from the beginning. However, there are some changes that affect

the agent's goals which are outside the scope of plan monitors, especially when the agent should consider switching to a pending goal. Hence the need for goal monitors which is the focus of this paper.

If the agent formulates a goal on its own, it should have reasons for doing so. These reasons establish the means for monitoring that the goal is still worth achieving. Explanation-based goal monitors are created from an explanation used to formulate that goal in response to a inconsistency between the agent's expectations and observations.

DiscoverHistory [Molineaux *et al.*, 2012] is a system that abductively reasons about unexpected events that occur during plan execution. When there are discrepancies between an agent's predictions and observations, DiscoverHistory hypothesizes events consistent with the observations that resolve the inconsistency.

Consider the following example in the Minecraft domain. While working, Steve's (our Minecraft agent) health decreases, and Steve finds this anomalous because he is expecting good health (a health-value ≥ 30). Instead he observes a lower health value. Low health is the consequence of some event that has occurred. The agent does not have full knowledge of his surroundings, but he explains this inconsistency by hypothesizing that he was shot by an archer skeleton or activated an arrow trap.

In this paper, we use an explanation of an anomaly to extract the reasons why the agent pursues a goal and generates monitors to observe these reasons. This approach provides the basis to change the goal when the goal is no longer useful in the world. Our contributions are the following:

- A goal monitoring approach that extracts possible reasons why the agent pursues a goal from explanations of a inconsistency and performs monitoring of these reasons to ensure the goal is still useful.
- An algorithm that creates explanation-based goal monitors to observe the causal structures of a formulated goal and change/drop the goal during the execution if the agent's beliefs change.
- An evaluation of an agent with explanation-based goal monitors in the Minecraft domain that outperforms a baseline agent without explanation-based goal monitors.

This paper is organized as follows: In the next section we review the MIDCA cognitive architecture and DiscoverHistory. The following section introduces the concepts, formalism and implementation of explanation-based goal monitors. We then present an ablation study demonstrating the benefits of an agent equipped with explanation-based goal monitors compared to a GDA agent without goal monitors. We follow with results and conclude with a discussion of future work.

2. Definitions and Formalism

We use the standard definitions from classical planning for predicates, atoms, planning operators and actions [Ghallab *et al.*, 2016]. A planning environment is partially observable if an agent only has access to the environment through observations which do not cover the complete state.

An event template is defined the same as a classical planning operator: e=(name(e), precond(e), effects(e)) where *name* is the name of the event, *preconds* and *effects* are the preconditions and

effects of the event respectively. An event occurs when all of its preconditions are met in the true world state. Since actions and events change the current state of the world, events can be triggered at any time.

Occurrence o refers to the occurrence of any observation obs, action a, or event e. An execution history is a finite sequence of observations and actions $\langle obs_0, a_1, obs_1, a_2, ..., a_k, obs_{k+1}\rangle$.

If the agent's expectations for the outcome of its actions are different from the actual observed outcomes in each new state, we say a inconsistency occurs [Dannenhauer and Munoz-Avila, 2015]. An inconsistency is a tuple (p, o, o'), where p is a predicate in the new observation o'. $p \in obs$ but p is not an effect of the previous occurrence o since o is an action a and $\neg p \in$ effects(a).

One way to resolve the inconsistency is to show that some occurrence changed the value of a literal in between the preceding occurrence o and the following occurrence o'. This occurrence must be an event e, such that effects(e) $\models p$. Another way to resolve the inconsistency is hypothesizing an initial value. In this case, a different initial occurrence o may be hypothesized. We use DiscoverHistory to generate these hypotheses.

We adopt the classical planning formalism [Ghallab *et al.*, 2004] where any given goal, $g_i \subseteq s \in S$, is a state or subset of a state that an agent tries to achieve. The agent's goal agenda $\hat{G} = \{g_1, ..., g_c, ..., g_n\}$ contains the current goal g_c and any pending goals it intends to pursue.

We formalize goal monitors as a goal operation [Cox *et al.*, 2017]. An individual goal operation, $\delta : G \rightarrow G$, is a function from one goal expression $g \in G \subset S$ to another $g' \in G \subset S$, where S is the set of all possible states. An operation is formalized as a transformation represented by δ = (head(δ), parameter(δ), pre(δ), res(δ)), where pre(δ) and res(δ) are its preconditions and result. The decisions $\langle \delta^1, \delta^2, ..., \delta^n \rangle$ applied in sequence to arrive at g' by $\delta_n(...\delta_2(\delta_1(g_c))) = g'$.

Table 1 represents the goal monitor operation as a goal transformation, δ^{mo}. The content of the goal does not change in this operation, rather the *is-monitored* attribute changes. This flag on the goal signals to an implementation that a monitor needs to be created. The monitor includes two major components. First the monitor encapsulates environmental conditions whose change signals the need for goal reconsideration. Second the monitor includes a specification of the response (e.g., goal abandonment) if the perception component detects the first condition.

In Table 1, the class hierarchy CL is used to represent a hierarchy of goal predicates r. The monitor operation, like all goal operations, has 4 parts: a name (i.e., head), a parameter (i.e., the input goal), a set of preconditions (here 3 of them), and a result (i.e., res). The hierarchy notation is more important for other operations such as goal change, where the goal may change from r to its immediate parent r, but in monitoring the preconditions assure that the goal predicate is not already monitored, and it results in one that is monitored.

3. Review of the MIDCA Architecture

The *metacognitive integrated dual-cycle architecture (MIDCA)* [Cox *et al.*, 2016; Paisner *et al.*, 2013] is a cognitive architecture that models both cognition and metacognition for intelligent agents. It consists of "action-perception" cycles at both the cognitive level and the metacognitive level (see Figure 1). In general, a cycle performs problem-solving to achieve its goals and tries to comprehend the resulting actions and those of other agents. The problem-solving mechanism of each cycle con-

Table 1. The goal monitor operation as transformation. In the notation of [Bergmann, 2002], **CL** is a class hierarchy having leaves $L_c \subset \mathbf{CL}$ and whose root class C has superclass $\top$, i.e., $C_{superclass} = \top$. Precondition pre_2 of the transformation assures that the goal is not already monitored.

$\delta^{mo}(g_c : G) : G$
head(δ^{mo})= monitor
parameter(δ^{mo})=g_c=$r(obj1, obj2)$
pre$_1$(δ^{mo})=$obj1 \in Objs \wedge obj2 \in Objs$
pre$_2$(δ^{mo}) = $\exists r, r', i | r \in CL \wedge r' \in CL \wedge r_{superclass} =$
$r' \wedge r = (r_{name}, r', (r.A_1, r.A_2, ..., r.A_m)) \wedge 1 \geq i \geq m$
$\wedge A_i$=is-monitored $\wedge \neg r.A_i$
pre$_3$(δ^{mo})=needs-monitoring(s, g_c)
pre(δ^{mo}) = $\{pre_1(\delta^{mo}), pre_2(\delta^{mo}), pre_3(\delta^{mo})\}$
res(δ^{mo}) = if $\forall x | x \in pre(\delta^{mo}) \wedge (s \models x)$ then
p.is-monitored $\leftarrow \top$
return g_c

sists of intention, planning, and action execution processes, whereas the comprehension mechanism consists of perception, interpretation, and goal evaluation.

In problem solving, the *Intend* phase commits to a current goal from those available. The *Plan* phase then generates a sequence of actions. The plan is executed by the *Act* phase to change the world through the effects of the planned actions. The agent will then use these expectations in the next cycle to evaluate the execution of the plan.

Comprehension starts with perception of the world. The *Interpret* phase takes the resulting predicates and the expectations in memory as input. The *Evaluate* phase determines whether the current goal is actually achieved in the world and removes it from memory if true.

The Interpret phase of MIDCA compares expectations from its plans and knowledge about the world to perceived observations. Discrepancies detected by these comparisons enable MIDCA to generate new goals and to change existing goals in a robust manner. The Interpret phase is implemented as a GDA procedure and analyzes the current state to determine which new goals should be pursued. In our scenario, this is the phase that detects an anomaly, forms explanation and formulates new goals in response.

DiscoverHistory is integrated with the Interpret phase in MIDCA to reason about the causes of inconsistencies between observations and expectations that arise during plan execution and generate abductive explanations. Interpret passes the execution history to DiscoverHistory, and DiscoverHistory generates successive explanations by attempting to resolve inconsistencies.

This is often useful to derive a causal structure for why the agent's plan fails. These structures provide the environmental conditions that must persist for the agent's goals to remain valid. Goal monitors that are created in the Interpret phase observe these conditions and provide the response if they change. In this paper, we examine what information these monitors observe, and how these monitors respond when they detect changes in these causal structures.

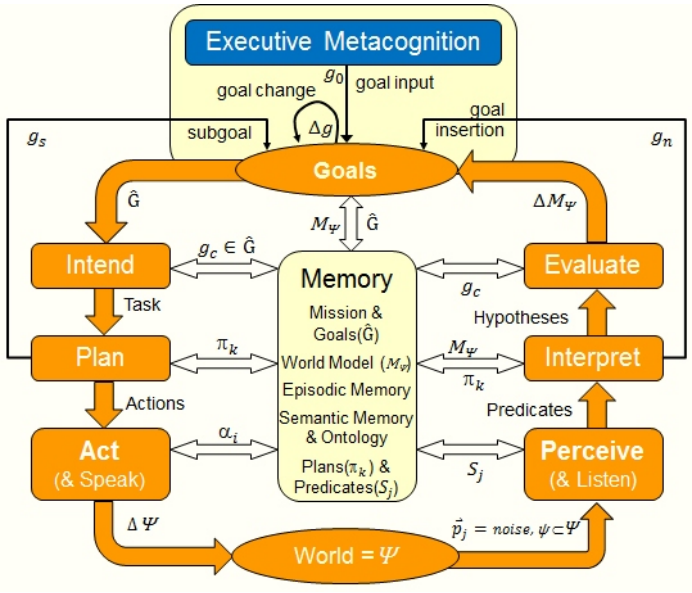

Figure 1. The action-perception cycle in MIDCA. Together Intend, Plan, and Act compose the problem-solving mechanism in the architecture, and Perceive, Interpret, and Evaluation constitute the comprehension mechanism. Explanations are embedded in Interpret. Hypotheses are the explanations that are generated in Interpret.

4. Goal Monitoring

In MIDCA, goals provide focus for the agent's reasoning and represent the desired future state that the agent seeks to achieve. We categorize goal monitors into two types based on how the goal is formulated: *operator-based* and *explanation-based*. Operator-based goal monitors [Cox and Dannenhauer, 2017] consist of a set of goal operators as rules that generate goals when their conditions are satisfied in the world. Formally, a goal operator o, is the tuple (name(o), precond(o), result(o)). The set of literals precond(o) represents the operator's preconditions. They specify what conditions the current state must satisfy in order for o to be applied. Operators exist for various goal types and data-driven context-sensitive rules spawn them given matching run-time observations. Goal monitors observe these conditions to make sure the goal is still valid in the new state. While previous work considered on operator-based goal monitors in a logistics domain setting, this is the first work (of which the authors are aware) on explanation-based goal monitors.

In partially observable domains, the GDA agent might consider multiple hypotheses. If there are two or more possible explanations, each could be equally likely initially. But if relevant new information is observed, the system should consider updating the explanations. A contribution of this paper is handling multiple explanations in situations where more than one hypothesis may be true. The goal monitors are created for all possible hypotheses, and when the agent observes new information, one or more of the goal monitors may fire which in turn will lead to a possible change in the goals the agent is pursuing.

In the Minecraft scenario mentioned earlier, after Steve observes his health is low, he picks one of the assumptions about the world and acts based on it by formulating a goal/plan. He also observes evidence related to other assumptions in case they end up being the cause of the problem. Considering multiple possible hypotheses enables the agent to react to the situation and generate a goal and plan to survive while observing other possible events/goals.

We now describe the execution monitoring framework for an agent equipped with explanation-based goal monitors. First, we describe how the explanation module generates the necessary information for goal monitors. Next, we describe how the monitors use this information to decide if the current goal is still valid in a dynamic world, and which goal the agent should pursue when there are multiple possible goals. Finally, we describe how the framework will respond to changes in the conditions that are being monitored.

4.1 Monitor Trigger Conditions

At each cycle of MIDCA, the execution history $\langle obs_0, a_1, ..., a_k, obs_{k+1} \rangle$ is passed to DiscoverHistory to detect discrepancies. When a new observation is inconsistent with the agent's expectation ($\neg p \in$ effect(a_k) and $p \in obs_{k+1}$), DiscoverHistory tries to resolve the inconsistency. DiscoverHistory deduces the possible worlds that result from different sets of assumptions.

Lets assume that there are n possible explanations, each has m assumed initial values $h = [p'_1, ..., p'_m]$ and one possible event $e_i, i \in [1..n]$. Assumptions across all explanations are added to MIDCA's set of hypotheses $hyps = [h_1...h_n]$.

In our scenario described in the introduction, the appearance of a skeleton leads to a skeleton attack event e_1 that includes effects that match the discrepancies in the observed state (effect$(e_1) \models p$, where p is low health). DiscoverHistory produces an explanation that includes the unobservable facts that would cause the event to occur. MIDCA adopts these facts as a hypothesis.

For example, these are explanations in the earlier scenario:

```
Explanation 1:
(ASSUME-INITIAL-VALUE
  (THING-AT SKELETON ADJ-M) T TIME 1)
(SKELETON-ATTACKED M1-1 ADJ-M 15 20
                   TIME 8)

Explanation 2:
(ASSUME-INITIAL-VALUE
  (IS-TRAP ADJ-M) T TIME 1)
(FALL-IN-TRAP M1-1 ADJ-M 15 20 TIME 8)
```

These two hypothetical facts are competing alternatives. The first explanation states that there is a skeleton in one of the adjacent tiles to the agent, and an event $e =$ `SKELETON-ATTACKED` occurred. Note that `(THING-AT SKELETON ADJ-M)` is an unobservable fact. The second explanation states that there is a trap in one of the adjacent tiles to the agent. MIDCA adds these assumed predicates to the set of hypotheses:

h_1 = (THING-AT SKELETON ADJ-M)
h_2 = (IS-TRAP ADJ-M)

An abstract goal g_a is generated in the Interpret phase in MIDCA in response to a inconsistency. After explaining the inconsistency, the agent formulates goals for each explanation (if any) and adds these goals to the set of g_a's subgoals. Algorithm 1 shows how g_a is decomposed into a set of subgoals $\langle g_1, ..., g_n \rangle$ using the causal structure of each explanation. It takes as input the abstract goal, g_a and the set of hypotheses for a inconsistency, $hyps$. First, the set of hypotheses $hyps$ are sorted based on some metrics like their possibility, danger, etc (line 2). Then, for each hypothesis a goal is formulated and is added to the set of g_a's subgoals (lines 4-5). Next, if Interpret decides to have a goal monitored a goal monitor for each assumed predicate in a hypothesis is created (lines 6-8). The first subgoal is selected as the current goal (line 9).

g_1 = ¬ (THING-AT SKELETON ADJ-M)
g_2 = ¬ (IS-TRAP ADJ-M)

When a goal g_i is generated in response to event e_i, Interpret formulates an explanation-based goal monitor GM for each predicate in the hypothesis h=$[p'_1, ..., p'_m]$. GM is a tuple (p', g_a, g) that consists of the predicate p' from a hypothesis related to event e_i, the abstract goal g_a, and the formulated goal g_i. The hypotheses provide the environmental conditions that must persist for the goal g_i to remain valid. Goal monitors observe these conditions and provide the response if they change. If these conditions change in the state, the goal monitor will fire, and the GDA process for goal management will know to reconsider pursuing the current goal.

m_1= ⟨(THING-AT SKELETON ADJ-M), g_a, g_1⟩
m_2= ⟨(IS-TRAP ADJ-M), g_a, g_2⟩

Algorithm 1 Elaborate the abstract goal and create explanation-based goal monitors.
Input: Abstract goal g_a, set of hypotheses $hyps$.

```
1: procedure ELABORATE-GOAL(g_a, hyps)
2:     sort hyps based on possibility
3:     for h in hyps do
4:         g_i ← formulate goal
5:         g_a.subgoals ← g_a.subgoals ∪ g_i
6:         if g_i.is-monitored then
7:             for p' in h do
8:                 mnts ← mnts ∪ (p', g_a, g_i)
9:     g_c ← g_a.subgoals[1]
```

4.2 Monitor Response

Algorithm 2 shows the firing condition for the explanation-based goal monitors. It takes as input the abstract goal, g_a and the set of goal monitors $mnts$. Goal monitors tie the predicates from hypotheses to goals, and then when a monitor fires the agent considers switching the goal. The first step in monitoring is to check if the hypothesized predicates are in the new observed state.

If a monitored predicate p' is observed, MIDCA adds that to the set of beliefs, s_b. If p' is an effect of event e and goal g, then the current goal should change to g since e is what really happened in the world (lines 3-4).

If a monitored predicate p' is believed to be false ($\neg p \in s_b$) then the corresponding goal should be abandoned (if it is current goal) and removed from the set of subgoals. Then, the algorithm changes the current goal to be another possible subgoal of g_a (lines 5-8). If there are not more subgoals in g_a, it will abandon g_a.

Algorithm 2 Check for fired Goal monitors.
Input: Abstract goal g_a, set of monitors $mnts$.

```
1: procedure FIRE-MONITOR(g_a, mnts)
2:     for (p', g_a, g) in mnts do
3:         if if p' ∈ s_b then
4:             g_c ← g
5:         if ¬p' ∈ s_b then
6:             g_a.subgoals ← g_a.subgoals − g
7:             if g_c = g then
8:                 g_c ← g_a.subgoals[1]
```

5. Empirical Evaluation

We claim that in partially observable domains with unexpected events an agent with goal monitors will outperform an agent without goal monitors. To evaluate this hypothesis, we conducted tests with MIDCA on a partially observable Minecraft-based domain. The MIDCA agents operate in planning domains using a PDDL domain definition [Fox and Long, 2003] representing actions, events, and predicates. For planning, we use the MetricFF planner [Hoffmann, 2003].

In the Minecraft domain, various unobservable events occur that our agent does not have sufficient knowledge to predict. To evaluate the impact of goal monitors, we compare our agent *GMAgent* equipped with explanation-based goal monitors against two baselines: a GDA agent with explanation capabilities but no Goal monitors which we call *EXAgent*, and an off-line planning agent named *PLAgent*. In this scenario, GMAgent should outperform PLAgent due to the environments' partial observability. EXAgent generates explanations when an anomaly happens and formulates goals in response to the anomalies, but it does not monitor the current goal or the set of pending goals. We hypothesize that GMAgent will outperform EXAgent in scenarios where unobserved facts affect the state and there are multiple hypotheses, due to goal monitoring, and it will outperform PLAgent where events occur outside of the agent's mission due to goal reasoning. We measure performance as progress towards achieving the initial goal, which we describe next.

5.1 The Minecraft Domain

In the Minecraft game[1], the character "Steve" explores an infinite 3D virtual world while gathering resources and surviving dangers. Minecraft has become a popular evaluation domain for artificial intelligence research, because it has some properties of real-world domains: partially observable, 3-dimensional, infinite state space, and real-time execution [Branavan *et al.*, 2012]. There are different factors that can damage Steve's health like falling in lava, getting shot by a skeleton archer, triggering an arrow trap, and low hunger level. In this evaluation, we only consider two possible events: getting shot by a skeleton archer or triggering an arrow trap. Both decrease the agent's health by two.

The agent is tasked with the initial goal to obtain seven pieces of wood. Wood is obtained by harvesting trees (via a chopping action which requires an axe). The location of trees is known by the agent beforehand.

We now walk through a scenario in which Steve should switch to a pending goal. While Steve is en route to a known tree's location, he observes his health has suddenly become low and his life is in danger. Steve's health begins at a value of thirty. When Steve's health value reaches zero, the agent dies and no more goals can be accomplished. Performance is calculated based on how many trees the agent can harvest. We gave the agent three lives; whenever the agent dies, it loses all his items and it starts form the beginning tile. However, the agent retains credit for trees harvested before death.

The PDDL planning model for this experiment consists of fifteen actions, and two events. We randomly generated fifty problems for each difficulty level. Difficulty levels differ only in the number of random traps placed. More skeletons also make the scenario more difficult in sense there is more danger, but since two agents EXAgent and GMAgent behave the same, we only make the scenario more difficult by varying the number of traps. In each problem, there are seven trees, and five skeletons. Each problem takes place in a 10×10 grid. The number of traps (or difficulty level) was varied from zero to six. In this experiment, if the agent goes to an adjacent tile with a skeleton or trap, he gets shot by an arrow. If there is a skeleton, the agent needs to shoot the archer skeleton using a bow. If the agent triggers a trap, then he needs to disarm the trap with an axe. Note that skeletons and traps are unobservable and the agent can see them only after performing sensing actions and facing the direction of the skeleton/trap.

5.2 Results

In each difficulty level, we performed 50 trials which were held constant across the three agents. Figure 2 shows the average performance of GMAgent, EXAgent and PLAgent. When there is no trap, GMAgent and EXAgent have almost the same performance, because they selected the right explanation (*skeleton-attacked*) in the beginning and generate a goal to deal with the external event.

When there are more traps, the GMAgent has a better overall performance score than the EXAgent. In the middle of plan execution a monitor fires if the GMAgent did not find a skeleton or if it receives a new observation that proves there is an arrow trap nearby. The agent responds by

1. http://www.minecraft.net/

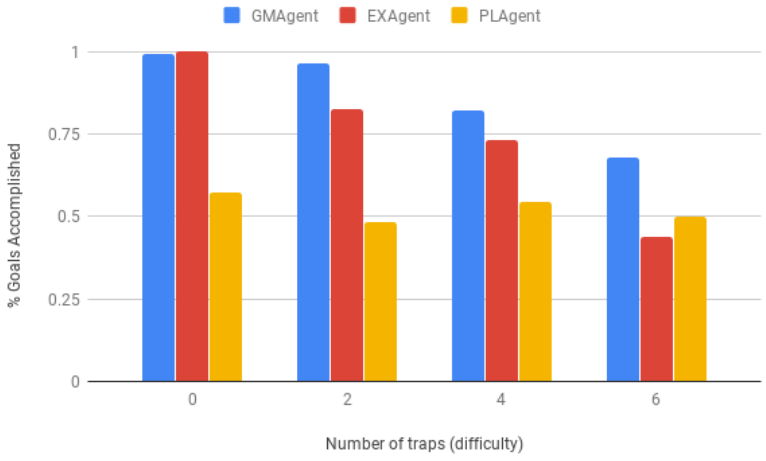

Figure 2. The performance of the GMAgent, EXAgent and PLAgent in the Minecraft domain. Difficulty is based on the number of traps (i=0,2,4,6). The chart shows the average of 50 runs at each difficulty level.

changing the goal to destroy the trap instead of the skeleton. In contrast, the EXAgent persists in pursuit of the goal (kill skeleton), but it dies before it gets to the real problem.

At difficulty level six, the performance of PLAgent is slightly better than EXAgent, because the EXAgent wastes time pursuing the wrong goal, which causes damage. The PLAgent ignores the skeleton and trap and pursues the original goal. This comparison shows that the GMAgent makes better progress towards achieving its goals compared to the EXAgent and PLAgent which supports our claim that using goal monitors improves the agent's performance.

6. Related Work

Much of the prior work in AI regarding interleaving planning and interpretation has been carried out in the form of cognitive architectures. Work on monitoring has mainly been done in individual systems (e.g., HOTRIDE [Ayan *et al.*, 2007]). The novelty of our work is the ability to generate goal monitors for goals which were formulated in response to hypotheses (explanations) about anomalous and surprising events. The effect of goal monitors leads agents to be able to better select among current and pending goals as new information is discovered, including observing new facts and/or verifying facts to not be true (i.e. as is the case when the agent realizes there is no skeleton nearby).

One important aspect of this work is related to cognitive architectures. Systems like ICARUS, PUG, SOAR and MIDCA are inspired from human cognition and are designed to accomplish multiple tasks [Langley and Choi, 2006; Laird, 2012; Laird *et al.*, 2012]. The ICARUS architecture [Langley and Choi, 2006] includes modules for conceptual inference, goal selection, skill execution, means-ends problem solving, and skill learning. The Inference module is a bottom-up processing of perceptual input with rules that fire in certain circumstances.

The PUG extension to the ICARUS cognitive architecture [Langley *et al.*, 2017] is the closest work to our MIDCA architecture in spirit. PUG integrates action execution, planning, and plan monitoring. The monitors check for the preconditions and effects of actions during execution, and if they find any anomaly (e.g. the preconditions of any action are not satisfied) it leads to replanning. They also monitor the preconditions of the rules that generate goals in PUG, and if the conditions change, the goal will be abandoned. Our work is related to their approach in that we address the problem of plan execution and goal monitoring. However, the nature of goal monitors presented in this paper is different since it is concerned with goals generated from dynamic explanatory structures rather than static rules. Additionally in our work, the response to changes is not only a choice between goal abandonment or replanning; our agent may change the goal itself in addition to simply dropping it.

Goal-driven autonomy [Munoz-Avila *et al.*, 2010; Aha *et al.*, 2010] is a kind of goal reasoning that focuses on explanation of discrepancies in order to formulate new goals. GDA agents generate goals as the agent encounters differences between the agentâĂŹs expectations for the outcome of its actions and the actual observed outcomes in each new state [Dannenhauer and Munoz-Avila, 2015]. When such a inconsistency occurs, GDA agents generate a causal explanation for the inconsistency, and generate a new goal based on the causal structure. ARTUE [Klenk *et al.*, 2013] is a domain independent autonomous agent with the capacity to dynamically determine which goals to pursue in unexpected situations. In these systems that agents do not consider explanations for pending goals or monitoring for pending goals, which is the focus of this paper.

Our work is related to the problem of execution monitoring. The problem of detecting failures during execution, diagnosing the reasons for these failures, and recovering from them [De Giacomo *et al.*, 1998; Ambros-Ingerson and Steel, 1988]. Expectation-based monitors detect failures in execution by comparing expectations generated by the planner to observations received during execution. Pablo Mendoza *et al.* (2015) focus on monitoring stochastic expectations and finding subtle anomalies from collections of observations, and adapting the model to improve performance based on experience. Another system like CASPER [Au *et al.*, 2004] refines the plan under construction when the external information changes during planning time. Neither of these systems perform monitoring on explicit goal structures.

7. Conclusion

An autonomous agent not only needs to generate plans to achieve different goals but also detect problems, formulate new goals, monitor the plans and goals, change the goal and replan when its beliefs change in order to operate in complex environments. In this paper, we described an execution monitoring framework for an agent equipped with explanation-based goal monitors. Our system supports monitoring during execution, and it detects unexpected changes in the agentâĂŹs goals.

Our system finds the discrepancies between its expectations and its observations during execution. It then explains these discrepancies using an environment model. It then formulates goals and monitors the justification for goal selection and abandons/changes the goal when justifications are not valid in the environment. In addition, we reported our system's operation on scenarios that

demonstrated its ability to handle the situation when there are multiple hypotheses about the world and switch goals when its beliefs change.

Future work should investigate reasoning over pieces of evidence to calculate the probability for each hypothesis; when new information is received the system should reason about it before deciding about dropping or changing its current goal. By providing a probability for each piece of evidence, we can use Bayesian probability to calculate the probability of each event and let the goal monitor decide if the goal change is needed when it observes new pieces of evidence.

8. Acknowledgments

This material is partially based upon work supported by AFOSR grant FA2386-17-1-4063, by ONR grant N00014-18-1-2009, and by DARPA contract number N6600118C4039.

References

DW Aha, M Klenk, H Munoz-Avila, A Ram, and D Shapiro. Goal-driven autonomy: Notes from the "aaai" workshop, 2010.

Jose A Ambros-Ingerson and Sam Steel. Integrating planning, execution and monitoring. In *AAAI*, volume 88, pages 21–26, 1988.

T Au, Dana Nau, and VS Subrahmanian. Utilizing volatile external information during planning. In *ECAI*, volume 16, page 647. Citeseer, 2004.

N Fazil Ayan, Ugur Kuter, Fusun Yaman, and Robert P Goldman. Hotride: Hierarchical ordered task replanning in dynamic environments. In *Planning and Plan Execution for Real-World Systems–Principles and Practices for Planning in Execution: Papers from the ICAPS Workshop. Providence, RI*, volume 38, 2007.

Ralph Bergmann. *Experience management: foundations, development methodology, and internet-based applications*. Springer-Verlag, 2002.

SRK Branavan, Nate Kushman, Tao Lei, and Regina Barzilay. Learning high-level planning from text. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 126–135. Association for Computational Linguistics, 2012.

Michael T Cox and Zohreh A Dannenhauer. Perceptual goal monitors for cognitive agents in changing environments. *Advances in Cognitive Systems (ACS-17)*, 2017.

Michael T Cox, Zohreh Alavi, Dustin Dannenhauer, Vahid Eyorokon, and Hector Munoz-Avila. MIDCA: A metacognitive, integrated dual-cycle architecture for self-regulated autonomy. In *AAAI*, 2016.

Michael T Cox, Dustin Dannenhauer, and Sravya Kontrakunta. Goal operations for cognitive systems. In *AAAI*, 2017.

Michael T Cox. Perpetual self-aware cognitive agents. *AI magazine*, 28(1):32, 2007.

Dustin Dannenhauer and Hector Munoz-Avila. Raising expectations in gda agents acting in dynamic environments. In *International Joint Conference on Artificial Intelligence (IJCAI-15)*, 2015.

Giuseppe De Giacomo, Raymond Reiter, and Mikhail Soutchanski. Execution monitoring of high-level robot programs. In *PRINCIPLES OF KNOWLEDGE REPRESENTATION AND REASONING-INTERNATIONAL CONFERENCE-*, pages 453–465. MORGAN KAUFMANN PUBLISHERS, 1998.

Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning: theory & practice*. Elsevier, 2004.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning and acting*. Cambridge University Press, 2016.

Jörg Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. 20:291–341, 2003.

Matthew Klenk, Matt Molineaux, and David W Aha. Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence*, 29(2):187–206, 2013.

John E Laird, Keegan R Kinkade, Shiwali Mohan, and Joseph Z Xu. Cognitive robotics using the soar cognitive architecture. *Cognitive Robotics AAAI Technical Report WS-12-06. Accessed*, pages 46–54, 2012.

John E Laird. *The Soar cognitive architecture*. MIT Press, 2012.

Pat Langley and Dongkyu Choi. A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1469. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

Pat Langley, Dongkyu Choi, Mike Barley, Ben Meadows, and Edward P Katz. Generating, executing, and monitoring plans with goal-based utilities in continuous domains. In *Proceedings of the Fifth Annual Conference on Advances in Cognitive Systems*, 2017.

Matthew Molineaux, Ugur Kuter, and Matthew Klenk. Discoverhistory: Understanding the past in planning and execution. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 989–996. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

Héctor Munoz-Avila, David W Aha, Ulit Jaidee, Matthew Klenk, and Matthew Molineaux. Applying goal driven autonomy to a team shooter game. In *FLAIRS Conference*, 2010.

Juan Pablo Mendoza, Manuela Veloso, and Reid Simmons. Plan execution monitoring through detection of unmet expectations about action outcomes. 2015:3247–3252, 06 2015.

Matt Paisner, Michael Maynord, Michael T Cox, and Don Perlis. Goal-driven autonomy in dynamic environments. In *Goal Reasoning: Papers from the ACS Workshop*, page 79. Citeseer, 2013.