

**Goal Reasoning:
Papers from the ACS Workshop**

(<http://mcox.org/g-reason>)

Baltimore, MD
14 December 2013

Workshop Chair and Editors

David W. Aha (Chair), Naval Research Laboratory (USA)
Michael T. Cox, University of Maryland (USA)
Héctor Muñoz-Avila, Lehigh University (USA)

Preface

This technical report contains the 11 accepted papers presented at the Workshop on *Goal Reasoning*, which was held as part of the 2013 Conference on Advances in Cognitive Systems (ACS-13) in Baltimore, Maryland on 14 December 2013. This is the third in a series of workshops related to this topic, the first of which was the AAAI-10 Workshop on *Goal-Directed Autonomy* while the second was the *Self-Motivated Agents* (SeMoA) Workshop, held at Lehigh University in November 2012.

Our objective for holding this meeting was to encourage researchers to share information on the study, development, integration, evaluation, and application of techniques related to **goal reasoning**, which concerns the ability of an intelligent agent to reason about, formulate, select, and manage its goals/objectives. Goal reasoning differs from frameworks in which agents are told what goals to achieve, and possibly how goals can be decomposed into subgoals, but not how to dynamically and autonomously decide what goals they should pursue. This constraint can be limiting for agents that solve tasks in complex environments when it is not feasible to manually engineer/encode complete knowledge of what goal(s) should be pursued for every conceivable state. Yet, in such environments, states can be reached in which actions can fail, opportunities can arise, and events can otherwise take place that strongly motivate changing the goal(s) that the agent is currently trying to achieve.

This topic is not new; researchers in several areas have studied goal reasoning (e.g., in the context of cognitive architectures, automated planning, game AI, and robotics). However, it has infrequently been the focus of intensive study, and (to our knowledge) no other series of meetings has focused specifically on goal reasoning. As shown in these papers, providing an agent with the ability to reason about its goals can increase performance measures for some tasks. Recent advances in hardware and software platforms (involving the availability of interesting/complex simulators or databases) have increasingly permitted the application of intelligent agents to tasks that involve partially observable and dynamically-updated states (e.g., due to unpredictable exogenous events), stochastic actions, multiple (cooperating, neutral, or adversarial) agents, and other complexities. Thus, this is an appropriate time to foster dialogue among researchers with interests in goal reasoning.

Research on goal reasoning is still in its early stages; no mature application of it yet exists (e.g., for controlling autonomous unmanned vehicles or in a deployed decision aid). However, it appears to have a bright future. For example, leaders in the automated planning community have specifically acknowledged that goal reasoning has a prominent role among intelligent agents that act on their own plans, and it is gathering increasing attention from roboticists and cognitive systems researchers.

In addition to a survey, the papers in this workshop relate to, among other topics, cognitive architectures and models, environment modeling, game AI, machine learning, meta-reasoning, planning, self-motivated systems, simulation, and vehicle control. The authors discuss a wide range of issues pertaining to goal reasoning, including representations and reasoning methods for dynamically revising goal priorities. We hope that readers will find that this theme for enhancing agent autonomy to be appealing and relevant to their own interests, and that these papers will spur further investigations on this important yet (mostly) understudied topic.

Many thanks to the participants and ACS for making this event happen!

David W. Aha
Baltimore, Maryland (USA)
14 December 2013

Table of Contents

Title Page	i
Preface	ii
Table of Contents	iii
Goal Substitution in Response to Surprises <i>Robert Bobrow, Marshall Brinn, Mark Burstein, & Robert Laddaga</i>	1
Question-Based Problem Recognition and Goal-Driven Autonomy <i>Michael T. Cox</i>	10
Inferring Actions and Observations from Interactions <i>Joseph P. Garnier, Olivier L. Georgeon, & Amélie Cordier</i>	26
Beyond the Rational Player: Amortizing Type-Level Goal Hierarchies <i>Thomas R. Hinrichs & Kenneth D. Forbus</i>	34
Situation Awareness for Goal-Directed Autonomy by Validating Expectations <i>Michael Karg & Alexandra Kirsch</i>	43
HALTER: Hierarchical Abstraction Learning via Task and Event Regression <i>Ugur Kuter & Héctor Muñoz-Avila</i>	53
Learning Models of Unknown Events <i>Matthew Molineaux & David W. Aha</i>	64
Goal-Driven Autonomy in Dynamic Environments <i>Matt Paisner, Michael Maynard, Michael T. Cox, & Don Perlis</i>	79
Hierarchical Goal Networks and Goal-Driven Autonomy: Going where AI Planning Meets Goal Reasoning <i>Vikas Shivashankar, Ron Alford, Ugur Kuter, & Dana Nau</i>	95
Breadth of Approaches to Goal Reasoning: A Research Survey <i>Swaroop Vattam, Matthew Klenk, Matthew Molineaux, & David W. Aha</i>	111
Towards Applying Goal Autonomy for Vehicle Control <i>Mark Wilson, Bryan Auslander, Benjamin Johnson, Thomas Apker, James McMahon, & David W. Aha</i>	127
Author Index	143

Goal Substitution in Response to Surprises

Robert Bobrow

RUSTY@BBN.COM

Marshall Brinn

MBRINN@BBN.COM

Raytheon BBN Technologies, Cambridge, MA 02138

Mark Burstein

BURSTEIN@SIFT.NET

SIFT, LLC. Lexington, MA 02421

Robert Laddaga

BOBLADDAGA@GMAIL.COM

Vanderbilt University, Nashville, TN 37235

Abstract

This paper looks at the problem of learning from and responding to surprise during task performance by agents in complex unfamiliar environments. In particular, we describe an architecture and a brief demonstration of a cognitive agent driving a car in a simulated city street grid, in which unexpected obstacles are placed. The agent is surprised by unexpected behavior of the car and its environment, and dynamically shifts, temporarily, to perform actions in terms of background safety goals even as it is learning how to behave appropriately to those unexpected conditions.

1. Introduction

Software systems are built for particular environments and applications, and are notoriously brittle in facing circumstances for which they were not explicitly designed or trained. When such systems are embedded in physically autonomous or distributed systems, UXVs or systems that interlink multiple different organizations, and are faced with dynamic, open environments and situations they were never tested on, they demonstrate their brittleness in numerous ways. For instance, a UAV might be programmed to perform reconnaissance tasks and be tested by flying solo over flat terrain but then be deployed in a mountainous setting where there are other UAVs in the same airspace. Autonomous cars may be tested on empty roads and then used in traffic. How do we ensure that such systems (a) maintain invariants, e.g., they do not run into each other or other unexpected obstacles, and (b) respond robustly to surprise or novel circumstances such as changes in maneuverability or control due to external factors? Fundamentally, these systems do more than simply adapt in order to make progress toward a current or primary goal (e.g., Georgeff et al, 1985, Remington et al, 2002). If evidence indicates that the agent's actions are not having the expected effect, then continuing to select actions that its model says will produce those effects leads only to perseveration. The architecture must change to include mechanisms that recognize when expectations fail and react by coping with these disconnect. In the short run this means considering safety goals, information gathering goals to aid in adapting and improving the agent's own models and learning goals specifically aimed at changing action preconditions, procedures, and object or state characterizations as ways of improving its behaviors over the

longer term. Such applications would then act more reliably, continually supporting high-level goals in the face of unfolding execution, and adjusting gradually to new environments. Even systems that have learned reactive controllers suffer from these issues when the environments they are employed in differ from their training. For such systems to succeed, they need to recognize and adapt to the unanticipated, that is, to surprise. They must also be capable of suspending their primary objectives in the face of such surprises, in order to maintain stability and safety invariants.

This paper briefly outlines an approach and abstract cognitive architecture for goal-directed task reasoning in environments where surprising or unexpected events will occur and must be handled. We illustrate our approach with an implementation of an agent using this architecture to handle a simulated driving task. The example shows how the agent reasons about suspending its primary goals to maintain safety using a behavior learned from previous surprises.

1.1 Goals of this research

Our overall objective has been to develop distributed autonomous systems that are more robust in response to surprising circumstances by providing each agent with a reflective software layer that maintains and updates models of the capabilities, goals and actions of the system itself, of others, and of the operating environment. By robust, we mean systems that, when encountering surprise,

- ***Don't fail*** the first time. Instead, they do something *reasonable* to allow them to keep on going while they reconsider their understanding of the world.
- ***Aren't surprised*** the second time. They have some memory of past events and can update their models so that previously surprising events are no longer outside the realm of possibility.
- ***Perform better*** each successive time. They learn continuously, improving their models of self, and others, their ability to represent and predict changes in the environment, and to plan and select actions in pursuit of goals.

Our technical approach rests on the use of a class of architectures that:

- contains a learned world-model that provides continued ***prediction*** of the evolution of the world (both as a result of actions by the agent, and as the result of endogenous processes in the world)
- utilizes these predictions as ***expectations***, which are compared against observations to detect ***surprises*** (expectation violations that may signify incompleteness or error of the world model, and
- reacts to surprise through rapid ***reflection*** that triggers ***reprioritization of goals, replanning,*** and ***control changes.***

Reliable software systems must be able to pursue goals and maintain policies even as they adapt in response to surprises. Our approach rests on the use of architectures that utilize *reflection* – diagnostic processes where one questions and tries to correct one's interpretation of the world, one's own behavior and capabilities – based on the detection of expectation violations - mismatches between predictions or assumptions and observations. In this paper we focus on how the system avoids major failure in the face of surprise. In particular, the system described in this paper makes use of a reflective control layer that manages self-diagnosis and adapts models and behavior under the time constraints imposed by the system's ongoing operations. Specifically,

this layer manages learning (model update) goals and safety/survival goals along-side task performance goals.

- **Surprise Modeling.** Surprise stems from expectations that have been violated in any of several ways. (e.g. [Horvitz et al, 2005, Shapiro et al, 2004])
 - Situations may be sufficiently uncertain or variable that they cannot be planned for.
 - Events may be considered to be impossible in that they contradict some internally modeled assumptions about the environment.
 - Events may be considered possible in general, but not predicted in specific circumstances because key indicators were not known or were not detected.
- **Assumption representation and re-evaluation.** In responding to surprises, the underlying cause for the expectation failure must be identified and corrected. To that end, the agent must reconsider its representation of assumptions about its environment, the reliability of its own behavior, and its expected impact on the environment given its actions. These assumptions must be systematically questioned and expanded to explain anomalous observations.
- **Memory-driven Expectation Monitoring.** In addition to general operating assumptions, these systems must develop expectations of specific operational contexts in order to be surprised. Agents must develop (learn) predictive experience-based memory models that enable it to have expectations with variable certainty in different contexts. Such models must characterize the capabilities of the agent itself, as well as its interactions with the environment and other agents.

Figure 1 shows, abstractly, the main control loop and reflective control loops that we envision managing the process of acting in the world, and the process of reflecting on our expectations about that world by comparing effects to intended (Δ_C or change in control) and expected (Δ_M or change in model) states while simultaneously reflecting on how accurately those expectations are met and what that implies for its model of the world, and the likely success of its plans. The reflective, cognitive control loop reacts to differences between observations and predictions (Δ_M) or expectations that it has generated based on its model of the actions it is performing. This loop is responsible for revising both the internal world model (predictors of the effects of its own actions and processes) and its plans for achieving goals. This model also includes what it knows about change processes operating in the world it is observing. When differences arise due to observations of outcomes, it must evaluate whether these differences are likely or unlikely. When the agent's actions are control changes, the differences it considers include changes in rates, not just positions. Statistically significant differences require a decision whether to simply replan or to also change its models of how the world reacts to its actions, and, conversely, of the preconditions for actions necessary in order to get a particular outcome. Model changes include changes to the likelihood of discrete or qualitative effects of actions, including attentive actions, as well as to the state of the agent's own effectors. They can also include the statistical ranges on quantitative effects. Model changes are context dependent, as we see in the case in the driving domain, discussed below, where applying brakes or turning the wheel does not have the usually predicted effect when the car is on ice. Alternate model changes and explanations for them must be considered, such as that could be that the brakes don't work when the car is on ice, or that the breaks are themselves suddenly broken. Occam's razor must be used to resolve these ambiguities.

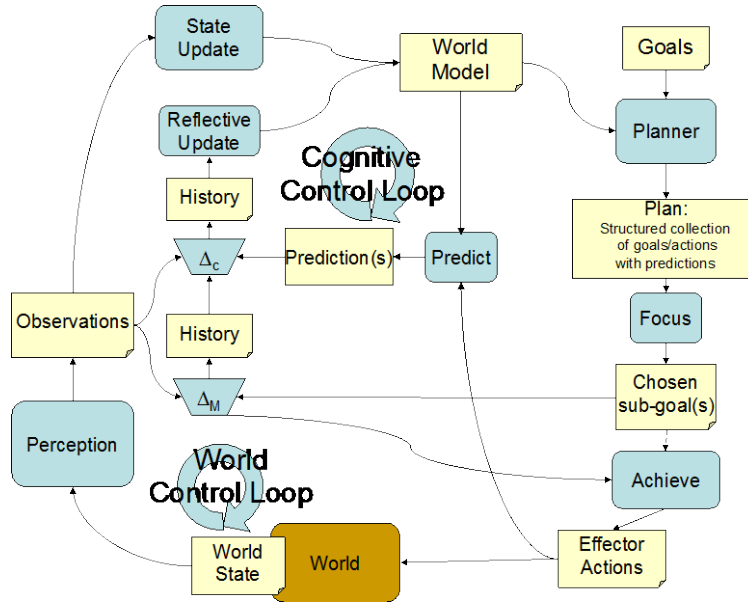


Figure 1. General COGENT Architecture

2. Sample Scenario

Consider an automated driver that knows the rudiments of driving a car in controlled test environments. It has enough knowledge to expect generally unobstructed well-paved roads, to know how to plan routes and follow directions to move through the streets, stay in lanes and follow other rules of the road, such as obeying traffic signals and signs. It also has rules associated with its and other's safety, such as knowing to slow down and stop if its path is blocked, go around minor obstacles within its path and keeping a safe distance from cars ahead of it. Though it didn't exist when we first considered this task, Google's driverless car now does reasonably well at these things. However, someone must still sit there ready to react to things that it cannot deal with, including such things as construction cones, human traffic directors, and pedestrians that come out from behind stopped vehicles ahead.

If such systems are to be able to gradually gain reliability in more realistic urban driving situations, they must learn from experience but also have the ability to reason about the expectations from prior knowledge and advice they were given to explain how new situations violated those expectations. Recognizing **expectation failures** (surprises) and using **reflective reasoning** (about potential flaws in its own behavior, knowledge, model of how the world works) to recover and learn from these surprises, should help them improve by being less surprised over time, even though the events themselves are rare.

Figure 2 shows our realization of the general COGENT (Cognitive Agent) architecture for our simplified driver agent operating in simulation. The simulated cars move forward an incremental amount based on their speed and direction at each tick, and the environmental controller makes observations and generated reactions at each step based on its observations and plans. The cognitive controller acts in response to differences between the predicted incremental effects of actions and the observed ones. When the cognitive controller is turned on, we refer to

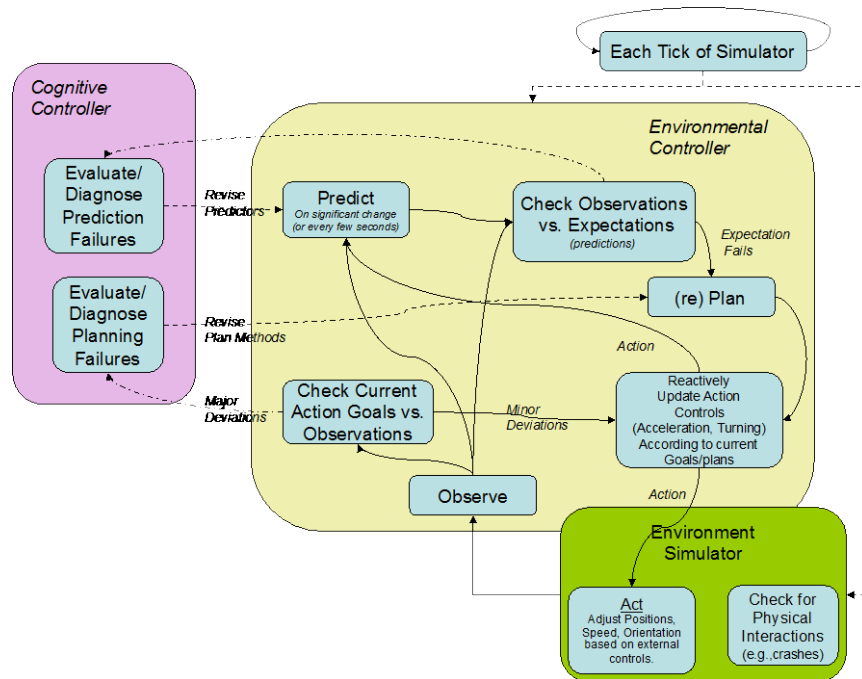
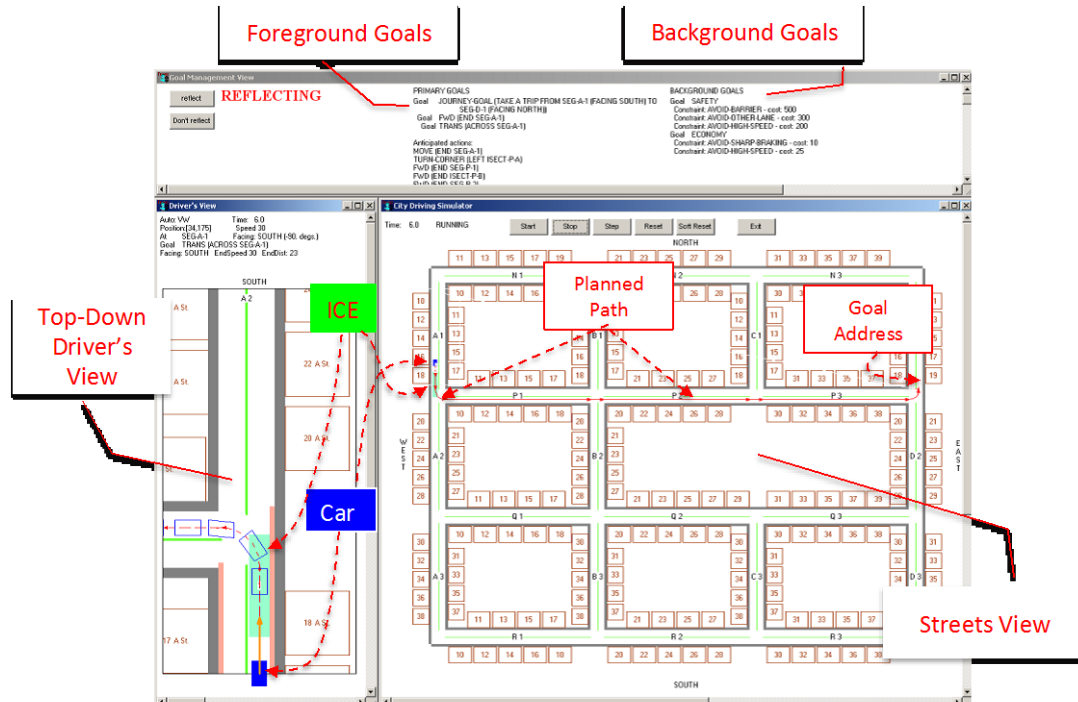


Figure 2. Simulated Driving Agent Architecture

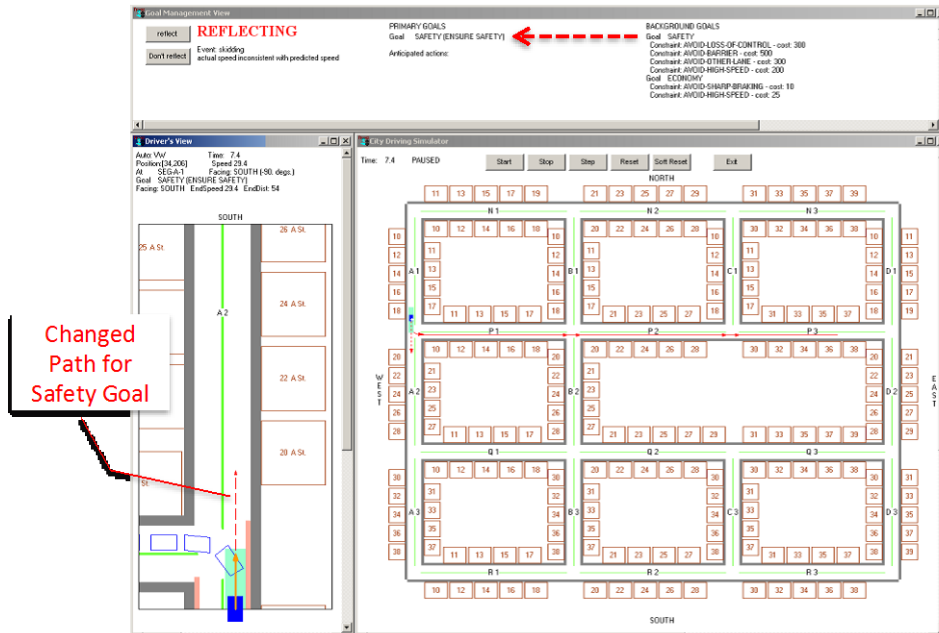
this as ‘reflective mode’ for the simulation. When prediction failures implicate the actions that the agent would use to replan for its current goal, safety goals are made primary instead.

Figure 3 shows several screen shots of the simulation in action, with and without the results of reflective control. The simulation environment was developed based on a simulation model provided by Pat Langley and described in (Choi et al, 2004). In the non-reflective case, when the driver begins a turn and detects loss of traction, it responds by turning the wheel more, the normal goal-driven course adjustment loop. The result is it crashes into the building on the far corner. When reflection is turned on, and loss of traction is detected, the foreground goal of navigating the route path is suspended temporarily, and the background safety goals of regaining control and crash avoidance are switched to the foreground. When the safety goals are primary, a set of policies for avoiding crashes is used to control the vehicle. Though the agent had not categorized the ice as a threat to control, it now recognizes control loss and uses rules to steer in the direction of movement and apply braking until control is regained. When the reflective driver does this, it misses the turn, but passes beyond the ice and stops, avoiding crashing. Once traction is restored, the reflective agent resumes its normal goals and the safety goals are pushed into the background. Now the driver realizes it has passed the intersection, and must compute a new path to get to the higher goal of arriving at its destination. Figure 4a and 4b shows these steps.

The other result of surprise is model repair. In this case, a search for differences between the state of the environment when control is maintained, and the current case of control loss, in order to explain the difference. Classifying the observed ice as the distinguished new element of the driving situation enables the reflective agent to add a new case to its model of road hazards, so that a subsequent encounter with a patch of ice causes the agent to go around the ice, rather than over it. (Figure 4c).



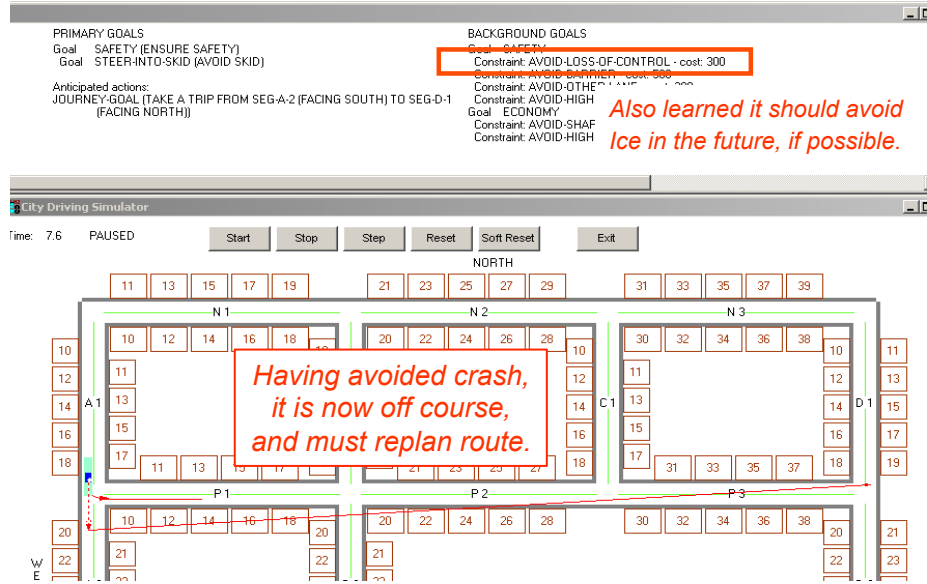
A) Car approaching patch of ice (not recognized as dangerous)



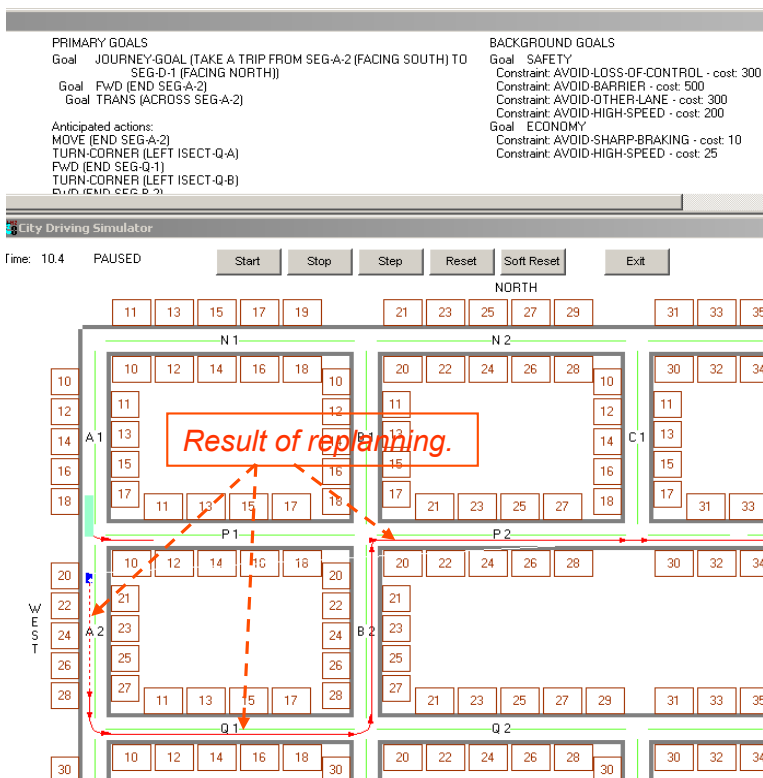
B) When simulator uses **REFLECTING** mode, Background Goal to ensure own safety is made primary when car does not behave as expected. Car breaks and does not attempt to turn.

Figure 3. Simulated reaction to hitting ice patch.

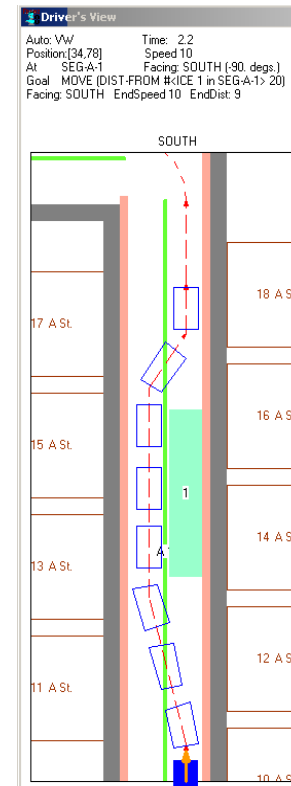
GOAL SUBSTITUTION IN RESPONSE TO SURPRISES



A) After car avoids crash, learns to attend to ice, and replans to recover original goal.



B) Replanned route, continuing from position beyond the intersection



C) Subsequent use of learned policy

Figure 4: Goal Recovery and Learning Goal Generation

3. Discussion

Though not previously published, the bulk of this work was done in 2004-2006. Our main research focus was on developing a framework for autonomous learning and adaptation in open environments where unexpected or surprising events, effects, or objects could impact the agent's goals, its model of itself, its environment, and its expected impact on that world. In this regard we are very much aligned with the work described in (Molineaux et al, 2011, Klenk et al 2013) and our architecture looks very similar to theirs. Due to the nature of the driving simulation we used as a motivating illustration, we were more concerned with particular kinds of expectation violation in continuous processes than with discrete, symbolic differences. For the driver, the presence of an unknown object (ice) plus the *unexpected rate of change* in speed and direction compared to change in controls was a primary issue. We also used a very simple model of foreground and background goals, and did not address the subtleties of reasoning about the quality of the various explanations of the observed explanation failure that could impact the choice of plans for safe recovery. Our learning was limited to reclassification of the environmental conditions impacting the operators used in the task, based on an identification of the novel features in that environment.

While there are many things that can trigger the reflective reasoning that causes shifts in goal priorities, surprise is a particularly interesting source of these responses. Although surprise is often linked solely with the expectation violations from unusual or low-probability events, in open worlds, it may also come from an inability to classify or explain objects or events of unknown types, or in unexpected contexts. We must also consider the impact on goal shifting of such things as the discovery of contradictory evidence from different sensors, and or failures in prediction due to incomplete knowledge of the world.

Our model is based heavily on the notion of reacting to and learning from expectation failures (Schank, 1982) and on the creation of internal *learning goals* (Hunter, 1989; Ram and Leake, 1995) when expectations are violated. Although our example illustrates the elevation of safety goals, in open environments surprises can be both negative and positive. Autonomous agents must have multiple goals, not all of which are active at all times. Surprises can include unexpected goal achievements and unexpected discovery of useful objects for goals that were temporarily suspended, or were associated with persistent interests or needs of the agent (food, fuel, relevant information). Reasoning to elevate and shift to serendipitous goals can occur in a fashion very similar to the activation of the safety goal in the example presented above. Surprise can also lead to learning goals for exploration and experimentation in service of knowledge gathering.

Other kinds of goal failures (especially knowledge failures) can result in shifts to plans for knowledge gathering. When the agent assumed that she had enough information to pursue the original goal, these too look very much like responses to 'surprise'.

Of course, not all goal shifts are due to surprise. In our highly multi-tasked electronically supported society, interruption is probably the number-one source of new goals and incomplete goals. If agents are working in a social setting with other agents, especially where cooperation is required, then reasoning about suspension of ones own goals to interact with and potentially address the needs of another agent is an important category of goal reasoning unrelated to surprise. Early work on agent teams such as (Tambe, 1997) illustrated this point, though without explicit goal shifts.

Acknowledgements

This work was performed in part with funding provided by DARPA under contract HR0011-04-C-0078.

References

- Choi, D., Kaufman, M., Langley, P., Nejati, N., & Shapiro, D. (2004). An architecture for persistent reactive behavior. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems* (pp. 988-995). New York: ACM Press.
- Georgeff, M., Lansky, A. & Bessiere, P. (1985) A procedural logic. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 921-927). Los Angeles, CA: Morgan Kaufmann.
- Horvitz, E., Apacible, J., Sarin, R. & Liao, L. (2005). Prediction, expectation, and surprise: Methods, designs, and study of a deployed traffic forecasting service. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*. Edinburgh, Scotland, UK.
- Howe, A.E. (1995). Improving the reliability of AI planning systems by analyzing their failure recovery. *IEEE Transactions on Knowledge and Data Engineering*, 7, 14-25.
- Hunter, L.E. (1989). *Knowledge acquisition planning: Gaining expertise through experience*. Doctoral dissertation: Department of Computer Science, Yale University, New Haven, CT.
- 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.
- 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.
- Ram, A., & Leake, D. B. (Eds.) (1995). *Goal-driven learning*. Cambridge, MA: MIT Press.
- Remington, R.W., Matessa, M.P, Freed, M. & Lee, S. (2003). Using Apex/CPM-GOMS to Develop human-like software agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems*. Melbourne, Australia.
- Schank, R. (1983). *Dynamic memory: A Theory of reminding and learning in computers and people*. New York: Cambridge University Press.
- Shapiro, D., Billman, D., Marker, M., & Langley, P. (2004). *A human-centered approach to monitoring complex dynamic systems* (Technical Report). Palo Alto, CA: Institute for the Study of Learning and Expertise.
- Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7, 83-124.

Question-Based Problem Recognition and Goal-Driven Autonomy

Michael T. Cox

MCOX@CS.UMD.EDU

Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742 USA

Abstract

Autonomy involves not only the capacity to achieve the goals one is given but also to recognize problems and to generate new goals of one's own that are worth achieving. The creation of goals starts with the recognition of a novel problem, and this recognition begins with the detection of anomalies represented as the difference between expectations and observations (or inferences). The expectation failure triggers the posing of questions; questions lead to explanations; and explanations form the basis for goals. I illustrate these principles with examples from a computational architecture called MIDCA.

1. Introduction

Have patience with everything that remains unsolved in your heart. Try to love the questions themselves, like locked rooms and like books written in a foreign language. Do not now look for the answers. They cannot now be given to you because you could not live them. It is a question of experiencing everything. At present you need to live the question. Perhaps you will gradually, without even noticing it, find yourself experiencing the answer, some distant day. — *Letter 4* (Rilke, 1986/1903)

Certainly one of the key elements of intelligence that separates us from the rest of the primate order is that we routinely question our world and ourselves. Yet some questions can take us in false directions and waste valuable time and effort. Other questions are rhetorical and serve quite different functions (e.g., speech acts). However I claim here that *questions frame an inquiry that lies at the heart of what it means to be intelligent*. Although the performance of the individual may benefit, it is not the efficiency of action that primarily motivates questioning and answering. Instead it is the agent in search of truth that operates behind the curtain. In seeking to understand the world and ourselves, we discover problems in our understanding of and in our world. By addressing these problems, we can better comprehend and independently manage the worlds within which we exist. Represented appropriately, these problems give rise to goals that enable change in our environment as well as change within ourselves.

Goals are key computational structures that drive problem solving, comprehension, and learning. In humans they constitute a special category of knowledge structure that represents desired and attainable states of affairs and that holds some measure of value for the individual (Kruglanski, 1996). For Kruglanski goals possess properties attributable to all knowledge, to goals as a class, and to the specific goal relation identified. Hierarchical systems of goal structures motivate and constrain reasoning and effective behavior (Kruglanski, Köpetz, Bélanger, Chun, Orehek, & Fishbach, 2013; Kruglanski, Shah, Fishbach, Friedman, Young, & Chun 2002). These

goal systems enable efficient self-regulation and self-control. Goal orientation has also been shown to be an important determinant for successful learning (Dweck, 1986). Indeed the claim has been made that virtually all human behavior involves the pursuit of goals, that goals are at the center of human learning, and thus that goals constitute an effective means by which to organize educational curricula (Schank, 1994).

In artificial cognitive systems, goals serve functions similar to those they do in people. In both cases goals provide focus, direction, and coordination for the allocation of computational resources during problem-solving and inference making. They also form a basis for the organization of and retrieval from memory (Schank & Abelson, 1977; Schank, 1982). Computationally goals reduce the amount of processing involved in decision-making; psychologically they afford attention and provide purpose. Further they also provide the means for explaining behavior (Malle, 2004; Ram, 1990; Schank, 1982; 1986). That is, agents do particular actions because they intend to achieve the states that result from such actions. Finally, many researchers have shown a positive relation between goal-focused behavior and deliberate, planful learning (Cox, 2007; Cox & Ram, 1999; Ram & Leake, 1995).

Most computational theories of cognition assume the existence of goal structures and concentrate on their application and use in problem-solving, planning and execution, learning and other activities. Except under two conditions, these theories generally do not account for how goals originate. One assumption is that goals are simply input by a human. A second possibility is that goals arise due to subgoaling. When action preconditions are unsatisfied in the environment, a sub-goal is spawned to achieve blocked preconditions. Here I will examine in detail an alternative cognitive process that creates novel goals for an autonomous agent.

Autonomy involves not only the capacity to achieve given goals, but it also concerns the ability to recognize new problems and to propose goals that are worth achieving. An independent cognitive system is autonomous to the extent that it understands the world and its relation to it and can act accordingly. Both opportunities and threats in the world require an agent to anticipate events within the context of its interests. Given an understanding of these opportunities and threats, an autonomous agent manages the goals and plans it has by adapting either its plans or goals and by abandoning old goals or by generating new ones.

This paper will examine these issues in the context of computational theories and will illustrate parts of our theory with various examples. The following section discusses what it means for a cognitive system to be autonomous, and it will challenge the current models. Here autonomy means being able to generate new goals rather than just following the goals of others. The subsequent section expands upon this model and proposes that new goals come from the recognition of problems in the environment and within one's knowledge and experience. The key is to identify when observations violate expectations, to ask why such an anomaly occurred, to answer by explaining the causes of the anomaly, and to generate a goal to remove the primary cause. This process either points to a deficiency of knowledge or a deficiency in the world. Next we examine an architecture called MIDCA that implements many of these ideas and processes and that provides a direction for our research. I close with summary and concluding remarks.

2. Autonomy

A common model of autonomy assumes an intelligent agent that can perform tasks given to it by a user and/or can automatically improve its performance in some environment over time (Maes, 1994; Wooldridge, 2002). The consensus asserts that agents exist within some environment (either

real or virtual) and can both sense and act within that environment (Franklin & Graesser, 1997; Weiss, 1999). Weiss (1999) distinguishes autonomous agents from mere programs, since agents decide whether or not to perform a request; whereas called functional programs have no say in the matter. Many researchers attempt to restrict autonomous agents to those that have their own agenda or otherwise do not need human intervention to perform some task (e.g., Franklin & Graesser, 1997). However this transfers the uncertainty from the question of “what is autonomy?” to “what is having an agenda?” In general, an autonomous agent is one that makes many of its own decisions but that has an explicit mechanism of control. Control is provided by human assigned tasks or goals or by programming and design specifications. Autonomy is provided procedurally by automated planning and learning mechanisms.

2.1 Goal-Following Autonomy

Consistent with these characterizations, I define *goal-following autonomy* as either hardware or software agents that can accept a goal from a user (or another agent) and can automatically achieve the goal by performing a sequence of actions in their environment.¹ The goal is simply some state configuration of the environment in terms of what needs to be satisfied. Functionally the agent implements the following characteristics.

- A means to accept a goal request for some environment;
- A decision whether to pursue the goal given its current situation and any prior requests;
- A capacity to plan for goal achievement;
- A capability to interact with the environment by executing a plan and perceiving results.

If the world does not cooperate, a flexible agent may even change its plan at execution time to accomplish the goal as specified. But once the goal is achieved, does the agent wait to be told what to do next? Does it halt? In either case, this does not appear to constitute full autonomy in the more substantial sense.

Practical problems also exist with the goal-following model. Most autonomous unmanned vehicles (AUVs) control their behaviors through preprogrammed mission plans that specify a set of waypoints, vehicle parameters, and tasks to perform at particular waypoints (Hagen, Midtgaard, & Hasvold, 2007). Complex tasks that cannot be fully specified in advance rely upon sporadic human intervention and communication. A large portion of the existing research effort into physical platforms revolves around motion planning and obstacle avoidance (e.g., see Berry, Howitt, Gu, & Postlethwaite, 2012; Minguez, Lamiroux, & Laumond, 2008). However an unrealistic burden on the user exists if a human must continuously monitor the behavior of an AUV. Furthermore in situations of low bandwidth or stealth, continuous human to machine communications cannot be assumed. Yet in many complex, dynamic environments, unusual situations arise on a regular basis as the world changes in unexpected ways. This quandary has been called *the brittleness problem* (Duda, & Shortliffe, 1983; Lenat & Guha, 1989). That is, agents and virtually all cognitive systems in complex environments are brittle except in narrow situations that have been foreseen and verified previously by the system designers. But problems will occur, and a truly autonomous system should be robust in the face of surprise.

¹ The goal-following model of autonomy also includes those agents that can accept tasks to perform rather than states to achieve. In either case, the agent takes some high-level description of the desired behavior and automatically computes how to instantiate the directive in a particular environment.

One widespread solution to the brittleness problem is machine learning technology (Maes, 1994; Holland, 1986; Stone & Veloso, 2000). Instead of explicitly programming an agent to select the optimal choice among large numbers of candidates across all possible situations, machine learning attempts to create generalizations for those conditions that apply to the largest set of possible contingencies. For example a classifier maps from a state of the world to some choice or decision outcome. In the case of AUVs, this choice could be in terms of a particular action to perform given the current state. Learning from experience then would develop new responses for unexpected situations. Much success has of course been reported in the machine learning and agent literature and also in the cognitive systems community (e.g., see Li, Stracuzzi, & Langley, 2012 and Laird, Derbinsky, & Tinkerhess, 2012 for the latter).

I do not dispute that autonomy is about choosing good actions given a goal and about flexibly learning to improve these choices. But the goal-following model is not complete, and it misses an important distinction when the environment is extremely uncooperative and complex. Interestingly DARPA (2012, p. 8) has pointed to this missing component of autonomy in its capabilities description for its latest X-ship project called the ASW Continuous Trail Unmanned Vessel or ACTUV. Among the many autonomous capabilities, they identified that ACTUV needed to be “capable of autonomous arbitration between competing mission and operating objectives based on strategic context, mission phase, internal state, and external conditions.” That is, autonomy implies the need to balance its own condition in relation to what is happening in the world with the strategic context of what it is trying to accomplish.

2.2 Goal-Driven Autonomy (GDA)

Broadly construed, the topic of *goal reasoning* concerns cognitive systems that can self-manage their goals (Vattam, Klenk, Molineaux, & Aha, in press). The topic is about high-level management of goals, plans, knowledge, and activities, not simply goal pursuit. In particular we focus on managing goals in the midst of failure. Failure is important for any cognitive system if it is to improve its performance in complex environments. First no programming no matter how extensive will guarantee success in non-trivial domains. Second failure points to those aspects of the system that are no longer relevant or contain some gap that needs filling in the new context. In particular cognitive systems (especially humans) generate expectations about what will or should occur in the world. Expectation failures drive much of cognition and a number of researchers have recognized this relationship (Anderson & Perlis, 2005; Birnbaum, Collins, Freed, & Krulwich, 1990; Cox & Ram, 1999; Perlis, 2011; Schank, 82; Schank & Owens, 1987).

The alternative model of autonomy I advocate here consists of self-motivated processes of generating and managing an agent’s own goals in addition to goal pursuit. This model - called *goal-driven autonomy (GDA)* (Cox, 2007; Klenk, Molineaux, & Aha, 2013; Munoz-Avila, Jaidee, Aha, & Carter, 2010) - casts agents as independent actors that can recognize problems on their own and act accordingly. Furthermore in this goal-reasoning model, goals are dynamic and malleable and as such arise in three cases: (1) goals can be subject to transformation and abandonment (Cox & Veloso, 1998; Talamadupula, et al., 2010); (2) they can arise from subgoaling on unsatisfied preconditions (e.g., see Veloso, 1994) or in response to impasses (Laird, 2012) during problem-solving and planning; and (3) they can be generated from scratch during interpretation (Cox, 2007; see also Norman, 1995).

For our purposes here, the most important of the above three cases is the third one. The idea is that given a problem in the world, an autonomous cognitive system must distinguish between

perturbations that require a change in plans for the old goal and those that require a new goal altogether. What is missing in the planning and agent communities is a recognition that autonomy is not just planning, acting and perceiving. It also must incorporate a first-class reasoning mechanism that interprets and comprehends the world as plans are executed (Cox, 2011). It is this comprehension process that not only perceives actions and events in the world, but can recognize threats to current plans, goals, and intentions. I claim that a balanced integration between planning and comprehension leads to agents that are more sensitive to surprise in the environment and more flexible in their responses.

In my approach, flexibility is realized through a technique I call *goal insertion*, where an agent inserts a goal into its planning process. In general goals are produced through *goal formulation* (c.f., Hanheide et al., 2010; Wilson, Molineaux, & Aha, 2013; Weber, Mateas, & Jhala, 2010), the process that includes the creation and deployment of autonomous goals where the agent takes initiative to establish new concerns and to pursue new opportunities. First an agent detects discrepancies between its observations and its expectations. The agent then explains what causes such discrepancies and subsequently generates a new goal to remove or mitigate the cause (Cox, 2007).

Consider a baby that is crying in public. Given that it normally is quite calm, the crying observation violates the mother’s expectation and represents an anomaly. The baby’s behavior could be explained in a couple of simple ways. If the baby is hungry then it cries, and if the baby’s diapers are dirty then it cries. The mother may check the diapers to eliminate that explanation and conclude that it is hungry. The resulting goal for the mother is to eliminate the hunger. A reasonable plan would be to get a bottle from her bag and feed the baby. Table 1 shows the distinct stages of the GDA process in an overly simplified manner.

Table 1. Crying baby example

Steps	Representations
Anomaly detection	Expect: calm (baby) Observe: cry (baby)
Anomaly explanation	hungry (baby) → cry (baby)
Goal formulation	\neg hungry(baby)
Plan	get (bottle) feed (baby, bottle)

In this example, the explanation is a basic rule, but in general it may be an arbitrary lattice or explanatory graph structure. Here the goal is just the negation of the rule antecedent (see Cox, 2007 for an accompanying algorithm), but for realistic situations, goal formulation must choose some salient prior state or set of states as the cause that requires attention for the problem to be solved. The next section will take a closer look at this issue.

3. Question-Based Problem Recognition

Years ago Getzels (Getzels & Csikszentmihalyi, 1975) described the early stages of problem solving as including a cognitive process they called *problem finding* (see also Runco & Chand, 1994). Problem finding involves the identification of the problem and precedes problem-

representation. Getzels classified problem finding into three distinct classes: problem presentation involves tasks given to the subject; problem discovery is the detection or recognition of the problem; and problem creation is a creative act that designs new problems.² This paper focuses on the second class and uses the more common term *problem recognition* (Pretz, Naples, & Sternberg, 2003).³

In a brief *Cognitive Science* article, Getzels (1979) notes that, despite the journal's broad coverage of cognitive processes and the eminent role that problems play in such processes, little research examines how problems are found or formulated. Surprisingly this negative condition is still true today. Despite the acknowledgement that goal formulation constitutes an initial stage of computation for agents (e.g., Russell & Norvig, 1995, p. 56), authors and AI researchers still fail to investigate its operation. Instead the technical focus is on high quality problem-representation (by researchers) and optimal problem-solving algorithms (by machines). Computational problems and goal states are typically assumed as given (presumably by an intelligent human). But our claim here is that the recognition of the problem and the generation of a goal are the hard problems rather than the generation of solutions.

The GDA approach starts with an expectation failure as an initial condition for anomaly detection. However not all anomalies are problems and not all problems are relevant to the agent. Furthermore the representation for problems is often overly simplified in the literature. For example the automated planning community represents problems as an initial state, a goal state, and a set of operators (i.e., action models) (see Ghallab, Nau, & Traverso, 2004). But the real issue with how problem instances are represented is that in general they are arbitrary. Consider the blocksworld domain. The initial states in this domain are random configurations of blocks, but so too are the goal states. For example in the first panel of Figure 1, the initial state is the arrangement of three blocks on the table and the goal state is to have block A on top of block B. The planner is given no reason for why this should be the case, unlike the second panel in the figure. In this situation the planner wishes to have the triangle D on the block A to keep water out. Here D represents the roof of the house composed of A, B, and C. Water getting into a person's living space is a problem; stacking random blocks is not.

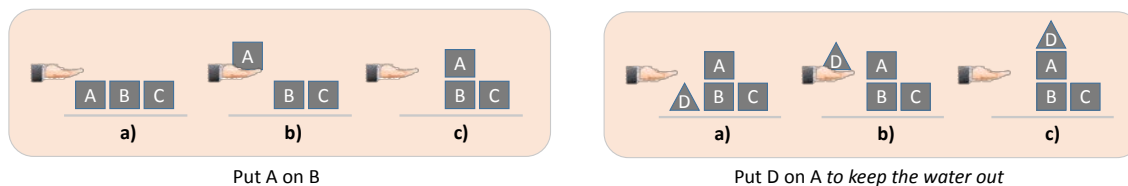


Figure 1. Blocksworld state sequences that distinguish justified problems from arbitrary problems

² Although none of these researchers take a computational approach, Hawkins, Best, & Coney (1989) actually come closest to our conception of the processes of problem recognition, but they do so from the perspective of a business analysis of consumer behavior. Their process starts with a discrepancy between what the consumer expects and what they perceive. Such problem recognition then leads to a solution in terms of a product to purchase. As an aside, it is interesting to note that it is in the interest of businesses and marketing firms to manufacture a consumer need by giving the consumer specific expectations. In this sense product marketing may be viewed as problem creation in Getzels classification.

³ See also Klein, Pliske, Crandall, & Woods (2005) for *problem detection*. Their perspective is similar to ours (although still not computational), but they insist that the detection is not solely about expectation violations.

In our earlier work on symbolic anomaly detection (Cox, Oates, Paisner, & Perlis, 2012), we simulated an anomaly by removing an operator from the set of action models for the logistics domain. The logistics problems were to transport packages from one location to another using planes and trucks. When the `unload-airplane` action was removed and the original problem set was presented to the planner, only a subset of problems could be solved. The new set of solutions then provided an anomalous series of observations relative to the normal set of solutions. Now consider how a GDA-based cognitive system might actually explain the planes not being unloaded at a particular airport.

Figure 2 shows a possible explanation. The airplanes are not unloaded, because no workers are at the airport. This is the case because the stevedores are on strike. The strike is caused by bad working conditions and low pay. For the management of the logistics company, this represents a serious and relevant problem. If it continues, profits will plummet and investor confidence will wane. Both are thus threatened. The question remains as to the management's goal however.

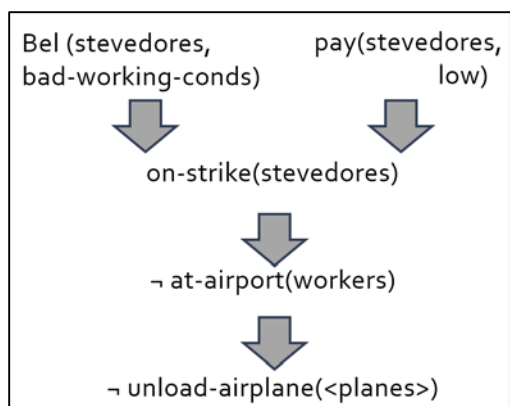


Figure 2. Explanation for why airplanes are not unloaded

Negating the pay predicate leads to higher wages; whereas negating the belief in poor conditions possibly leads to misinformation. An alternative is to negate `¬at-airport(workers)` which is to imply workers should be present. This could be achieved by bringing in scabs. Finally the management might address the actual working conditions. In this last case, the goal would be `¬bad-working-conds`. The choice of which cause of the problem to attack is unclear. A cost benefit analysis might be used to differentiate between choices, but one must be sure not to require a cost or benefit of the resulting plan or solution, only the goal. Otherwise one would need to perform planning before goal formulation – a clear case of the cart before the horse.

Instead an intelligent agent should ask the right question in the first place. It observes no planes unloading when it expects normal activity at the airport and experiences an expectation failure and hence an anomaly. If it asks why the planes were not unloaded it would get an explanation of how this was not the case. But if it asks why the workers chose not to perform their duty, it would not get an explanation that focused on their wages. They have unloaded planes in the past at these same wages. The answer would focus on the working conditions. The agent would then recursively question whether it was the perception of the conditions that changed (i.e., the belief) or the conditions themselves that changed. This secondary question would get to the core cause of the problem. Given an answer to the question and an appropriate explanation, the goal would be simple. Removing the cause of the grievance would be the goal that the autonomous agent should prefer. Planning and hence action would follow directly and effectively.

In the next section I will examine a cognitive architecture that integrates problem-solving as planning and comprehension as goal-driven autonomy. The architecture includes a cognitive cycle for action and perception and an analogous metacognitive cycle for meta-level control and introspective monitoring. I will concentrate on the former cycle to illustrate the concepts presented above.

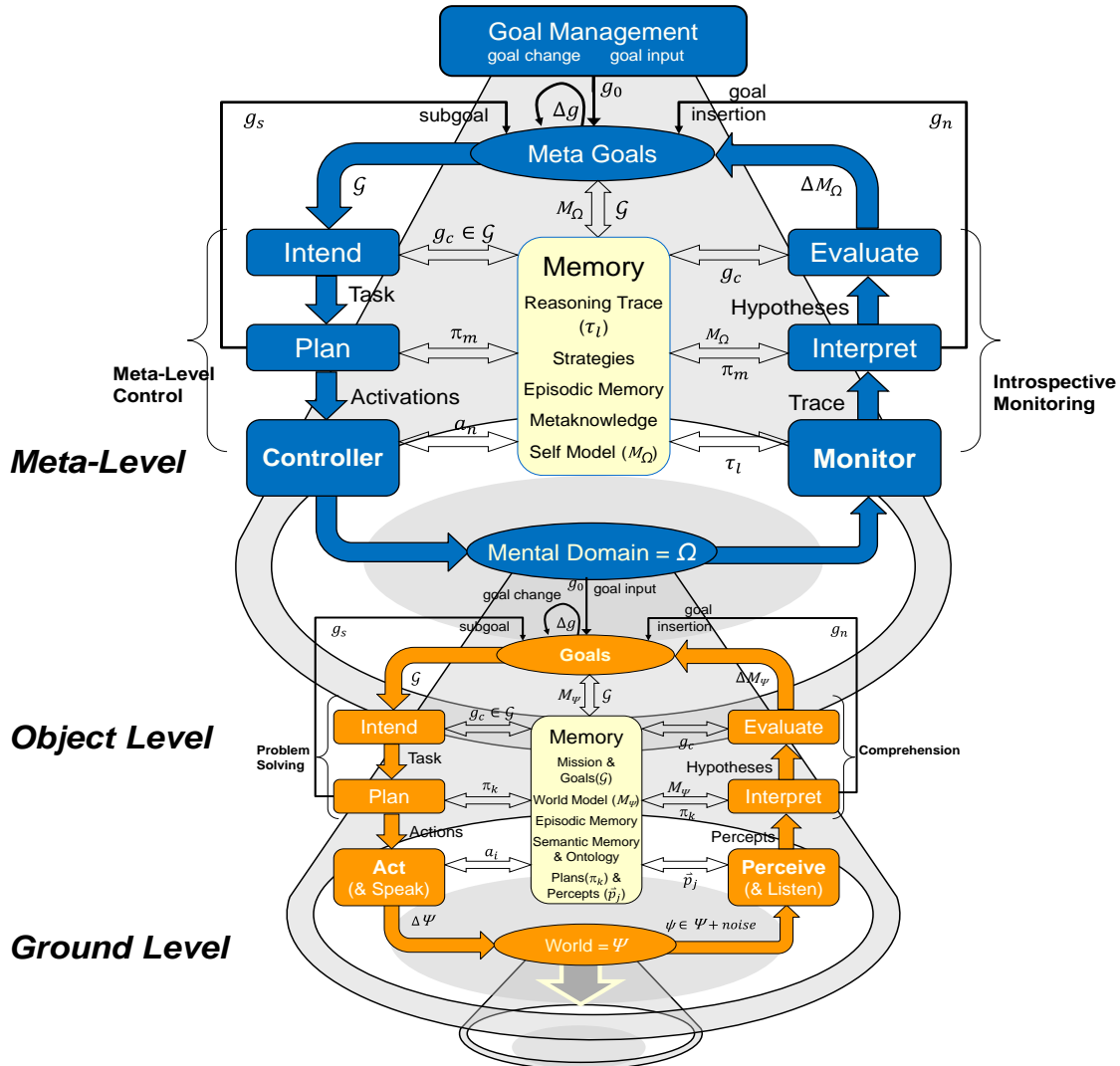


Figure 3. The MIDCA architecture

4. MIDCA

The *Metacognitive, Integrated, Dual-Cycle Architecture (MIDCA)* (Cox, Maynard, Paisner, Perlis, & Oates, 2013; Cox, Oates, & Perlis, 2011) consists of “action-perception” cycles at both the cognitive (i.e., object) level and the metacognitive (i.e., meta-) level (see Figure 3). The output side of each cycle consists of intention, planning, and action execution, whereas the input side consists of perception, interpretation, and goal evaluation. A cycle selects a goal and commits to achieving it. The agent then creates a plan to achieve the goal and subsequently executes the planned actions to make the domain match the goal state. The agent perceives changes to the environment resulting from the actions, interprets the percepts with respect to the plan, and evaluates the interpretation

with respect to the goal. At the object level, the cycle achieves goals that change the environment (i.e., ground level). At the meta-level, the cycle achieves goals that change the object level. That is, the metacognitive “perception” components introspectively monitor the processes and mental state changes at the cognitive level. The “action” component consists of a meta-level controller that mediates reasoning over an abstract representation of the object level cognition.

To appreciate the distinctions in the relationship between levels, examine the finer details of the object level as shown in Figure 3. Here the meta-level executive function manages the goal set \mathcal{G} . In this capacity, the meta-level can add initial goals (g_0), subgoals (g_s) or new goals (g_n) to the set, can change goal priorities, or can change a particular goal (Δg). In problem solving, the *Intend* component commits to a current goal (g_c) from those available by creating an intention to perform some *Task* that can achieve the goal (Cohen & Levesque, 1990). The *Plan* component then generates a sequence of *Actions* (π_k , e.g., a hierarchical-task-net plan, see Nau, et al., 2003) that instantiates that *Task* given the current model of the world (M_ψ) and its background knowledge (e.g., semantic memory and ontologies). The plan is executed by the *Act* component to change the actual world (Ψ) through the effects of the planned *Actions* (a_i). Problem solving stores the goal and plan in memory to provide the agent expectations about how the world will change in the future. Then given these expectations, the comprehension task is to understand the execution of the plan and its interaction with world with respect to the goal so that success occurs.

Comprehension starts with perception of the world in the attentional field via the *Perceive* component. The *Interpret* component takes as input the resulting *Percepts* (i.e., \vec{p}_j) and the expectations in memory (π_k and g_c) to determine whether the agent is making sufficient progress. A GDA interpretation procedure implements the comprehension process. The procedure is to *note* whether an anomaly has occurred; *assess* potential causes of the anomaly by generating explanatory *Hypotheses*; and *guide* the system through a response. Responses can take various forms, such as (1) test a *Hypothesis*; (2) ignore and try again; (3) ask for help; or (4) insert another goal (g_n). Otherwise given no anomaly, the *Evaluate* component incorporates the concepts inferred from the *Percepts* thereby changing the world model (ΔM_ψ), and the cycle continues. This cycle of problem-solving and action followed by perception and comprehension functions over discrete state and event representations of the environment.

Likewise introspective monitoring starts with “perception” of the self (Ω) via the *Monitor* component. The *Interpret* component takes as input the resulting *Trace* (i.e., τ_l) and the expectations in memory (π_m and g_c) to determine whether the reasoning is making sufficient progress. The *Interpret* procedure is to *detect* a reasoning failure; *explain* potential causes of the failure by generating explanatory *Hypotheses*; and *generate* a learning goal or attainment goal. Reasoning about the self (e.g., am I knowledgeable about the domain) and the reasoning task enables the agent to determine the difference (i.e., learning vs. attainment goal). If MIDCA produces a learning goal, the meta-level control will create and execute a learning plan to change its knowledge. Attainment goals are passed through to the object level. Given no anomaly, the *Evaluate* component incorporates the concepts inferred from the *Trace* thereby changing the self-model (ΔM_Ω), and the cycle continues.

4.1 Implementation: MIDCA_1.1

MIDCA_1.1 (Paisner, Maynard, Cox, & Perlis, in press) is a simplified version of the complete MIDCA architecture shown in the schematic of Figure 3. It is currently composed of the cognitive (object level) cycle components shown in Figure 3. The implementation effort has concentrated on

a simulator that generates successor states based on valid actions taken in the blocksworld domain; a state interpretation component; and a planner. For the planner, we used SHOP2 (Nau et al., 2003), a domain-independent task decomposition planner. Whereas the full MIDCA architecture has a meta-cognitive component that manages goals, MIDCA_1.1 has no goal management, and simply passes any new goals from the interpreter directly to the planner. In MIDCA_1.1, the Interpret component consists of an integration of bottom up and top down process as explained below. The Act component is incorporated into the blocksworld simulator, and the Perceive component is implicit in the transfer of world state representation to the interpreter.

The GDA interpretation procedure at the object level has two variations that represent a bottom-up, data-driven track and a top-down, knowledge rich, goal-driven track (Cox, Maynard, Paisner, Perlis, & Oates, 2013). The data-driven track we call the *D-track*; whereas the knowledge rich track we call the *K-track*. The D-track is implemented by a bottom-up GDA process as follows. A statistical anomaly detector constitutes the first step, a neural network identifies low-level causal attributes of the anomaly, and a machine learning goal classifier provides the goal formulation. The K-track is implemented as a case-based explanation process. The representations for expectations significantly differ between the two tracks. K-track expectations come from explicit knowledge structures such as action models used for planning and ontological conceptual categories used for interpretation. Predicted effects form the expectations in the former and attribute constraints constitute expectation in the latter. D-track expectations are implicit by contrast. Here the implied expectation is that the probabilistic distribution of observations will remain the same. When statistical change occurs instead, an expectation violation is raised.

The D-track interpretation procedure uses a novel approach for noting anomalies. We apply the statistical metric called the *A-distance* to streams of predicate counts in the perceptual input (Cox, Oates, Paisner, & Perlis, 2012; 2013). This enables MIDCA to detect regions whose statistical distributions of predicates differ from previously observed input. These regions are those where change occurs and potential problems exist.

When a change is detected, its severity and type can be determined by reference to a neural network in which nodes represent categories of normal and anomalous states. This network is generated dynamically with the growing neural gas algorithm (Paisner, Perlis, & Cox, 2013) as the D-track processes perceptual input. This process leverages the results of analysis with A-distance to generate anomaly archetypes, each of which represents the typical member of a set of similar anomalies the system has encountered. When a new state is tagged as anomalous by A-distance, the GNG net associates it with one of these groups and outputs the magnitude, predicate type, and valence of the anomaly.

Goal generation is done through a conjunction of two machine learning algorithms both of which work over symbolic predicate representations of the world (Maynard, Cox, Paisner, & Perlis, 2013). Given a world state interpretation, the state is first classified using a decision tree into one of multiple state classes, where each class has an associated goal generation rule generated during learning. Given an interpretation and a class, different groundings of the variables of the rule are permuted until either one is found which satisfies that rule (in which case a goal can be generated) or until all permutations of groundings have been attempted (in which case no goal can be generated). This approach to goal insertion is naïve in the sense that it constitutes a mapping between world states and goals which is static with respect to any context.

The K-track GDA procedure is presently implemented with the Meta-AQUA system (Cox & Ram, 1999). In Meta-AQUA frame-based concepts in the semantic ontology provide constraints on expected attributes of observed input and on expected results of planned actions. When the

system encounters states or actions that diverge from these expectations, an anomaly occurs. Meta-AQUA then retrieves an explanation-pattern that links the observed anomaly to the reasons and causal relationships associated with anomaly. A goal is then generated from salient antecedents of the instantiated explanation pattern (see also Cox, 2007).

4.2 Firefighting Example: Autonomous goal formulation

To generate goals autonomously, one might statistically train a classifier to recognize a goal given an arbitrary state representation.⁴ Maynard, Cox, Paisner, & Perlis (2013) demonstrated the potential of this approach using a knowledge structure called a *TF-Tree*. As mentioned in the previous section, the TF-Tree combines the results of two machine learning algorithms to detect those conditions that warrant the generation of a new goal. Given multiple examples of state-goal pairs, the classifier learns to generate appropriate goals when presented with novel states.

For the implementation of MIDCA_1.1 we used a modified blocksworld for the domain. This version of blocksworld includes both rectangular and triangular blocks that compose the materials for simplified housing construction (see Figure 1). The initial goals for problems in this domain are to build houses consisting of towers of blocks with a roof on each. In addition the possibility exists that blocks may catch fire (set by a hidden arsonist). Furthermore there are additional actions added to the standard blocksworld operators. One action will put out fires, and another will find and capture the arsonist. In the amalgamated firefighting/house-construction/blocksworld domain, the TF-Trees will learn to generate a goal to have a fire extinguished when given a state containing a block on fire.

The fires are problems because of the effect on housing construction and the supposed profits of the housing industry, and they pose threats to life and property. The approach to understanding the fire problems is to ask *why* the fires were started and not just *how*.⁵ An explanation of how the fire started would relate the presence of sufficient heat, fuel, and oxygen with the combustion of the blocks. Generating the negation of the presence of the oxygen for example would result in the goal \neg oxygen and therefore put out the fire. But this does not get to the reason the fire started in the first place. To ask why the fire was started would result in possibly two hypotheses or explanations. Poor safety conditions can lead to fire or the actions of arsonists can result in fire. In this latter case, the arsonist causes the presence of the heat through some hidden lighting action. Given this explanation the agent can anticipate the threat of more fires and generate a goal to remove the threat by finding the arsonist. Apprehending the arsonist then removes the potential of fires in the future rather than just reacting to fires that started in the past.

Empirical results show that a GDA approach to goal formulation significantly outperforms the statistical approach with fewer actions required for the same amount of housing construction. In brief the housing domain goes through a cycle of three state classes in building new “houses.”

⁴ One might also enumerate all possible goals and the conditions under which they are triggered. Tac-Air Soar (Jones, Laird, Nielsen, Coulter, Kenny, & Koss, 1999) takes this approach. Operators exist for various goal types and data-driven context-sensitive rules spawn them given matching run-time observations. However even if one could engineer all relevant goals for a domain and all the conditions under which they apply, an expectation failure or surprise may occur if the domain shifts (e.g., the introduction of novel technology). The recognition of new problems and the explanation of their causes would enable the formulation of a goal from first principles, even under conditions not envisioned by the agent designer.

⁵ See Leake (1991) for extensive discussion of explanation types and the conditions under which they are appropriate.

Figure 4 shows three instantiated states classified by the TF-Trees and the grounded goals that each tree recognizes.

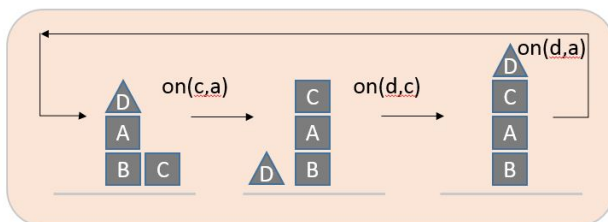


Figure 4. Three classifiers exist that recognize goals to get to the next state

Using the GDA method, over 1000 time steps results in an average improvement of 245.4% relative to a baseline method that lacks goal generation altogether; whereas the statistical method results in a 54.2% improvement over the baseline (see Paisner, Maynard, Cox, & Perlis, in press, i.e., this volume, for technical details). The specific results are less important, however, than the conceptual framing of the problem. Many

methods exist to implement a GDA approach to planning and action (e.g., see Aha, Klenk, Muñoz-Avila, Ram, & Shapiro, 2010; Aha, Cox, & Munoz-Avila, in press). The crucial point for autonomous cognitive systems is to consider how they can be made to understand and represent the problem not just optimize a solution.

5. Conclusion

This paper is not a technical analysis of autonomy, nor is it an empirical investigation of specific algorithms and data structures; rather it makes a sometimes intuitive case for a different approach to autonomy and intelligent agency. The proposal is that autonomy is chiefly about recognizing problems independently and generating goals to solve them. The specific technical details as they exist now are to be found in the references cited throughout this article. In lieu of these details here, I have tried instead to explore an alternative that lies relatively unexamined and to shed some light on the issues. For the most part, my proposition is novel, because few have thought much about it computationally; the research that does exist is found mainly in the social psychology literature. Also it is still unclear how many of the propositions and claims herein can be implemented fully, although further examples exist, especially in some of the earlier AI literature (e.g., Leake, 1991; Ram, 1991; Schank, 1986). A few new researchers have done work under the GDA topic, but much of the focus has been on goal formulation, explanation, and case learning rather than problem recognition and question posing. So as it stands now, much research remains, not the least of which is to clarify the role of question asking and experience in problem recognition.

No current theory explains precisely why humans question the world and themselves. What is the exact relationship between question posing, problem recognition, and motivation in an intelligent agent? Why does an agent actively look for problems to begin with? That is why are we so motivated (sometimes compelled) to solve problems? Certainly a benefit accrues or some personal utility exists in the cost-benefit calculations that percolate in our mind, but this may not constitute the whole story. Perhaps at least part of the answer can be seen in the explanations provided by Higgins (2012).

Higgins proposes that human intelligence is motivated by three things: value, truth, and fit. Value corresponds to the utility (i.e., value) we experience in things (e.g., objects, events, or states). Truth corresponds to the capability to accurately interpret the world with respect to one's own experience and memory. Fit refers to the appropriateness of strategy. The AI community focusses on the first factor; whereas this paper has focused on the second. The motivation of truth impels us

to look clearly at the world. We seek to discern where our observations agree with and where they differ from our knowledge of the world with the aim of improving our understanding. As I see it, value and truth correspond to the problem solving and problem comprehension divisions within the MIDCA architecture. An autonomous agent values effective performance and finds “truth” in effective interpretation. One is as a complement to the other.

Acknowledgements

This material is based upon work supported by ONR Grants # N00014-12-1-0430 and # N00014-12-1-0172 and by ARO Grant # W911NF-12-1-0471. I thank Michael Maynard, Tim Oates, Don Perlis and the anonymous reviewers for comments on the content of this paper.

References

- Aha, D. W., Cox, M. T., & Munoz-Avila, H. (Eds.) (in press). *Proceedings of the 2013 Annual Conference on Advances in Cognitive Systems: Workshop on Goal Reasoning*. College Park, MD: University of Maryland.
- Aha, D. W., Klenk, M., Muñoz-Avila, H., Ram, A., & Shapiro, D. (Eds.) (2010). *Goal-Directed Autonomy: Papers from the AAI Workshop*. Menlo Park, CA: AAI Press.
- Anderson, M., & Perlis, D. (2005). Logic, self-awareness and self-improvement. *Journal of Logic and Computation* 15, 21–40.
- Berry, A. J., Howitt, J., Gu, D.-W., & Postlethwaite, I. (2012). A continuous local motion planning framework for unmanned vehicles in complex environments. *Journal of Intelligent & Robotic Systems* 66(4), 477-494.
- Birnbaum, L., Collins, G., Freed, M., & Krulwich, B. (1990). Model-based diagnosis of planning failures. In *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 318-323). Menlo Park, CA: AAI Press.
- Cohen, P. R. & Levesque, H. J. (1990). Intention is choice with commitment. *Artificial Intelligence* 42(2-3), 213 – 261.
- Cox, M. T. (2007). Perpetual self-aware cognitive agents. *AI Magazine* 28(1), 32-45.
- Cox, M. T. (2011). Metareasoning, monitoring, and self-explanation. In M. T. Cox & A. Raja (Eds.) *Metareasoning: Thinking about thinking* (pp. 131-149). Cambridge, MA: MIT Press.
- Cox, M. T., Maynard, M., Paisner, M., Perlis, D., & Oates, T. (2013). The integration of cognitive and metacognitive processes with data-driven and knowledge-rich structures. In *Proceedings of the Annual Meeting of the International Association for Computing and Philosophy*.
- Cox, M. T., Oates, T., Paisner, M., & Perlis, D. (2012). Noting anomalies in streams of symbolic predicates using A-distance. *Advances in Cognitive Systems* 2, 167-184.
- Cox, M. T., Oates, T., Paisner, M., & Perlis, D. (2013). Detecting change in diverse symbolic worlds. In L. Correia, L. P. Reis, L. M. Gomes, H. Guerra, & P. Cardoso (Eds.), *Advances in Artificial Intelligence, 16th Portuguese Conference on Artificial Intelligence* (pp. 179-190). University of the Azores, Portugal: CMATI.

- Cox, M. T., Oates, T., & Perlis, D. (2011). Toward an integrated metacognitive architecture. In P. Langley (Ed.), *Advances in Cognitive Systems: Papers from the 2011 AAAI Fall Symposium* (pp. 74-81). Technical Report FS-11-01. Menlo Park, CA: AAAI Press.
- Cox, M. T., & Ram, A. (1999). Introspective multistrategy learning: On the construction of learning strategies. *Artificial Intelligence*, 112, 1-55.
- Cox, M. T., & Veloso, M. M. (1998). Goal transformations in continuous planning. In M. desJardins (Ed.), *Proceedings of the 1998 AAAI Fall Symposium on Distributed Continual Planning* (pp. 23-30). Menlo Park, CA: AAAI Press.
- Defense Advanced Research Projects Agency (2012). *ASW Continuous Trail Unmanned Vessel (ACTUV) Phases 2 through 4*. DARPA-BAA-12-19. Arlington, VA: DARPA. <https://www.fbo.gov/utills/view?id=2935bca24073347c8fd1ae0820cc20f8>
- Duda, R. O., & Shortliffe, E. H. (1983). Expert systems research. *Science* 220, 261-268.
- Dweck, C. S. (1986). Motivational processes affecting learning. *American Psychologist*, 41, 1040-1048.
- Franklin, S., & Graesser, A. (1997) Is it an agent, or just a program?: A taxonomy for autonomous agents, *Intelligent agents III*. Berlin: Springer, 21-35.
- Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated planning: Theory and practice*. San Francisco: Morgan Kaufmann.
- Getzels, J. W., & Csikszentmihalyi, M. (1975). From problem solving to problem finding. In I. A. Taylor & J. W. Getzels (Eds.), *Perspectives in creativity* (pp. 90-116). Chicago: Aldine.
- Getzels, J. W. (1979). Problem finding: A theoretical note. *Cognitive Science* 3, 167-172.
- Hagen, P. E., Midtgaard, O., & Hasvold, O. (2007). Making AUVs truly autonomous. In *Proceedings of the MTS/IEEE Oceans Conference and Exhibition* (pp. 1-4). Red Hook, NY: Curran Associates.
- Hanheide, M., Hawes, N., Wyatt, J., Gobelbecker, M., Brenner, M., Sjoo, K., Aydemir, A., Jenselt, P., Zender, H. and Kruiff, G. (2010). A framework for goal generation and management. In D. W. Aha, M. Klenk, H. Muñoz-Avila, A. Ram, & D. Shapiro (Eds.), *Goal Directed Autonomy: Papers from the AAAI Workshop*. Menlo Park, CA: AAAI Press.
- Hawkins, D. I., Best, R. J., & Coney, K. A. (1989). *Consumer behavior: Implications for marketing strategy*, 4ed. Boston: Business Publications.
- Higgins, E. T. (2012). *Beyond pleasure and pain: How motivation works*. New York: Oxford University Press.
- Holland, J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. Michalski, J. Carbonell & T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*, Vol. 2 (pp. 593-623). San Mateo, CA: Morgan Kaufmann Publishers.
- Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P., & Koss, F. V. (1999). Automated intelligent pilots for combat flight simulation. *AI Magazine*, 20(1), 27-41.
- Klein, G., Pliske, R. Crandall, B., & Woods, D. D. (2005). Problem detection. *Cognition, Technology & Work* 7(1), 14-28.
- Klenk, M., Molineaux, M., & Aha, D. (2013). Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence*, 29(2), 187-206, 2013.

- Kruglanski, A. W. (1996). Goals as knowledge structures. In P. M. Gollwitzer & J. A. Bargh (Eds.), *The psychology of action: Linking cognition and motivation to behavior* (pp. 599-618). New York: Guilford Press.
- Kruglanski, A. W., Köpetz, C., Bélanger, J. J., Chun, W. Y., Orehek, E., & Fishbach, A. (2013). Features of multifinality. *Personality and Social Psychology Review*, 17(1) 22–39.
- Kruglanski, A. W., Shah, J. Y., Fishbach, A., Friedman, R., Young, W., & Chun (2002). A theory of goal systems. In M. P. Zanna (Ed.), *Advances in experimental social psychology* (pp. 331-378). New York: Academic Press.
- Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.
- Laird, J. E., Derbinsky, N., & Tinkerhess, M. (2012). Online determination of value-function structure and action-value estimates for reinforcement learning in a cognitive architecture. *Advances in Cognitive Systems* 2, 221-238.
- Leake, D. (1991). Goal-based explanation evaluation. *Cognitive Science* 15, 509-545.
- Lenat, D., & Guha, R. (1989). *Building large knowledge-based systems*. Menlo Park, CA: Addison-Wesley.
- Li, N., Stracuzzi, D. J., & Langley, P. (2012). Improving acquisition of teleoreactive logic programs through representation extension. *Advances in Cognitive Systems* 1, 109–126.
- Maes, P. (1994). Modeling adaptive autonomous agents. *Artificial Life* 1 (1-2), 135-162.
- Malle, B. F. (2004). *How the mind explains behavior: Folk explanations, meaning, and social interaction*. Cambridge, MA: MIT Press/Bradford Books.
- Maynard, M., Cox, M. T., Paisner, M., & Perlis, D. (2013). Data-driven goal generation for integrated cognitive systems. In C. Lebiere & P. S. Rosenbloom (Eds.), *Integrated Cognition: Papers from the 2013 Fall Symposium* (pp. 47-54). Menlo Park, CA: AAAI Press.
- Mínguez, J., Lamiroux, F., & Laumond, J.-P. (2008). Motion planning and obstacle avoidance. In B. Siciliano & O. Khatib (Eds.), *Springer handbook of robotics* (pp. 827-852). Berlin: Springer.
- Munoz-Avila, H., Jaidee, U., Aha, D. W., Carter, E. (2010). Goal-driven autonomy with case-based reasoning. In *Case-Based Reasoning, Research and Development, 18th International Conference on Case-Based Reasoning, ICCBR 2010* (pp. 228-241). Berlin: Springer.
- Nau, D., Au, T., Ilghami, O., Kuter, U., Murdock, J., Wu, D., & Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20, 379–404
- Norman, T. (1995). *Motivation-based direction of planning attention in agents with goal autonomy*. PhD thesis, Department of Computer Science, University College London.
- Paisner, M., Maynard, M., Cox, M. T., & Perlis, D. (in press). Goal-driven autonomy in dynamic environments. To appear in D. W. Aha, M. T. Cox, & H. Munoz-Avila (Eds.), *Proceedings of the 2013 Annual Conference on Advances in Cognitive Systems: Workshop on Goal Reasoning*. College Park, MD: University of Maryland.
- Paisner, M., Perlis, D., & Cox, M. T. (2013). Symbolic anomaly detection and assessment using growing neural gas. In *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence* (pp. 175-181). Los Alamitos, CA: IEEE Computer Society.
- Perlis, D. (2011). There's no 'me' in meta - or is there? In Cox, M. T., and Raja, A., (Eds.), *Metareasoning: Thinking about thinking* (pp. 15–26). Cambridge, MA: MIT Press.

- Pretz, J. E., Naples, A. J., & Sternberg, R. J. (2003). Recognizing, defining, and representing problems. In J. E. D. a. R. J. Sternberg (Ed.), *The psychology of problem solving* (pp. 3-30). Cambridge, UK: Cambridge University Press.
- Ram, A. (1990). Decision models: A theory of volitional explanation. In *Proceedings of Twelfth Annual Conference of the Cognitive Science Society* (pp. 198-205). Hillsdale, NJ: LEA.
- Ram, A. (1991). A theory of questions and question asking. *Journal of the Learning Sciences*, 1, (3&4), 273-318.
- Ram, A., & Leake, D. (1995). Learning, goals, and learning goals. In A. Ram & D. Leake (Eds.), *Goal-driven learning* (pp. 1-37). Cambridge, MA: MIT Press/Bradford Books.
- Rilke, R. M. (1986). *Letters to a young poet*. New York: Vintage Books. Originally published 1903.
- Runco, M. A., & Chand, I. (1994). Problem finding, evaluative thinking, and creativity. In M. A. Runco (Ed.), *Problem finding, problem solving, and creativity* (pp. 40-76). Norwood, NJ: Ablex.
- Russell, S. & Norvig, P. (1995). *Artificial intelligence: A modern approach*. Upper Saddle River, NJ: Prentice Hall.
- Schank, R. C. (1982). *Dynamic memory: A theory of reminding and learning in computers and people*. Cambridge, MA: Cambridge University Press.
- Schank, R. C. (1986). *Explanation patterns: Understanding mechanically and creatively*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Schank, R. C. (1994). Goal-based scenarios: A radical look at education. *The Journal of the Learning Sciences*, 3(4), 429-453.
- Schank, R. C., & Abelson, R. P. (1977). *Scripts, plans, goals and understanding: An inquiry into human knowledge structures*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Schank, R. C., & Owens, C. C. (1987). Understanding by explaining expectation failures. In R. G. Reilly (Ed.), *Communication failure in dialogue and discourse*. New York: Elsevier Science.
- Stone, P. & Veloso, M. M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8, 345-383.
- Talamadupula, K., Schermerhorn, P., Benton, J., Kambhampati, S., & Scheutz, M. (2011). Planning for agents with changing goals. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling* (pp. 71-74). Menlo Park, CA: AAAI Press.
- Vattam, S., Klenk, M., Molineaux, M., & Aha, D. (in press). Breadth of approaches to goal reasoning: A research survey. In D. W. Aha, M. T. Cox, & H. Munoz-Avila (Eds.), *Proc. of the 2013 Annual Conference on ACS: Workshop on Goal Reasoning*. College Park: Univ. Maryland.
- Veloso, M. (1994). *Planning and learning by analogical reasoning*. Berlin: Springer-Verlag.
- Weber, B. G., Mateas, M., & Jhala, A. (2010). Case-based goal formulation. In *Proceedings of the AAAI Workshop on Goal-Driven Autonomy*.
- Weiss, G. (1999). *Multiagent systems: A modern approach to distributed artificial intelligence*. Cambridge, MA: MIT Press.
- Wilson, M., Molineaux, M., & Aha, D. W. (2013). Domain-independent heuristics for goal formulation. *Proceedings of the Twenty-Sixth Florida Artificial Intelligence Research Society Conference* (pp. 160-165). Menlo Park, CA: AAAI Press.
- Wooldridge, M. (2002). *An introduction to multiagent systems*. Hoboken, NJ: Wiley.

Inferring Actions and Observations from Interactions

Joseph P. Garnier

Olivier L. Georgeon

Amélie Cordier

Université de Lyon, CNRS

Université Lyon 1, LIRIS, UMR5205, F-69622, France

JOSEPH.GARNIER@LIRIS.CNRS.FR

OLIVIER.GEORGEON@LIRIS.CNRS.FR

AMELIE.CORDIER@LIRIS.CNRS.FR

Abstract

This study follows the Radial Interactionism (RI) cognitive modeling paradigm introduced previously by Georgeon and Aha (2013). An RI cognitive model uses sensorimotor interactions as primitives—instead of observations and actions—to represent Piagetian (1955) sensorimotor schemes. Constructivist epistemology suggests that sensorimotor schemes precede perception and knowledge of the external world. Accordingly, this paper presents a learning algorithm for an RI agent to construct observations, actions, and knowledge of rudimentary entities, from spatio-sequential regularities observed in the stream of sensorimotor interactions. Results show that the agent learns to categorize entities on the basis of the interactions that they afford, and appropriately enact sequences of interactions adapted to categories of entities. This model explains rudimentary goal construction by the fact that entities that afford desirable interactions become desirable destinations to reach.

1. Introduction

Georgeon and Aha (2013) introduced a novel approach to cognitive modeling called Radical Interactionism (RI), which invites designers of artificial agents to consider the notion of *sensorimotor interaction* as a primitive notion, instead of perception and action. A sensorimotor interaction represents an indivisible cognitive cycle, consisting of sensing, attending, and acting. Within constructivist epistemology, it corresponds to a Piagetian (1955) sensorimotor scheme from which the subject constructs knowledge of reality. RI suggests a conceptual inversion of the learning process as compared to traditional cognitive models: instead of learning sensorimotor interactions from patterns of observations and actions, RI recommends constructing observations and actions as secondary objects. This construction process rests upon regularities observed in sensorimotor experience, and happens concurrently with the construction of knowledge of the environment. Figure 1 illustrates the RI cognitive modeling paradigm.

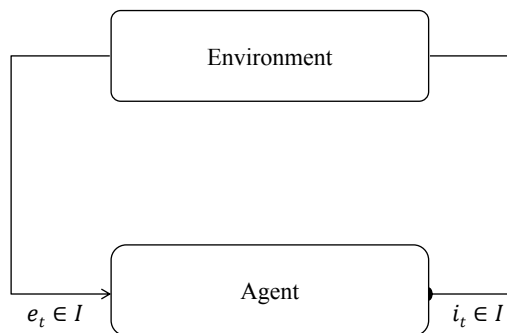


Figure 1. The Radical Interactionism modeling paradigm (adapted from Georgeon & Aha 2013). At time t , the agent chooses an *intended primitive interaction* i_t from among the set of interactions I . The attempt to enact i_t may change the environment. The agent then receives the *enacted primitive interaction* e_t . If $e_t = i_t$ then the attempt to enact i_t is considered a *success*, otherwise, a *failure*. The agent's "perception of its environment" is an internal construct rather than the input e_t .

The algorithm begins with a predefined set of sensorimotor interactions I , called *primitive interactions*. At time t , the agent chooses a primitive interaction i_t that it intends to enact, from among I . The agent ignores this enaction's meaning; that is, the agent has no rules that would exploit knowledge of how the designer programmed the primitive interactions through actuator movements and sensory feedback (such as: "if a specific interaction was enacted then perform a specific computation"). As a response from the tentative enaction of i_t , the agent receives the *enacted interaction* e_t , which may differ from i_t . The enacted interaction is the only data available to the agent that carries some information about the external world, but the agent ignores the meaning of this information.

An RI agent is programmed to learn to anticipate the enacted interactions that will result from its intentions, and to tend to select intended interactions that are expected to succeed ($e_t = i_t$). Such a behavior selection mechanism implements a type of self-motivation called autotelic motivation (the motivation of being "in control" of one's activity, Steels, 2004). Additionally, the designer associates a numerical valence with primitive interactions, which defines the agent's behavioral preferences (some primitive interactions that the agent innately *likes* or *dislikes*). Amongst sequences of interactions that are expected to succeed, an RI agent selects those that have the highest total valence, which implements an additional type of self-motivation called interactional motivation (Georgeon, Marshall, & Gay, 2012).

Our previous RI agents (Georgeon & Ritter, 2012; Georgeon, Marshall, & Manzotti, 2013) learned to organize their behaviors so as to exhibit rudimentary autotelic and interactional motivation without constructing explicit observations and actions. Here we introduce an extension to construct instances of objects (in the object-oriented programming sense of "object") that represent explicit observations and actions learned through experience. Our motivation is to design future RI agents that will use these to learn more sophisticated knowledge of their environment and develop smarter behaviors. In particular, we address the problem of autonomous goal construction by modeling how an observable entity in the environment that affords positive interactions can become a desirable destination to reach.

2. Agent

Our agent has a rudimentary visual system that generates *visual interactions* with entities present in the environment. A visual interaction is a sort of sensorimotor interaction generated by the relative displacement of an entity in the agent’s visual field as the agent moves. The agent is made aware of the approximate relative direction of the enacted visual interaction e_t by being provided with the angular quadrant ρ_t in which e_t was enacted. Additionally, the agent is made aware of its displacement in space through the angle of rotation $\theta_t \in \mathbb{R}$ induced by the enaction of e_t . The information θ_t corresponds to the information of relative rotation given by the vestibular system in animals. It can be obtained through an accelerometer in robots. Figure 2 illustrates these additions to the RI model.

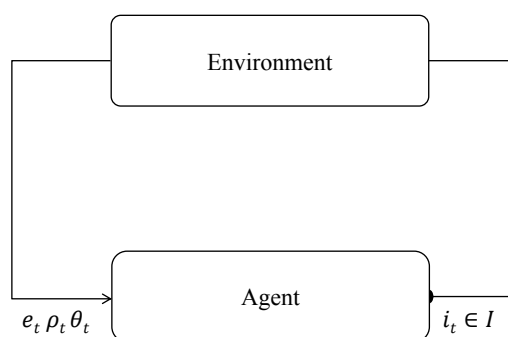


Figure 2. The Directional Radical Interactionism (DRI) model. Compared to RI (Figure 1), the DRI model provides additional directional information ρ_t and θ_t when a primitive interaction e_t is enacted at time t . ρ_t represents the directional quadrant where the interaction e_t is enacted relative to the agent, and θ_t the angle of rotation of the environment relative to the agent, generated by the enaction of e_t .















3. Experiment

We propose an implementation (see Figure 3) using the DRI model to study how agents constructs observations and actions from spatio-sequential regularities observed in its stream of sensorimotor interactions. This experiment was implemented in Java in our environment using the Enactive Cognitive Architecture (ECA). ECA is a cognitive architecture based on sensorimotor modeling, inspired by the Theory of Enaction, to control an agent that learns to fulfill its autotelic and interactional motivation. Also, ECA allows implementing self-motivation in the agent¹(Georgeon, Marshall, & Manzotti, 2013). The environment consists of a grid of *empty cells* (white squares) where the *agent* (represented by the brown arrowhead) tries to move one cell forward, turn to left or to the right. The experimenter can flip any cell from empty to wall or vice versa by clicking on it at any time. Also, the environment is composed of walls (gray squares) where the agent could bump if it tries to move through them.

1. <http://e-ernest.blogspot.fr/2013/09/ernest-12.html>

The agent has a rudimentary distal sensory system was inspired by the visual system of an arachaic arthropod, the limulus: the limulus's eyes responds to movement, and the limulus has to move to "see" immobile things. The agent "likes" to eat blue fish (called target). When the agent reaches a target, the target disappears as if the agent had eaten it. The experimenter can introduce other targets by clicking on the grid. The agent's visual system consists of one simple detector (violet half-circle on the agent) for detecting target. His detector covers a 180° span. This visual system is not sensitive to static elements of the visual field (such as the presence and the position of the target) but to changes in the visual field as the agent moves: closer, appears, unchanged and disappeared. Moreover, the agent divides his visual field in three area: *A*, *B* and *C*. These area inform the agent in which directional quadrant the entity is detected.

The designer can also specify the numerical valence associated with primitive interactions before running the simulation. The values chosen implement a behavioral proclivity to move towards targets because the agent has positive satisfaction when the targets appears or closer, and negative satisfaction when the target disappears.

a) Primitive interactions (valence)		<i>Meaning (ignored by the agent)</i>
i_1  (10)	i_2  (10)	<i>turn right target closer, turn left target closer</i>
i_3  (3)	i_4  (3)	<i>turn right target appears, turn left target appears</i>
i_5  (-1)	i_6  (-1)	<i>turn right visual field unchanged, turn left visual field unchanged</i>
i_7  (-1)	i_8  (-1)	<i>turn right target disappeared, turn left target disappeared</i>
i_9  (15)	i_{10}  (-1)	<i>move forward target eaten, bump</i>
i_{11}  (-5)	i_{12}  (-5)	<i>move forward target disappeared, move forward target appears</i>
i_{13}  (-6)	i_{14}  (-6)	<i>move forward visual field unchanged, move forward target closer</i>

b) Environment:

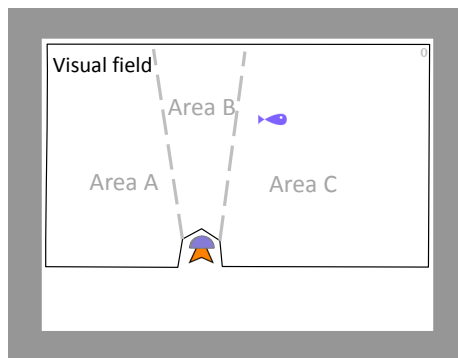


Figure 3. a) The 14 primitive interactions available to the agent with their numerical valence in parentheses, set by the experimenter. This valence system implements the motivation to move towards targets because the valence is positive when the target appears or approaches, and negative when the target disappears. b) The agent in the environment with the agent's visual field overprinted. There are three directional quadrants in which visual interactions can be localized: $\rho_t \in \{A, B, C\}$. Non-visual interactions are localized in a fourth abstract quadrant labeled "O".

To understand how the agent, during interactions with the environment, constructs its actions and its observations, we propose a simplified UML model and an example in Figure 4, and finally the algorithm.

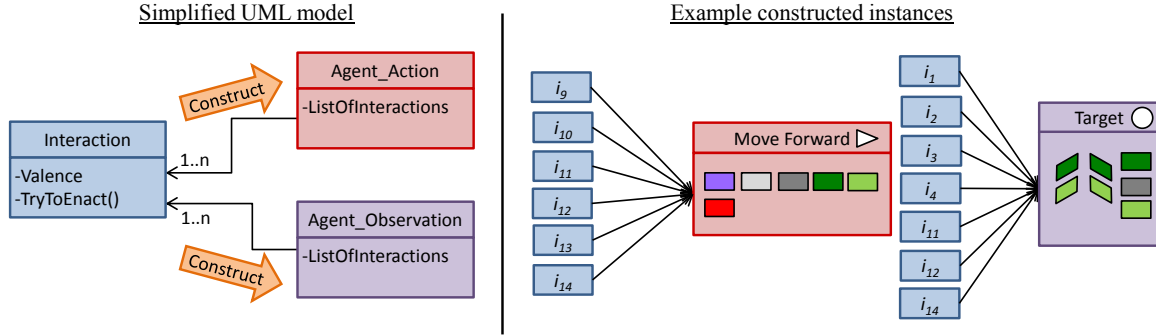


Figure 4. Simplified UML model (left): the modeler defines primitive interactions as instances of subclasses of the *Interaction* class (left) and programs their effects in the *TryToEnact()* method. The agent constructs actions as instances of the *Agent_Action* class (top-right) and observations as instances of the *Agent_Observation* class (bottom-right) from sequential and spatial regularities observed while enacting interactions. Example constructed instances (right): the action *Move Forward* can be enacted through the interactions $i_9, i_{10}, i_{11}, i_{12}, i_{13}, i_{14}$. The observation *Target* affords interactions $i_1, i_2, i_3, i_4, i_{11}, i_{12}, i_{14}$.

To interact with the environment, the agent utilizes a set of interactions defined by the designer. The designer programs an interaction as an action on the environment and an observation. But, the agent originally ignores this distinction and must learn that some interactions inform it about the presence of an entity in its surrounding space, while simultaneously learning to categorize these entities. Each interaction can be afforded by a specific type of entity. In using the model DRI, see section 2, at decision step t , the agent tries to enact an *intended interaction* i_t and get the actually enacted interaction, *enacted interaction*, e_t at the end of step t . If the enacted interaction differs from the intended interaction ($e_t \neq i_t$) then the agent considers that these interactions produce two different actions a_1, a_2 . Thus, a first action is represented by interaction e_t and a second action is represented by interaction i_t ($a_1 = \{e_t\}$ and $a_2 = \{i_t\}$). In case of $e_t = i_t$, the agent considers that these interactions produce the same action, which can be represented by the set of these interactions ($a_1 = a_2 = \{e_t, i_t\}$).

A type of entity present in the world affords a collection of interactions. When a set of interactions consistently overlaps in space, the agent infers the existence of a kind of entity that affords these interactions. To be concrete, a physical object would be an entity that is solid and persistent. The agent uses spatial information from DRI model to learn to categorize the entity with it can interact, according to the collection of interactions that this entity affords. At decision step t the agent tries to enact an *intended interaction* i_t and get the interaction effectively enacted, *enacted interaction*, e_t at the end of step t . In each *enacted interaction* there is the directional quadrants (A, B, C or O) where it enacted. If the enacted interaction e_t is in the same area that enacted interaction e_{t-1} then the agent considers that these interactions are afforded by the same entity

($entity_1 = entity_2 = \{e_t, e_{t-1}\}$). In case of these interactions are enacted in two different area, the agent infer it exists two kind of entity ($entity_1 = \{e_t\}$ and $entity_2 = \{e_{t-1}\}$).

4. Result

During the learning phase, the agent learns a behavior that it then uses to reach subsequent targets introduced by the experimenter. Different instances of agents may learn different behaviors as a result of having different learning experiences. Figure 5 and 6 show traces of two behaviors learned by two different agents. Once a behavior has been learned, the agent keeps using it indefinitely to reach subsequent targets.

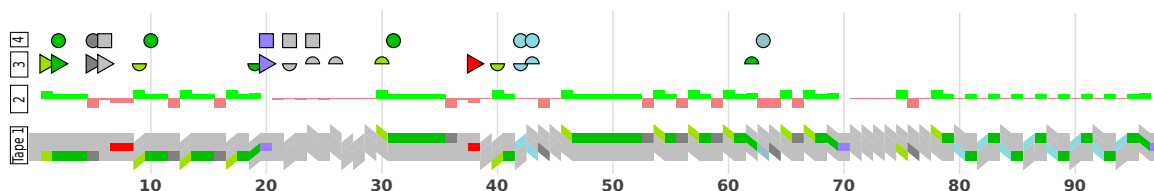


Figure 5. First 97 steps in Example 1. Tape 1 represents the primitive interactions enacted, in directional quadrant *A* (top), *B* (center), *C* (bottom), with the same symbols as in Figure 3. Tape 2 represents the valence of the enacted primitive interactions as a bar graph (green when positive, red when negative). Tape 3 represents the progressive aggregations of interactions to form actions. The shape represents the action and the color is the color of the enacted interaction aggregated to this action at a particular time step. The triangles correspond to the *move forward* action, the inferior half-circles to the *turn right* action, and the superior half-circles to the *turn left* action. Tape 4 represents the progressive aggregation of interactions to form observations. The shape represents the category of observation and the color is the color of the enacted interaction aggregated to this category of observation at a particular time step. The circles represent the observation of a *target*, and the squares the observation of *void*. The agent also constructs a third category of observation: the observation of *walls*. However, since walls are only observable through a single interaction (i_{10} , red rectangles), there is no aggregation of other interactions to the wall observation. In this example, the agent ate the first target on step 20 (blue rectangle in Tape 1). The experimenter introduced the second target on step 30, and the agent ate it on step 70. The third target was introduced on step 74 and eaten on step 97. The agent learned to reach the target through a "stair step" behavior consisting of repeating the sequence turn left - move forward - turn right - move forward, until it aligns itself with the target and then keeps moving forward until it reaches the target (steps 78 to 97).

A different choice of valence or modification of the environment by the experimenter at different times shows that behavior depends on the motivation that drives the agent and environment configuration. For example, if the experimenter add a target earlier than in Example 1, the agent acts differently. This behavior has been observed in Experiment 2 illustrated by the example trace in Figure 6.

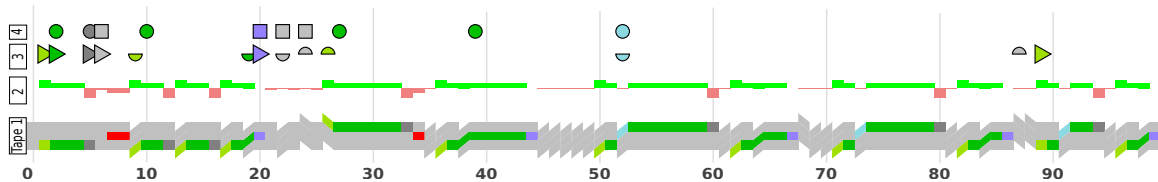


Figure 6. First 99 steps in Example 2. The behavior is the same as in Example 1 up to step 25. The experimenter introduced the second target on step 26 rather than 30 in Example 1. This difference caused the agent to learn a different behavior to reach the target, consisting in moving in a straight line until the target disappears from the visual field, then getting aligned with the target by enacting the sequence turn right – turn right – move forward – turn right, then keeping moving forward until it reaches the target (episodes 26 to 44, 50 to 67, 71 to 86 and 89 to 99).

This experiment also demonstrates the interesting property of individuation: different instances of agents with the same algorithm may learn different behaviors due to the specific learning experiences that they had. From step 26, behaviors are different. Such individuation occurs through "en habitus deposition" as conceptualized by the theory of enaction.

5. Conclusion

This work addresses the problem of implementing agents that learn to *master the sensorimotor contingencies* afforded by their coupling with their environment (O'Regan & Noë, 2001). In our approach, the modeler specifies the low-level sensorimotor contingencies through a set of sensorimotor interactions, which corresponds to what Buhrmann, Di Paolo, and Barandiaran (2013) have called the *sensorimotor environment*. The learning consists for the agent to simultaneously learn actions and categories of observable entities as second-order constructs. Here, we use the concept of action in its cognitive sense of "intentional action" (Engel et al., 2013). Our algorithm offers a solution to implements Engel et al.'s (2013, p203) view that "agents first exercise sensorimotor contingencies, that is, they learn to associate movements with their outcomes, such as ensuing sensory changes. Subsequently, the learned patterns can be used for action selection and eventually enable the deployment of intentional action".

Our agent has no pre-implemented strategy to fulfill his inborn motivation (approaching the target). We show two examples in which the agent learns two different deployments of actions to fulfill this motivation (Figure 5 and 6). These deployments of actions can be considered intentional because the agent anticipates the consequences of actions and use anticipation to select actions. In future studies, we plan on designing agents capable of reasoning upon their intentionality to learn to explicitly consider observable entities as possible goals to reach. We expect that emergent intentionality associated with explicit goal construction will make the agents capable of exhibiting more sophisticated behaviors in more complex environments, and contribute more broadly to the research effort on goal reasoning.

Acknowledgements

This work was supported by the French Agence Nationale de la Recherche (ANR) contract ANR-10-PDOC-007-01.

References

- Buhrmann, T., Di Paolo, E. A., & Barandiaran, X. (2013). A dynamical systems account of sensorimotor contingencies. *Frontiers in psychology, 4*.
- Engel, A. K., Maye, M., Kurthen, M., & König, P. (2013). Where's the action? the pragmatic turn in cognitive science. *Trends in Cognitive Sciences, 17*, 202–209.
- Georgeon, O. L., & Aha, D. (2013). The radical interactionism conceptual commitment. *Journal of Artificial General Intelligence*, In press.
- Georgeon, O. L., Marshall, J., & Gay, S. (2012). Interactional motivation in artificial systems: Between extrinsic and intrinsic motivation. *International Conference on Development and Learning (ICDL-Epirob)*.
- Georgeon, O. L., Marshall, J., & Manzotti, R. (2013). Eca: An enactivist cognitive architecture based on sensorimotor modeling. *Biologically Inspired Cognitive Architectures, 6*, 46–57.
- Georgeon, O. L., & Ritter, F. (2012). An intrinsically-motivated schema mechanism to model and simulate emergent cognition. *Cognitive Systems Research, 15-16*, 73–92.
- O'Regan, J. K., & Noë, A. (2001). A sensorimotor account of vision and visual consciousness. *Behavioral and Brain Sciences, 24*, 939–972.
- Piaget, J. (1955). *The construction of reality in the child*. London: Routledge and Kegan Paul.
- Steels, L. (2004). The autotelic principle. In F. Iida, R. Pfeifer, L. Steels, & Y. Kuniyoshi (Eds.), *Embodied artificial intelligence*, Vol. 3139 of *Lecture Notes in Computer Science*, 231–242. Springer Berlin Heidelberg.

Beyond the Rational Player: Amortizing Type-Level Goal Hierarchies

Thomas R. Hinrichs

Kenneth D. Forbus

EECS, Northwestern University, Evanston, IL 60208 USA

T-HINRICHS@NORTHWESTERN.EDU

FORBUS@NORTHWESTERN.EDU

Abstract

To what degree should an agent reason with pre-computed, static goals? On the one hand, efficiency and scaling concerns suggest the need to avoid continually re-generating subgoals, while on the other hand, flexible behavior demands the ability to acquire, refine, and prioritize goals dynamically. This paper describes a compromise that enables a learning agent to build up a type-level goal hierarchy from learned knowledge, but amortizes the cost of its construction and application over time. We present this approach in the context of an agent that learns to play the strategy game Freeciv.

1. Introduction

One of the premises of goal reasoning is that explicit, reified goals are important for flexibly driving behavior. This idea is almost as old as AI itself and was perhaps first analyzed and defended by Alan Newell (1962), yet it is worth revisiting some of the rationale. Explicit goals allow an agent to reason about abstract future intended states or activities without making them fully concrete. A goal may be a partial state description, or it may designate a state in a way that is not amenable to syntactic matching (e.g., to optimize a quantity).

A challenge arises when a domain is dynamic and involves creating new entities on the fly, and therefore potentially new goals, or simply involves too many entities to permit reifying propositional goals. For example, in playing Freeciv¹, new cities and units are continually being created and it is not possible to refer to them by name before they exist. Moreover, each terrain tile in a civilization may be relevant to some goal, but it is infeasible to explicitly represent them all. In short, static goal trees can be inflexible and reified propositional goal trees can be prohibitively large. Goal representations may not be amenable to matching, and dynamically inferring subgoals can be expensive.

These challenges have led us to a compromise solution with six parts:

1. Elaborate a goal lattice from a given performance goal and a learned qualitative model.
2. Represent goals at the type level.
3. Indexicalize goals for reuse across game instances.
4. Reify denotational terms (names) for goals to reduce computational cost of subgoaling.
5. Identify and reify goal tradeoffs.
6. Index goals and actor types by capability roles.

¹ <http://freeciv.wikia.com>

Given this information, a very simple agent can efficiently make informed decisions with respect to goals. We call such an agent a *rational player*. A rational player has the property that its actions can be explained or justified with respect to a hierarchy of domain goals. This is not a planner per se, because it does not necessarily project future states of the world, but a rational player prefers actions that it believes will positively influence higher-level (and therefore more important) goals. As we progress beyond the rational player, this weak method for deciding what to do can be incrementally augmented and overridden by learned knowledge that looks more like task-decomposition plans and policies for choosing among tradeoffs.

This paper presents our type-level goal representation and describes how it is produced and used by a rational player in the context of learning to play Freeciv, and previews our extension to a more reflective player.

2. The Freeciv Domain

Freeciv is an open-source implementation of the popular Civilization series of games. The main objective is to build a large civilization over several simulated millennia and conquer all other civilizations. This involves many types of decisions, long-term goals, and tradeoffs. Examples of some of these types of decisions can be seen in Figure 1.

There are clear challenges in learning a game like this. The sheer diversity of activities, complexity of the world model, uncertainty of adversarial planning and stochastic outcomes, incomplete information due to the "fog of war", and dynamic nature of the game makes it fundamentally different from games like chess or checkers.

Some salient features of this game are that many actions are durative, actions have delayed effects, and the game simulates a number of quantitative systems. This makes it relatively easy to model in terms of *continuous processes* (Forbus, 1984). In fact, one of our hypotheses has been that by learning a qualitative model of the game, a simple game playing agent should be able to exploit the model to achieve flexible behavior in far fewer trials than it would need using reinforcement learning. An additional benefit of an explicit model is that it can be extended through language-based instruction or by reading the manual, and its behavior is explainable with respect to comprehensible goals.

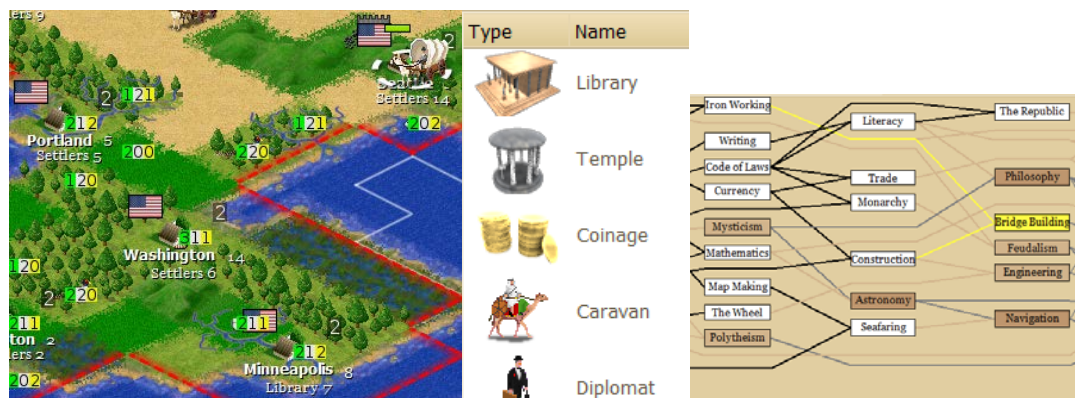


Figure 1: Freeciv map, typical production decisions, and partial technology tree

3. Representing a Reified Goal Lattice

We assume that games are defined in terms of one or more top-level goals for winning along with a system of rules. We also assume that there is a set of discrete primitive actions with declarative preconditions, plus an enumerable set of quantity types whose value can be sampled. Beyond that, we assume almost nothing about the domain. Initially, therefore, there is no goal lattice, no HTN plans, or goal-driven behavior. The system must learn to decompose goals to subgoals by observing an instructor’s demonstration and inducing a qualitative model of influences between quantities and the effects of actions on quantities. This directed graph of influences is translated into a lattice of goals. So for example, given a positive influence between food production on a tile and food production in a city, a goal of maximizing city food production is decomposed into a subgoal of maximizing food production on its tiles. This lattice-building process starts with a high-level goal and works down, bottoming out in leaves of the influence model. Notice that the construction of this goal lattice does not suggest that the system knows how to *pursue* any particular goal or goal type. It merely provides a general way to search for subgoals that might be operational – there might be a learned plan for achieving a goal or a primitive action that has been empirically discovered to influence a goal quantity.

As we talk about goals in this paper, we will occasionally refer to *performance* goals, as opposed to *learning* goals. Performance goals are domain-level goals that contribute to winning the game. Learning goals, on the other hand, are knowledge acquisition goals that the agent posts to guide experimentation and active learning. In addition, the performance goals in the lattice can be further partitioned into *propositional* goals (achieve, maintain, or prevent a state) and *quantitative* goals (maximize, balance, or minimize a quantity). Identifying subgoal relations between propositional and quantitative goals involves a combination of empirical and analytical inferences. In this section, we describe how this works and how these goals are decomposed and represented.

3.1 The learned qualitative model

A key idea is that subgoal decompositions can be derived directly from a learned qualitative model of the domain (Hinrichs & Forbus 2012a). The system induces qualitative relations in the game by observing an instructor’s demonstration and tracking how quantities change in response to actions and influences over time, in a manner similar to BACON (Langley et al., 1987). Influences are induced at the *type-level* using higher-order predicates (Hinrichs & Forbus 2012b). For example:

```
(qprop+TypeType (MeasurableQuantityFn cityFoodProduction)
  (MeasurableQuantityFn tileFoodProduction)
  FreeCiv-City FreecivLocation cityWorkingTileAt)
```

This represents the positive indirect influence between the food produced at a tile worked by a city and the total food produced by the city. This second-order relation implicitly captures a universal quantifier over all cities and locations related by the `cityWorkingTileAt` predicate. We refer to this as a type-level relation because it does not reference any instance-level entities such as particular cities or locations in the game. In addition to influences, we also induce qualitative *process limits* by looking for actions and events that bookend monotonic trends.

3.2 Building the Goal Lattice from the Qualitative Model

Given a model of type-level qualitative influences and a performance goal, such as maximizing food production in cities, it is straightforward to infer that one way to achieve this is to maximize the food produced on tiles worked in cities. This inference is more complex when the parent goal is quantitative and the subgoal is propositional, or vice versa. In the former case, the influence model may contain a *dependsOn* relation, which is a very general way of saying that a quantity is influenced by a proposition, for example, the food production on a tile depends on the presence of irrigation. Our system currently learns this through language-based instruction.

The opposite case is more complex. What does it mean for a propositional goal to have a quantitative subgoal? We've identified two ways this happens: maximizing the *likelihood* of achieving or preventing the proposition and maximizing the *rate* of achieving it. Our current implementation focuses on subgoaling via the rate. For example, given a propositional goal such as `(playerKnowsTech ?player TheRepublic)`, it identifies this as the outcome of a durative action whose process is directly influenced by the global science rate (learned empirically). Maximizing this will achieve the proposition sooner, and is therefore a subgoal.

We found such inferences became unacceptably slow when they are re-computed for each decision and transitively propagated through the influence model. To avoid repeatedly instantiating intermediate results, we translate the influence model into an explicit goal lattice offline, store it with the learned knowledge of the game, and elaborate it incrementally as the learned model is revised. A partial goal lattice for Freeciv is shown in Figure 2.

3.3 Representing goals at the type level

Because type-level goals are not tied to particular entities that change across instances of the game, they remain valid without adaptation. For example, a goal to maximize surplus food in all cities could be represented as:

```
(MaximizeFn
  ((MeasurableQuantityFn cityFoodSurplus)
   (GenericInstanceFn FreeCiv-City))
```

Here, `GenericInstanceFn` is used to designate a skolem to stand in for instances of a city, as if there were a universally quantified variable.

3.4 Indexical Goal Representations

One problem with the goal above is that it does not restrict the cities to those owned by the current player whose goal this is. The goal representation must relate everything back to the current player, except that the player's name changes in each game instance. The player may be called Teddy Roosevelt in one game and Attila the Hun the next. Consequently, we introduce an *indexical* representation for the current player as:

```
(IndexicalFn currentPlayer)
```

where `currentPlayer` is a predicate that can be queried to bind the player's name. Now, instead of our skolem being simply every city, we must construct a specification that relates the subset of cities to the indexical:


```
(CollectionSubsetFn FreeCiv-City
  (TheSetOf ?var1
    (and (isa ?var1 FreeCiv-City)
      (ownsCity (IndexicalFn currentPlayer)?var1))))
```

This subset description is constructed by recursively walking down from the top-level indexicalized game goal through the type-level qualitative influences. These influences relate the entities of the dependent and independent quantities. These relations are incrementally added to the set specification to form an audit trail back to the player. Thus the translation from model to goal involves no additional domain knowledge.

3.5 Denotational Terms for Goals

The fully-expanded representation for a simple quantity goal starts to look rather complex. We found that simply retrieving subgoal relationships at the type level suffered from the cost of unifying such large structures. As a consequence, we construct explicit denotational terms for type-level goals and use these more concise names to capture static subgoal relationships. The `goalName` predicate relates the denotational term to the expanded representation:

```
(goalName (GoalFn 14)
  (MaximizeFn
    ((MeasurableQuantityFn cityFoodSurplus)
      (GenericInstanceFn
        (CollectionSubsetFn FreeCiv-City
          (TheSetOf ?var1
            (and (isa ?var1 FreeCiv-City)
              (ownsCity (IndexicalFn currentPlayer)?var1))))))))))
```

Representing the subgoal relationship is now more concise:

```
(subgoal (GoalFn 14) (GoalFn 15))
```

This leads to more efficient retrieval, unification and reasoning. The savings in run time to retrieve a fact of this form versus compute it ranges from a factor of about 36 to a factor of 250.

3.6 Reifying Goal Tradeoffs

Goals are often in conflict. Resource allocation decisions in particular almost always involve tradeoffs. Most commonly, there's an opportunity cost to pursuing a goal. The appropriate balance usually changes over time and often varies across different entities based on extrinsic properties such as spatial location. In the process of building the goal lattice, we also record whether sibling goals involve a tradeoff. In Figure 2, this is indicated by an oval node shape.

The tradeoffs are partitioned along two dimensions: *partial/total* indicates whether every instance of the goal is affected or only some, while *progressive/abrupt* indicates whether the tradeoff happens gradually or instantaneously. So a total-abrupt tradeoff describes mutually exclusive goals, while total-progressive describes a global balance that could shift over time (such as changing tax allocations). A partial-abrupt tradeoff instantaneously switches the goal preference for some subset of possible entities and not others (such as suddenly putting all coastal

cities on a defensive war footing), whereas a partial-progressive tradeoff gradually changes the goal preference over time and for different entities.

The existence of a tradeoff is inferred based on whether two quantities are percentages of a common total, inversely influence each other, or constitute a *build-grow* pattern that trades off the number of entities of a quantity type versus the magnitude of that of that quantity for each entity.

3.7 Indexing Capability Roles

While the goal network amortizes subgoal relations, assignment decisions can also benefit from reifying type-level information. We treat the Freeciv game player as a kind of multi-agent system to the extent that there are different units and cities that can behave independently or in concert. The different unit types have different capabilities such that, for example, only Settlers can found cities, and only Caravans or Freight can establish trade routes. Static analysis run before the first game inspects the preconditions of primitive actions and clusters unit types into non-mutually exclusive equivalence categories with respect to groups of actions.

4. Reasoning with the Goal Lattice

To illustrate how the goal lattice can be used flexibly and efficiently, we present a sequence of

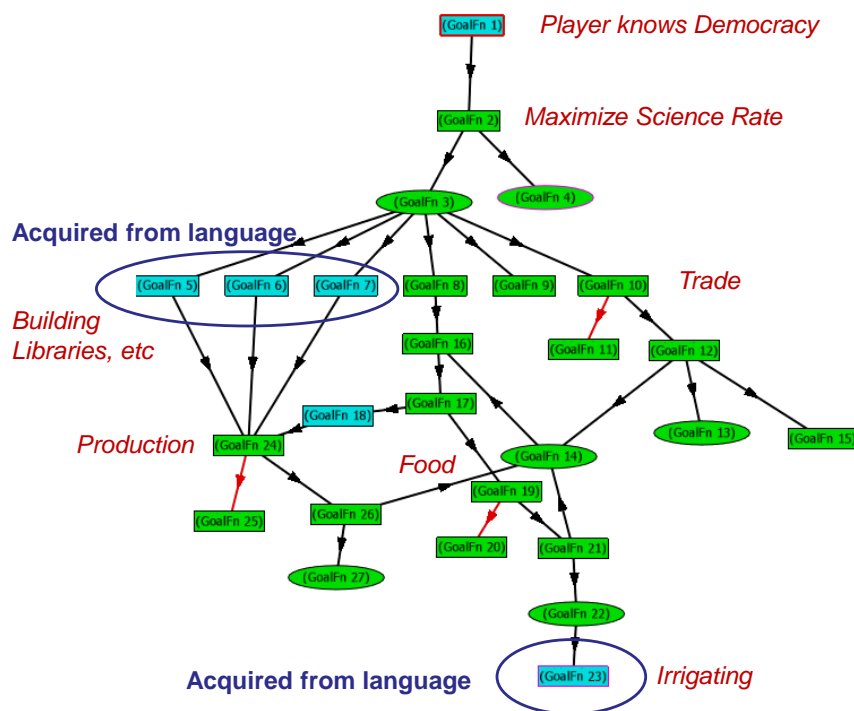


Figure 2: Reified Freeciv Goal Lattice. Oval shapes indicate goals with tradeoffs. Full goal representations would be too large to display, but see section 3.5 for the expansion of (GoalFn 14).

game-playing interpreters. We do not refer to these as planners because they do not project future states. They make individual decisions or expand learned HTN tasks to execute primitives, after which they re-plan to adapt to the dynamically changing environment.

4.1 The Legal Player

The legal player is essentially a strawman game interpreter. It works by enumerating all legal actions on each turn for each agent and picking randomly. This provides a point of comparison to show the effect of domain learning, and also serves to illustrate the size of the problem space. On the initial turn, with five units and no cities, there are 72 legal actions. This problem space quickly grows as new units and cities are constructed so that on a typical game, after 150 turns there are over 300 legal actions (388 on a recent game, but this depends on how many units and cities a civilization has). The gameplay is abysmal, as expected. In particular, it tends to continually revise decisions about what to research, what to build and what government to pursue.

4.2 The Rational Player

The rational player is the simplest player that makes use of goals when deciding what to do. It uses the goal lattice in two ways: it makes *event-triggered* decisions and it *polls* for things for agents to do. To clarify this distinction, consider that some decisions can legally be made at any time, such as choosing what to build, selecting a government or a technology to research. To avoid constantly churning through these decisions on every turn, we learn via demonstration what domain events trigger these decision types. So for example, we choose what a city should produce when it is first built or whenever it finishes building something. By factoring out the sequencing and control issues, the rational player can focus on making these individual decisions with respect to its goal lattice. It does this by enumerating the alternative choices and searching breadth-first down the goal lattice. Recall that the lattice reflects the causal influence structure of the domain model. In other words, traversing the goal lattice is a kind of regression planning from desired effects to causal influences. At each level in the lattice, it collects the sibling goals and uses simple action regression planning to gather plans that connect the decision choices to the goals. If no such connection is found, it proceeds to the next set of subgoals and tries again. Alternatively, if more than one action leads to a goal at that level, the competing plans are used to populate a dynamic programming trellis in order to select a preferred choice. Due to the dynamic nature of the domain, we use a heuristic that prefers choices that maximize future options. So for example, to achieve `TheRepublic`, a good place to start is by researching `TheAlphabet` because it enables more future technologies sooner than do the alternatives.

The other mode of operation is resource-driven. The rational planner enumerates the available actors in the game and assigns them to goals by again searching breadth-first through the goal lattice and finding the highest-level operational goal with which the actor can be tasked. Here, the notion of an operational goal is different than it is for decision tasks. Instead of applying action regression and dynamic programming, it looks for primitive actions and learned HTN plans that have been empirically determined to achieve the goal type. The most common learned plans are those that achieve the preconditions of primitives. So for example, the goal of having irrigation in a location is operational because of the existence of a plan for achieving the preconditions of the `doIrrigate` primitive. The precondition plan is learned from demonstration and by reconciling the execution trace against the declarative preconditions of the action.

Consequently, when the rational planner searches the goal lattice, it finds the goal to achieve irrigation, which it can directly perform.

In these ways, the rational player uses learned knowledge both in the decomposition of goals and in the construction of plans to achieve them. It does not, however, use known goal tradeoffs to search for appropriate balances between goals.

4.3 The Reflective Player

The reflective player is a superset of the rational player. It is designed to allow learned knowledge to override the blind search methods of the rational player. Although still under development, the reflective player will exploit higher-level strategies acquired by operationalizing domain-independent abstract strategies. Operationalization works by reconciling demonstration and instruction against the known goals and goal tradeoffs. Our belief is that effective, coordinated strategies are not something that people invent from scratch when they learn a game, but instead they are adapted from prior background knowledge. This is markedly different from a reinforcement learning approach that re-learns each game anew.

5. Related Work

To the extent that we are concerned with rational behavior in a resource-bounded agent, our approach can be viewed as a kind of Belief-Desires-Intentions (BDI) architecture. As such, it is novel in what it chooses to represent persistently in terms of goals (the type level lattice), and how it breaks behavior down into action plans and decision tasks and what or how much to re-plan on each turn.

Goel and Jones (2011) investigated meta-reasoning for self-adaptation in an agent that played Freeciv. We are pursuing similar kinds of structural adaptation, while trying to learn higher-level strategies from demonstration and instruction.

Many researchers have explored using reinforcement learning for learning to play games. In the Freeciv domain, Branavan et al. (2012) combine Monte-Carlo simulation with learning by reading the manual. While information from the manual accelerates learning, it still requires many trials to learn simple behaviors and the learned knowledge is not in a comprehensible form that can serve to explain its behavior.

Hierarchical Goal Network planning (HGN) is a formalism that incorporates goals into task decomposition planning (Shivashankar et al, 2012). It combines the task decomposition of HTN planning with a more classical planning approach that searches back from goal states. This is similar to the way we use HTN decomposition to set up regression planning from propositional goals, however we do not produce a complete plan this way and execute it. Instead, we use the planner to identify the best initial step and execute that, and then re-plan. This works well when the actions are durative, such as researching a technology. In other cases, where actual procedures make sense, we bottom out at learned HTN plans, rather than classical planning.

6. Summary and Future Work

We have described a compromise between pre-computing or providing static goals versus completely inferring them on the fly. Since the structure of the game doesn't change, the lattice

of goal types can be saved for efficient reuse without over-specifying particular entities. This goal lattice can be used in a simple reasoning loop to guide planning and decision making.

This does not preclude learning more strategic and reactive behaviors. Our current focus is on learning more coordinated and long-term strategies through language, demonstration, and experimentation. We believe that a key benefit of this weak-method for applying learned knowledge will be that as it learns to operationalize higher-level goals, it will be able to short-circuit more of the reasoning. This is a step towards our main research goal of long-term, high-performance learning.

Acknowledgements

This material is based upon work supported by the Air Force Office of Scientific Research under Award No.FA2386-10-1-4128.

References

- Branavan, S.R.K, Silver, D. & Barzilay, R., (2012). Learning to Win by Reading Manuals in a Monte-Carlo Framework. *Journal of Artificial Intelligence Research*, 43, 661-704.
- Forbus, K. D. (1984). Qualitative process theory. *Artificial Intelligence*, 24, 85–168.
- Forbus, K., Klenk, M., & Hinrichs, T. (2009). Companion Cognitive Systems: Design Goals and Lessons Learned So Far. *IEEE Intelligent Systems*, 24, no. 4, pp. 36-46, July/August.
- Goel, A. & Jones, J. (2011). Metareasoning for Self-Adaptation in Intelligent Agents. In M. T. Cox & A. Raja (Eds.), *Metareasoning: Thinking about Thinking*. Cambridge, MA: MIT Press.
- Hinrichs, T. & Forbus K. (2012) Learning Qualitative Models by Demonstration. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence* (pp. 207-213), Toronto, CA.
- Hinrichs, T. & Forbus, K. (2012). Toward Higher-Order Qualitative Representations. In *Proceedings of the 26th International Workshop on Qualitative Reasoning*, Playa Vista, CA.
- Langley, P., Simon, H.A., Bradshaw, G.L. & Zytkow, J.M. (1987). *Scientific Discovery: Computational Explorations of the Creative Processes*. Cambridge, MA: MIT Press.
- Newell, A. (1962) Some Problems of Basic Organization in Problem-Solving Programs. RAND Memo RM-3283.
- Rao, A. S. & Georgeff, M. P., (1995). BDI Agents: From Theory to Practice. In *Proceedings of the First International Conference on Multi-Agent Systems* (pp. 312-319).
- Shivashankar, V., Kuter, U., Nau, D., & Alford, R. (2012). A Hierarchical Goal-Based Formalism and Algorithm for Single-Agent Planning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*.

Situation Awareness for Goal-Directed Autonomy by Validating Expectations

Michael Karg

KARGM@IN.TUM.DE

Institute for Advanced Study, Technische Universität München, Lichtenbergstrasse 2a, D-85748 Garching, Germany, <http://hcai.in.tum.de>

Alexandra Kirsch

ALEXANDRA.KIRSCH@UNI-TUEBINGEN.DE

Department of Computer Science, University of Tübingen, Sand 14, D-72076 Tübingen, Germany, <http://www.hci.uni-tuebingen.de>

Abstract

Robots that are supposed to work in everyday environments are confronted with a wide variety of situations. Not all such situations can be taken into account by a programmer when implementing the system. This is why robots often show strange behavior when they encounter situations that their programmer has not expected. We propose a knowledge-based approach to explicitly represent *expectations* in the robot program. Comparing those expectations to the current situation allows the robot itself to detect unusual situations and react appropriately. Our general framework can incorporate expectations from different knowledge sources and offers a flexible combination of different expectations. We demonstrate the feasibility of the approach in the context of a household robot in simulation. Finally we discuss the adequacy of our proposed solution and open questions for further research.

1. Introduction

Goal-directed agents must be able to detect situations that require them to generate new goals or change existing goals. For example, one component in the framework of goal-driven autonomy (GDA) (Molineaux, Klenk, and Aha, 2010) is a discrepancy detector that compares expectations to the current situation in the world. The output of the discrepancy detector is used to generate an explanation, which then generates new goals.

In this paper, we introduce a general framework for detecting anomalies by comparing expectations to the situation in the world. We represent expectations explicitly in the robot program, using different knowledge sources. Figure 1 illustrates some categories of expectations: In the left picture there are objects floating in the air. This clearly violates the laws of physics and would contradict a naive physics reasoner. The human lying on the floor obeys the laws of physics, but is unusual behavior. The image on the right shows a less surprising situation, even though a pile of boxes on a table might not be a very normal thing in a household while they would be expected in a warehouse. In general, unexpected situations can have different causes including failures in the robot program (caused by inaccurate sensing or unreliable action execution); unexpected human behavior, such

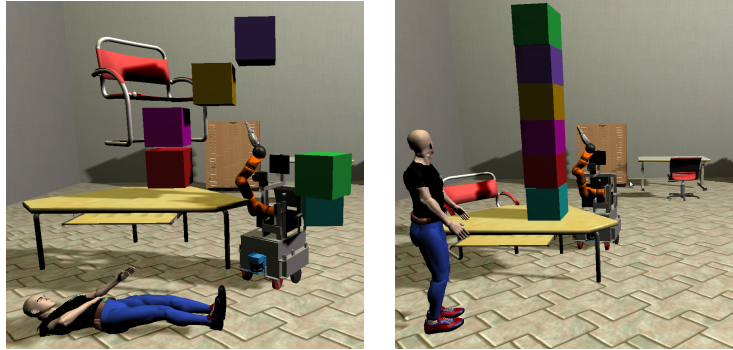


Figure 1. Left: An "abnormal" simulated scene in human robot interaction. Right: A "normal" simulated scene in human robot interaction.

as dropping objects or forgetting things (the latter is especially relevant in the care of people with dementia); and events in the environment such as someone calling at the door or a storm coming up. We propose a framework that combines different kinds of knowledge to represent different classes of expectations that can differ according to the specific application. Promising candidates to provide such knowledge are naive physics reasoning (Akhtar and Kuestenmacher, 2011), simulation-based projection (Kunze et al., 2011), geometric reasoning (Mösenlechner and Beetz, 2013) or knowledge based inference (Kunze, Tenorth, and Beetz, 2010). Expectations can also arise from the execution context: If the robot uses a planner, the effects of the actions are expectations of properties of the world after executing the action. Each of these methods can detect certain kinds of anomalies in the specific situation, but none of them covers all possible sources of surprise.

An unusual situation is not necessarily an error or a failure. Some events may not require any reaction of the robot at all (when the doorbell rings and an inhabitant of the household opens the door, the robot has no need for action). Others may require clarification actions by the robot such as asking the human lying on the floor what is wrong. In this paper, we do not deal with the explanation of the observed discrepancy or appropriate reactions. However, the framework represents all expectations in a normality tree that can serve as a starting point to analyze the found discrepancies and to generate or change goals.

We first present related work of automatic detection of failures or surprise in general. Then we present our framework, in particular options for representing expectation models and how to combine different expectations. After that we demonstrate the feasibility of the approach for a household robot in a simulated world. The paper ends with a discussion of the benefits and limits of our approach, pointing out further research questions.

2. Related Work

General, formal approaches for failure detection in technical systems follow a similar aim as our proposed expectations framework and include research from model-based programming (Struss, 2008) and discrete control theory (Kelly et al., 2009). In both approaches, the system is described

by a formal model like a state automaton with probabilistic transitions or Petri nets. By explicitly modeling failures, a system can avoid such states or diagnose the reasons for encountering a failure.

In the field of robotics, such approaches are common for the diagnosis of internal faults of single components of a robot. Gerald Steinbauer (2005) introduces *observers* to perform model based diagnosis on robot components without affecting the control system, while Kuhn et al. (2008) propose the paradigm of pervasive diagnosis to simultaneously enable active diagnosis and model based control. Williams et al. (2003) use model based autonomy to enable autonomous systems to be aware of their states and possible errors in uncertain environments. They use models of the nominal behavior of a system as well as models of common failure modes to perform extensive reasoning and recognize and recover from failures. While the approaches mentioned so far consider only internal faults of technical systems, Akhtar and Kuestenmacher (2011) use qualitative reasoning on naive physics concepts for diagnosis for the prediction of external faults of autonomous systems.

However, these approaches are not suitable to failures in the high-level behavior of autonomous service robots. First, defining the interaction of a human and a robot in a finite state machine would mean a huge modeling effort. Whereas in discrete control theory the formal model can often be extracted directly from the system specification (which is usually given in a work-flow programming language), such an automaton would have to be hand-coded in the case of a robotic system interacting with humans. And this would mean that a designer would have to take all possible failure situations into account when modeling the behavior. Even more important, it is often impossible for a robot to decide whether it is in a failure state. A typical failure state would be “human has abandoned task”. But the fact that a person has left the room can mean anything from fetching a necessary object to abandoning the joint task. And modeling on a direct observation-level would lead to extremely complex models, which would have to take into account the situational context.

Therefore, we propose to detect failures without an explicit model of failure states. Similar to the models of nominal behavior of technical systems of Williams et al. (2003), we define a potential failure as all observations that are not compliant with the robot’s experience and knowledge about how the world and in particular humans should behave. Combining different evidences and considering the degree of divergence from the robot’s expectations prevents the robot from being overcautious. Work on such an explicit use of expectations has been done by Minnen et al. (2003). They use extended stochastic grammars to introduce constraints into activity recognition to recognize a human playing “Towers of Hanoi” from video data by analyzing object interaction events. They find that humans have strong prior expectations about actions in activities and technical systems performing activity recognition can benefit from explicit models of expectations about high-level activities. Another example for the use of expectations in autonomous robots can be found in (Maier and Steinbach, 2010). Here Maier and Steinbach generate expectations to enable a mobile robot to detect unexpected scenes from video data. They use a dense map of images of the robot’s environment and comparisons of luminance and chrominance values of the images at different times. This enables them to detect changes in the robot’s environment and make assumptions about the expectations and uncertainties in the environment. Kurup et al. 2012 propose that cognitive systems can benefit from constantly generating expectations and matching them against observations to be able to react when discrepancies are detected. They evaluate their system in the

ACT-R cognitive architecture and create expectations about the tracks of pedestrians that cross an intersection.

These approaches work well for diagnosing faults in specific domains. But to our knowledge there is no approach that incorporates a combination of different models, including of humans, to enable robots performing diagnosis on cooperative everyday tasks.

3. Expectations Framework

We define an expectation as any piece of knowledge that describes normal mechanisms of the world, such as physical laws, robot behavior or habitual human behavior. The *normality* of a situation is assessed by comparing each expectation with the current situation and combining the single values of experience matching into one overall value.

Our framework offers a common interface for all expectation and implement a validation method that returns a normality value between 0 and 1. A normality value of 1 means that the expectation is met perfectly in the current situation and a value of 0 means that the expectation is currently not at all fulfilled. Validation of expectations can be triggered at different points in time depending on the type of expectation. Imagine for example expectations about physical models in contrast to expectations about future locations of a person. While it makes sense to validate physical constraints continually, we can only validate the expectation that the human will go to the refrigerator next as soon as we know if he/she actually went to the refrigerator.

All expectations are stored in an *expectations pool* that can be modified dynamically. Depending on the context and the progression of the situation, expectations can be added, adapted and removed. For example, when a user enters the room, the robot should add expectations about human behavior, which can be removed when the person has left again.

Similar expectations can be grouped in categories, which are treated as composite objects with the same interface as single expectations. The normality value of a category is computed recursively from the normality values of all the experiences in that category. One such category could be “Human Activity Expectations” to group several expectations about human activities and get an impression about how well the human actions observed so far generally fulfill the expectations of the robot. The introduction of expectation categories as well as the different types of expectations themselves strongly depend on the scenario of application.

3.1 Expectation Models

We already mentioned that the validation method of an expectation returns a value between 0 and 1 to express the degree to which expectation is fulfilled. For our household robot application we differentiate between logical, temporal and probabilistic expectation representations. *Logical expectations* consist of logical propositions and return 1 on validation if the proposition is evaluated to be true and 0 otherwise. “Cup on table”, for example is a logical expectation that is either true or false. *Temporal expectations* extend logical expectations by a duration during which an expectation is expected to hold. As long as the duration during which the described fact holds is smaller than the expected duration, the validation of such an expectation returns 1. When the duration exceeds the expected duration, the normality value starts decreasing. We define the decrease of the normality

linearly by default, although one might as well think of other functions depending on the intended application. One could for example think of modeling the time that a human is standing in front of a drawer when picking up objects from it as a temporal expectation. *Probabilistic Expectations* model situations where we expect multiple possibilities to be likely to happen, some possibly more probable than others. Probabilistic expectations have probability distributions over random variables instead of binary propositions assigned to them. The probability distribution assigns a probability to each event describing how likely we expect single events to occur. One might for example expect a human to clean the table after breakfast or directly leave the room without cleaning (when he is in a hurry) while the former is more likely than the latter. Validation of a probabilistic expectation will return a value between 0 and 1 corresponding to the probability that was assigned to the expectation that was observed to be true. The implications of such a validation are discussed in section 5.

3.2 Expectation Validation

We assume that the overall normality of a situation results from the combination of all expectations in the expectations pool. We construct a *normality tree* of expectations values by recursively validating all known expectations. The normality value of an composite expectation class is computed by recursively validating and combining all single expectations contained in it. The framework leaves the method for combining these values open. For our trials we have so far simply averaged the values from the child nodes. But other methods such as weighted sums, thresholds or maximum are also conceivable. Also the depth of the hierarchy is not restricted, since classes of expectation classes can be defined. In our experiments we always used three layers as shown in the exemplary normality tree in Figure 2. Here, a situation is observed that mostly fulfills the robot's expectations and thus leads to the high overall normality of 0.95. In the example, the robot's expectations about the objects of interest are fulfilled since the table did not move and it has detected the cup on the table as expected, resulting in a normality value of 1 for the expectations about the objects. Also the validation of expectations about the human, consisting of two probabilistic expectations, return the values of 0.92 and 0.85, resulting in an average normality of 0.89 for the human normalities. The normality tree is updated as soon as one of the validated expectations changes its value so the robot will obtain an estimation of the normality of the current situation after every new event for which an expectation exists.

The normality tree allows a limited, but general way of finding the cause of surprising events. So when the overall normality value drops below a certain threshold, the robot should consider taking some action. By traversing the normality tree it can diagnose which of its expectations has been violated and act accordingly, for example by further clarifying the situation or taking immediate action.

4. Application: Expectations for a Household Robot

To evaluate the applicability of our expectations framework we set up our framework in a simulated apartment scenario using different types of expectations. The simulated apartment environment

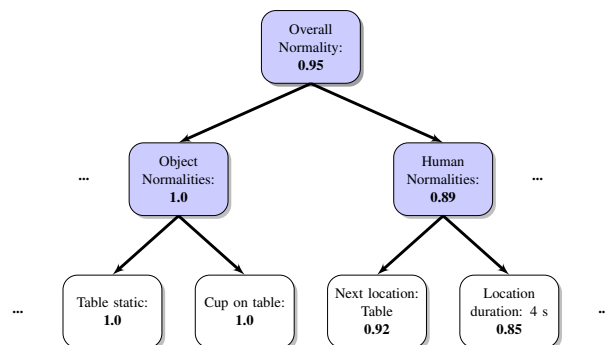


Figure 2. A normality tree generated by the validation of different types of expectations. The highest level of the tree describes the overall normality of the situation whereas the lower levels allow for a more fine-grained description of the normalities of specific expectations. Blue nodes represent normalities generated from composite expectations, white nodes correspond to normalities from single expectations.

includes a person and a PR2 robot. We use the realistic physical simulator MORSE¹ that includes a human avatar that can be controlled like in modern 3D computer games (Lemaignan et al., 2012) and enables a user to perform pick- and place actions, open doors, cupboards and drawers and operate switches in the simulated scenario. The robot is equipped with a simulated object detection sensor that returns the name of objects that are within the field of view of the robot as well as their positions. Furthermore, the robot is able to detect if doors are open or closed and the positions of humans, while it is unable to distinguish between different persons. The robot moves using the ROS-based 2D navigation “move_base”². A video of the simulated scenario is available online³.

Our household robot — having nothing else to do — is guarding the apartment while the human is sleeping. It constantly patrols several locations to check if everything is as expected and it stays at each location for two seconds. We use five expectations in this scenario: The TV is expected to be in the living room and it is expected not to move. Humans are not expected to be anywhere else but in the bedroom or outside of the house. The entrance door of the apartment is locked and the robot navigation works normally. For the robot navigation, we generate a temporal expectation each time the robot starts moving to another waypoint and queries the navigation path planner for a new path plan. It then estimates the time to the location using the length of the path plan and its average speed and generates a temporal expectation. The expectation gets removed when a goal point is reached, meaning that during the two seconds where the robot is standing still at each location, only the four expectations mentioned before are active.

In this experiment, we use the human avatar to simulate a burglar that enters the apartment during the robot’s patrol. He opens the entrance door, passes the hallway to get into the living room, picks up the TV and passes the hallway again, carrying the TV out of the apartment. The simulated burglary is illustrated in Figure 3. The average normality of the robot during this burglary plotted over time is shown in Figure 4. The normality drops as soon as the robot detects the open

1. <http://www.openrobots.org/wiki/morse>

2. http://wiki.ros.org/move_base

3. <http://vimeo.com/hcai/expectations>



Figure 3. The patrol robot scenario. Left: The robot detects the open door and the human in the hallway. Middle: The robot detects the TV moving and not being in the living room. Right: The burglar has left the apartment with the TV.

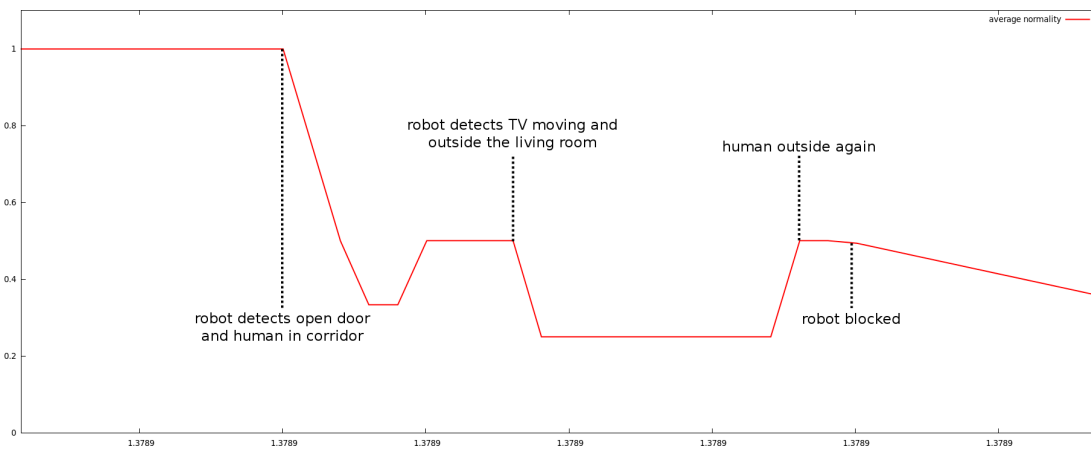


Figure 4. Average normality of the patrol robot scenario plotted over time.

entrance door and the human in the hallway. It drops even further when it detects that the TV is not in the living room any more and has moved since the last detection. However, the normality never drops to 0 since the navigation of the robot is still working correctly. When leaving the apartment with the TV, the burglar had to pass the robot really closely, getting into the sensor range of the simulated laser scanner thus the robot's navigation is blocked due to security reasons. This caused the navigation to take longer as expected and the temporal expectation starts decreasing linearly towards the end.

5. Discussion

The proposed framework of expectations along with their validation is designed to offer a general, modular and knowledge-based way to monitor and react to unusual situations. We make no assumptions about the knowledge used for representing expectations except that it can be used to quantify the normality of a situation. With the modular definition of expectations the framework can be used and extended for arbitrary applications. The combination of normality estimates in the normality tree provides some guidance towards suitable reactions. By exploiting knowledge that the robot may

also need for its decision making or state estimation, the recognition of unusual situations can be done by the robot itself, thus freeing engineers from the burden of considering every single situation the robot might encounter.

One open question is how to get the necessary expectations. The number and types of expectations a robot needs to identify abnormal situations largely depends on the type of tasks it is designed to perform. While for example a vacuuming robot can benefit from expectations about the humans' future locations in order to avoid vacuuming in close proximity to the human, expectations about the location of a cup will be of no interest for such a robot. In contrast, a more elaborate household robot that is designed for household activities like setting the table, expectations about the location of a cup can prove useful. In our current application scenarios, we partly manually define expectations that seem useful in specific situations. We also use learned models about the expected behavior and future locations of humans performing different activities and give an example about how we can use learned models about human activities in combination with activity recognition to generate and validate expectations about human task performance. Another way of generating expectations is by inference using common sense knowledge. Tenorth et al. (2010), for example, use Knowledge-Linked Semantic Object Maps in the knowledge processing system KnowRob to infer likely storage locations of objects in a kitchen which can be modeled as expectations.

No matter how expectations are defined and represented, a common problem is the comparison of real-world sensor data with the expectations. For intelligent behavior, the most useful expectations are defined on abstract levels, giving the robot some global insights in what is happening in the world. However, such abstract information is hard to extract reliably from sensor data. So comparing the current activity of a user to some expected activity requires a stable recognition of human actions and also a representation of acceptable variations. Expectations that are represented on the level of sensor data are easier to evaluate, but it is more difficult to define a baseline expectation. Even though humans can easily recognize activities of other humans, they would hardly be able to predict every movement of a person. Thus, expectations defined on sensor values will almost always be inaccurate and thus expectations can hardly be met.

The uncertainty of expectations also leads to an implicit weighting factor. When using probabilistic expectations, the normality factor can at most be the probability of the most likely event. So when the robot considers several next actions of the user as equally likely, the normality should intuitively be 1 if one of those actions is observed. However, because of the uncertainty in the expectation, the normality value will get a small value.

The choice of the combination function for normality values in general is an open question and it is not clear whether our proposed representation in the normality tree is enough to cover all necessary interactions of normality values. We chose the average as the combination function, which can be justified by the tallying heuristic of humans, where weights are ignored for decision making (Gigerenzer and Gaissmaier, 2011). However, in some cases weights may be necessary and some kind of non-linear combination may be appropriate. For example when expectations are violated that imply a state of danger, just averaging may mean that the overall normality value is not affected strongly enough. So when the robot detects smoke it should not wait until its overall averaged normality factor drops below a threshold, but act immediately. One way of achieving such a shortcut is by not only monitoring the overall normality value, but also single critical values.

Our expectations framework only considers one aspect of goal-directed autonomy. It leaves open the question of what to do when expectations are not met. The normality tree offers a straightforward way for diagnosis by identifying those branches of the tree that have a low normality value. One way of generating reactions would be to associate with each node in the normality tree a module in the robot program that is not only responsible for evaluating the normality, but also to offer actions for re-establishing normality. For example, one such component may be a task planner. It can monitor whether the effects of its actions are indeed the expected ones. If this is not the case, it can generate a new plan that considers the changed circumstances.

Such a local reconsideration of goals is not always possible. Sometimes the normality value drops because several modules detect small deviations from expectations, which would not be alarming in isolation, but which sum up to an overall low normality. For example if the user performs certain activities more slowly than normally and drops objects more often, the single deviations from the normality would not require any action. But the combination of these deviations may indicate some deterioration in the user's health and should be reacted to. To deal with such complex cases, additional reasoning mechanisms and knowledge would be required.

Even without being able to deal with all situations autonomously, the pure recognition of unexpected situations can make robots safer and more reliable for realistic tasks. When not understanding the situation, a robot could prompt the user for help or access some external help desk, where operators could either provide the robot with additional knowledge to understand the situation or manually set robot goals, so that the situation will get back to normal.

In all, our framework is a first step towards a general treatment of failures and unusual situations by autonomous robots. It can be easily integrated with traditional methods of failure recognition and handling, which is still the most robust way to handle frequent failures or dangerous situations. But our knowledge-based approach can cover the large variety of situations that an engineer would not think of or for which specific coding would be too costly.

6. Conclusion

This paper presented a promising first step towards a general framework that enables robot assistants to measure the normality of situations by validating a combination of different expectations. We showed how different types of expectations can be created, grouped and validated, giving a robot an impression of the overall normality of the situation as well as the normality of different categories or single expectations. In a simulated apartment, we showed how a robot can use a variety of expectations to detect situational anomalies and we illustrated how it can get a continuous impression of the normality of human behavior by observing motion tracking data in a sensor equipped kitchen. Future work includes other combinations of normality values as well as improved validation techniques for probabilistic expectations.

Acknowledgements

With the support of the Technische Universität München - Institute for Advanced Study, funded by the German Excellence Initiative, and the Bavarian Academy of Sciences and Humanities.

References

- Akhtar, N., and Kuestenmacher, A. 2011. Using naive physics for unknown external faults in robotics. In *22nd International Workshop on Principles of Diagnosis (DX-2011)*, volume 1, 23.
- Gigerenzer, G., and Gaissmaier, W. 2011. Heuristic decision making. *Annual Review of Psychology* 62:451–482.
- Kelly, T.; Wang, Y.; Lafortune, S.; and Welsh, M. 2009. A formal foundation for failure avoidance and diagnosis. (HPL-2009-203).
- Kuhn, L.; Price, B.; de Kleer, J.; Do, M.; and Zhou, R. 2008. Pervasive diagnosis: Integration of active diagnosis into production plans. In *proceedings of AAAI*.
- Kunze, L.; Dolha, M. E.; Guzman, E.; and Beetz, M. 2011. Simulation-based temporal projection of everyday robot object manipulation. In Yolum; Tumer; Stone; and Sonenberg., eds., *Proc. of the 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*. Taipei, Taiwan: IFAAMAS.
- Kunze, L.; Tenorth, M.; and Beetz, M. 2010. Putting People’s Common Sense into Knowledge Bases of Household Robots. In *33rd Annual German Conference on Artificial Intelligence (KI 2010)*, 151–159. Karlsruhe, Germany: Springer.
- Kurup, U.; Lebiere, C.; Stentz, A.; and Hebert, M. 2012. Using expectations to drive cognitive behavior. In *AAAI*.
- Lemaignan, S.; G., E.; Karg, M.; Mainprice, M.; Kirsch, A.; and Alami, R. 2012. Human-robot interaction in the morse simulator. In *Proceedings of the 2012 Human-Robot Interaction Conference (late breaking report)*.
- Maier, W., and Steinbach, E. 2010. A probabilistic appearance representation and its application to surprise detection in cognitive robots. *IEEE Transactions on Autonomous Mental Development* Vol. 2, No. 4, pp. 267 - 281.
- Minnen, D.; Essa, I.; and Starner, T. 2003. Expectation grammars: Leveraging high-level expectations for activity recognition. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, II–626. IEEE.
- Molineaux, M.; Klenk, M.; and Aha, D. 2010. Goal-driven autonomy in a navy strategy simulation. In *AAAI Conference on Artificial Intelligence*.
- Mösenlechner, L., and Beetz, M. 2013. Fast temporal projection using accurate physics-based geometric reasoning. In *IEEE International Conference on Robotics and Automation (ICRA)*.
- Steinbauer, G., and Wotawa, F. 2005. Detecting and locating faults in the control software of autonomous mobile robots. In *19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, 1742–1743. Citeseer.
- Struss, P. 2008. Model-based problem solving. 395–465.
- Tenorth, M.; Kunze, L.; Jain, D.; and Beetz, M. 2010. KNOWROB-MAP – Knowledge-Linked Semantic Object Maps. In *Proceedings of 2010 IEEE-RAS International Conference on Humanoid Robots*.
- Williams, B.; Ingham, M.; Chung, S.; Elliott, P.; Hofbaur, M.; and Sullivan, G. 2003. Model-based programming of fault-aware systems. *AI Magazine* 24(4):61.

HALTER: Hierarchical Abstraction Learning via Task and Event Regression

Ugur Kuter

UKUTER@SIFT.NET

SIFT, LLC, 211 North 1st Street, Suite 300, Minneapolis, MN 55401-2078

Hector Munoz-Avila

MUNOZ@LEHIGH.EDU

Lehigh University, Dept. of Computer Science & Engineering, 19 Memorial Drive West, Bethlehem, PA 18015-3084

Abstract

In this paper we present HALTER, an algorithm capable of learning task-subtasks from input action traces. HALTER iteratively learns how task subtraces can be folded to achieve tasks of increasing level of abstraction. HALTER also learns applicability conditions to determine when a task can be decomposed into a task subtrace. We examine the expressiveness of HALTER and observe this is the first HTN learning algorithm capable of learning nonregular languages.

1. Introduction

One of the basic questions agents face when interacting in an environment is the goal formulation question: what goals should it try to achieve next to achieve some over-arching goals? the answer depends on multiple factors such as the underlying planning paradigm (i.e., if the planner is totally-ordered it needs to consider how achieving goals might interact with the current partial plan generated) and the overarching goals or objectives (i.e., whether achieving a goal will benefit the overall goals or objectives). Planning paradigms offer a variety of answers to the question. For example, heuristic planning techniques may estimate the effort to reach a goal state from a state achieving an intermediate goal (i.e., selecting the state that has less estimated effort). Landmark planning will identify intermediate facts that must be fulfilled regardless of the solution (e.g., going through a choke point when navigating out of a maze).

This paper presents a procedure for learning hierarchical task networks (HTNs). Hierarchical task network (HTN) planning side-steps reasoning about concrete goals and instead reasons on how to achieve tasks. Plans are generated in a top-down manner by decomposing complex tasks into simpler ones, which are themselves recursively decomposed until so-called primitive tasks corresponding to actions are generated. In our work, we view tasks as goals of increasing level of abstraction. This view is not unusual: ICARUS (Langley & Choi, 2006a) and works combining HTN and STRIPS planning (Shivashankar et al., 2013; Kambhampati & Srivastava, 1995) all use HTN planning formalisms where the tasks are STRIPS goals. Unlike STRIPS planners where the planner automatically finds the goal-subgoal relations as part of the planning process, in HTN planning's users must define these goal-subgoal relations manually. Hence, we view our work on learning

HTNs in the same vein as existing work on *learning goal formulation knowledge* (e.g., (König & Laird, 2006; Jaidee, Muñoz-Avila, & Aha, 2011)).

HTN learning has been a recurrent research topic. There are two camps for HTN learning research: those that use existing HTNs and generalize them to be reused in different situations and those that learn the HTN structures from scratch (i.e., from input plans). The former is motivated by domains such as project planning where the project plans, i.e., hierarchy of steps that were followed to organize an event, are readily available. The latter is motivated by domains where only the actions are available but not the hierarchy that could lead to generate them. We will expand on this discussion in the related work section.

We present HALTER (Hierarchical Abstraction Learning via Task and Event Regression). HALTER bridges these two camps. It can exploit existing HTNs, when available, to generalize and reuse in new situations. But it can also learn new task decompositions, not existing in any previously seen HTN, to acquire new knowledge that can be used to generate plans which cannot be generated with the current HTN knowledge.

2. Basics

We use the usual definitions for HTN planning as in Chapter 11 of (Ghallab, Nau, & Traverso, 2004). A *state* s is a collection of ground atoms. A *planning operator* is a 3-tuple $o = (h, pre, del, add)$, where h (the *head* of the operator) is a logical expression of the form $(n\ arg_1 \dots arg_k)$ such that n is a symbol denoting the name of the operator and each argument arg_i is either a logical variable or constant symbol. The *preconditions*, *delete list* and *add list* of the planning operator, pre , del , and add respectively, are logical formulas over literals.

An *action* a is a ground instance of a planning operator. An action is *applicable* to a state s if its preconditions hold in that state. The result of applying a to s is a new state $\gamma(s, a) = (s \setminus del) \cup add$. A *plan* π is a sequence of actions.

A *task* is a symbolic representation of an activity in the world, formalized as an expression of the form $(t\ arg_1 \dots arg_k)$ where t is a symbol denoting the name of the activity and each arg_i is either a variable or a constant symbol. A task can be either *primitive* or *nonprimitive*. A primitive task corresponds to the head of a planning operator and denotes an instantaneous action that can be directly executed in the world. A nonprimitive task cannot be directly executed; instead, it needs to be decomposed into simpler tasks until primitive ones are reached.

Each nonprimitive task t is associated with two *task indicators*, $start_t$ and end_t , indicating the starting and ending points of the nonprimitive task t in a plan trace. A *primitive task trace* is a sequence $\langle t_1, t_2, \dots, t_k \rangle$ where each t_i is either a primitive task or a task indicator. A *nonprimitive task trace* may contain primitive and nonprimitive tasks. If the starting indicator for a task t occurs in a task trace, its ending task indicator must appear later in the task trace. These task indicators are primitive tasks but do not reflect any action in the domain.

We assume a *failure* in a task trace is designated by identifying a primitive task (i.e., an action) as *failed* in that trace. There are multiple reasons why an action in a trace fails. Potential reasons for action to fail include: (1) the preconditions of the action are not applicable in the state resulting after the previous action in the trace was applied, and (2) applying the action sends the

remaining actions in the trace towards a state that will not fulfill the goals. An example of the latter is a navigation domain where the action turns a vehicle in the wrong direction and the vehicle ends in a different location from the destination location. Had the action been to turn in a different direction, the vehicle would have reached the destination by following the remaining actions in the trace.

More formally, we say that given a set of goals, g and a primitive task trace τ , τ is a failed trace if executing τ from the state s_0 yields a state that doesn't satisfy one or more of the goals in g .

In this paper, we restrict ourselves to the Ordered Task Decomposition formalism of HTN planning (Nau et al., 1999). In this formalism, tasks are executed in the order they are achieved. An *HTN method* is a procedure that describes how to decompose nonprimitive tasks into simpler ones. Formally, a *method* is a triple $m = (h, pre, subtasks)$, where h is a nonprimitive task (the *head* of the method), pre is a logical formula denoting the *preconditions* of the method, and $subtasks$ is a totally-ordered sequence of *subtasks*. A method m is *applicable* to a state s and task t if the head h of the method matches t and the preconditions of the method are satisfied in s . The result of applying a method m to a state s and task t is the state s and sequence of *subtasks*.

An *HTN planning problem* is a 4-tuple $P^h = (\mathcal{T}, s_0, T, O, M)$, where \mathcal{T} is the finite set of all possible tasks, s_0 is the initial state, T is the initial sequence of tasks, and O and M are sets of planning operators and methods respectively. A *solution for the HTN planning problem* P^h is a plan (i.e., a sequence of actions) π that, when executed in the initial state, performs the desired initial tasks T . If there is a solution for the HTN planning problem P^h , then P^h is *solvable*.

A *learning example* is a tuple of the form (s_0, τ, g) where s_0 is a state, τ is a primitive task trace, and g is a set of goals, such that $\gamma(s_0, \tau) \models g$. An *HTN learning problem* is a 4-tuple $P = (T, O, M, L)$, where T is the finite set of tasks and task indicators, and O and M are sets of planning operators and (possibly empty) initial set of methods respectively. L is the set of learning examples.

A *solution* to an HTN learning problem $P = (T, O, M, L)$ is a set M' of methods such that $M \subseteq M'$ and for every state s and goal g , any plan π generated by a provable correct HTN planner is a solution for (s, T, O, M') and $\gamma(s, \pi) \models g$.

3. Algorithms

HALTER is an incremental learning algorithm that produces a knowledge base of HTN methods from an input solution plan to a classical planning problem and successively updates its knowledge base when presented with solutions to new classical planning problems in the same planning domain.

The basis for the high-level HALTER algorithm is a variant of the well-known *chart parsing* technique (Charniak, Goldwater, & Johnson, 1998). A chart parser is a type of parser suitable for ambiguous grammars (including grammars of natural languages). It uses a *dynamic programming* approach - partial hypothesized results are stored in a structure called a chart and can be re-used. This eliminates backtracking and prevents a combinatorial explosion in the search space since the parser does not generate the same grammar rules multiple times in different parts of the search space.

Algorithm 1: A high-level description of the HALTER learning procedure.

```

1 Procedure HALTER( $\mathcal{T}, O, M, L$ )
2 begin
3   foreach learning example  $(s, \tau, g) \in L$  do
4      $plan\_tree \leftarrow Learn\_Structure(\mathcal{T}, s, \tau, g)$ 
5      $M' \leftarrow Learn\_HTN\_Methods(plan\_tree, s, g, M)$ 
6      $M \leftarrow Inductively\_Generalize(M')$ 
7   return  $M$ 
8 end

```

Algorithm 2: A high-level description of structure learning in HALTER.

```

1 Procedure LEARN_STRUCTURE( $T, s, \tau, g$ )
2 begin
3    $plan\_tree \leftarrow \emptyset$ 
4    $task\_trace \leftarrow \tau$ 
5   loop until no new task can be learned
6      $X \leftarrow \emptyset$ 
7     foreach task in  $T$  do
8        $children \leftarrow \text{find children of task in } task\_trace$ 
9        $\text{insert}(task, children)$  into  $X$ 
10     $task\_trace \leftarrow X$ 
11     $\text{insert } task\_trace$  into  $plan\_tree$ 
12  return  $plan\_tree$ 
13 end

```

Algorithm 1 shows a high-level description of the HALTER algorithm. For each learning example given to the algorithm, HALTER first learns a task hierarchy from that learning example. The task hierarchy for the example includes all of the primitive tasks that are specified by the task trace of the example, the nonprimitive tasks from \mathcal{T} that can be inferred from the example, and the parent-child relationships among those nonprimitive and primitive tasks.

Algorithm 2 shows a high-level description of structure learning in HALTER that produces a task hierarchy from a learning example. In the initial iteration, the procedure starts from the primitive task trace and infers all of the nonprimitive tasks such that all of the children of each such nonprimitive task is in the task trace. Then, the algorithm replaces those children with their nonprimitive parents. This creates the next task trace from which the algorithm repeats the above process. The learning process terminates when there is no new nonprimitive task for which the algorithm can learn children.

Once the task hierarchy that corresponds to input learning example is learned, HALTER uses this task hierarchy in order to learn new HTN methods. The procedures for learning HTN methods depends on whether the learning example was positive or negative; thus we discuss each separately in the subsequent sections.

HALTER

HALTER uses a particular type of chart parsing for learning HTNs. It infers task hierarchies from the trace, in a layer by layer fashion. Given a task trace (see Algorithm 2): $\langle t_1, t_2, \dots, t_k \rangle$, it performs the following steps:

- Line 7 of Algorithm 2 replaces subtraces in the task layer $\langle t_1, t_2, \dots, t_k \rangle$ with their parent tasks. This creates a new task trace $\langle t'_1, \dots, t'_n \rangle$ (with $n \leq k$) where each t' is either one of tasks t_k or is parent task for a subtrace. That is, by replacing a subtrace $\langle start_{t'}, \dots, end_{t'} \rangle$ with t' . Hence, $\langle t'_1, \dots, t'_n \rangle$ may contain both primitive and nonprimitive tasks.
- HALTER recursively repeats the task replacement process with $\langle t'_1, \dots, t'_n \rangle$. The structural learning process terminates when pre-specified top-level tasks are reached, or no new non-primitive tasks can be inferred.

The hierarchical task structure inferred is a tree. Each interior node in the tree is a nonprimitive task t' and its children are the tasks in the subtrace that t' folds. The leaves are the primitive tasks in the input trace $\langle t_1, t_2, \dots, t_k \rangle$.

To learn the set of preconditions P of a method, HALTER uses *goal regression* over a task trace $\langle t'_1, \dots, t'_n \rangle$ by recursively constructing P , which is initially empty. Starting from the last task (i.e., $k = n$) in the trace and ending in the first task (i.e., $k = 1$), for each task t'_k it removes from P any effect added by t'_k and adds to P the preconditions of t'_k .

Once HALTER learns preconditions and effects for each nonprimitive task in the hierarchy, it generates HTN methods from them. At this point, however, the HTN methods generated for each nonprimitive task are still ground; i.e., they only contain planning domain objects as arguments for the task, its subtasks and for the precondition predicates. HALTER uses similar inductive generalization techniques as in HTN-Maker (Hogg, Muñoz-Avila, & Kuter, 2008) in order to generalize the objects into variable symbols, while ensuring the soundness of the method.

Finally, the algorithm returns the set of the generalized methods as the HTN library learned from the input task trace.

As mentioned above, HALTER can also learn HTN methods from failed task traces. Given a learning example (s_0, τ, g) , τ is a failed trace if executing τ from the state s_0 yields a state that doesn't satisfy one or more of the goals in g . The learning process in HALTER assumes an action in the trace has been marked as a failed action. This is used to learn HTN methods that ensure that such a failure will not happen in a planning episode.

We assume that if an action fails, then any action that contributes to the application of that action in the particular state it fails is a potential cause for the failure. In other words, if a fails, and there is another action a' in the trace that appears before a and $\text{eff}(a') \models \text{pre}(a)$, then a' is a potential cause for the failure of a . Starting from the failed action, HALTER performs a variant of goal regression in order to find the earliest possible action a' that is found to be a cause for the failure in the trace. Then, the algorithm uses the task hierarchy learned from successful traces in order to identify the ancestor tasks of the action a' in the method library. The ancestor tasks include an immediate parent of a' in the task hierarchy, the parents of that immediate parent, and so on. For each such ancestor, HALTER learns additional preconditions that captures the reasons why that task failed. For example, suppose the failed action has a precondition p . Then the algorithm infers

a new precondition of the form (*not p*) for the parent task. This additional precondition ensures that the methods for that parent task are not going to be applied in states where *p* is true, preventing the failure from happening.

Once the methods from failed traces are learned, they are added to the method library and the algorithm returns the library.

4. Preliminary Results and Discussion

In this section, we discuss some of the properties of the HALTER learning algorithm.

We define a *classical planning problem* as a 3-tuple $P^c = (s_0, g, O)$, where s_0 is the initial state, g is the *goals* represented as a conjunction of logical atoms, and O is a set of planning operators defined in Section 2. A *solution for the classical planning problem* P^c is a plan $\pi = \langle a_1, \dots, a_k \rangle$ such that the state $s' = \gamma(\gamma(\dots, (\gamma(s_0, a_1), a_2), \dots), a_k)$ satisfies the goals g .

Suppose a set of goal atoms g are specified for some task t . Then, the *HTN-equivalent planning problem* to a classical planning problem $P^c = (s_0, g, O)$ is an HTN planning problem $P^h = (\mathcal{T}, s_0, \{t\}, O, M)$ such that M is a set of HTN methods.

Proposition 1. *Let O be a set of planning operators for a planning domain. Suppose HALTER is given a set of learning examples, and it produces a set M of HTN methods. Then, for any classical planning problem P^c in the domain, a solution π produced by a sound HTN planning algorithm on the HTN-equivalent problem P^h using the methods in M will be a solution to the classical planning problem.*

A comparison with HALTER and its predecessor HTN-Maker is in order. Unlike HTN-Maker (Hogg, Muñoz-Avila, & Kuter, 2008), HALTER can learn both from successful and failed traces (HTN-Maker only learns from positive examples). Furthermore, HALTER does not require semantically-annotated task definitions required by HTN-Maker. This enables HALTER to learn more expressive HTNs compared to those learned by HTN-Maker, whose HTNs can express regular expressions only. The same holds for ICARUS since the tasks are goals, hence its task decomposition structure is equivalent to STRIPS sub-goaling.

In HTN-Maker, an annotated task t describes the preconditions that must hold in the state prior to pursuing to achieve t and the effects in the state after achieving t . Hence, the HTNs learned by HTN-Maker are strictly equivalent to STRIPS planning. Indeed, HTN-Maker can only learn task structures equivalent to regular languages (i.e., right recursive methods). In contrast, HALTER can learn structures that are equivalent to context-free grammars. To see this, assume the following two traces are given as input to HALTER (t is a nonprimitive task and t_1, t_2, t_3 and t_4 are primitive tasks):

$$\langle start_t, t_2, t_3, end_t \rangle$$

$$\langle start_t, t_1, t_2, t_3, t_4, end_t \rangle$$

From the first trace HALTER learns a method decomposing t into $\langle t_2, t_3 \rangle$. From the second trace it will learn a second method decomposing t into $\langle t_1, t, t_4 \rangle$. This means that the second method's structure captures the context-free grammar rule: $t \rightarrow \langle t_1, t, t_4 \rangle$. Hence, repeated application of the method will decompose t into $\langle t_1^n, t, t_4^n \rangle$ for an arbitrary n which is a non-regular

expression. It is impossible to generate such non-regular expression with STRIPS planning. The following theorem states this result.

Proposition 2. *HALTER learns HTN methods that can be used to generate non-regular expressions.*

The ability to learn plans exhibiting such non-regular expressions is important to represent many realistic situations. A simple situation is to plan an NPC in a video game to repeatedly patrol a circuit of locations (e.g., starting and ending in the same location) and to break this patrol process when a particular situation occurs (e.g., the NPC sees an intruder, in which case it sounds an alarm). Another example is planning to control a robotic arm that must distribute some chemical across some containers of equal but not pre-determined size. A third example, is one where we need to plan a gate in a network that distributes messages uniformly across a number of predefined channels until a certain message (e.g., a terminate message) is received that closes the gate.

5. Related Work

Goal-Driven Autonomy (GDA) is a reflective model of goal reasoning that controls the focus of an agent’s planning activities by dynamically resolving unexpected discrepancies in the world state (Muñoz-Avila et al., 2010; Molineaux, Klenk, & Aha, 2010). Current research on goals in GDA are mostly restricted to STRIPS goals representations in which the goal is either a desired state or an atom in the state. However, ARTUE and variants use HTN representations (Molineaux, Klenk, & Aha, 2010; Powell, Molineaux, & Aha, 2011).

One difficulty in GDA research is for the developer the need to manually specify the GDA elements such as actions’ expectations, explanations for discrepancies between the action’s expectations and their actual outcomes, and goal formulation knowledge. Research has been conducted to learn these elements. Notably, Jaidee et al (2011) reports on a system learning some GDA elements. It uses STRIPS representations for the GDA elements. For example, the expectation are the collection of atoms resulting from applying an action (Jaidee, Muñoz-Avila, & Aha, 2011). Another notable system, reported by Weber and Mateas (2012) learn all the main GDA elements although this is restricted to a vector of state-value representations and a representational bias towards increasing these value (Weber, Mateas, & Jhala, 2012). For example, values represent the number of the GDA agent’s own units in a combat-based game; actions are expected to increase the number of these units or at least not to decrease them. Our work will enable learning of HTNs that could be used by GDA researchers.

Hierarchical decompositions representations, such as HTNs, in which complex tasks are decomposed into simpler ones are a common representation paradigm in many cognitive agent architectures (Langley, Cummings, & Shapiro, 2004). SOAR uses a representation of operator hierarchies to capture complex interrelation between goals. Hierarchies in SOAR are used to fill gaps in SOAR’s production rule knowledge. Abstract operators are seen as subgoals that enables the system to move forward towards achieving its goals (Laird, 2012). ICARUS follows a converse idea: it uses HTNs to generate plans and whenever it finds gaps in its HTN knowledge, it falls back to standard (STRIPS) planning to fill this gap (Langley & Choi, 2006b). LIGHT uses a similar mechanism to ICARUS

but enables indexing goals (Nejati, Langley, & Konik, 2006). ACT-R uses production composition to represent hierarchical knowledge (Anderson, 1993); at the bottom level of the hierarchy, when production rules are triggered they generate new goals to achieve. Applicability conditions are evaluated against the current goals the system is trying to achieve. Hence, at the bottom level, triggering production rules is reminiscent of backward goal chaining. At higher levels in the hierarchy, triggering production rules generates new conditions, which in turn can be used to trigger other production rules. Hence, high level production rules capture knowledge about complex relations between the goals. The Companion cognitive architecture aims at developing agents that assist humans in their problem solving efforts (Forbus, Klenk, & Hinrichs, 2008). As such, its HTN planning process is built on top of a truth-maintenance system that enables Companion to automatically build explanations about the chain of inferences that led to a plan. This enables Companion to justify to the user the reasons for decisions made. It also maintains and/or branches that enable exploration of alternative decompositions when needed. The Disciple cognitive architecture uses generalization hierarchies to represent concepts in increasingly high level of generalization (G. & Boicu, 2008). Problem-solving is done in a divide-and-conquer manner by decomposing the problems into simpler ones although it doesn't use HTN or other similar hierarchical planning techniques.

HTN planning origins started with seminal work in the systems NOAH (Sacerdoti, 1977) and NONLIN (Tate, 1977). Part of the recurrent interest in HTN planning is due to the fact that its semantics are well understood (Erol, Hendler, & Nau, 1994), the simplicity of its representation (Nau et al., 1999) and recurrent reports on its applications (Wilkins & Myers, 1998; Currie & Tate, 1985; Nau et al., 2005).

Learning HTNs have been a recurrent research topic over the years. Some research focuses on generalizing given hierarchies. Examples of situations where such hierarchies are available include work-breakdown structures, which represent a hierarchy of steps to produce a one-of-a-kind project and can be edited with commercial-off-the-shelf software such as Microsoft Project. A system in this camp is CAMEL (Ilghami et al., 2005). CAMEL receives as input plans annotated with the intermediate states between the actions and the HTNs that were used to generate these plans. It propagates the intermediate states upwards in the hierarchy to infer the applicability conditions of task decompositions given HTNs. Since the same task decomposition might occur multiples times and the applicability conditions might not be the same, it uses version spaces to obtain the most general conditions that are consistent with the examples generated so far. To ensure convergence to a single set of conditions, it assumes that negative examples are also given. That is, incorrect applicability conditions for the task decompositions. Another algorithm of this kind is DINCAT (Xu & Muñoz-Avila, 2005). Unlike CAMEL, it doesn't assume that negative examples are given. DINCAT uses inductive generalization techniques to infer generalized conditions, but unlike CAMEL, it can't guarantee that the learned conditions are the most specific that are consistent with the given examples. Unlike these systems, HALTER can also learn new hierarchical decompositions.

Other HTN learning systems learn the hierarchical decomposition. One of the early systems in this category is X-LEARN (Reddy & Tadepalli, 1997). It uses bootstrap learning by learning increasingly more complex hierarchies from carefully ordered training examples in the form of plans. ICARUS also learn the task decomposition knowledge (Langley & Choi, 2006b). ICARUS identifies gaps in the HTN knowledge and uses given concepts, represented as Horn clauses, to learn

hierarchies of skills filling this gap. The way this works is by attempting to generate a plan using the current HTN knowledge. When the plan can't be generated, ICARUS uses STRIPS planning techniques to generate the plan. It then carefully analyzes the resulting plan to learn new task decompositions. In ICARUS the tasks correspond directly to goals and, hence, problems solved by using the HTNs can also be generated by using STRIPS planning. However, using HTN planning provides two advantages: (1) it leads to speed-up in problem solving and (2) the HTNs provide a rationale explaining why the plans are generated. Another system in this camp is HTN-MAKER (Hogg, Muñoz-Avila, & Kuter, 2008). Like LIGHT, HTN-MAKER follows a bottom-up process to learn HTNs from given plans. But unlike LIGHT, HTN-MAKER can learn HTN knowledge that can solve problems not expressible in STRIPS planning. Unlike these systems, HALTER can also exploit given hierarchical decompositions when available to learn more general applicability conditions.

6. Final Remarks and Future Work

In this paper, we presented HALTER, an HTN learning algorithm that learns goal formulation knowledge by indicating the subtasks that must be achieved in order to achieve tasks, which are goals of higher level of abstraction. HALTER carefully analyzes how tasks were achieved in input traces and pinpoints parent-child task relations when a subtrace of the former subsumes a subtrace of the latter. HALTER can extract such relations over multiple levels and in any order of the traces. As a result, it can learn hierarchical structures that are strictly more expressive than regular languages. This is the first HTN learner that is capable of learning such expressive HTNs.

For the near future work, we will like to conduct experiments where we compare the performance of the HTN planner SHOP when using the HTN methods learned with HALTER versus when using the HTN methods learned by a state-of-the-art HTN learner such as HTN-Maker (Hogg, Muñoz-Avila, & Kuter, 2008). The experiments will also include comparisons of HTNs learned by HALTER and those used by SHOP2 in the past International Planning Competitions. The latter HTN libraries are written by human SHOP2 users, and thus, would provide a benchmark to evaluate how efficient HALTER can learn HTNs compared to those written by human experts. In these experiments, we are interested in comparing (1) the speed of convergence towards learning a domain (e.g., the percentage of problems that each can solve), (2) the running time speed of SHOP when solving these problems, and (3) the quality of the solutions obtained (measured in terms of the plan length).

Another research direction relates to work integrating HTN and STRIPS planning (Shivashankar et al., 2013; Kambhampati & Srivastava, 1995). These works do not specify hierarchies in terms of nonprimitive or primitive tasks, but in terms of goals. We are interested in generalizing HALTER to learn goal hierarchies directly. Learning goal hierarchies has been studied before. For example, (König & Laird, 2006) uses inductive logic programming techniques as opposed to HALTER's explanation-based parsing techniques such as chart parsing and goal regression. It would be interesting to generalize HALTER for goal hierarchies and compare both approaches both theoretically and experimentally.

Acknowledgements

This research is funded in part by Contract N00014-12-C-0239 with Office of Naval Research (ONR). The views expressed are those of the authors and do not reflect the official policy or position of the U.S. Government.

References

- Anderson, J. R. (1993). *Rules of the mind*. New York.
- Charniak, E., Goldwater, S., & Johnson, M. (1998). *Edge-based best-first chart parsing*, 127–133. Association for Computational Linguistics.
- Currie, K., & Tate, A. (1985). O-plan—control in the open planning architecture. In *Expert systems*, 225–240.
- Erol, K., Hendler, J., & Nau, D. S. (1994). HTN planning: Complexity and expressivity. *National Conference on Artificial Intelligence (AAAI)*.
- Forbus, K., Klenk, M., & Hinrichs, T. (2008). Companion cognitive systems: Design goals and some lessons learned. In *the AAAI Fall Symposium on Naturally Inspired Artificial Intelligence*. Washington D.C.
- G., G. T., & Boicu, M. (2008). *A guide for ontology development with disciple* (Technical Report). Learning Agents Center, George Mason University.
- Ghallab, M., Nau, D. S., & Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann.
- Hogg, C., Muñoz-Avila, H., & Kuter, U. (2008). HTN-MAKER: Learning HTNs with minimal additional knowledge engineering required. *Conference on Artificial Intelligence (AAAI)* (pp. 950–956). AAAI Press.
- Ilgami, O., Muñoz-Avila, H., Nau, D. S., & Aha, D. W. (2005). Learning approximate preconditions for methods in hierarchical plans. *International Conference on Machine Learning (ICML)* (pp. 337–344). Bonn, Germany.
- Jaidee, U., Muñoz-Avila, H., & Aha, D. W. (2011). Integrated Learning for Goal-Driven Autonomy. *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three* (pp. 2450–2455).
- Kambhampati, S., & Srivastava, B. (1995). Universal classical planner: An algorithm for unifying state-space and plan-space planning. *European Workshop on Planning (EWSP)*.
- König, T., & Laird, J. E. (2006). Learning goal hierarchies from structured observations and expert annotations. *Machine Learning*, 64, 263–287.
- Laird, J. (2012). *The soar cognitive architecture*. MIT Press.
- Langley, P., & Choi, D. (2006a). Learning recursive control programs from problem solving. *The Journal of Machine Learning Research*, 7, 493–518.
- Langley, P., & Choi, D. (2006b). Learning recursive control programs from problem solving. *Journal of Machine Learning Research*, 7, 493–518.

- Langley, P., Cummings, K., & Shapiro, D. (2004). Hierarchical skills and cognitive architectures. *Proceedings of the Twenty-Sixth Annual Conference of the Cognitive Science Society*.
- Molineaux, M., Klenk, M., & Aha, D. W. (2010). Goal-Driven Autonomy in a Navy Strategy Simulation. *AAAI*.
- Muñoz-Avila, H., Jaidee, U., Aha, D., & Carter, E. (2010). Goal-Driven Autonomy with Case-Based Reasoning. In *Case-based reasoning, research and development*, 228–241. Springer.
- Nau, D. S., Au, T.-C., Ilghami, O., Kuter, U., Muñoz-Avila, H., Murdock, J. W., Wu, D., & Yaman, F. (2005). Applications of SHOP and SHOP2. *IEEE Intelligent Systems*, 20, 34–41.
- Nau, D. S., Cao, Y., Lotem, A., & Muñoz-Avila, H. (1999). SHOP: Simple hierarchical ordered planner. *International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 968–973). Morgan Kaufmann.
- Nejati, N., Langley, P., & Konik, T. (2006). Learning hierarchical task networks by observation. *International Conference on Machine Learning (ICML)* (pp. 665–672).
- Powell, J., Molineaux, M., & Aha, D. W. (2011). Active and interactive discovery of goal selection knowledge. *FLAIRS Conference*.
- Reddy, C., & Tadepalli, P. (1997). Learning goal-decomposition rules using exercises. *International Conference on Machine Learning (ICML)*.
- Sacerdoti, E. (1977). *A structure for plans and behavior*. American Elsevier.
- Shivashankar, V., Alford, R., Kuter, U., & Nau, D. (2013). The godel planning system: a more perfect union of domain-independent and hierarchical planning. *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence* (pp. 2380–2386).
- Tate, A. (1977). Generating project networks. *International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 888–893).
- Weber, B. G., Mateas, M., & Jhala, A. (2012). Learning from demonstration for goal-driven autonomy. *AAAI*.
- Wilkins, D. E., & Myers, K. L. (1998). A multiagent planning architecture. *International Conference on AI Planning Systems (AIPS)* (pp. 154–162).
- Xu, K., & Muñoz-Avila, H. (2005). A domain-independent system for case-based task decomposition without domain theories. *National Conference on Artificial Intelligence (AAAI)*.

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 $(S, A, E, O, \gamma, \omega)$ with states S , actions A , exogenous events E , observations O , state transition function $\gamma: S \times (A \cup E) \rightarrow S$, and observation function $\omega: S \rightarrow O$. The transition function γ describes how an action’s execution (or an event’s occurrence) transforms the

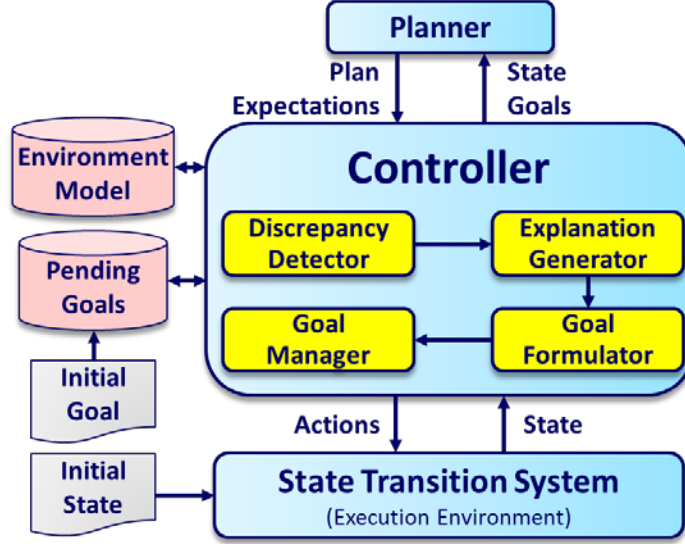


Figure 1: Conceptual Model for Goal-Driven Autonomy (GDA)

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 $A_c = [a_{c+1}, \dots, a_{c+n}]$, and (2) a corresponding sequence of expectations $X_c = [x_{c+1}, \dots, x_{c+n}]$, 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 $[a_1, \dots, a_n]$ and observations $[\mathbf{obs}_0, \dots, \mathbf{obs}_c]$ 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 **effects**⁻ and **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** $\mathcal{T} = \{t_0, t_1, t_2, \dots, t_n\}$ and an **ordering relation** between two such points, denoted as $t_1 < t_2$, where $t_1, t_2 \in \mathcal{T}$.

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** as a function such that **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 $\mathbf{obs}_0; a_1; \mathbf{obs}_1; a_2; \dots; a_k; \mathbf{obs}_{k+1}$. 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 $i = 0, \dots, k-1$ and each action a_j for $j = 1, \dots, k$ 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 $\mathbf{occ}(o_i) < \mathbf{occ}(o_j)$ such that $o_i, o_j \in C$. As a shorthand, we will sometimes write $o_i < o_j$ if and only if $\mathbf{occ}(o_i) < \mathbf{occ}(o_j)$.

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:

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

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:

$$\begin{aligned} \mathbf{prior}(o, p) &= \{o' \mid \mathbf{relevant}(p, o') \wedge \nexists o'' \text{ s.t. } \mathbf{relevant}(p, o'') \wedge o' < o'' < o\}. \\ \mathbf{next}(o, p) &= \{o' \mid \mathbf{relevant}(p, o') \wedge \nexists o'' \text{ s.t. } \mathbf{relevant}(p, o'') \wedge o < o'' < o'\}. \end{aligned}$$

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 $o < o' < (e, t)$.

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 $o < o'' < o' \in R$ and p is relevant to o'' .

An explanation $\chi = (C, R)$ is **plausible** if and only if the following holds:

1. There are no inconsistencies in χ .
2. Every event occurrence $(e, t) \in \chi$ has a proximate cause in χ .
3. For every pair of simultaneous occurrences such that $o, o' \in C$ and $\mathbf{occ}(o) = \mathbf{occ}(o')$, there may be no conflicts before or after. That is, for all p , $\mathbf{knownafter}(p, o) \Rightarrow \neg \mathbf{knownafter}(\neg p, o')$, and $\mathbf{knownbefore}(p, o) \Rightarrow \neg \mathbf{knownbefore}(\neg p, o')$.
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 **expectations**(α, t), and its observations at time t as **observations**(α, 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:

$$\mathbf{surprise}_x(\alpha, t) \equiv \mathbf{expectations}(\alpha, t) \cup \mathbf{observations}(\alpha, t) \cup \beta \models \perp. \quad (1)$$

In terms of a GDA agent, we can describe **expectations**(α, t) as X_t and **observations**(α, t) as **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 **derivedexpectations**(α, θ, 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:

$$\mathbf{surprise}_m(\alpha, t) \equiv \forall \theta \subset \Theta: [\mathbf{derivedexpectations}(\alpha, \theta, t) \cup \mathbf{observations}(\alpha, t) \cup \beta \models \perp] \quad (2)$$

From this, we can derive the fact that

$$\begin{aligned} [\exists \theta \subset \Theta: \mathbf{expectations}(\alpha, t) = \mathbf{derivedexpectations}(\alpha, \theta, t)] \\ \models (\mathbf{surprise}_m(\alpha, t) \models \mathbf{surprise}_x(\alpha, t)). \end{aligned} \quad (3)$$

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 $\mathbf{effects}^+(e_h) = \{p\}$ and $\mathbf{precond}(e_h) = \{\neg p\}$, and t' is an occurrence point such that $\mathbf{occ}(o) < t' < \mathbf{occ}(o')$. 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 $\mathbf{effects}(e_u) = \{\text{(sinking-rapidly Buttercup)}\}$. Here, the target concept is the event that causes the effect literal. The positive example literal here would be $(\text{event-occurs sinking-rapidly Buttercup } t' \text{ PrincessBride})$.

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

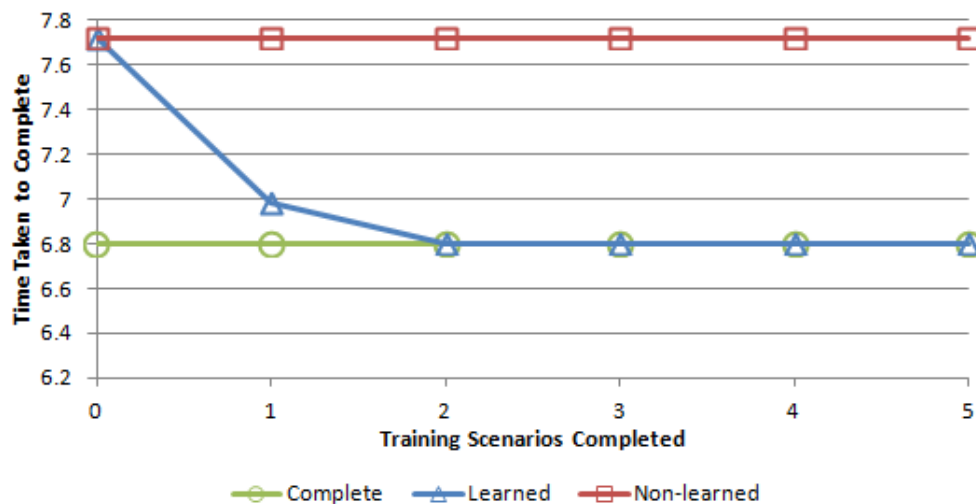


Figure 2: Time required by FOOLMETWICE to complete a scenario.

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.

Goal-Driven Autonomy in Dynamic Environments

Matt Paisner

Michael Maynard

Michael T. Cox*

Don Perlis

MATTHEW.PAISNER@GMAIL.COM

MAYNORD@UMD.EDU

MCOX@CS.UMD.EDU

PERLIS@CS.UMD.EDU

Department of Computer Science, University of Maryland, College Park, MD 20742 USA

*Institute for Advanced Computer Studies, University of Maryland, College Park, MD USA

Abstract

Dynamic environments are complex and change in often unexpected ways. Given such an environment, many autonomous agents have difficulty when the world does not cooperate with design assumptions. We present an approach to autonomy that seeks to maximize robustness rather than optimality on a specific task. Goal-driven autonomy involves recognizing possibly new problems, explaining what causes the problems, and autonomously generating goals to solve the problems. We present such a model within the MIDCA cognitive architecture and show that under certain conditions this model outperforms a less flexible approach to handling unexpected events.

1. Introduction

Humans are astonishingly versatile, dealing with a wide range of unanticipated circumstances on the fly while still making headway on high-level goals. Humans can also recognize new problems and opportunities when they arise and react appropriately to them. Yet for the most part our machines cannot; they are like idiot-savants, very good at one narrow task and useless for anything else, even tasks which differ only slightly from the task they were designed for. This is the so-called *brittleness problem* (Lenat & Guha, 1989), a major stumbling block for AI. What we appear to need is the opposite of expert systems: machines that might not be experts at anything, but that can muddle through a wide range of circumstances and keep a strategic perspective. Yet more than 50 years of intense effort has failed to produce such machines. One approach to the problem of brittleness combines what we call *goal-driven autonomy* and the *metacognitive loop*. Here we will show how together they can address significant issues of performance in dynamic environments.

In our view, the community has not fully exploited the computational power and promise of metacognition. We propose to test the hypothesis that metacognitive monitoring and control of reasoning can help confront the brittleness problem, leading to autonomous systems that are much more robust and long-lived than current systems, and that require significantly less domain-specific insight on the part of system designers. The Metacognitive Loop (MCL) (Anderson & Perlis, 2005) involves a three-step approach: First an agent notes any failure of match (a so-called *anomaly* – whether in its reasoning, its KB, or its behavior) between what it expects and what it observes; second it assesses the anomaly in causal terms of the failure; and third it guides an appropriate

response to repair or learn from the failure. This is the Note-Assess-Guide procedure. A particularly useful instantiation of the procedure can be found in the methods of goal-driven autonomy.

Goal-Driven Autonomy (GDA) is a unique conception that gives more independence to autonomous agents (Cox, 2007; Klenk, Molineaux, & Aha, 2013; Munoz-Avila, Jaidee, Aha, Carter, 2010). Here the MCL steps of Note-Assess-Guide map onto the tasks of recognizing a problem, explaining what causes the problem, and generating a goal to remove the problem. That is, rather than arbitrary anomaly-detection, the agent detects a problem (possibly relevant to its current goals and mission). Not all anomalies are problems, nor are all anomalies important enough to attend to. Rather than general assessment, the agent should abductively explain the causal factors giving rise to the problem situation. The response to the problem may be to generate a (possibly new) goal that solves the problem (e.g., by removing its supporting conditions). In these terms, GDA is as much about problem recognition as it is problem-solving (Cox, in press). Planning for the goal as a result of intending to achieve it would presumably result in actions that then bear on the problem. But we argue that the key to dealing effectively with anomalous situations in dynamic environments is for the agent to understand problems that the world presents; the success of planning and acting will then follow (or not). But without recognition of a problem and of a goal to solve it, planning and acting make no sense at all.

Consider a fire that breaks out at a construction site. This is a problem in many ways, not the least of which is that preconditions for actions (e.g., the integrity of building materials) will become unsatisfied. A subgoal to remove the burning condition is thus warranted in such cases, and the fire will be extinguished. However a subsequent fire might prompt the recognition by a supervisor of a long term threat to the construction site. An arsonist explanation would hypothesize the presence of criminal activity and license the goal of having the perpetrator in jail. Hence a direct reactive approach to fires would be to put them out; a GDA approach to this same situation would recognize the larger problem in terms of its threat to the future of the enterprise.

This paper will examine the distinction between such approaches to intelligent reasoning and behavior in a metacognitive architecture called MIDCA and will report the results of a simple empirical study to evaluate these differences. In section 2, we present the MIDCA architecture containing an implemented instantiation of the GDA model. In section 3 we evaluate the performance of systems making use of three distinct goal generation methods: exogenous goals; statistically generated goals; goals produced by a knowledge rich explanation system. Section 4 presents an overview of future work, section 5 surveys related work, and section 6 concludes.

2. Goal-Driven Autonomy in a Cognitive Architecture

The *Metacognitive, Integrated, Dual-Cycle Architecture (MIDCA)* (Cox, Maynard, Paisner, Perlis, & Oates, 2013; Cox, Oates, & Perlis, 2011) consists of “action-perception” cycles at both the cognitive (i.e., object) level and the metacognitive (i.e., meta-) level. Figure 1 shows the implemented components of the object level with the meta-level abstracted. The output side of each cycle consists of intention, planning, and action execution, whereas the input side consists of perception, interpretation, and goal evaluation. A cycle selects a goal and commits to achieving it. The agent then creates a plan to achieve the goal and subsequently executes the planned actions to make the domain match the goal state. The agent perceives changes to the environment resulting from the actions, interprets the percepts with respect to the plan, and evaluates the interpretation with respect to the goal. At the object level, the cycle achieves goals that change the environment (i.e., ground level). At the meta-level, the cycle achieves goals that change the object level. That is,

the metacognitive “perception” components introspectively monitor the processes and mental state changes at the cognitive level. The “action” component consists of a meta-level controller that mediates reasoning over an abstract representation of the object level cognition.

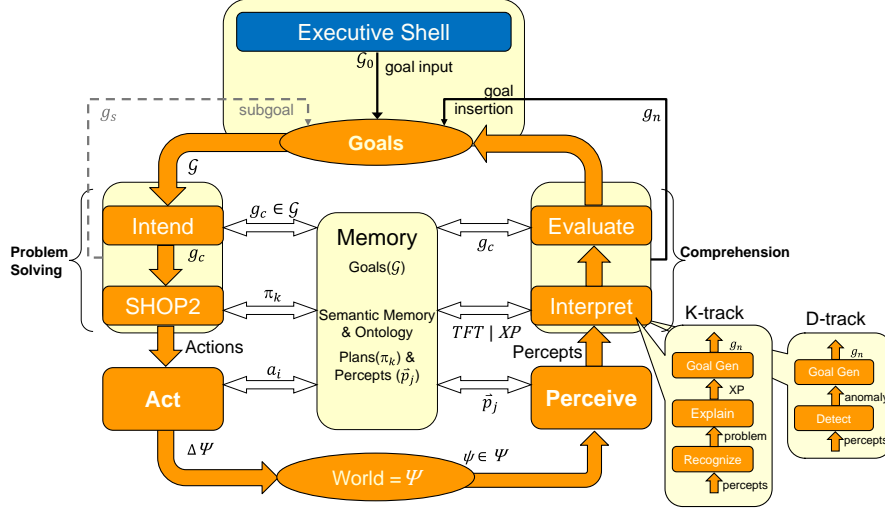


Figure 1. The MIDCA_1.1 object level structure. Note that execution-time subgoaling (dashes) is not currently implemented. TFT stands for TF-Tree (see section 2.2.1), and XP stands for eXplanation Pattern (see section 2.2.2).

Comprehension starts with perception of the world in the attentional field via the *Perceive* component. The *Interpret* component takes as input the resulting *Percepts* (i.e., \vec{p}_j) and the expectations in memory (π_k and g_c) to determine whether the agent is making sufficient progress. A GDA interpretation procedure implements the comprehension process. The procedure is to *note* whether an anomaly has occurred; *assess* potential causes of the anomaly by generating explanatory *Hypotheses*; and *guide* the system through a response. Responses can take various forms, such as (1) test a Hypothesis; (2) ignore and try again; (3) ask for help; or (4) insert another goal (g_n). Otherwise given no anomaly, the *Evaluate* component incorporates the concepts inferred from the *Percepts* thereby changing the world model (ΔM_Ψ), and the cycle continues. This cycle of problem-solving and action followed by perception and comprehension functions over discrete state and event representations of the environment.

Here the blue meta-level executive shell simply provides the goal set \mathcal{G} . In this capacity, the meta-level can add initial goals (g_0) or new goals (g_n) to the set. In problem solving (left side), the *Intend* component commits to a current goal (g_c) from those available. The *Plan* component then generates a sequence of *Actions* (π_k , i.e., a hierarchical-task-net plan). The plan is executed by the *Act* component to change the actual world (Ψ) through the effects of the planned *Actions* (a_i). The agent stores the goal and plan in memory to provide it with expectations about how the world will change in the future. The agent will then use these expectations in the next cycle to evaluate the execution of the plan and its interaction with the world with respect to the goal.

2.1 Metacognitive, Integrated, Dual-Cycle Architecture Version 1.1

MIDCA_1.1 is an early version of the architecture whose components are shown in the schematic of Figure 1. It implements each phase of the cognitive loop, allowing the MIDCA agent to notice, analyze and respond to events in a simple blocksworld domain. MIDCA_1.1 employs a central memory structure through which the components implementing each individual box shown in Figure 1 interact. Currently, memory is implemented as a simple key/value map through which individual data structures can be accessed. In future versions of MIDCA, we plan to add several capabilities to this, including a model of time and a system for reasoning about it as well as a detailed accounting of the sources of the systems knowledge. These plans will be discussed in greater detail in the future work section.

2.1.1 Performance Domain

To evaluate the performance of MIDCA in goal generation, we place the system in a modified blocksworld domain. This version of blocksworld includes both rectangular and triangular blocks, which compose the materials for simplified housing construction. The initial goals for problems in this domain are to build houses consisting of towers of blocks with a roof (triangle) on each. Specifically, the housing domain goes through a cycle of three state classes in building new “houses.” Figure 2 shows three intermediate states and the goals that transition the system between such states.

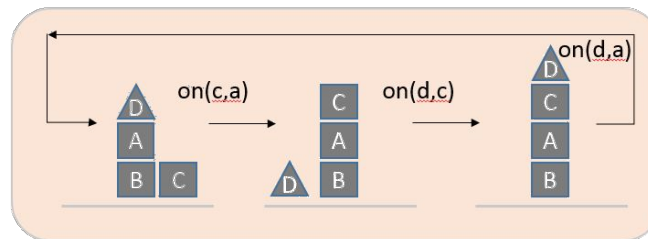


Figure 2. Three classifiers exist that recognize goals to get to the subsequent state

We use a simple world simulator that simulates actions specified using predicate logic. The types of actions which can be performed are specified prior to startup in a domain file. Actions MIDCA produces during the act phase will be simulated, as well as actions performed by other agents and natural events. For the purpose of generating interesting anomalies for MIDCA to deal with, we have added the possibility that blocks may catch fire (set by a hidden arsonist). Furthermore there are three additional actions available to MIDCA to deal with fires. The three new types of actions, then, are as follows.

- `light-on-fire(block-b)` if block-b is not on fire, lights it; performed only by the arsonist
- `put-out-fire(block-b)` if block-b is on fire, extinguishes it
- `apprehend(arsonist-a)` imprisons arsonist, rendering incapable of lighting any more fires

MIDCA_1.1’s task is to build “towers” of blocks while also dealing appropriately with the fire. In the next subsection, we describe the techniques it uses in achieving this.

2.1.2 MIDCA_1.1 Reasoning Components

MIDCA_1.1 is implemented by a series of components, centered about a core memory structure. Each of these components implements a single phase of the MIDCA cognitive loop shown in Figure 1. Running MIDCA_1.1 is equivalent to running each of these components in the order shown in Figure 1. This cycle repeats at each time step, beginning with the perceive component. Components interact by storing information in memory, where it can be accessed and used later in the cycle and in future cycles. The individual implementations are described below.

Perceive. The perceive phase is implemented very simply. The perceive component makes a copy of the current world state (defined in a predicate logic representation) and stores it in memory. As a result, MIDCA_1.1 always has a perfect, noise-free view of the current world state that is complete except for the direct actions of the arsonist. In the future work section, we describe plans to move away from this model to one in which perception may be flawed. In our simple blocksworld example, MIDCA_1.1 would copy to memory the same predicate representation of block relationships and attributes that the world simulator maintained as the current state.

Interpret. The interpret phase has been at the core of our research efforts. It is implemented as a GDA interpretation procedure that uses both a bottom-up, data-driven track and a top-down, knowledge rich, goal-driven track (Cox, Maynard, Paisner, Perlis, & Oates, 2013). MIDCA_1.1 uses both of these processes to analyze the current world state and determine which, if any, new goals it should attempt to pursue. The details of this process are described below. In our example, this would be the phase in which MIDCA_1.1 noticed an anomaly in the blocksworld – e.g., a block was on fire – and decided what to do about it.

Evaluate. In the evaluate phase, the goal generated during the previous step is evaluated. The system searches through the world representation stored during the perceive phase, and checks if the goal predicate exists in the world state. If so, MIDCA_1.1 notes that goal has been achieved. Additionally, during evaluate MIDCA_1.1 checks on the progress of its broader goals and updates the relevant performance metrics. In blocksworld, MIDCA_1.1 checks if its current goal, for example $on(A, B)$ has been achieved. It then checks to see if a new tower has been built and if so how many blocks in it are on fire. All this data is stored in memory, and used later to score MIDCA_1.1's success at achieving its goals in the face of problems.

Intend. The intend component determines which goals to pursue. If the evaluate phase reports that the previous goal has been achieved, MIDCA_1.1 checks to see if a new goal was generated during the interpret phase. If so, it adopts that goal. Otherwise, it will do nothing until a new goal is generated. If the previous goal has not been achieved, it will also do nothing. In blocksworld, this component converts the goal that has been generated into a task that can be taken as input by the planner. For example, the goal $not(onfire(A))$ would be transformed into $put-out-fire(A)$.

Plan. For the planner, we use SHOP2 (Nau et al., 2003), a domain-independent task decomposition planner. If the intend component specified a new task, SHOP2 generates a plan to achieve that task given the current world state stored in memory. Otherwise, it does nothing. The actions and methods that are used to achieve each task in blocksworld are specified in a domain file that we supply.

Act. MIDCA chooses the next action from the current plan, if one exists. Otherwise, it does not perform an action. If an action is chosen, it is sent to the world simulator, which uses it to compute the next world state. An example of such an action might be $unstack(A, B)$ if SHOP2 had generated a plan containing that step.

Apart from the central memory structure, MIDCA is designed to be highly modular. Each phase of the MIDCA loop is implemented by a component which is passed to the system at startup, and these components can be easily swapped in and out. Therefore, the component behaviors described above for MIDCA_1.1 can be easily expanded or modified to use different algorithms or approaches. For example, in the experiments described in Section 3, we achieve the different experimental setups described simply by passing MIDCA_1.1 different components corresponding to the interpret phase. We will now describe the nature of those different components.

2.2 Interpretation

The interpret phase of MIDCA has been the subject of much of our work, and is the focus of the experiments described below. It is implemented by two processes that combine to generate new goals based on the features of the world the agent observes. We call these processes the D-track, which is a data driven, bottom-up approach, and the K-track, which is knowledge rich and top-down. The D-track is implemented by a bottom-up GDA process as follows. A statistical anomaly detector constitutes the first step, a neural network identifies low-level causal attributes of the anomaly, and a goal classifier, constructed using methods from machine learning, provides the goal formulation. The K-track as it currently exists is implemented as a case-based explanation process. The representations for expectations significantly differ between the two tracks. K-track expectations come from explicit knowledge structures such as action models used for planning and ontological conceptual categories used for interpretation. Predicted effects form the expectations in the former and attribute constraints constitute expectation in the latter. D-track expectations are implicit by contrast. Here the implied expectation is that the probabilistic distribution from which observations are sampled will remain the same. When the difference between the expected and the perceived distribution is statistically significant, an expectation violation is raised.

2.2.1 D-Track Goal Generation

The D-track interpretation procedure uses a novel approach for noting anomalies. We apply the statistical metric called the A-distance to streams of predicate counts in the perceptual input (Cox, Oates, Paisner, & Perlis, 2012; 2013). This enables MIDCA to detect regions whose statistical distributions of predicates differ from previously observed input. These regions are those where change occurs and potential problems exist.

When a change is detected, its severity and type can be determined by reference to a neural network in which nodes represent categories of normal and anomalous states. This network is generated dynamically with the growing neural gas algorithm (Paisner, Perlis, & Cox, 2013) as the D-track processes perceptual input. This process leverages the results of analysis with A-distance to generate anomaly prototypes, each of which represents the typical member of a set of similar anomalies the system has encountered. When a new state is tagged as anomalous by A-distance, the GNG net associates it with one of these groups and outputs the magnitude, predicate type, and valence of the anomaly.

Goal generation is achieved in MIDCA_1.1 using *TF-Trees* (Maynard, Cox, Paisner, & Perlis, 2013), a machine-learning classification structure that results from the conjunction of two algorithms both of which work over the predicate representation of the blocks world domain. The first of these algorithms is Tilde (Blockeel, & De Raedt, 1997), which is itself a generalization of the standard C4.5 (Quinlan, 1993) decision tree algorithm. The second algorithm is FOIL (Quinlan, 1990), an algorithm which, given a set of examples in predicate representation reflecting some

concept, induces a rule consisting of conjunctions of predicates that identify the concept. Given a world state, a TF-Tree first uses Tilde to classify the state into one of a set of scenarios, where each scenario is associated with a rule generated by FOIL. Once that rule is obtained, groundings of the arguments of the predicates in that rule are permuted until either a grounding that satisfies the rule is found (in which case a goal is generated) or until all permutations have been eliminated as possibilities (in which case no goal is generated). This approach to goal insertion is naïve in the sense that it constitutes a mapping between world states and goals which is static with respect to any context. The structure of a TF-Tree is a tree where in internal nodes are produced by Tilde and leaf nodes are rules produced by FOIL. Figure 3 depicts the structure of the TF-Tree MIDCA_1.1 uses in cycling through the 3 block arrangements. For example given the middle state of Figure 2, **triangle-D** is clear, it is on the table, and the table is a table. Thus we take the right branch labeled “yes.” Now **triangle-D** is also a triangle, so again we take the “yes” branch to arrive at the right-most leaf of the tree. The leaf rule then binds the variable Y to the clear **square-C**, and the resulting goal is to have **triangle-D** on **square-C**.

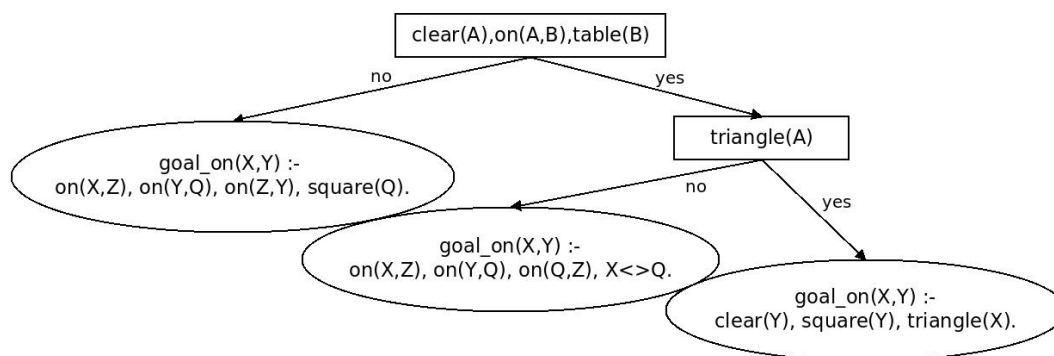


Figure 3. Depiction of the TF-Tree used in cycling through the 3 block arrangements used in MIDCA_1.1

The construction of a TF-Tree requires a training corpus consisting of world states and associated correct and incorrect goals. In simple worlds TF-Trees can be constructed which have perfect or near perfect accuracy using small training corpora. Corpora have to be constructed by humans, as labels need to be attached to potential goals in various world states. For simple worlds corpora construction does not carry an excessive burden, the burden of corpora construction however increases with the complexity of the world. Because a TF-Tree is a static structure trained on the specifics of the world, when the world changes, even in minor ways, a new training corpus has to be constructed and a new TF-Tree trained. However, the corpus to create a simple tree for reacting to fires (see Figure 4) consisted of only four examples.

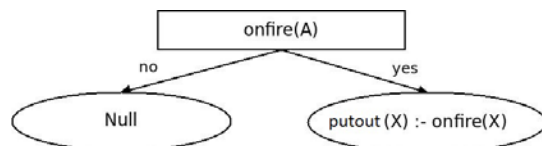


Figure 4. TF-Tree that generates goals to put out fires

2.2.2 K-Track Goal Generation

The K-track GDA procedure uses the XPLAIN system (Cox & Burstein, 2008), an explanation subsystem of the POIROT (Burstein, et al., 2008) multi-strategy learning system for Web Service procedures. XPLAIN is built on top of the Meta-AQUA introspective story understanding system (Cox and Ram 1999; Lee and Cox 2002) and is used in MIDCA to detect and explain problems in the input perceptual representations. The system's interpretation task is to "understand" input by building causal explanatory graphs that link the individual subgraph representations of the events into a coherent whole where an example measure of coherency is minimizing the number of connected components. The sub-system uses a multistrategy approach to comprehension. Thus, the top-level goal is to choose a comprehension method (e.g., script processing, case-based reasoning, or explanation generation) by which it can understand an input. When an anomalous or otherwise interesting input is detected, the system builds an explanation of the event, incorporating it into the preexisting model of the story. XPLAIN uses case-based knowledge representations implemented as frames tied together by *explanation-patterns* (Cox & Ram, 1999; Schank, 1986; Schank, Kass, and Riesbeck 1994) that represent general causal structures.

XPLAIN relies on general domain knowledge, a case library of prior plan schemas and a set of general explanation patterns that are used to characterize useful explanations involving that background knowledge. These knowledge structures are stored in a (currently) separate memory sub-system and communicated through standard socket connections to the rest of MIDCA_1.1. XPLAIN uses an interest-driven, variable depth, interpretation process that controls the amount of computational resources applied to the comprehension task. For example an assertion that **triangle-D** is picked up generates no interest, because it represents normal actions that an agent does on a regular basis. But XPLAIN classifies **block-A** burning to be a violent action and, thus according to its interest criterion, interesting. It explains the action by hypothesizing that the burning was caused by an arsonist. An abstract explanation pattern (see Table 1), or XP, retrieved from memory instantiates this explanation, and the system incorporates it into the current model of the actions in the input "story" and passes it as output to MIDCA.

The ARSONIST-XP asserts that the lighting of the block caused heat that together with oxygen and fuel (the block itself) caused the block to burn. The arsonist did this action because he wanted the blocks burning state that resulted from the burning. The objective is to counter a vulnerable antecedent of the XP. In this case the deepest antecedent is the variable binding =I-O or the lighting-object action. This can be blocked by either removing the actor or removing the ignition-device. The choice is the actor, and a goal to apprehend the arsonist is generated.

Table 1. The arsonist explanation pattern

```

(define-frame ARSONIST-XP
  (isa (value (xp)))
  (actor (value (criminal-volitional-agent)))
  (object (value (physical-object)))
  (antecedent (value (ignition-xp
    (actor (value =actor))
    (object (value =object))
    (ante (value (light-object =l-o
      (actor (value =actor))
      (instrumental-object (value (ignition-device))))))
    (conseq (value =heat))))))
  (consequent (value (forced-by-states
    (object (value =object))
    (heat (value =heat))
    (conseq (value (burns =b
      (object (value =object))))))))))
  (heat (value (temperature (domain (value =object))
    (co-domain (value very-hot.0))))))
  (role (value (actor (domain (value =ante))
    (co-domain (value =actor))))))
  (explains (value =role))
  (pre-xp-nodes (value (=actor =consequent =object =role)))
  (internal-nodes (value nil.0))
  (xp-asserted-nodes (value (=antecedent)))
  (link1 (value (results
    (domain (value =antecedent))
    (co-domain (value =consequent))))))
  (link2 (value (xp-instrumental-scene->actor
    (actor (value =actor))
    (action (value =l-o))
    (main-action (value =b))
    (role (value =role))))))
  (links (value (=link1 =link2))))

```

3. Evaluation: Autonomous goal formulation

The fires are problems because of the effect on housing construction and the supposed profits of the housing industry, and they pose threats to life and property. The approach to understanding fire problems is to ask why the fires were started and not just how. An explanation of how the fire started would relate the presence of sufficient heat, fuel, and oxygen with the combustion of the blocks. Generating the negation of the presence of the oxygen for example would result in the goal \neg oxygen and therefore put out the fire. But this does not get to the reason the fire started in the first place. To ask why the fire was started would result in possibly two hypotheses or explanations. Poor safety conditions can lead to fire or the actions of arsonists can result in fire. In this latter case, the arsonist causes the presence of the heat through some hidden lighting action. Given this explanation the agent can anticipate the threat of more fires and generate a goal to remove the threat by finding the arsonist. Apprehending the arsonist then removes the potential of fires in the future rather than just reacting to fires that started in the past.

We tested the effectiveness of three methods for goal generation under these conditions. The first method was a simple baseline using predetermined, exogenous goals. The second method used the statistical, D-Track GDA method described in Section 2.2.1. The third method combined the D-Track approach with additional analysis using K-Track GDA as described in Section 2.2.2. Details appear in Table 2. For each test, MIDCA was run for 1000 time steps (equivalent to

executing 1000 actions). At each step, the arsonist would have a probability p of starting a fire unless he had previously been apprehended. The value of p in the experiments described below was 0.4, allowing for enough fires to be significant without precluding progress in the tower construction project.

Table 2. Methods for goal generation

Exogenous Goals	MIDCA used a predetermined goal list that cycled between the three states constructing towers. MIDCA did not deviate from this list in response to the appearance of fires. The goal list was simply: [on(C,A), on(D,C), on(D,A), on(C,A),...on(D,A)], as in Figure 2.
D-Track GDA Goal Generation	MIDCA generated goals using TF-Trees. These trees were trained and implemented such that when no fire was present, they would generate the next goal in the 3-part cycle, but when a fire was present, they would instead generate a goal to put it out.
Dual-Track GDA Goal Generation	MIDCA generated goals using a combination of TF-Trees and a K-Track GDA approach using XPLAIN. XPLAIN contained knowledge about the possible role of arsonists in starting fires and therefore suggested a goal to search for and apprehend an arsonist when a fire started. TF-Trees generated other goals as in 2.

We tracked three scoring metrics: the number of towers completed; the overall prevalence of fires; and a combined score measuring completion of fire-free towers. Details on each scoring metric are shown in Table 3. At each time step in which a tower was completed – e.g. a triangular block was placed on a stack of rectangular blocks, – all fires were automatically put out, and the agent started on a new construction project.

Table 3. Scoring metrics for testing

Towers Completed	Total number of 3- and 4-block towers completed in 1000 cycles
Fire Prevalence	The number of blocks on fire times the number of time steps they were on fire. So, if 3 blocks were allowed to burn for three time steps and then put out simultaneously, the fire prevalence score would be $3 * 3 = 9$.
Overall Score	This score awarded 1 point for each block in a completed tower that was not on fire at the time the tower was completed. So, a 4-block tower in which two blocks were on fire would add $4 - 2 = 2$ points to this score.

Preliminary empirical results show that GDA approaches using only the D-Track as well as using both D-Track and K-Track perform significantly better than a baseline that does not use GDA. Also, the combined D- and K-Track implementation outperforms the purely statistical variant by a large margin.

Figure 4 shows the detailed results of testing. The agent that used only exogenous goals completed the most towers, but, because it did not deal with fires in any way, most of the towers were burning as they were completed and received very low scores. Certainly, this baseline behavior does not seem to be sufficient for a fully autonomous house construction agent. The second agent used behavior dictated by TF-Trees trained using methods from machine learning to fight fires directly. It did not finish as many towers because it divided its attention between building

and extinguishing towers, but those towers it did construct were much less likely to be on fire. Its total score was 367, 54.2% better than the baseline agent. Finally, the dual-track GDA agent analyzed the problem logically using XPLAIN, and thereby suggested an explanation of the fires as potentially caused by arson. As such, it generated a goal early in the process to apprehend the arsonist. This took some time, but afterwards it was able to devote its full attention to house construction without devoting time to firefighting. It completed very nearly as many towers as the baseline agent, and did so with almost no incidence of fire, since no fires started after the arsonist was apprehended. The dual-track agent achieved a score of 584, 245.4% better than the baseline agent.

It should not be surprising that an agent that is capable of reacting to the unanticipated problem posed by fire performs better than one that heedlessly continues on a predetermined course of action. Perhaps more telling is the large advantage gained by the dual-track agent, which has the knowledge to identify and address the true source of the problem, rather than simply treating its symptoms. Though this example is too simple to easily generalize, these results at least suggest the importance of combining a knowledge-rich approach with low-level data analysis to achieve the best possible results. In the next section, we discuss the steps we intend to take in order to duplicate these results in a domain that allows for a more convincing role for the data side of this equation.

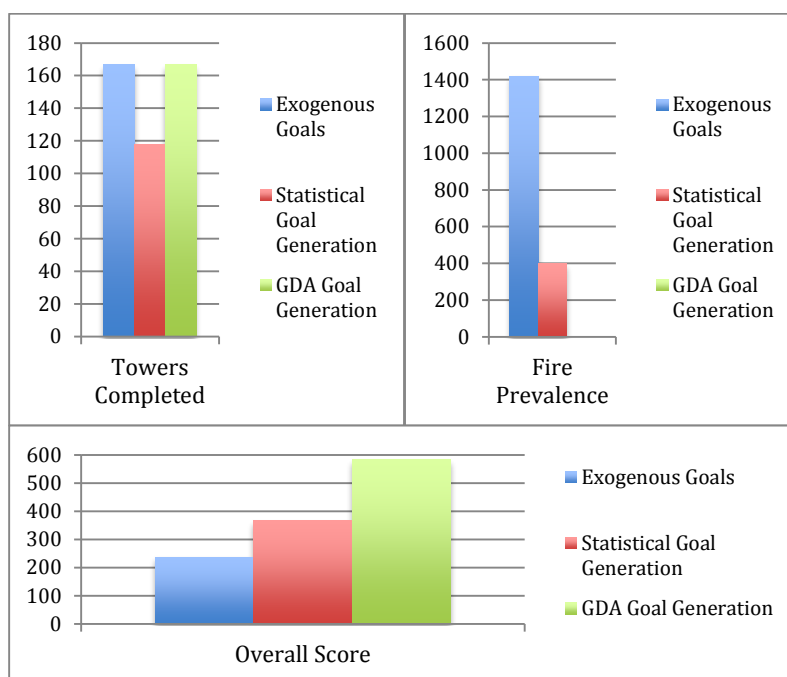


Figure 4. Results of testing using 3 methods. Note that the value of GDA Goal Generation in the Fire Prevalence panel is 2, which is too small to show clearly in the graph. Future Research

MIDCA_1.1 was designed with an eye towards expansion into both additional domains and added capabilities. We have several ideas about how to effect such expansion. The reader will likely have observed that the domain described in this paper is quite simplistic. In the future we plan to adapt

MIDCA for a robotic domain. Specifically, we plan to incorporate the MIDCA architecture into a PR2 robot (Cousins, 2010) and attempt to replicate results like those described in this paper. Some changes will be necessary (for example, lighting blocks on fire in the lab might not be advisable), but we might have the robot attempt to stack blocks into towers and face the unexpected problems that occur when the stacked blocks fall.

Solving this version of the block-stacking problem would require several additions to the current version of MIDCA. The PR2 would first need to find and identify the blocks. One of many ways to accomplish this could be by using blob detection combined with a set of rules that define what type of object a block is. Since the PR2 also has sensors and software that generate point-cloud distance measurements, size and location could also be used as determiners, if, for example, we constrain blocks to be small enough to lift and to be located on the table. Once blocks are identified, the next step would be to differentiate between them, and to characterize simple predicate relationships like those in the modified blocks world domain of this paper such as **on** or **on-table**. Given location information and a frame of reference, this might be accomplished using simple rules, so long as the environment was sufficiently uncomplicated.

Once the robot has an idea of the objects in its environment, it can generate a plan to achieve a tower construction goal. Each step of this plan must be translated from a predicate representation to individual actions the robot can perform. So, `unstack(b1, b2)` might become something like `[locate(b1), locate(b2), move-arm(near b1), grasp(b1), move-arm(rest-loc)]`. Additionally, the Evaluate phase of MIDCA would have to determine not only whether goals had been achieved, but also if individual actions had in fact had their anticipated effects. If not, for example if the robot accidentally knocked down a block while trying to pick it up, MIDCA would have to not only replan, but also consider generating a goal to relearn or modify its grasping algorithm.

The transition being discussed above will not be a trivial one, and the methods described are only examples of approaches we might take. Our overall purpose is the integration of a low-level, data driven approach with a high-level, knowledge rich one in an environment that is amenable to both. These approaches have complimentary strengths. Low-level approaches tend to be robust with respect to noise but not in the face of fundamental shifts or unexpected variables. In contrast, reasoning-based approaches can provide insight into how to accommodate or (as in the fire example) counteract major changes or surprises, but are often difficult to apply to complex, noisy data. Uniting the two methodologies in increasingly complex, realistic and interesting ways and leveraging the enormous work that has been done on each individually will be a major focus of our research going forward.

Another area of future research is expanding and refining the capabilities of MIDCA's memory structures. First, we have begun work on adding a concept of time, so that each piece of knowledge that MIDCA acquires would be tied to two times – the time it was learned, and the range of times it applies to. Using this information, we could leverage work in active logic (see Elgot-Drapkin, Kraus, Miller, Nirkhe, Perlis, 1999; Anderson, Gomaa, Grant, Perlis, 2008) to allow MIDCA to make inferences about events in its past or future. For an autonomous agent that is expected to reason effectively about the consequences of its actions, the inclusion of temporal information in that reasoning is essential.

This is especially true if such an agent is expected to deal with humans, who build implicit and explicit temporal constraints into everything. For example, consider a robot which is attempting to choose between two goals it has been given: “Fetch Mary a soda” and “Make sure Joe gets this important document before he leaves.” In this case, the robot can only make an appropriate decision about which to pursue first by considering the time. If it is noon and Joe usually leaves at 5 PM, it

should probably fetch the soda first. Not only does it have plenty of time to deliver the document, but it may know that at noon, people have a high likelihood of being at lunch and unavailable to receive documents. Mary, on the other hand, might appreciate a soda with her lunch, and the time might give the robot a clue about where to start searching for her.

A second useful addition to the capabilities of MIDCA's memory would be an account of the sources of its knowledge. This is useful in instances in which an agent discovers that something it believed was actually false. For example, an agent told that mice were ten feet tall might infer that mice lived only outdoors and had no fear of cats. Upon seeing a mouse, it could still, without any extra information on knowledge sources, eliminate the original faulty fact from its knowledge base. However, unless a connection persisted between the original fact about mouse size and the conclusions that the agent inferred from it, it would have no way of eliminating those facts as well. In a world in which even humans are constantly wrong and need to readjust to new data, the ability to connect a faulty assumption to series of faulty conclusions that spring from it is valuable.

In addition to expansions of MIDCA's memory architecture, it may be desirable to update MIDCA's goal generation method to counter the current method's limitations of scale and the requirement for a training corpus which covers most situations. By making use of explanation patterns (XP's), a goal generation scheme which is in part logical in nature can be constructed based on the concept of precondition negation. An anomalous situation depends on various preconditions, and these causal relations can be represented in an XP. Negating one or more of these preconditions can then remedy the anomaly. The question of how to determine which precondition(s) expressed in the XP to negate is non-trivial, will require extensive research, and is highly sensitive to the number and nature of XP's available for examination.

4. Related Work

An initial introspective cognitive agent called INTRO (Cox, 2007) combines planning and understanding within a Wumpus World environment (Russell & Norvig, 2003) by integrating the PRODIGY planning and learning architecture (Veloso, Carbonell, Perez, Borrajo, Fink, & Blythe, 1995) with the Meta-AQUA story understanding and learning system (Cox & Ram, 1999). Rather than input all goals for the agent to achieve, the understanding component compares expected states and events in the world with those actually perceived to create an interpretation. When the interpretation discloses divergence from those expectations, INTRO generates its own goals to resolve the conflict. These new goals are then passed back to the PRODIGY component so that a plan can be generated and then executed.

Work has been done to expand and better the capacities of agents by making use of goal manipulations. (Hanheide et al., 2010) created a framework for managing goals to be used by a robot exploring an unknown space which autonomously classifies rooms into categories. They ran the robot with and without the framework, and concluded that a framework for goal management increases the performance of the robot. Our work too demonstrates that by allowing an agent to dynamically alter its goal structure performance can be greatly improved.

Schermerhorn, Benton, Scheutz, Talamadupula, & Kambhampati (2009) seek to use modification of a robot's goal structure to confront the challenges of a domain which is partially observable, non-deterministic, where prior knowledge about the domain is limited, knowledge acquisition is non-monotonic, planning is subject to real time constraints, and goals and utilities can dynamically change during execution. Counterfactuals are used to determine actions that could lead to goal opportunities, and when opportunities are detected, the goal structure can be modified.

Other work has taken advantage of the GDA model which we use in our work. For example, Munoz-Avila, Jaidee, and Aha (2010) merged the GDA framework with case based reasoning (CBR) and ran a comparison between a GDA system using CBR, a rule based variant of GDA, and a non-GDA based agent. The CBR based GDA system outperformed the others, and functioned by making use of a case base that mapped goals to expectations and a case base that mapped mismatches to new goals. Our work as well shows that a GDA based system outperforms non-GDA based systems.

The ARTUE (Autonomous Response to Unexpected Events) system (Molineaux, Klenk, & Aha, 2010) instantiates the GDA model. Its purpose is to be a domain independent autonomous agent with the capacity to dynamically determine which goals to pursue in unexpected situations. ARTUE uses hierarchical task networks for planning, takes advantage of explanations, and manages goals.

5. Conclusion

Full implementation of the MIDCA architecture will be a large project. In this paper we have introduced MIDCA_1.1, a working version of the cognitive layer of MIDCA in a very simple domain. Our experimental results to this point should be taken as illustrative rather than probative – they are intended to demonstrate the kind of problem we intend for MIDCA to be able to solve and the ways in which it might do so. We hope to build on the potential for goal driven autonomy to increase the flexibility and adaptability of agents in complex, changing environments. The demonstrated success of this method in our simple example should be a building block for similar work in more challenging domains.

The second major feature of our method is the synergy between D-track and K-track approaches. We have described the use of data-driven techniques in anomaly detection (A-distance), neural networks (growing neural gas), and machine learning (Tilde; FOIL) as well as a predicate logic state representation and techniques for explanation generation (Meta-AQUA) and planning (SHOP2) that rely on such high level formalisms. Both high level and low level approaches to AI have been used with great success in their individual spheres. We believe that the integration of these approaches is one of the most promising opportunities in modern AI, and one of the central focuses of MIDCA.

Another factor in the relevance of this project is the emergence of robots like the PR2 with advanced built-in capabilities in image processing, localization, locomotion and object manipulation. Two-way translation between raw input data and reasonable formalisms, and the integrated analysis of both levels of information to generate specific commands a robot can follow, are now, while still difficult, much more approachable tasks than they have been in the past. Our hope is that MIDCA can effect this translation. Our vision is of an agent whose autonomy and usefulness grows gradually as we increase both the quality of the algorithms governing each of its individual functions and the level of integration between them. This paper has presented the first iteration of that process.

Acknowledgements

This material is based upon work supported by ONR Grants # N00014-12-1-0430 and # N00014-12-1-0172 and by ARO Grant # W911NF-12-1-0471. We thank the anonymous reviewers for their comments and suggestions.

References

- Anderson, M. L., Gomaa, W., Grant, J., & Perlis, D. (2008). Active logic semantics for a single agent in a static world. *Artificial Intelligence*, 172(8), 1045-1063.
- Anderson, M., & Perlis, D. (2005). Logic, self-awareness and self-improvement. *Journal of Logic and Computation* 15, 21–40.
- Bloekel, H., & De Raedt, L. (1997). Lookahead and discretisation in ILP. In N. Lavrac & S. Džeroski (Eds.), *Proceedings of the seventh international workshop on inductive logic programming* (pp. 77–84) LNAI: Vol. 1297. Berlin: Springer
- Burstein, M. H., Laddaga, R., McDonald, D., Cox, M. T., Benyo, B., Robertson, P., Hussain, T., Brinn, M., & McDermott, D. (2008). POIROT - Integrated learning of web service procedures. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence* (pp. 1274-1279). Menlo Park, CA: AAAI Press.
- Cousins, S. (2010). ROS on the PR2 [ROS Topics]. *IEEE Robotics & Automation Magazine*, 17(3):23-25.
- Cox, M. T. (2007). Perpetual self-aware cognitive agents. *AI Magazine* 28(1), 32-45.
- Cox, M. T. (in press). Question-based problem recognition and goal-driven autonomy. To appear in D. W. Aha, M. T. Cox, & H. Munoz-Avila (Eds.), *Proceedings of the 2013 Annual Conference on Advances in Cognitive Systems: Workshop on Goal Reasoning* (Tech. Rep. No. CS-TR-5029). College Park, MD: University of Maryland, Department of Computer Science.
- Cox, M. T., & Burstein, M. H. (2008). Case-based explanations and the integrated learning of demonstrations. *Künstliche Intelligenz (Artificial Intelligence)* 22(2), 35-38.
- Cox, M. T., Maynard, M., Paisner, M., Perlis, D., & Oates, T. (2013). The integration of cognitive and metacognitive processes with data-driven and knowledge-rich structures. In *Proceedings of the Annual Meeting of the International Association for Computing and Philosophy*.
- Cox, M. T., Oates, T., Paisner, M., & Perlis, D. (2012). Noting anomalies in streams of symbolic predicates using A-distance. *Advances in Cognitive Systems* 2, 167-184.
- Cox, M. T., Oates, T., Paisner, M., & Perlis, D. (2013). Detecting change in diverse symbolic worlds. In L. Correia, L. P. Reis, L. M. Gomes, H. Guerra, & P. Cardoso (Eds.), *Advances in Artificial Intelligence, 16th Portuguese Conference on Artificial Intelligence* (pp. 179-190). University of the Azores, Portugal: CMATI.
- Cox, M. T., Oates, T., & Perlis, D. (2011). Toward an integrated metacognitive architecture. In P. Langley (Ed.), *Advances in Cognitive Systems: Papers from the 2011 AAAI Fall Symposium* (pp. 74-81). Technical Report FS-11-01. Menlo Park, CA: AAAI Press.
- Cox, M. T., & Ram, A. (1999). Introspective multistrategy learning: On the construction of learning strategies. *Artificial Intelligence*, 112, 1-55.
- Elgot-Drapkin, J., Kraus, S., Miller, M., Nirkhe, M., & Perlis, D. (1999). *Active logics: A unified formal approach to episodic reasoning* (Tech. Rep. No. CS-TR-4072). College Park, MD: University of Maryland, Department of Computer Science.
- Hanheide, M., Hawes, N., Wyatt, J., Göbelbecker, M., Brenner, M., Sjöö, K., & Aydemir, A. (2010). A framework for goal generation and management. *AAAI Workshop on Goal-Directed Autonomy*.

- Klenk, M., Molineaux, M., & Aha, D. (2013). Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence*, 29(2), 187–206, 2013.
- Lee, P., & Cox, M. T. (2002). Dimensional indexing for targeted case-base retrieval: The SMIRKS system. In S. Haller & G. Simmons (Eds.), *Proceedings of the 15th International FLAIRS Conference* (pp. 62-66). Menlo Park, CA: AAAI Press.
- Lenat, D., & Guha, R. (1989). *Building large knowledge-based systems*. Menlo Park, CA: Addison-Wesley.
- Maynord, M., Cox, M. T., Paisner, M., & Perlis, D. (2013). Data-driven goal generation for integrated cognitive systems. In C. Lebiere & P. S. Rosenbloom (Eds.), *Integrated Cognition: Papers from the 2013 Fall Symposium* (pp. 47-54). Menlo Park, CA: AAAI Press.
- Molineaux, M., Klenk, M., Aha, D. (2010). Goal-driven autonomy in a Navy strategy simulation. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.
- Munoz-Avila, H., Jaidee, U., Aha, D. (2010). Goal-Driven autonomy with case-based reasoning. In I. Bichindaritz & S. Montani (Eds.), *Proceedings of the 18th international conference on Case-Based Reasoning Research and Development (ICCBR'10)* (pp. 228-241). Berlin: Springer.
- Munoz-Avila, H., Jaidee, U., Aha, D. W., Carter, E. (2010). Goal-driven autonomy with case-based reasoning. In *Case-Based Reasoning. Research and Development, 18th International Conference on Case-Based Reasoning, ICCBR 2010* (pp. 228-241). Berlin: Springer.
- Nau, D., Au, T., Ilghami, O., Kuter, U., Murdock, J., Wu, D., & Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20, 379–404
- Paisner, M., Perlis, D., & Cox, M. T. (2013). Symbolic anomaly detection and assessment using growing neural gas. In *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence* (pp. 175-181). Los Alamitos, CA: IEEE Computer Society.
- Quinlan, J. (1993). *Programs for machine learning*. San Francisco, CA: Morgan Kaufmann.
- Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning* 5, 239-266.
- Russell, S., & Norvig, P. (2003). *Artificial intelligence: A modern approach (2ed)*. NJ: Prentice.
- Schank, R. C. (1986). *Explanation patterns: Understanding mechanically and creatively*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Schank, R. C., Kass, A., & Riesbeck, C. K. (1994). *Inside case-based explanation*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Schermerhorn, P., Benton, J., Scheutz, M., Talamadupula, K., Kambhampati, S. (2009). Finding and exploiting goal opportunities in real-time during plan execution. In *Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems (IROS'09)* (pp. 3912-3917). Piscataway, NJ: IEEE Press.
- Veloso, M., Carbonell, J. G., Perez, A., Borrajo, D., Fink, E., & Blythe, J. (1995). Integrating planning and learning: The PRODIGY architecture. *Journal of Theoretical and Experimental Artificial Intelligence*, 7(1), 81-120.

Hierarchical Goal Networks and Goal-Driven Autonomy: Going where AI Planning Meets Goal Reasoning

Vikas Shivashankar

SVIKAS@CS.UMD.EDU

Department of Computer Science, University of Maryland, College Park, MD 20742

Ron Alford

RONWALF@CS.UMD.EDU

Department of Computer Science, University of Maryland, College Park, MD 20742

Ugur Kuter

UKUTER@SIFT.NET

SIFT, LLC, 211 North 1st Street, Suite 300, Minneapolis, MN 55401-2078

Dana Nau

NAU@CS.UMD.EDU

Department of Computer Science and Institute for Systems Research, University of Maryland, College Park, MD 20742

Abstract

Planning systems are typically told what goals to pursue and cannot modify them. Some methods (e.g., for contingency planning, dynamic replanning) can respond to execution failures, but usually ignore opportunities and do not reason about the goals themselves. Goal-Driven Autonomy (GDA) relaxes some common assumptions of classical planning (e.g., static environments, fixed goals, no unpredictable exogenous events). This paper describes our Hierarchical Goal Network formalism and algorithms that we have been working on for some time now and discusses how this particular planning formalism relates to GDA and may help address some of GDA's challenges.

1. Motivation

In the real world, intelligent agents typically have to operate in environments that are open-world, partially observable and dynamic. Therefore, pure offline planning or even online plan-repair isn't sufficient to handle unanticipated situations in such environments: external events often necessitate postponing or even abandoning your current goals in order to pursue newer goals of higher priority. *Goal-driven autonomy* (GDA) (Molineaux, Klenk, & Aha, 2010) is a conceptual model of such a goal reasoning process.

As one can imagine, enabling agents to operate as described above requires advanced planning capabilities, in particular capabilities that are not well supported by the current state of the art automated planning formalisms and algorithms. In particular, most existing planning approaches assume *inactive* goals; i.e., goals for planning problems are specified at the start of the planning process and never change during the course of that process. Reasoning about *active* goals and

planning for them is a core foundation for GDA, and we believe, there should be more research on integrating AI planning techniques with active goal reasoning.

In this paper, we will firstly pin down some key capabilities for active goal reasoning in planning that would be useful in the GDA architecture. We will then describe our ongoing research on a knowledge-based planning formalism called **Hierarchical Goal Network (HGN)** planning that makes first steps towards achieving some of these capabilities. HGN planning already supports some of the features desirable from a GDA standpoint such as:

- **Specification of complex domain-specific knowledge as HGN methods** These are similar to methods in Hierarchical Task Network (HTN) planning (Erol, Hendler, & Nau, 1994), but decompose goals into subgoals (instead of tasks into subtasks). When multiple methods can be used to achieve the given goal, we can use domain-independent heuristics to automatically detect the most promising methods.
- **Planning with Incomplete Models.** Knowledge-based planning formalisms are often too dependent on the input domain-specific knowledge; in particular, any missing/incorrect knowledge can break the planner. HGN planning on the other hand allows specification of arbitrary amounts of planning knowledge: it uses the given knowledge whenever applicable and falls back to domain-independent planning techniques including landmark reasoning (Richter & Westphal, 2010) and action chaining to fill in the gaps.
- **Managing Agenda of Goals.** In the GDA model, managing the agenda of goals and deciding which goals to achieve next is outside the purview of the planner. However, generating plans for each candidate goal and comparing the results is often the best way to decide which goal should be pursued next. The core data structure in HGN planning is a *goal network*, which captures precisely this functionality.

HGNs also have a potential to help achieve more advanced planning functionalities including:

- **Goal-based Plan Repair.** In dynamic environments, it is hard for the domain author to anticipate and provide knowledge for all the various scenarios that the agent could face. Since HGN planning is robust to incomplete and incorrect domain models, it is much better equipped to plan in unanticipated situations for which the user has not provided any domain knowledge.
- **Temporal and Cost-Aware Planning.** Since HGN planning uses goals, it is easier to adapt techniques from domain-independent planning literature related to cost-aware planning and temporal planning than in HTN planning.

Planning algorithms in the GDA model ought to be able to continuously assess their current situation, recognize and reason about both serendipitous and harmful events that might change the world around them exogenously, adapt their goals in accordance with these events, and finally generate plans to achieve these goals. HGNs provide a formal foundation for developing algorithms to provide these capabilities.

The remainder of this paper is structured as follows. Section 2 provides some background on GDA. Section 3 presents the HGN planning formalism (Shivashankar et al., 2012). Section 4 then

reviews **Goal Decomposition Planner (GDP)**, the algorithm we proposed to solve HGN planning problems in (Shivashankar et al., 2012). Section 5 then presents the **Goal Decomposition with Landmarks (GoDeL)** algorithm, an extension of GDP that can work with partial domain-specific knowledge which we proposed in (Shivashankar et al., 2013). Section 6 discusses the potential role of HGNs in providing planning capabilities within GDA. Finally, we conclude in Section 7.

2. GDA Preliminaries

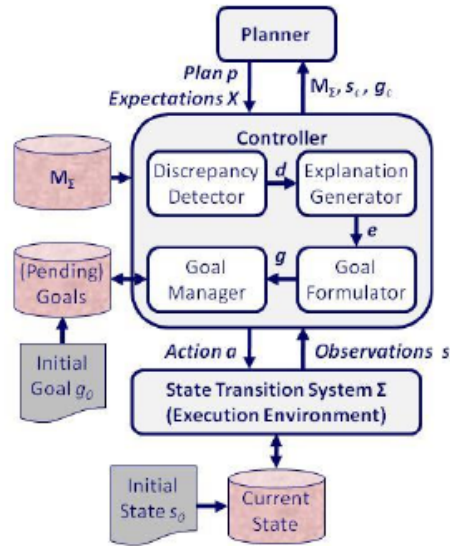


Figure 1. Conceptual Model of GDA (Molineaux, Klenk, & Aha, 2010)

Figure 1 shows a schematic representation of the GDA model. It consists of three components:

- **Planner** This module takes in the current planning problem $P = (M_\Sigma, s_c, g_c)$, where M_Σ is the current model of the environment and s_c, g_c respectively are the current state and goal, and computes a plan p for the agent to execute. It also returns a sequence of expectations $X_c = [x_c, \dots, x_{c+n}]$ which models constraints on the states $[s_c, \dots, s_{c+n}]$ that the planner expects the agent to pass through when executing p .
- **Controller** This takes the plan p and sends the actions one by one to the environment Σ and processes the resulting state observations as follows:
 - *Discrepancy Detector* compares actual state observations with its expectations and generates a set of discrepancies D ,
 - *Explanation Generator* hypothesizes one or more explanations E for D ,
 - *Goal Formulator* generates zero or more goals G in response to E ,

- *Goal Manager* updates G_{pend} , the set of pending goals, with G and optionally changes the set of active goals to be given to the planner.
- **Environment** This takes an action and executes it in the current state and returns observations from the resulting state.

From this model, we can infer some key capabilities that the Planner module has to support:

- **Compatibility with Goals** Since problems in GDA are modeled as goals that need to be achieved, planners need to be able to solve goal achievement problems. In addition, planners should also support complex goal representations such as open-world quantified goals (Talamadupula et al., 2010) and maintenance-style goals such as in Linear Temporal Logic (LTL).
- **Plan Generation** Given a planning problem, the planner should be able to compute plans efficiently.
- **Online Plan Repair** In case exogenous events break the current plan or necessitate change in goals, the planner should be able to repair the current plan.
- **Online Plan Optimization** The planner should continuously optimize the current plan during execution as more time is given. Additionally, It should be able to take advantage of serendipitous events to improve the quality of the plan.
- **Goal Management** In the original GDA model, the goal management is done outside of the planner. However, it is often hard to decide which goal to plan for next among multiple alternatives without actually planning for these goals and comparing the plan qualities. Therefore, we believe that managing the set of goals and deciding which goal to pursue next should also be done by the planner.

2.1 Evaluating Suitability of Domain-Independent Planning for GDA

Domain-Independent planning (Ghallab, Nau, & Traverso, 2004) algorithms in principle fits very well into the GDA model since they work with goals. However, since they cannot exploit any domain-specific knowledge outside of the action models, the performance of these planners rarely scales in complex, real-world domains. Moreover, most domain-independent planners follow assumptions from *classical planning* which restrict goals to conjunctive boolean formulas.

There exists some work on augmenting domain-independent planners with plan repair capabilities (Fox et al., 2006; Hammond, 1990) which perform better than simple replanning strategies. There also exist anytime planning algorithms such as LAMA (Richter & Westphal, 2010) which compute initial solutions and then proceed to optimize these solutions if given additional time. However, similar scalability problems exist as with the plan generation algorithms.

2.2 Evaluating Suitability of Hierarchical Task Network Planning for GDA

HTN planning (Erol, Hendler, & Nau, 1994; Nau et al., 2003), in contrast to domain-independent planning, models planning problems as accomplishment of complex tasks. Another difference

from domain-independent planning is that HTN planning takes as input additional domain-specific knowledge in the form of *HTN methods* which provide ways to decompose tasks into simpler sub-tasks. HTN planning algorithms thus can be finely tuned to achieve much better performance, especially in complex domains. However, HTN planning algorithms are extremely sensitive to the correctness and completeness of the input method set (Shivashankar et al., 2012): missing/incorrect methods can result in the algorithm not finding valid solutions¹.

Moreover, while there were a number of attempts to combine HTN planning with domain-independent planning (Kambhampati, Mali, & Srivastava, 1998; Mccluskey, 2000; Biundo & Schatzenberg, 2001; Alford, Kuter, & Nau, 2009; Gerevini et al., 2008) to work with incomplete HTN method sets, the lack of correspondence between tasks and goals has interfered with these efforts necessitating several *ad hoc* modifications and restrictions.

The disconnect between tasks and goals has also interfered with efforts to augment HTN planning with plan repair capabilities (Ayan et al., 2007; Warfield et al., 2007). (Hawes, 2001; Hawes, 2002) provide HTN planning algorithms that work in an anytime manner. These algorithms however, instead of generating a concrete solution and then iteratively improving it in the remaining time, return a plan at varying levels of abstraction based on when the planner is interrupted; this limits its usefulness since applications typically require a concrete plan that is executable.

3. Hierarchical Goal Networks

As Sections 2.1 and 2.2 indicate, domain-independent planning and HTN planning formalisms have different (and in fact, complementary) advantages and disadvantages. Therefore, we attempted to come up with a hybrid formalism that is hierarchical, but is based on goals rather than tasks. This idea is not new; SIPE (Wilkins, 1984) and PRS (Georgeff & Lansky, 1987; Ingrand & Georgeff, 1992) both use goal decomposition techniques in order to build planning systems. Instead, our contributions lie in (1) building a formal theory and analyzing properties of a hierarchical goal-based planning framework, and (2) exploiting connections between HGNs and techniques in domain-independent planning such as state-space heuristics and landmark reasoning. Below we formalize HGN planning.

Classical planning. Following Ghallab, Nau and Traverso (2004), we define a classical planning domain D as a finite state-transition system in which each state s is a finite set of ground atoms of a first-order language L , and each action a is a ground instance of a planning operator o . A planning operator is a triple $o = (\text{head}(o), \text{precond}(o), \text{effects}(o))$, where $\text{precond}(o)$ and $\text{effects}(o)$ are sets of literals called o 's *preconditions* and *effects*, and $\text{head}(o)$ includes o 's *name* and *argument list* (a list of the variables in $\text{precond}(o)$ and $\text{effects}(o)$).

An action a is executable in a state s if $s \models \text{precond}(a)$, in which case the resulting state is $\gamma(a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$, where $\text{effects}^+(a)$ and $\text{effects}^-(a)$ are the atoms and negated atoms, respectively, in $\text{effects}(a)$. A plan $\pi = \langle a_1, \dots, a_n \rangle$ is executable in s if each a_i is executable in the state produced by a_{i-1} ; and in this case we let $\gamma(s, \pi)$ be the state produced by executing the entire plan.

1. For example, SHOP (Nau et al., 2001) was disqualified from the 2000 International Planning Competition because of an error made by SHOP's authors when they wrote their HTN version of one of the planning domains.

<p>Method for using truck ?t to move crate ?o from location ?l1 to location ?l2 in city ?c:</p> <p><i>Head:</i> (move-within-city ?o ?t ?l1 ?l2 ?c) <i>Pre:</i> ((obj-at ?o ?l1) (in-city ?l1 ?c) (in-city ?l2 ?c) (truck ?t ?c) (truck-at ?t ?l3)) <i>Sub:</i> ((truck-at ?t ?l1) (in-truck ?o ?t) (truck-at ?t ?l2) (obj-at ?o ?l2)))</p> <p>Method for using airplane ?plane to move crate ?o from airport ?a1 to airport ?a2:</p> <p><i>Head:</i> (move-between-airports ?o ?plane ?a1 ?a2) <i>Pre:</i> ((obj-at ?o ?a1) (airport ?a1) (airport ?a2) (airplane ?plane)) <i>Sub:</i> ((airplane-at ?plane ?a1) (in-airplane ?o ?plane) (airplane-at ?plane ?a2) (obj-at ?o ?a2)))</p> <p>Method for moving ?o from location ?l1 in city ?c1 to location ?l2 in city ?c2, via airports ?a1 and ?a2:</p> <p><i>Head:</i> (move-between-cities ?o ?l1 ?c1 ?l2 ?c2 ?a1 ?a2) <i>Pre:</i> ((obj-at ?o ?l1) (in-city ?l1 ?c1) (in-city ?l2 ?c2) (different ?c1 ?c2) (airport ?a1) (airport ?a2) (in-city ?a1 ?c1) (in-city ?a2 ?c2)) <i>Sub:</i> ((obj-at ?o ?a1) (obj-at ?o ?a2) (obj-at ?o ?l2)))</p>

Figure 2. HGN methods for transporting a package to its goal location in the Logistics domain.

A *classical planning problem* is a triple $P = (D, s_0, g)$, where D is a classical planning domain, s_0 is the initial state, and g (the *goal formula*) is a set of ground literals. A plan π is a solution for P if π is executable in s_0 and $\gamma(s_0, \pi) \models g$.

HGN planning. An *HGN method* m has a head $\text{head}(m)$ and preconditions $\text{precond}(m)$ like those of a planning operator, and a sequence of subgoals $\text{subgoals}(m) = \langle g_1, \dots, g_k \rangle$, where each g_i is a goal formula (a set of literals). We define the *postcondition* of m to be $\text{post}(m) = g_k$ if $\text{subgoals}(m)$ is nonempty; otherwise $\text{post}(m) = \text{precond}(m)$. See Figure 2 for an example set of HGN methods for the Logistics domain.

An action a (or grounded method m) is *relevant* for a goal formula g if $\text{effects}(a)$ (or $\text{post}(m)$, respectively) entails at least one literal in g and does not entail the negation of any literal in g .

Some notation: if π_1, \dots, π_n are plans or actions, then $\pi_1 \circ \dots \circ \pi_n$ denotes the plan formed by concatenating them.

An *HGN planning domain* is a pair $D = (D', M)$, where D' is a classical planning domain and M is a set of methods.

A *goal network* is a way to represent the objective of satisfying a partially ordered sequence of goals. Formally, it is a pair $gn = (T, \prec)$ such that:

- T is a finite nonempty set of nodes;
- each node $t \in T$ contains a *goal* g_t that is a DNF (disjunctive normal form) formula over ground literals;
- \prec is a partial order over T .

An *HGN planning problem* is a triple $P = (D, s_0, gn)$, where D is a planning domain, s_0 is the initial state, and $gn = (T, \prec)$ is a goal network.

Definition 1. *The set of solutions for P is defined as follows:*

Case 1. *If T is empty, the empty plan is a solution for P .*

Case 2. *Let t be a node in T that has no predecessors. If $s_0 \models g_t$, then any solution for $P' = (D, s_0, (T', \prec'))$ is also a solution for P , where $T' = T - \{t\}$, and \prec' is the restriction of \prec to T' .*

Case 3. *Let a be any action that is relevant for g_t and executable in s_0 . Let π be any solution to the HGN planning problem $(D, \gamma(s_0, a), (T', \prec'))$. Then $a \circ \pi$ is a solution to P .*

Case 4. *Let m be a method instance that is applicable to s_0 and relevant for g_t and has subgoals g_1, \dots, g_k . Let π_1 be any solution for (D, s_0, g_1) ; let π_i be any solution for $(D, \gamma(s_0, (\pi_1 \circ \dots \circ \pi_{i-1})), g_i)$, $i = 2, \dots, k$; and let π be any solution for $(D, \gamma(s_0, (\pi_1 \circ \dots \circ \pi_k)), (T', \prec'))$. Then $\pi_1 \circ \pi_2 \circ \dots \circ \pi_k \circ \pi$ is a solution to P .*

In the above definition, the relevance requirements in Cases 2 and 3 prevent classical-style action chaining unless each action is relevant for either the ultimate goal g or a subgoal of one of the methods. This requirement is analogous to (but less restrictive than) the HTN planning requirement that actions cannot appear in a plan unless they are mentioned explicitly in one of the methods. As in HTN planning, it gives an HGN planning problem a smaller search space than the corresponding classical planning problem.

4. Goal Decomposition Planner

Algorithm 1 is GDP, our HGN planning algorithm. It works as follows:

In Line 3, if gn is empty then the goal has been achieved, so GDP returns π . Otherwise, GDP selects a goal g in gn without any predecessors (Line 4). If g is already satisfied, GDP removes g from gn and calls itself recursively on the resulting goal network.

In Lines 7-8, if no actions or methods are applicable to s and relevant for g , then GDP returns failure. Otherwise, GDP nondeterministically chooses an action/method u from U .

If u is an action, then GDP computes the next state $\gamma(s, u)$ and appends u to π . Otherwise u is a method, so GDP inserts u 's subgoals at the front of g . Then GDP calls itself recursively on gn .

Algorithm 1: A high-level description of GDP. Initially, D is an HGN planning domain, s is the initial state, gn is the goal network, and π is $\langle \rangle$, the empty plan.

```

1 Procedure GDP( $D, s, gn, \pi$ )
2 begin
3   if  $gn$  is empty then return  $\pi$ 
4    $g \leftarrow$  goal formula in  $gn$  with no predecessors
5   if  $s \models g$  then
6     | remove  $g$  from  $gn$  and return GDP( $D, s, gn, \pi$ )
7    $U \leftarrow$  {actions and method instances that are relevant for  $g$  and applicable to  $s$ }
8   if  $U = \emptyset$  then return failure
9   nondeterministically choose  $u \in U$ 
10  if  $u$  is an action then append  $u$  to  $\pi$  and set  $s \leftarrow \gamma(s, u)$ 
11  else insert  $\text{subgoals}(u)$  as predecessors of  $g$  in  $gn$ 
12  return GDP( $D, s, gn, \pi$ )
13 end

```

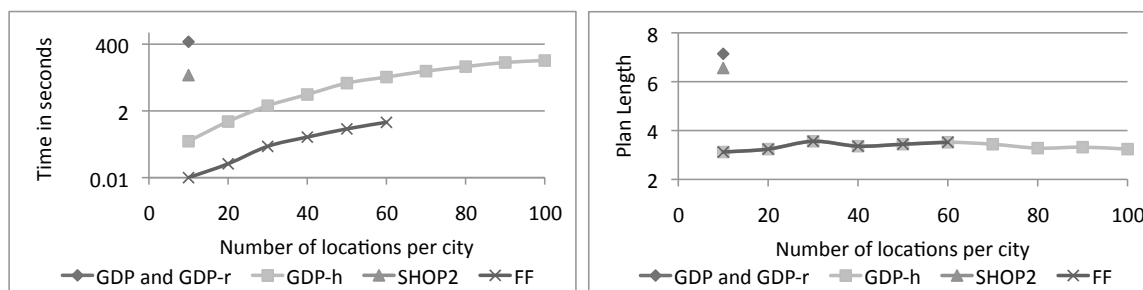


Figure 3. Average running times (in logscale) and plan lengths in the 3-City Routing domain, as a function of the number of locations per city. Each data point is an average of 25 randomly generated problems. FF couldn't solve problems involving more than 60 locations while GDP and SHOP2 could not solve problems with more than 10 locations. SHOP2- r could not solve a single data point, because of which it was excluded from the figure.

Experimental Results. To evaluate the performance of GDP, we compared it with the state-of-the-art HTN planner SHOP2 (Nau et al., 2003) and the domain-independent planner FF (Hoffmann & Nebel, 2001) across five domains. GDP- h represents GDP enhanced with a variant of the relaxed planning graph heuristic used in FF. GDP- r and SHOP2- r are identical to GDP and SHOP2 respectively, but with the input method sets randomly reordered. We did this to investigate the effect of methods being provided in the wrong order on the planner's performance. Here we present results from only one domain: *One-City Routing*; the complete experimental study is detailed in (Shivashankar et al., 2012).

We constructed the *3-City Routing* domain in order to examine the performance of the planners in a domain with a weak domain model. In this domain, there are three cities c_1 , c_2 and c_3 , each containing n locations internally connected by a network of randomly chosen roads. In addition,

there is one road between a randomly chosen location in c_1 and a randomly chosen location in c_2 , and similarly another road between locations in c_2 and c_3 . The problem is to get from a location in c_1 or c_3 to a goal location in c_2 .

We randomly generated 25 planning problems for each value of n , with n varying from 10 to 100. For the road networks, we used near-complete graphs in which 20% of the edges were removed at random. Note that while solutions to such problems are typically very short, the search space has extremely high branching factor, i.e. of the order of n . For GDP and GDP- h , we used a single HGN method, shown here as pseudocode:

- **To achieve** at (b)
 precond: at (a), adjacent (c, b)
 subgoals: achieve at (c) and then at (b)

By applying this method recursively, the planner can do a backward search recursively from the goal location to the start location.

To accomplish the same backward search in SHOP2, we needed to give it three methods, one for each of the following cases: (1) goal location same as the initial location, (2) goal location one step away from the initial location, and (3) arbitrary distance between the goal and initial locations.

As Figure 3 shows, GDP and SHOP2 did not solve the randomly generated problems except the ones of size 10, returning very poor solutions and taking large amounts of time in the process. GDP- h , on the other hand solved all the planning problems quickly, returning near-optimal solutions. The reason for the success of GDP- h is that the domain knowledge specified above induce an unguided backward search in the state space and the planner uses the domain-independent heuristic to select its path to the goal.

GDP- r performed at par with GDP since there was only one method, and hence reordering did not affect it. SHOP2- r , on the other hand, could not solve a single problem, thus illustrating the high sensitivity to the correctness of not only the methods, but also the *order* in which they are provided. We believe that GDP is also similarly sensitive to the method order; on the other hand, GDP- h , due its domain-independent heuristic-based method ordering technique, is agnostic to the method ordering provided by the domain author.

FF was able to solve all problems up to $n = 60$ locations, after which it could not even complete parsing the problem file. This has to do with FF grounding all the actions right in the beginning, which it could not do for the larger problems.

Domain Authoring. When writing the domain models for our experiments, it seemed to us that writing the GDP domain models was easier than writing the SHOP2 domain models—so we made measurements to try to verify whether this subjective impression was correct. Figure 4 compares the sizes of the HGN and HTN domain descriptions of the planning domains. In almost all of them, the domain models for GDP were much smaller than those for SHOP2. This is the case because the HGN task and method semantics obviates the need for a plethora of extra methods and bookkeeping operations needed in HTN domain models (see (Shivashankar et al., 2012) for a more detailed explanation).

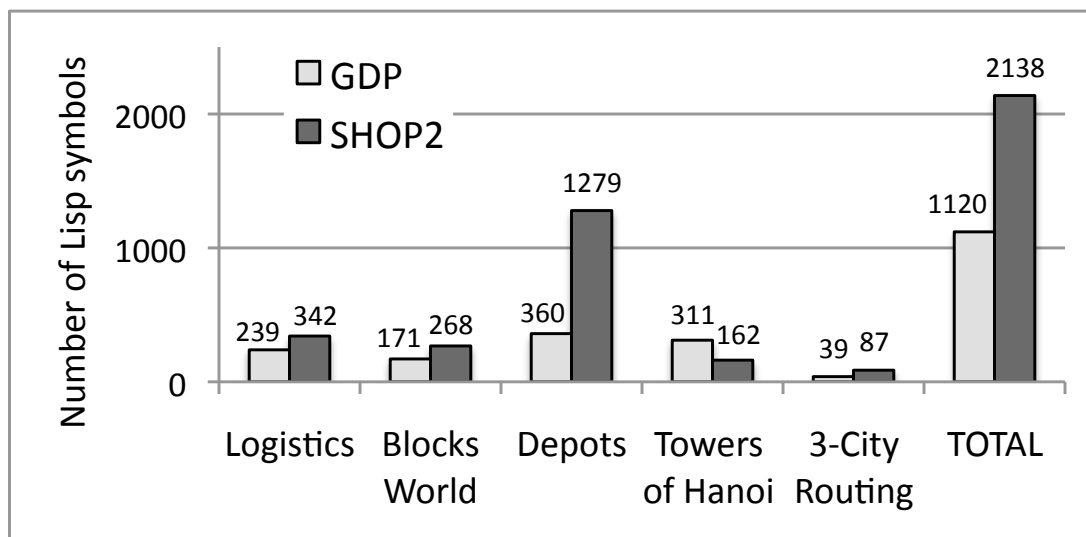


Figure 4. Sizes (number of Lisp symbols) of the GDP and SHOP2 domain models.

5. The GoDeL Planning Formalism and Algorithm

GDP is similar to SHOP/SHOP2 in that it is sound and complete only with respect to the methods provided to it. It therefore provides no completeness guarantees if the HGN methods are incomplete. GoDeL is an extension of GDP that does provide completeness guarantees even if given incomplete or empty method sets. It accomplishes this by interleaving method decomposition with subgoal inference using landmarks and action chaining.

Explanation of pseudocode. Algorithm 2 is the GoDeL planning algorithm. It takes as input a planning problem $P = (D, s, G)$, a set of methods M , and the partial plan π generated so far.

Lines 3 – 6 specify the base cases of GoDeL. If these are not satisfied, the algorithm nondeterministically chooses a goal g with no predecessors and generates \mathcal{U} , all method and operator instances applicable in s and relevant to g . It then nondeterministically chooses a $u \in \mathcal{U}$ to progress the search (Lines 7 – 9). If u is an action, the state is progressed to $\gamma(s, u)$ (Line 10). Else, the subgoals of u are added to G , adding edges to preserve the total order imposed on $\text{subgoals}(u)$ (Line 12). In either case, GoDeL is invoked recursively on the new planning problem (Lines 12 and 14).

If GoDeL fails to find a plan in the previous step, it then attempts to infer a partially ordered set of subgoals lm that are to be achieved enroute to achieving g using the Infer-Subgoals procedure (Line 15). The subgoals are added to G , adding edges to preserve the partial order in lm . The algorithm is then recursively invoked on the new planning problem (Line 17). If this call also returns failure, then the algorithm falls back to action chaining (Lines 19 – 22), returning failure if no actions are applicable in s .

Below, we shall motivate and describe the Infer-Subgoals procedure in greater detail.

Subgoal inference. Sometimes, the planner may have methods that tell how to solve some subproblems, but not the top-level problem. For instance, the first method in Figure 2 would not

Algorithm 2: A nondeterministic version of GoDeL. Initially, (D, s, G) is the planning problem, M is a set of methods, and π is $\langle \rangle$, the empty plan.

```

1 Procedure GoDeL( $D, s, G, M, \pi$ )
2 begin
3   if  $G$  is empty then return  $\pi$ 
4   nondeterministically choose a goal formula  $g$  in  $G$  without any predecessors
5   if  $s \models g$  then
6     return GoDeL( $D, s, G - \{g\}, M, \pi$ )
7    $\mathcal{U} \leftarrow \{\text{operator and method instances applicable to } s \text{ and relevant to } g\}$ 
8   while  $\mathcal{U}$  is not empty do
9     nondeterministically remove a  $u$  from  $\mathcal{U}$ 
10    if  $u$  is an action then
11       $\text{res1} \leftarrow \text{GoDeL}(D, \gamma(s, u), G, M, \pi \circ u)$ 
12    else
13       $\text{res1} \leftarrow \text{GoDeL}(D, s, \text{subgoals}(u) \circ G, M, \pi)$ 
14    if  $\text{res1} \neq \text{failure}$  then return  $\text{res1}$ 
15   $\text{lm} \leftarrow \text{Infer-Subgoals}(D, s, g)$ 
16  if  $\text{lm} \neq \emptyset$  then
17     $\text{res2} \leftarrow \text{GoDeL}(D, s, \text{lm} \circ G, M, \pi)$ 
18    if  $\text{res2} \neq \text{failure}$  then return  $\text{res2}$ 
19   $\mathcal{A} \leftarrow \{\text{operator instances applicable to } s\}$ 
20  if  $\mathcal{A} = \emptyset$  then return failure
21  nondeterministically choose an  $a \in \mathcal{A}$ 
22  return GoDeL( $D, \gamma(s, a), G, M, \pi \circ a$ )
23 end

```

be applicable to problems involving transporting packages across cities, but it *is* applicable to the subproblems of moving the package between the start and goal locations and the corresponding airports. A natural question that then arises is the following: *How can we automatically infer these subproblems for which the given methods are relevant?*

To answer the above question, we use landmarks. A *landmark* for a planning problem P (Hoffmann, Porteous, & Sebastia, 2004; Richter & Westphal, 2010) is a fact that is true at some point in every plan that solves P . A *landmark graph* is a directed graph whose nodes are *landmarks* and edges denote orderings between these landmarks. Therefore, if there is an edge between two landmarks l_i and l_j , this implies that l_i is true before l_j in every solution to P .

Therefore, a landmark for a planning problem P can be thought of as a subgoal that every solution to P must satisfy at some point. We can, as a result, use any landmark generation algorithm (for example, (Hoffmann, Porteous, & Sebastia, 2004; Richter & Westphal, 2010)) to automatically infer subgoals (and orderings between them) for which the given methods are relevant.

Algorithm 3: Procedure to deduce possible subgoals for GoDeL to use. It takes as input a planning problem $P = (D, s, g)$, and outputs a DAG of subgoals. It uses LMGGEN, a landmark generation algorithm that takes P and generates a DAG (V, E) of landmarks for it.

```

1 Procedure Infer-Subgoals  $(D, s, g)$ 
2 begin
3    $(V, E) \leftarrow \text{LMGGEN}(D, s, g)$ 
4    $L \leftarrow \{v \in V : s \not\models \text{fact}(v), g \not\models \text{fact}(v) \text{ and } \exists \text{ a method } m \in D \text{ s.t. } \text{goal}(m) \text{ is relevant}$ 
   to  $\text{fact}(v)\}$ 
5   if  $L = \emptyset$  then return  $\emptyset$ 
6    $E_L \leftarrow \{(u, v) : u, v \in L \text{ and } v \text{ is reachable from } u \text{ in } (V, E)\}$ 
7   return  $(L, E_L)$ 
8 end

```

Algorithm 3 is Infer-Subgoals, the subgoal inference procedure. It uses a landmark generation procedure LMGGEN (such as ones in (Richter & Westphal, 2010; Hoffmann, Porteous, & Sebastia, 2004)) that takes as input a classical planning problem P and generates a DAG of landmarks.

Infer-Subgoals begins by computing LM_graph , the landmark graph for the input problem P . It then computes L , the set of nodes in LM_graph that have relevant methods (Line 4). It does not consider trivial landmarks such as literals true in the state s or part of the goal g . E_L is the set of all edges between landmarks $l_i, l_j \in L$ such that there exists a path from l_i to l_j in LM_graph (Line 6). Infer-Subgoals returns the resulting partial order $\text{lm} = (L, E_L)$.

Experimental Results. The main purpose of this experimental study was to investigate the performance of GoDeL with varying amounts of domain knowledge. In particular, we hypothesized that GoDeL would (1) perform at par with state-of-the-art domain-independent planners when given low/no knowledge, (2) improve performance as more knowledge is given, and finally (3) perform at par with state-of-the-art hierarchical planners when given complete knowledge.

To verify this hypothesis, we compared (1) GDP- h , the heuristic enhanced version of GDP from Section 4, (2) the state of the art domain-independent planner LAMA (Richter & Westphal, 2010) and (3) GoDeL run with three different amounts of domain knowledge: GoDeL with complete knowledge (GoDeL- C), moderate knowledge (GoDeL- M), and low knowledge (GoDeL- L). We compared these planners across three domains; we present the results only for Depots. The full experimental study is reported in (Shivashankar et al., 2013).

The experimental results on the Depots domain are consistent with our hypotheses. As shown in Figure 5, GoDeL- C has the best running times in Depots, and is the only planner to solve all of the problems. Even GoDeL- M significantly outperformed GDP- h , which had access to the full method set. GoDeL- L , which was given only one Blocks-World method, solved significantly fewer problems. LAMA solved the fewest problems, having not solved any problems containing 24 blocks or more. With respect to plan lengths, GoDeL- C , GoDeL- M and GDP- h produced plans of similar lengths for the problems they could solve. GoDeL- L and LAMA however generate significantly longer plans for the problems they could solve.

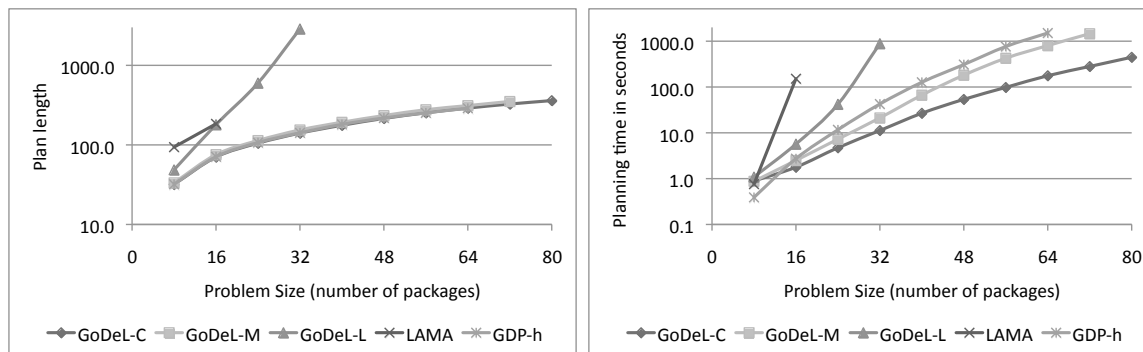


Figure 5. Average plan lengths and running times (both in logscale) in the Depots domain, as a function of the number of packages to be delivered. Each data point is an average of 25 randomly generated problems. GoDeL-M could not solve 80-package problems; GDP-h, GoDeL-L and LAMA could not solve problems of size > 64, 32 and 16 respectively.

6. Role of HGNs in GDA

We presented the HGN planning formalism in Section 3 and two HGN planning algorithms in Sections 4 and 5. However, the question that still remains to be answered is: *Why use HGNs in GDA?* In this section, we address this by discussing the main features of HGN planning algorithms and how they help address some of the planning challenges posed in GDA. We shall also discuss how HGN planning is in a unique position to solve some of the more advanced planning problems in goal reasoning.

6.1 Modeling Domain-Specific Knowledge using HGN Methods

One of the strengths of using HTNs is that one can model complicated pieces of knowledge as HTN methods, thus not only speeding up search but also enabling fine-tuning of the types of solutions you want the planner to return. HGNs also provide the same amount of control, but with the added benefit of retaining the semantics of goals in the methods (see (Shivashankar et al., 2012) for a discussion about incompatibilities between task and goal semantics). We show in (Shivashankar et al., 2012) that in fact HGNs are equal in expressivity to total-order HTN planning, thus indicating that one will not lose any expressivity when moving from HGNs to HTNs.

6.2 Managing Agenda of Goals using HGNs

In the GDA model, managing the agenda of goals and deciding which goals to achieve next is outside the purview of the planner. As we mentioned previously, sometimes generating plans for the candidate goals and comparing them is often the best way to decide which goal should be pursued next. Moreover, certain goal orderings work much better than others; determining a feasible order in which goals should be achieved is a problem the planner can help solve. HGNs provide a framework to do this reasoning: since problems are modeled as $(state, goalnetwork)$ pairs, the agenda of goals can be managed within the planner as a goal network and the planner could use its

heuristics in combination with any other domain-specific goal choice strategy to choose next goals. In particular, the definition of a goal network could be extended to include *utilities* over the goals, which could even change over time (and thus potentially trigger replanning). (Wilson, Molineaux, & Aha, 2013) adopt a similar approach of letting the planner evaluate candidate next goals in a domain-independent fashion in the context of HTNs. They do not, however, factor in implications that achieving a particular goal might have on the (un)solvability of another goal downstream.

6.3 Seamless Use of Incomplete HGN Domain Models

As mentioned previously, one of the main criticisms of HTN planning is the brittleness of the planners due to the over-reliance on the correctness and completeness of HTN methods. GoDeL on the other hand can work with any arbitrary sets of HGN methods. This is possible mainly due to goals in HGN methods which makes it immediately compatible with domain-independent planning. Therefore, domain engineers can provide domain knowledge only for the most complicated parts of the domain and rely on GoDeL to figure out the rest, thus easing the burden of domain authoring. This becomes even more crucial in complex real-world domains where it's unrealistic to assume that a complete suite of domain knowledge can be made available to the planner.

6.4 Plan Repair in Dynamic Environments

The domain authoring issue is further exacerbated in dynamic environments since other agents, exogenous events, and erroneous sensor and action models can bring about unanticipated situations which the domain author cannot anticipate offline. Therefore, it is even more critical in such cases that the planner has a fallback mechanism in case the provided knowledge is insufficient. Therefore, GoDeL augmented with plan repair capabilities will be perfectly suited for such use-cases.

6.5 Temporal Planning and Cost-Aware Planning

One of the advantages in moving from task networks to goal networks is that we can adapt existing planning heuristics from the domain-independent planning literature to work with HGNs. We provided a proof-of-concept of this by augmenting GDP, our first HGN planner, with a variant of the *relaxed planning graph heuristic* used in the FastForward planning system (Hoffmann & Nebel, 2001). This helps the planner in cases when multiple methods are applicable at a certain point in the search and the planner needs a way to decide which methods to try out first.

Along the same lines, we can adapt *admissible* heuristics (Karpas & Domshlak, 2009) that, if used with an algorithm like A*, can assure cost-optimal plans. We can also aim for a principled compromise via anytime planning techniques (Richter & Westphal, 2010), where we do not assure cost optimality, but instead try to improve the current best solution by exploring promising parts of the search space yet unexplored in an anytime fashion and asymptotically reach the optimal solution.

We can also augment the HGN formalism to reason about temporal aspects of planning via durative actions and deadlines on goals. These extensions, we think, will be very useful from a GDA standpoint since agents often have to reason about resource constraints, deadlines, and action costs; therefore it will be helpful to have planners that can actively optimize with respect to these metrics in a scalable manner.

7. Final Remarks and Future Work

We have described our planning formalism, called *Hierarchical Goal Networks (HGNs)*, and our planning algorithm based on this formalism. We argued that HGNs are particularly relevant and useful for GDA because they provide flexibility and reasoning capabilities in goal reasoning, during both planning and execution.

We're currently developing a generalization of HGNs for temporal goal reasoning. This work involves extending the HGN formalism described here with durative tasks and deadlines on goal achievement. The GoDeL algorithm for HGN planning must be generalized for reasoning with such temporal constructs. One of the challenges in this work is to model, correctly, the temporal semantics between planning and execution that will yield sound behavior in the generated plans.

Furthermore, to facilitate GDA, we want to extend HGN planning to include ways of reasoning and planning with goals under execution failures. Execution failures typically stem from the discrepancies from the planning and goal model that a system has and the ground-truth discovered during execution. We're interested in investigating how a planner can help reorder goal preferences during execution when such discrepancies occur, and repair the existing plans accordingly.

Finally, humans can reason about goals in very abstract levels and semantically, whereas automated planning and learning algorithms are computationally effective but has to stay in the realm of logical formulations. This comparison is perhaps similar to the comparisons between humans and AI in image processing tasks. For example, during their planning and acting, humans can identify a goal that may not be on their initial agenda but is a "low-hanging fruit." We're currently investigating how to model this concept in an AI planning and execution algorithm.

Acknowledgements. This work was supported in part by ARO grant W911NF1210471 and ONR grants N000141210430 and N000141310597. The information in this paper does not necessarily reflect the position or policy of the funders; no official endorsement should be inferred.

References

- Alford, R., Kuter, U., & Nau, D. S. (2009). Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. *IJCAI*.
- Ayan, N. F., Kuter, U., Yaman, F., & Goldman, R. (2007). Hotride: Hierarchical ordered task replanning in dynamic environments. *Proceedings of the ICAPS-07 Workshop on Planning and Plan Execution for Real-World Systems – Principles and Practices for Planning in Execution..*
- Biundo, S., & Schattenberg, B. (2001). From abstract crisis to concrete relief—a preliminary report on combining state abstraction and HTN planning. *Proc. of the 6th European Conference on Planning* (pp. 157–168).
- Erol, K., Hendler, J., & Nau, D. S. (1994). HTN planning: Complexity and expressivity. *AAAI*.
- Fox, M., Gerevini, A., Long, D., & Serina, I. (2006). Plan stability: Replanning versus plan repair. *ICAPS* (pp. 212–221).
- Georgeff, M. P., & Lansky, A. L. (1987). Reactive reasoning and planning. *AAAI* (pp. 677–682).
- Gerevini, A., Kuter, U., Nau, D. S., Saetti, A., & Waisbrot, N. (2008). Combining domain-independent planning and HTN planning. *ECAI* (pp. 573–577).

- Ghallab, M., Nau, D. S., & Traverso, P. (2004). *Automated planning: Theory and practice*.
- Hammond, K. J. (1990). Explaining and repairing plans that fails. *Artif. Intell.*, *45*, 173–228.
- Hawes, N. (2001). Anytime planning for agent behaviour. *Twelfth Workshop of the UK Planning and Scheduling Special Interest Group* (pp. 157–166).
- Hawes, N. (2002). An anytime planning agent for computer game worlds. *Proceedings of the Workshop on Agents in Computer Games at The 3rd International Conference on Computers and Games*, 1–14.
- Hoffmann, J., & Nebel, B. (2001). The FF planning system. *JAIR*, *14*, 253–302.
- Hoffmann, J., Porteous, J., & Sebastia, L. (2004). Ordered landmarks in planning. *JAIR*, *22*, 215–278.
- Ingrand, F., & Georgeff, M. (1992). An architecture for real-time reasoning and system control. *IEEE Expert*, *6*, 33–44.
- Kambhampati, S., Mali, A., & Srivastava, B. (1998). Hybrid planning for partially hierarchical domains. *AAAI* (pp. 882–888).
- Karpas, E., & Domshlak, C. (2009). Cost-optimal planning with landmarks. *IJCAI* (pp. 1728–1733).
- Mccluskey, T. L. (2000). Object transition sequences: A new form of abstraction for HTN planners. *AIPS* (pp. 216–225).
- Molineaux, M., Klenk, M., & Aha, D. W. (2010). Goal-driven autonomy in a navy strategy simulation. *AAAI*. AAAI Press.
- Nau, D. S., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., & Yaman, F. (2003). SHOP2: An HTN planning system. *JAIR*, *20*, 379–404.
- Nau, D. S., Cao, Y., Lotem, A., & Muñoz-Avila, H. (2001). The SHOP planning system. *AI Mag*.
- Richter, S., & Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res. (JAIR)*, *39*, 127–177.
- Shivashankar, V., Kuter, U., Nau, D. S., & Alford, R. (2012). A hierarchical goal-based formalism and algorithm for single-agent planning. *AAMAS* (pp. 981–988).
- Shivashankar, V., Kuter, U., Nau, D. S., & Alford, R. (2013). The GoDeL planning system: A more perfect union of domain-independent planning and hierarchical planning. *IJCAI*.
- Talamadupula, K., Benton, J., Kambhampati, S., Schermerhorn, P. W., & Scheutz, M. (2010). Planning for human-robot teaming in open worlds. *ACM TIST*, *1*, 14.
- Warfield, I., Hogg, C., Lee-Urban, S., & Muñoz-Avila, H. (2007). Adaptation of hierarchical task network plans. *FLAIRS* (pp. 429–434). AAAI Press.
- Wilkins, D. E. (1984). Domain-independent planning: Representation and plan generation. *Artif. Intell.*, *22*, 269–301.
- Wilson, M. A., Molineaux, M., & Aha, D. W. (2013). Domain-independent heuristics for goal formulation. *FLAIRS Conference*. AAAI Press.

Breadth of Approaches to Goal Reasoning: A Research Survey

Swaroop Vattam

SWAROOP.VATTAM.CTR.IN@NRL.NAVY.MIL

Naval Research Laboratory, NRC Postdoctoral Fellow, Washington, DC 20375

Matthew Klenk

KLENK@PARC.COM

Palo Alto Research Center, Embedded Reasoning Area, Palo Alto, CA 94304

Matthew Molineaux

MATTHEW.MOLINEAUX@KNEXUSRESEARCH.COM

Knexus Research Corporation, Springfield, VA 22150375

David W. Aha

DAVID.AHA@NRL.NAVY.MIL

Naval Research Laboratory, Navy Center for Applied Research in AI, Washington, DC 20375

Abstract

Goal-directed behavior is a hallmark of intelligence. While the majority of artificial intelligence research assumes goals are static and externally provided, many real-world applications involve unanticipated changes in the environment that may require changes to the goals themselves. Goal reasoning, which emphasizes the explicit representation of goals, their automatic formulation and dynamic management, is considered an important aspect of high-level autonomy. Building from these three basic requirements, we describe and apply a framework for surveying research related to goal reasoning that focuses on triggers and methods for goal formulation and goal management. We also summarize current research and highlight potential areas of future work.

1. Introduction

It is generally acknowledged that goal-directed behavior is a hallmark of intelligence (Newell & Simon 1972; Schank & Abelson 1977). Goal-directed behavior has usually been interpreted as *autonomy of actions* - an intelligent agent should be able to reason about actions in an autonomous manner in order to change the state of the world (including itself) as a means to satisfying a *given* goal. On the one hand, this interpretation has provided a clear focus, guiding much AI research from early problem solvers to modern day automated planners. On the other hand, it has also limited the reach and richness of AI systems by ignoring *goals*; it is often assumed that an external user or system is responsible for providing goals that remain static over a problem-solving episode. *Goal reasoning* (e.g., Norman & Long, 1996; Cox, 2007; Hawes, 2011; Klenk, Molineaux, & Aha, 2013; Jaidee, Muñoz-Avila, & Aha, 2013) challenges this interpretation and strives for *autonomy of goals* - in addition to autonomy of actions, an intelligent agent should be aware of its own goals and deliberate upon them. As we start to consider designs for intelligent systems that are more autonomous and use multiple interacting competencies to solve a wider variety of problems in the real world, it becomes increasingly difficult to ignore the issue of goal reasoning.

To illustrate the importance of goal reasoning for intelligent behavior, consider a fishing craft in the Gulf of Mexico. While carrying out a plan to achieve the goal of catching fish, the fishermen receive reports of an explosion on a nearby offshore oil rig. Upon hearing the reports, the fishermen change their goal from “catch fish” to “rescue the rig’s workers”. This goal change results in a far superior outcome, rescued workers, but is outside the scope of the original mission, catching fish.

In this paper, we present a preliminary analysis of research related to goal reasoning in the context of planning and problem-solving. (Due to space limitations, we do not also examine research on the role of goals in human and machine learning (e.g., Leake 1991; Leake & Ram 1995).) We begin by describing

the Goal Reasoning Analysis Framework (GRAF) and use it to focus on the tasks of goal formulation and goal management. Next we survey approaches and techniques for these tasks in terms of this framework. Finally, we briefly discuss current goal reasoning research and highlight potential areas for future work.

2. Goal Reasoning Analysis Framework (GRAF)

Because the notion of *goal reasoning* is polymorphous and often interpreted and applied differently in different research contexts, it is productive to think about a common framework for analyzing and comparing the various techniques and approaches related to goal reasoning. We propose the Goal Reasoning Analysis Framework (GRAF) as a first step in this direction. We develop this framework by first identifying the following three minimum requirements for goal reasoning.

Explicit goals: First, the system should explicitly represent and reason about goals.

Goal formulation: Second, the system should be able to formulate *goals*. Once we require an intelligent system to have explicit goals, we require processes that can generate or identify and select them dynamically. We shall refer to these processes as *goal formulation processes*. Where goals come from is often overlooked in intelligent system, which motivated us to address it in this survey.

Goal management: Third, the system should manage goals and select the ones that should be acted upon. An independent goal formulation process can lead to multiple goals. Therefore we require some form of management system that accepts goals produced by goal formulation processes, selects which goal(s) should be pursued (with reference to any ongoing goal-directed behavior), and triggers the appropriate plan generation mechanism to achieve the selected goal. If the goal formulation processes produce goals dynamically, asynchronously and in parallel, the management system must accept and manage new goals in this manner too. It should not block the operation of the goal formulation processes, as this would interfere with the system's ability to respond to new situations.

This set of requirements is consistent with those proposed by Hawes (2011). There is a fourth core requirement: the system should generate goal-directed behavior from a collection of goals and available resources. However, to simplify, we will ignore this requirement and assume that it is fulfilled by a planner with its execution system.

Our framework, GRAF (Table 1), is obtained by applying the five questions *What*, *Where*, *Why*, *When* and *How* to the three requirements of explicit goals, goal formulation and goal management.

Table 1. A tabular representation of GRAF.

Questions Requirements	What	Where	Why	When	How
Explicit goals	Representation	Source			
Goal formulation			Rationale	Triggers	Methods
Goal management					Methods

What is a goal? This applies to the requirement of explicit goals and refers to the nature and representation of a goal. Explicit goals can be of two kinds. A *declarative* goal is a description of the state of the world which is sought and a *procedural* goal is a set of intended tasks to be solved. Consensus has it that most declarative goals are *attainment* goals. These are states an agent should achieve through plan execution. Declarative goals can also include *maintenance* and *prevention* goals, which refer to states to maintain over time or to prevent from occurring. Given our assumption that the required process which translates goals into behavior is a planning system, the nature of a goal and how it is explicitly represented in a system depends on that planner.

Where does a goal come from? This also applies to the requirement of explicit goals and refers to a goal's source. We identify three sources of goals: external, self, and hybrid. The goals can be supplied to the intelligent system by an *external* source in the environment (e.g., user or peer agents). Goals can also be *self-initiated* by the goal formulation process. While a majority of intelligent system designs assume the

former, goal reasoning architectures focus on the latter. For the sake of completion, we also envision a *hybrid* situation where the goals can be both externally and internally initiated.

Why self-formulate a goal? This is applicable to the requirement of goal formulation. One reason to formulate goals is *rational anomaly response*: to better respond to developing situations that threaten an agent's interests. A second reason is *graceful degradation*: while the current goals may no longer be achievable, intelligent action may be achieved by degrading them (e.g., "submitting a full report" is predicted to fail given the time constraints, but "submitting a draft report" may be achievable). A third reason for goal formulation is better *future performance*: we want intelligent systems to avoid dead-ends with respect to the current goals, and also to avoid states that jeopardize goal achievement in the future. Furthermore, it may be desirable to take actions that increase the system's capabilities for more actions and more potential goals. A fourth reason for goal formulation is *societal norms*: as the scope of the agent's operation becomes broader and its lifespan longer, humans that interact with autonomous agents will have expectations about their behavior. Goals have to be accommodated to meet those expectations.

When is a goal formulated? This also applies to the requirement of goal formulation and refers to triggers for goal formulation. Typically, goal formulation is considered when an anomaly is detected and/or the system is self-motivated to explore its actions in the world.

How are goals formulated? This applies to the requirement of goal formulation and refers to methods for achieving the function of goal formulation.

How are goals managed? This also applies to the requirement of goal management and refers to methods for achieving the function of goal management.

In this survey, we primarily focus on the questions of *When* and *How*. That is, our emphasis is on triggers of goal formulation, methods for goal formulation, and methods for goal management.

3. Triggers for Goal Formulation

Typically, goal formulation can occur when an anomaly is detected and/or the system is self-motivated to explore its actions in the world. In most current implementations a goal is formulated when no active goal exists and the intelligent system is self-motivated to pursue additional goals, or an active goal exists but an anomaly is detected, and pursuing alternate goals is considered advantageous in light of the anomaly. Because a majority of existing approaches are anomaly-driven, we will focus on the latter. A non-exhaustive list of anomalies could include:

- An active plan fails (or is predicted to fail or perform suboptimally) and no contingency plan exists.
- An affordance is perceived (i.e., pursue a better goal that the agent was considering but hadn't been able to pursue).
- An opportunity is detected (i.e., pursue a better goal that the agent wasn't planning to pursue).
- An internal drive of a system requires attention (e.g., a battery's energy level is low and the system has an internal drive to maintain its energy level).

Anomaly triggered goal formulation requires a discussion about how anomalies are detected. Anomaly detection typically relies on various kinds of monitoring processes, including the following:

- **Plan monitoring**: One source of information for detecting anomalies comes from the plan itself. Changes in the environment may prevent the execution of a plan's future actions. In plan monitoring, the agent monitors the plan's execution by assessing whether its remaining actions' preconditions are satisfied in the current state or achievable as an effect of a preceding planned action. If not, a plan fails. Similarly, plans may also fail because an agent's actions do not achieve their intended effects. Action monitoring algorithms ensure that the last action was successfully executed (i.e., the effects of the action are true in the environment).

In addition to monitoring the validity of plans during execution, research has identified methods for monitoring plan optimality during execution. Fitz & McIlraith (2007) define plan optimality and

describe a state space planner that monitors the utility of the current plan with respect to alternatives using a variant of A* search. In this context, the agent should replan when it predicts that the plan will fail or execute sub-optimally.

Plan failure has been the subject of replanning and plan repair in traditional AI planning research from the beginning (Russell & Norvig, 2003). For example, Darmok implements action monitoring in an online case-based planner for a real-time strategy (RTS) game (Ontañón et al., 2010). If an action fails, Darmok extends its current plan with new actions to satisfy the failed action's goal. Also focusing on replanning, HOTRiDE employs action monitoring in simulated noncombatant evacuation operation planning (Ayan et al., 2007). When an action fails, HOTRiDE uses a dependency graph to determine which task decompositions are no longer valid and must be replanned.

When a plan fails or is predicted to fail (or be suboptimal), replanning systems try to generate new plans or repair existing plans using the original goal. In contrast, goal reasoning systems instead reason about their goals and try to formulate new goals. For example, ARTUE (Klenk et al., 2013) finds discrepancies (for discrete states) using a set difference operation between the expected and observed literals. For continuous states, the observed and expected value of each fluent is compared; a discrepancy is considered to occur whenever their values differ by more than 0.1% of the (absolute) observed value. When a discrepancy is detected, its anomaly response mechanism performs anomaly explanation and goal formulation.

- **Periodic monitoring:** Instead of focusing solely on the current plan and its execution, agents may monitor the entire environment to determine if new goals should be considered. In periodic monitoring, the agent considers the current state at set intervals. Periodic monitoring is frequently used in systems that perform real-time response. For example, Burkhard et al. (1998) illustrate how Belief-Desire-Intention (BDI) agents (Rao & Georgeff, 1995) monitor the environment for changes in their beliefs. Their RoboCup soccer agents receive new sensor information every 300ms. PROSOCS uses a sensing, revision, planning, and execution cycle to periodically monitor the environment (Mancarella et al., 2005). At the start of each cycle, new sensor information is received that can inform execution, plan revision, and future planning. A final example is the cognitive architecture ICARUS, which executes periodic monitoring during its recognize-act cycle (Langley & Choi, 2006).
- **Expectation monitoring:** Expectations are driven by experience from problem solving or interacting with an environment. Problem-solving experience can set expectations that can be monitored. A change in expectations can then trigger changes in behavior. For example, Veloso, Pollack and Cox (1998), in their rationale-based plan monitoring architecture, showed that plan rationales often include expectations that result in the adoption of the current plan at the expense of an alternative plan. Such expectations lead to (1) generating monitors that represent environmental features which affect plan rationale, (2) deliberating, whenever a monitor fires, about whether to respond to it, and (3) transforming plans as warranted by modifying goals. Expectation-driven goal-oriented behavior based on problem-solving experience is a hallmark of Schank's approach to intelligent systems (Schank 1982; Schank & Owens 1987), which is highly relevant to goal reasoning.

Agents can also learn a model of how the environment changes through experience from interacting with their environment. Expectation monitoring uses this model to assess the nature and relevance of a discrepancy. In robotic navigation, Bouguerra, Karlsson, and Saffiotti (2008) used semantic knowledge to generate expectations concerning objects that may be encountered during plan execution. For example, when moving into a living room, the robot expects to see objects typical to that location (e.g., a TV, a sofa). From a cognitive science perspective, INTRO uses a rule-based model to generate expectations and detect discrepancies in a Wumpus World environment (Cox, 2007). Kurup et al. (2012) introduce a cognitive model of expectation-driven behavior in ACT-R. It generates future states called *expectations*, matches them to observed behavior, and reacts when a difference exists between them.

Expectation monitoring can be implemented using anomaly recognition techniques. Typically, these approaches can be divided into three groups: (1) signature detection, which matches the current

situation to known deviant patterns, (2) anomaly detection, which compares the current situation to baseline patterns, and (3) hybrid methods, which include both (Patcha & Park, 2007).

- **Domain-specific monitoring:** Monitoring for expectation failures is difficult in environments whose future states are difficult to predict. Therefore, some agents utilize domain-specific monitoring strategies, which periodically test values of specified state variables during plan execution. Many researchers use domain-specific monitoring to directly link unanticipated states to new goals. In a simulated rover domain, MADBot uses motivations to monitor specified values in the environment (e.g., when the battery’s charge falls below 50%, a new goal is created to recharge it) (Coddington, 2006). M-ARTUE (Wilson, Molineaux, & Aha, 2013) similarly represents drives to direct goal formulation. While MADBot uses domain-specific drives, M-ARTUE does not represent motivations using domain knowledge, and is not limited to generating goals for achieving threshold values. Dora the Explorer (Hawes et al., 2011) encodes motivators that formulate goals related to exploring space and determining the function of rooms, similar to M-ARTUE’s exploration motivator. However, Dora’s functions are also domain-specific. Finally, Hawes’s (2011) survey of motivation frameworks defines goal management and goal formulation in terms of goal generators or drives. It relates many systems in terms of these concepts, and proposes a design for future “motive management frameworks”.
- **Object-based monitoring:** In domain-specific monitoring, the monitors specify particular state variables. Object-based monitoring also includes the set of objects in the environment. The detection of new objects may interrupt plans or cause the creation of new goals. Object-based monitoring systems specify which types of new objects to consider as discrepancies. Goldman (2009) describes an HTN planner with universally quantified goals that uses loops and other control structures to plan for sets of entities whose cardinality is unknown at planning time. Similarly, Cox and Veloso (1998) and Veloso, Pollack, and Cox (1998) also discuss and implement universally quantified goals where some objects (and hence goals) are not known. Dora generates a goal to explore each newly detected room (Hanheide et al., 2010). Open world quantified goals extend these approaches to include knowledge about how new objects may be detected (Talamadupula et al., 2010). For example, in an urban search and rescue task, plans must be generated to locate objects that are unknown prior to execution (i.e., the victims). In real-time games like GRUE (Gordon & Logan, 2004), a more typical approach for this kind of monitoring is by authoring game AI using a teleo-reactive program (TRP) (Benson & Nilsson, 1995). TRPs dictate which actions to take in specific world states (e.g., if the agent is running past a weapon it does not have, then it should pick up the weapon).

4. Methods for Goal Formulation

We identify six types of goal formulation methods based on the knowledge they use.

- **State-Based Goal Formulation:** The most straightforward method for generating goals is to pre-specify links between specific state variables and specific goals. Consider a helicopter’s low-fuel indicator light. When it flashes, the agent pilot may generate a goal to refuel. The new goal depends solely on a single variable in the current state (i.e., the low-fuel indicator).

These approaches are typically applied in fully observable environments. For example, game designers who have complete access to the environment can use behavior trees (Champanand, 2007) to control non-player characters; this is done in many modern video games. To increase reusability and make plans interruptible, Cutumisu & Szafron (2009) use multiple behavior trees to control characters interacting in a restaurant. Working with the internal state of the rover, AgentSpeak-MPL (Meneguzzi & Luck, 2007) uses motivations to formulate new goals when the value of particular state variables drops below individual thresholds. ICARUS (Choi 2010) uses a reactive goal management procedure to nominate and prioritize new top-level goals in which <condition, goal> pairs in long-term goal memory are considered for nomination at every reasoning step. This resembles rule-based goal-formulation, as used in ARTUE (Klenk et al. 2013). M-ARTUE (Wilson et

al. 2013) includes a motivation subsystem that formulates goals based on the psychological notion of drives, which constitute a hierarchy of heuristic functions representing both external and internal needs. M-ARTUE differs from ARTUE only in the way goals are formulated; instead of using reactive rules, it uses domain independent heuristics to evaluate potential goals. This approach is similar in spirit to CLARION’s goal formulation mechanism (Sun, 2009), where drives are represented sub-symbolically and they set the level of activation for explicit goals according to the world state. The primary difference between M-ARTUE and CLARION is that the representations of internal needs are domain independent and domain dependent, respectively.

- **Interactive Goal Formulation:** In realistic domains it is often infeasible to provide goal formulation knowledge for every situation. To address this, T-ARTUE (Powell, Molineaux, & Aha, 2011) and EISBot (Weber, Mateas, & Jhala, 2012) learn this knowledge from humans: T-ARTUE learns from criticism and answers to queries, while EISBot learns from human demonstrations. Each provides a domain-independent method for acquiring formulation knowledge, but neither system reasons about internal needs alongside external goals. Although based on the GDA model, GDA-C (Jaidee et al. 2013) differs substantially from ARTUE and M-ARTUE. GDA-C learns its goal selection function using Q-learning. While this increases autonomy, it employs a domain dependent reward function; indirectly, GDA-C’s goal selection strategy is guided by a human.
- **Object-Based Goal Formulation:** While specifying a goal for each state provides an agent designer with considerable control over an agent’s actions, these methods are inflexible and difficult to author. To promote reuse and flexibility, several systems rely on rules or schemas that specify how to formulate goals for a range of possible states. One important problem this solves is the generation of goals in response to the discovery of new objects in the environment that were unknown at planning time. Consider a robot on a search and rescue mission. Prior to plan execution, the number of rooms to search is unknown. Goal formulation allows the robot to formulate an initial plan to detect rooms, and then assert new goals to search the rooms as they are located.

Recently, several researchers have proposed extensions to goal specifications to account for unknown objects. For example, goal generators produce goals when new objects are detected that satisfy a set of conditions (Hanheide et al., 2010). For example, when a new region is detected by a mobile robot, a goal will be generated to identify that region. In addition to generating goals based on newly detected objects, open-world quantified goals provide information about sensing actions for planning (Talamadupula et al., 2010). Each of these approaches extends the goal specification to specify the importance of the newly generated goal.

- **Belief-Based Goal Formulation:** In addition to the observed state, an agent may formulate goals using its beliefs about the current state. Representing knowledge about the environment that is not directly observed, beliefs are generally output by an inference process such as explanation or state elaboration. For example, on observing a lightning strike, an agent might infer a belief that a storm is approaching. This belief could lead to the formulation of a goal to seek shelter.

Recent work has demonstrated the effectiveness of this approach in dynamic environments. After using explanation to update its beliefs, ARTUE uses rules to specify how to formulate goals based on the observed state and the agent’s beliefs (Molineaux et al., 2010). An alternative method for generating beliefs is through state elaboration. Using forward inference rules over the observed state, ICARUS creates a set of beliefs, which are used by reactive goal management to nominate goals from long term memory for use in a simulated driving task (Choi, 2010).

- **Case-Based Goal Formulation:** Case-based goal formulation stores applicable goals in cases. During goal formulation, a case matching the cue is retrieved and the associated goal is reused in the current situation. For example, when a submarine disappears, an agent pilot might remember a previous situation in which searching for the submarine with a helicopter was a useful goal to pursue.

Case-based goal formulation methods differ in their retrieval cues and types of goals generated. In EISBot (Weber, Mateas, & Jhala, 2010), the current state is used as a cue to retrieve a gameplay

trace, which is a state sequence recorded from an expert's game play. EISBot selects a future state from the trace as the current goal. It performed well in StarCraft games against the built-in AI and human players. In another strategy game, CB-gda uses observed discrepancies as a retrieval cue to generate task goals (Muñoz-Avila et al., 2010). Each of these approaches requires minimal knowledge engineering as the retrieved cases may be automatically collected by observing human-provided traces of activities.

- **Explanation-Based Goal Formulation:** While the methods described above require knowledge engineering for each possible goal, an alternative approach focuses on explaining a discrepancy when generating a goal. When the observed discrepancy may prevent the agent from achieving its goals, the agent can generate a new goal by reasoning over its explanation. Consider a helicopter that is losing fuel. An agent pilot might explain this anomaly by inferring a leak in the fuel tank. Using this explanation, a goal could be generated to stop this leak.

Explanation-based methods use the explanation to generate goals. For example, INTRO (Cox, 2007) generates a goal by negating the antecedent of the explanation. In the Wumpus World domain, the discrepancy of the screaming wumpus would yield a goal to negate their hunger. In pervasive diagnosis, goals are generated to collect information based on the current diagnosis of faults in the device (Kuhn et al., 2008). The purpose is to generate plans to achieve production goals while refining its explanation for any faults. By focusing on the syntax of the explanation, these approaches can be easily applied to new domains.

Here we discuss four types of methods for explanation generation in response to an anomaly.

- a. **Propositional Causal Models:** In such models, p causes q implies that p is always followed by q . A causal model is typically encoded as a set of rules, provided by a domain expert, which is used to infer the cause underlying a set of observations. This approach is exemplified by expert systems, such as the MYCIN medical diagnosis system (Shortliffe, 1976). Another deterministic approach uses truth-maintenance systems (Forbus & de Kleer, 1993), where facts are either assumptions provided to the system or consequences computed by a set of rules. For any consequence, it is possible to trace the rules and assumptions that support it.

Intelligent agents have used deterministic causal models to improve their performance in problem-solving domains and simulated environments. For example, using explanation-based learning (DeJong, 1993), CASCADE applied overly-general rules to model human learning in physics problem solving (VanLehn et al., 1992). The goal reasoning agent ARTUE uses an abductive explanation (Josephson & Josephson, 1994) process to assume hidden facts that could cause a discrepancy (Molineaux et al., 2010). Using the environment model, ARTUE selects assumptions that, if true in the prior state, would predict the discrepancy d and the current state.

- b. **Probabilistic Explanation Models:** Unlike deterministic models, probabilistic explanation models explicitly quantify uncertainty. In probabilistic models, p causes q implies that the occurrence of q increases the probability of p . Probabilistic explanation typically uses graphical models, such as Bayesian networks (Pearl, 2000), to determine the likely causes of individual propositions. These models rely on conditional independence between causes and the subjective probabilities can be learned by applying Bayes' rule with experience and a given prior probability. A probabilistic model of a ship explosion would include facts describing the likelihood of an explosion given a gas leak (or a fuel leak) as high, and the prior probability of a gas leak as higher than the prior probability of a torpedo. An agent would reason from this model that both a gas leak and a torpedo are possible explanations, with a gas leak being more likely.

Probabilistic models have been adopted in many AI subfields. In planning under uncertainty, the environment is frequently modeled as a partially observable Markov decision process (Kaelbling, Littman, & Cassandra, 1998). A typical agent using this model will update an internal belief state after each action, which characterizes the probability of the agent being in each possible environment state. The update of this belief state is a form of explanation in which the observations are explained to result from a given state trajectory. From a goal reasoning

perspective, pervasive diagnosis maintains a set of probabilities indicating the likelihood that each potential system fault has occurred based on prior observations (Kuhn et al., 2008).

- c. **Qualitative Explanation Models:** This kind of model provides an alternative approach for describing uncertainty by allowing an agent to reason about changes to continuous quantities without using precise quantitative measurements. Quantity $q1$ is qualitatively proportional to quantity $q2$ if, all things being equal, an increase in $q1$ causes an increase in $q2$ (Forbus, 1984). A qualitative model may explain a ship explosion as the result of a decrease in the engine oil pressure that caused its temperature to rise above its flashpoint.

Qualitative models are useful in domains where numerical models are unknown, inaccurate, or computationally expensive. For example, MAYOR (Fasciano, 1996) explains its expectation failures in managing a simulated city using a qualitative economic model (e.g., high crime decreases housing demand). Using a different qualitative economic model for cities, Hinrichs and Forbus (2007) use qualitative explanations to overcome local maxima in a worker placement task in the Freeciv turn-based strategy game.

- d. **Example-specific Explanation Models:** Due to the difficulty of obtaining complete and correct models from domain experts (Watson, 1997), another approach is to rely on example-specific models, which are easier to elicit from experts. An expert may state that p causes q for a particular situation(s), and this knowledge may be used inductively to infer p' as a cause for q' in a new situation. For example, when faced with a new situation, case-based reasoning (Leake & McSherry, 2005) and analogical reasoning (Falkenhainer et al., 1989) approaches retrieve a similar example and reuse its example-specific explanation. Examples may be labeled with a cause, which can allow supervised learning approaches to infer causes for new instances (Mitchell, 1997). To explain a ship's explosion, an agent may recall another ship that was sunk by a submarine's torpedo and conclude that an enemy submarine is within range of the ship.

The transfer of example-specific models has been used to improve the performance of AI systems. PHINEAS (Falkenhainer, 1988) creates analogies between qualitative behaviors to transfer explanatory models in physical domains. META-AQUA uses explanation patterns (Cox 2007), which are a type of case for explaining expectation violations. Muñoz-Avila and Aha (2004) define a taxonomy of explanation types pertinent to case-based planning for games.

5. Methods for Goal Management

In goal reasoning, agents may need to consider many goals. Given a set of pending goals, goal management selects which goal(s) should be pursued. Goal management can be a continuous ongoing process or triggered by certain events. For example, Veloso, Pollack and Cox (1998) discuss the use of rationale-based planning monitors as triggers for goal change, while Jones et al. (1999) represent goals as operators which are triggered at run-time by rules that match predefined states and sensor readings.

We identify seven types of plan-invariant methods (i.e., approaches that focus solely on pending goals) for goal management. They differ in how they store pending goals and how they select which goals to pursue. Shapiro et al. (2012) provide formal semantics for goal management by dropping or modifying intentions in the context of BDI agents, some of which are applicable to the methods discussed below.

- **Replacement:** Replacement remembers and plans for one goal at a time; if a new goal arises, it immediately replaces the existing goal. These approaches are useful when the set of goals is small, and the agent actively switches between them. For example, in Baltes's (2002) RoboCup soccer agent, the agent switches frequently between offense and defense based on the state of the field.
- **Stack (consider execution history):** In lieu of strict replacement, an agent may use a stack to manage its goals. In this approach, the execution history is taken into account: a newly generated goal is accomplished first, after which the agent pursues the pending goals beginning with the goal that was being pursued when the most recent goal was generated. This is a common approach in cognitive architectures and other agents focused on long term execution. For example, both SOAR (Laird,

2008) and ACT-R (Anderson & Lebiere, 1998) agents have used this strategy to manage their goals in a wide range of domains. The same strategy is employed by the rover agents discussed previously (Coddington, 2006).

- **Rule-based (consider the state):** In rule-based goal management systems, a set of rules is used to change the system's active goals. Each rule is a condition-action pair, where a condition is a statement about an event or a world state that, if true, results in an action to modify (e.g., add, drop, change) the current goals.

Rule-based approaches have been used in reactive-planning agent architectures. While typical BDI agents (Rao & Georgeff, 1995) change their procedural goals as a result of observed events, CANPlan illustrates how observed events can trigger declarative goals that can be reasoned about using planning (Sardina & Padgham, 2010). Extending the semantics, the abstract agent language CAN specifies abstract goal states (pending, waiting, active, and suspended) for three different types of goals (achievement, task, and maintenance) and transitions among them (Harland et al., 2010).

- **Oversubscription planning (consider quantitative goals):** Classical planning focuses on generating plans that achieve a conjunctive set of goals. If no such plan exists, then classical planning fails. Oversubscription planning (Smith, 2004) relaxes this all-or-nothing constraint, and instead focuses on generating plans that achieve the "best" subset of goals (i.e., the plan that gives the maximum trade-off between total achieved goal utility and total incurred action cost). While rule-based approaches do not include quantitative information in the goals themselves or how they are evaluated in a given state, oversubscription planning includes quantitative information in goals. This goal management strategy requires that each goal have an associated utility and each action have an estimated cost.

While this greatly increases the computational complexity of finding an optimal plan, some heuristic approaches have been used for oversubscription planning. For example, heuristic Partial Satisfaction Planning approaches have been shown to generate plans of similar quality to optimal plans (van den Briel et al., 2004). Much of the research in this area has focused on describing the soft constraints that impact action costs and goal utilities. For example, *goal dependencies* (Do et al., 2007) involve constraints among goals (e.g., mutually exclusive goals), further complicating the goal selection process. While most oversubscription approaches do not consider changes to the agent's goals during execution, Han & Barber (2005) introduce a desire-space framework that accounts for goal dependencies. A desire-space is a Markov decision process (Sutton & Barto, 1998) in which each node is a set of achieved goals and the links between them are costs of a macro-operator that achieves the goals in the destination node. This enables the application of decision theory to determine which goals are worth the cost of achieving. Cushing, Benton, and Kambhampati (2008) describe an extension of oversubscription planning that includes replanning, which is cast as a process of reselecting goals. Each top-level goal is associated with rewards and penalties. Rewards are accrued when objectives are achieved and penalties otherwise. Newly arriving goals are modeled as rewards while existing plan commitments are modeled as penalties. The planner continually improves its current plan in an anytime fashion, while monitoring to see if any selected goal is still appropriate. Replanning occurs whenever a situation deviates significantly from the model, causing the selection of a new set of objectives.

- **Spreading activation (consider execution history and state):** While the prior methods use only the time of the goal's formulation to determine the planner's goals, spreading activation methods determine the most relevant goals using the current context of the agent's working memory. In this approach, goals are associated with concepts in a semantic network. The concepts currently in working memory spread activation through the network to individual goals. The goal with the highest activation is selected for consideration by the agent. Motivated by psychological results which indicate that a goal stack insufficiently models human goal processing, some researchers have extended ACT-R's goal management system to select goals based on spreading activation in its declarative memory (Anderson & Douglass, 2001; Altmann & Trafton, 2002). Activation is spread between goals and cues based on associative links, which are formed when they enter working

memory at the same time. This view of goal reasoning emphasizes the role of the environment to supply cues that activate the appropriate goals.

- **Priority queue (domain specific methods that incorporate execution history and state to prioritize goals):** Priority queues generalize spreading activation to allow the ordering of goals along any preference metric (i.e., for each goal a number can be generated by some method using the current beliefs about the environment). The highest scoring goal is the one that should be pursued. Unlike the priority queue data structure, these approaches allow the priority of goals to change after being added to the queue. Therefore, each time an agent selects new goals, it must recompute the existing goals' priorities using its current beliefs about the environment.

This approach has been used in research systems in robotics and game AI, some of which reason with learned priorities. For example, goal intensity allows a simulated rover agent to order its goals using the goals themselves and its beliefs about the environment (Meneguzzi & Luck, 2007). In robotics, the affective goal management method (Scheutz & Schermerhorn, 2009) maintains a recent history of previous successes and failures for each action type and uses these to estimate the expected utility for each goal. Instead of focusing solely on successes and failures, some systems incorporate appraisal theories (Roseman & Smith 2001). For example, the FearNot! framework selects goals related to the strongest emotions (Aylett, Dias, & Paiva, 2006), and SOAR 9 uses appraisals for intrinsically motivated reinforcement learning (Marinier, van Lent, & Jones, 2010). In game AI, GRUE (Gordon & Logan 2004) allows for concurrent goals to be pursued, but does so in a non-compensatory manner (i.e., goals with higher priorities receive preference for resources over all other goals). Similarly, the multi-queue approach to behavior trees (Cutumisu & Szafron, 2009) makes use of qualitative priorities between types of goals, and uses quantitative distinctions within each grouping to select the current goals. Young and Hawes's (2012) work on using evolutionary approaches to determine the priorities of high-level tasks in QUORUM also fits into this approach.

- **Goal transformation:** Goal transformation involves changing the current goals to enable plan generation (Cox & Veloso, 1998). Research on this topic has focused on defining the space of transfer formations and methods for applying them. For example, Cox & Veloso (1998) create a taxonomy of 13 goal transformations and demonstrate how they allow for graceful performance degradation in an air superiority planning task (e.g., in air combat planning, if insufficient resources are available to destroy a bridge, a new goal to damage the bridge can be generated). Goal Morph introduces costs and utilities to goal transformations in a web service composition application (Vukovic & Robinson, 2005). After constraining the space of applicable transformations using the context, Goal Morph applies the transformation that yields the goals with the highest utility.

6. Discussion

Goal formulation determines how an agent responds to an explained discrepancy. Many discrepancies do not require goal change. That is, the agent may continue executing the same plan, or it may generate a new plan for the same goals. While pure replanning approaches, such as FF-Replan, have been effective in many domains, they are susceptible to failures due to execution dead-ends (i.e., states from which the current goals cannot be achieved) (Yoon, Fern, & Givan, 2007). In addition to providing information about the environment, discrepancies may present threats to current or future plans, opportunities or obligations. One reason to formulate goals is to respond to developing situations that threaten the agent's interests, similar to the function of maintenance goals (Dastani, van Riemsdijk, & Meyer, 2006). There are other reasons for formulating goals: (1) graceful degradation, (2) improved future performance, and (3) societal norms. These other reasons have not been investigated sufficiently in goal reasoning research, which provides opportunities for future work.

With its focus on dynamic, uncertain, and open environments, goal reasoning seeks to increase autonomy through a knowledge intensive process. Therefore, goal formulation should not rely solely on the observed state, but also on the agent's beliefs about the environment, as in (Molineaux et al., 2010). In addition, it is difficult to specify all potential goals for an agent. Therefore, an important area of future

research is to reduce the knowledge engineering burden by learning goal formulation methods, as in (Weber et al., 2010).

The need to consider competing goals is a primary motivation for goal reasoning. Simple replacement and stack approaches are well understood, but are too inflexible for more complex tasks. When planning failures occur, autonomous behavior requires a graceful degradation of performance, which may be achieved (at least partially) through existing oversubscription planning and goal transformation approaches. While oversubscription planning endows an agent with a rational method for selecting goals based on utility, it is insufficient when the set of goals is dependent on the agent's continuing observations of the environment (i.e., goals are subject to change at plan execution time). Approaches combining goal transformations with a definition of goal utility captured in a priority queue appear to be promising for handling larger classes of problems.

Future research should also investigate the interaction of goal reasoning components with traditional planning systems. Due to the separation of goal reasoning from planning, it should be possible to integrate a single goal reasoning method with multiple planners. Given that a state, a goal, and an environment model constitute a planning problem, it is worth exploring whether particular goal reasoning methods favor particular planners. In conducting this survey, we observed that the same or similar goal reasoning components may be used with the tasks of HTN planning (Molineaux et al., 2010) and the state-based goals used in many planning approaches (Hanheide et al., 2010). This suggests that goal reasoning is a distinct process worthy of independent investigation.

Evaluating goal reasoning systems is inherently difficult. AI researchers have produced many discussions on agent evaluation strategies (Kaminka & Burghart, 2007). In ablation experiments (e.g. Molineaux et al., 2010), a system's performance is evaluated through a series of trials during which components are removed to measure their contribution to the entire system. While there has been some research on discrepancy detection, explanation, goal formulation, and goal management, evaluating how each component performs within integrated intelligent systems will inform the design of future systems. Alternatively, Cassimatis, Bello, and Langley (2008) suggest comparing intelligent systems via metrics for capabilities, breadth, and parsimony. These metrics can provide evaluations based on a different view. Given the scope of the claims made about goal reasoning agents, a wide array of evaluation methodologies is needed to assess them.

7. Conclusion

Goal reasoning is motivated by four challenges to traditional planning approaches:

- *Nondeterministic partially observable environments*: An agent's observations of the current state are incomplete and the results of its actions are not deterministic. Furthermore, the environment may exhibit unbounded indeterminacy: it is not possible to fully enumerate the future states as a result of an agent's actions.
- *Dynamic environments*: The environment changes as a result of actions executed by the agent, events in the environment, or actions executed by other agents.
- *Incomplete knowledge*: In complex real-world domains, contingencies arise frequently but the knowledge of those contingencies may be limited. Furthermore, during execution, environment changes may present unidentifiable world states.
- *Knowledge engineering*: Capturing complete planning knowledge in complex real-world domains may require capturing wickedly large models for exogenous change, a prohibitively large number of contingencies, and probabilistic effects of actions. These can each present tremendous knowledge engineering challenges.

To enable intelligent action in these types of situations, we propose that agents should formulate and reason about their goals based on environmental changes. Goal reasoning is expected to provide two benefits to intelligent agents. First, goal reasoning should enable agents to better respond to unexpected circumstances. Second, goal reasoning should decrease the knowledge engineering burden in complex real-world domains for a given system by shifting the burden from capturing knowledge for exhaustive

planning to that of coding models used by goals reasoning, which we conjecture to be an inherently simpler task. While there is some initial evidence supporting each claim (Handheide et al., 2010; Molineaux et al., 2010; Muñoz-Avila et al., 2010; Weber et al., 2010), further investigations are required.

As intelligent systems execute for longer periods without human intervention on a wide range of tasks, it becomes increasingly difficult to pre-specify all its possible goals and contingencies. Therefore, the current state-of-the-art relies on human operators to oversee an agent's execution on narrower tasks. But due to the proliferation of robotic and software agents in work, social, and residential environments, utilizing omnipresent human operators is not a viable option. Also, creating many systems for narrower tasks is inefficient and poses a usability challenge as people interact with each new system. Advances in goal reasoning should alleviate these bottlenecks to promote intelligent system development and deployment by increasing an agent's autonomy.

Acknowledgements

Thanks to OSD ASD (R&E) for sponsoring this research and to Michael Cox for his extensive recommendations. Swaroop Vattam performed this work while an NRC post-doctoral researcher located at NRL. 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

- Altmann, E. M., & Trafton, J. G. (2002). Memory for goals: An activation-based model. *Cognitive Science*, 26, 39–83.
- Anderson, J.R., & Douglass, S. (2001). Tower of Hanoi: Evidence for the cost of goal retrieval. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 27, 1331–1346.
- Anderson, J.R., & Lebiere, C. (Eds.) (1998). *The atomic components of thought*. Hillsdale, NJ: Erlbaum.
- Ayan, N.F., Kuter, U., Yaman F., & Goldman R. (2007). Hotride: Hierarchical ordered task replanning in dynamic environments. In F. Ingrand, & K. Rajan (Eds.) *Planning and Plan Execution for Real-World Systems – Principles and Practices for Planning in Execution: Papers from the ICAPS Workshop*. Providence, RI.
- Aylett, R.S, Dias, J & Paiva, A. (2006). An affectively-driven planner for synthetic characters. *Proceedings of Sixteenth International Conference on Automated Planning and Scheduling* (pp. 2-10). Cumbria, UK: AAAI Press.
- Baltes, J. (2002). Strategy selection, goal generation, and role assignment in a robotic soccer team. *Proceedings of the Seventh International Conference on Control, Automation, Robotics and Vision* (pp. 211-214). Singapore: IEEE Press.
- Benson, S. & Nilsson, N. (1995). Reacting, planning and learning in an autonomous agent. In K. Furukawa, D. Michie, & S. Muggleton (Eds.) *Machine Intelligence 14*. Oxford, UK: Clarendon Press, Oxford.
- Bouguerra, A. Karlsson, L., and Saffiotti, A. (2008). Active execution monitoring using planning and semantic knowledge. *Robotics and Autonomous Systems*. 56(11), 942-954.
- van den Briel, M., Sanchez Nigenda, R., Do, M.B., & Kambhampati, S. (2004). Effective approaches for partial satisfaction (over-subscription) planning. *Proceedings of the Nineteenth National Conference on Artificial Intelligence* (pp. 562-569). San Jose, CA: AAAI Press.
- Burkhard, H. D., Hannebauer, M. & Wendler, J. (1998). Belief-desire-intention deliberation in artificial soccer. *AI Magazine*, 19(3), 87-93.
- Cassimatis, N., Bello, P., & Langley, P. (2008). Ability, breadth and parsimony in computational models of higher-order cognition. *Cognitive Science*, 32(8), 1304-1322.

- Champandard, A. (2007). Behavior trees for next-gen game AI. In *Proceedings of the Game Developers Conference*, Lyon, France.
- Choi, D. (2010). *Coordinated execution and goal management in a reactive cognitive architecture*. Doctoral dissertation, Department of Aeronautics and Astronautics, Stanford University, Stanford, CA.
- Coddington, A.M. (2006). Motivations for MADbot: A motivated and goal-driven robot. *Proceedings of the Twenty-Fifth Workshop of the UK Planning and Scheduling Special Interest Group* (pp. 39-46). Nottingham, UK: [<http://www.cs.nott.ac.uk/~rxq/PlanSIG/PlanSIG06.htm>].
- Cox, M.T. (2007). Perpetual self-aware cognitive agents. *AI Magazine*, 28(1), 32-45.
- Cox, M.T., & Veloso, M.M. (1998). Goal transformations in continuous planning. In M. desJardins (Ed.), *Proceedings of the Fall Symposium on Distributed Continual Planning* (pp. 23-30). Menlo Park, CA: AAAI Press.
- Cushing, W., Benton, J., & Kambhampati, S. (2008). *Replanning as a deliberative re-selection of objectives* (Technical Report). Computer Science and Engineering Department, Arizona State University, Tempe, AZ.
- Cutumisu, M., & Szafron, D. (2009). An architecture for game behavior AI: Behavior multi-queues. *Proceedings of the Fifth AAAI Artificial Intelligence and Interactive Digital Entertainment Conference* (pp. 20-27). Stanford, CA: AAAI Press.
- Dastani, M., van Riemsdijk, B., & Meyer, J.-J. (2006). Goal types in agent programming. *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems* (pp. 1285-1287). Hakodate, Japan: ACM Press.
- DeJong, G. (1993). *Investigating explanation-based learning*. Norwell, MA: Kluwer Academic Publishers.
- Do, M.B., Benton, J., van den Briel, M., & Kambhampati, S. (2007). Planning with goal utility dependencies. *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence* (pp. 1872-1878). Hyderabad, India: Professional Book Center.
- Falkenhainer, B. (1988). *Learning from physical analogies* (Technical Report UIUCDCS-R-88-1479). Department of Computer Science, University of Illinois, Urbana-Champaign, IL.
- Falkenhainer, B., Forbus, K.D., & Gentner, D. (1989). The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41(1), 1-63.
- Fasciano, M.J. (1996). *Real-time case-based reasoning in a complex world* (Technical Report TR-96-05). Computer Science Department, the University of Chicago, Illinois.
- Forbus, K. (1984). Qualitative process theory. *Artificial Intelligence*, 24, 85-168.
- Forbus, K. & de Kleer, J. (1993). *Building problem solvers*. Cambridge, MA: MIT Press.
- Fritz, C., and McIlraith, S. A. (2007). Monitoring plan optimality during execution. *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling* (pp. 144-151). Providence, Rhode Island: ACM Press.
- Goldman, R.P. (2009). Partial observability, quantification, and iteration for planning: Work in progress. In C. Fritz, S. McIlraith, S. Srivastava, & S. Zilberstein (Eds.) *Generalized Planning: Macros, Loops, Domain Control: Papers from the ICAPS Workshop*. Thessaloniki, Greece: [<http://www.cs.berkeley.edu/~siddharth/genplan09/>].
- Gordon, E. & Logan, B. (2004). Game over: You have been beaten by a GRUE. In D. Fu & J. Orkin (Eds.) *Challenges in Game AI: Papers of the AAAI'04 Workshop* (Technical Report WS-04-04). San José, CA: AAAI Press.
- Han, D. & Barber, K. (2005). Desire-space analysis and action selection for multiple dynamic goals. In *Computational Logic in Multi-Agent Systems* (pp. 249-264). Berlin: Springer.
- Hanheide, M., Hawes, N., Wyatt, J., Göbelbecker, M., Brenner, M., Sjöö, K., Aydemir, A., Jensfelt, P., Zender, H., and Kruijff, G-J. (2010). A framework for goal generation and management. In D.W. Aha,

- M. Klenk, H. Muñoz-Avila, A., Ram, & D. Shapiro (Eds.) *Goal-directed autonomy: Notes from the AAAI Workshop (W4)*. Atlanta, GA: AAAI Press.
- Harland, J., Thangarajah, J., Morley, D., and Yorke-Smith, N. (2010). Operational behaviour for executing, suspending, and aborting goals in BDI agent systems. In A. Omicini, S. Sardina, & W. Vasconcelos (Eds.) *Declarative Agent Languages and Technologies: Papers from the AAMAS Workshop*. Toronto, CA: [<http://goanna.cs.rmit.edu.au/~ssardina/DALT2010>].
- Hawes, N. (2011). A survey of motivation frameworks for intelligent systems. *Artificial Intelligence*, 175(5-6), 1020-1036.
- Hawes, N., Hanheide, M., Hargreaves, J., Page, B., Zender, H., & Jensfelt, P. (2011). Home alone: Autonomous extension and correction of spatial representations (pp. 3907-3914). In *Proceedings of the IEEE International Conference on Robotics and Automation*. Shanghai, China: IEEE Press.
- Hinrichs, T.R., & Forbus, K.D. (2007). Analogical learning in a turn-based strategy game. *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence* (pp. 853-858). Hyderabad, India: Professional Book Center.
- Jaidee, U., Munoz-Avila, H., & Aha, D.W. (2013). Case-based goal-driven coordination of multiple learning agents. *Proceedings of the Twenty-First International Conference on Case-Based Reasoning* (pp. 164-178). Saratoga Springs, NY: Springer.
- Jones, R.M., Laird, J.E., Nielsen, P.E., Coulter, K.J., Kenny, P., & Koss, F.V. (1999). Automated intelligent pilots for combat flight simulation. *AI Magazine*, 20(1), 27-41.
- Josephson, J.R., & Josephson, S.G. (1994). *Abductive inference*. Cambridge, UK: Cambridge University Press.
- Kaelbling, L.P., Littman, M.L. & Cassandra, A.R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 99-134.
- Kaminka, G.A., & Burghart, C.R. (Eds.) (2007). *Evaluating architectures for intelligence: Papers from the AAAI Workshop* (Technical Report WS-07-04). San Mateo, CA: AAAI Press.
- 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.
- Kuhn, L., Price, B., de Kleer, J., Bo, M., & Zhou, R. (2008). Pervasive diagnosis: The integration of diagnostic goals into production plans. *Proceedings of the Twenty-Third Conference of the Association for the Advancement of Artificial Intelligence* (pp. 1306-1312). Chicago, IL: AAAI Press.
- Kurup, U., Lebiere, C. Stentz, A. & Hebert, M. (2012). Using expectations to drive cognitive behavior. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*. Ontario, Canada: AAAI Press.
- Laird, J.E. (2008). Extending the Soar cognitive architecture. *Proceedings of the First Artificial General Intelligence Conference* (pp. 224-235). Memphis, TN: IOS Press.
- Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. *Proceedings of the Twenty-First National Conference on Artificial Intelligence*. Boston, MA: AAAI Press.
- Leake, D. (1991). Goal-based explanation evaluation. *Cognitive Science*, 15, 509-545.
- Leake, D. B., & Ram, A. (1995). Learning, goals, and learning goals: a perspective on goal-driven learning. *Artificial Intelligence Review*, 9(6), 387-422.
- Leake, D. & McSherry, D. (2005). Introduction to the special issue on explanation in case-based reasoning. *Artificial Intelligence Review*, 24(2), 103-108.
- Mancarella, P., Sadri, F., Terreni, G., & Toni, F. (2005). Planning partially for situated agents. In *Computational Logic in Multi-Agent Systems* (pp. 230-248). Berlin: Springer.
- Marinier, B., van Lent, M., and Jones, R. (2010). Applying appraisal theories to goal directed autonomy. In D.W. Aha, M. Klenk, H. Muñoz-Avila, A., Ram, & D. Shapiro (Eds.) *Goal-directed autonomy: Notes from the AAAI Workshop (W4)*. Atlanta, GA: AAAI Press.

- Meneguzzi, F.R., & Luck, M. (2007). Motivations as an abstraction of meta-level reasoning. *Proceedings of the Fifth International Central and Eastern European Conference on Multi-agent Systems* (pp. 204-214). Leipzig, Germany: Springer.
- Mitchell, T. (1997). *Machine learning*. Columbus, Ohio: McGraw-Hill.
- Molineaux, M., Klenk, M., & Aha, D.W. (2010). Goal-driven autonomy in a Navy strategy simulation. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. Atlanta, GA: AAAI Press.
- Muñoz-Avila, H., & Aha, D.W. (2004). On the role of explanation for hierarchical case-based planning in real-time strategy games. In D. McSherry & P. Cunningham (Eds.), *Explanation in Case-Based Reasoning: Papers from the ECCBR Workshop* (Technical Report 142-04). Madrid, Spain: Universidad Complutense de Madrid, Departamento de Sistemas Informáticos y Programación.
- Muñoz-Avila, H., Jaidee, U., Aha, D.W., & Carter, E. (2010). Goal directed autonomy with case-based reasoning. *Proceedings of the Eighteenth International Conference on Case-Based Reasoning* (pp. 228-241). Alessandria, Italy: Springer.
- Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.
- Norman, T.J., & Long, D. (1996). Alarms: An implementation of motivated agency. In *Intelligent Agents II: Agent Theories, Architectures, and Languages* (pp. 219-234). Berlin: Springer.
- Ontañón, S., Mishra, K., Sugandh, N., & Ram, A. (2010). On-line case-based planning. *Computational Intelligence*, 26(1), 84-119.
- Patcha, A., & Park, J.-M. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51, 3448-3470.
- Pearl, J. (2000). *Causality: Models, reasoning and inference*. Cambridge, UK: Cambridge University Press.
- Powell, J., Molineaux, M., & Aha, D.W. (2011). Active and interactive learning of goal selection knowledge. In *Proceedings of the Twenty-Fourth Florida Artificial Intelligence Research Society Conference*. West Palm Beach, FL: AAAI Press.
- Rao, A., & Georgeff, M. (1995). BDI agents: From theory to practice. *Proceedings of the First International Conference on Multi-agent Systems* (pp. 312-319). Menlo Park, CA: AAAI Press.
- Roseman, I. & Smith, C. A. (2001). Appraisal theory: Overview, assumptions, varieties, controversies. In K. R. Scherer, A. Schorr, & T. Johnstone (Eds.) *Appraisal Processes in Emotion: Theory, Methods, Research* (pp. 3-19). New York and Oxford: Oxford University Press.
- Russell, S., & Norvig, P. (2003). *Artificial intelligence: A modern approach* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.
- Sardina, S., & Padgham, L. (2010). A BDI agent programming language with failure handling, declarative goals, and planning. *Autonomous Agents and Multi-Agent Systems*, 23(1), 18-70.
- Schank, R. C., & Abelson, R. P. (1977). *Scripts, plans, goals and understanding: An inquiry into human knowledge structures*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Schank, R. C. (1982). *Dynamic memory: A theory of reminding and learning in computers and people*. Cambridge, MA: Cambridge University Press.
- Schank, R. C., & Owens, C. C. (1987). Understanding by explaining expectation failures. In R. G. Reilly (Ed.), *Communication Failure in Dialogue and Discourse*. New York: Elsevier Science.
- Scheutz, M. & Schermerhorn, P. (2009). Affective goal and task selection for social robots. In J. Vallverdú & D. Casacuberta (Eds.) *The Handbook of Research on Synthetic Emotions and Sociable Robotics*. Hershey, PA: IGI Publishing.
- Shapiro, S., Sardina, S., Thangarajah, J., Cavedon, L., & Padgham, L. (2012). Revising conflicting intention sets in BDI agents. *Proceedings of the Eleventh International Conference on Autonomous*

- Agents and Multiagent Systems* (pp. 1081-1088). Valencia, Spain: International Foundation for Autonomous Agents and Multiagent Systems.
- Shortliffe, E.H. (1976). *Computer-based medical consultations: MYCIN*. New York: Elsevier/North Holland.
- Smith, D.E. (2004). Choosing objectives in over-subscription planning, *Proceedings of Fourteenth International Conference on Automated Planning and Scheduling* (pp. 393 – 401). Whistler, British Columbia, Canada: AAAI Press.
- Sun, R. (2009). Motivational representations within a computational cognitive architecture. *Cognitive Computation*, 1(1), 91-103.
- Sutton, R.S., & Barto, A.G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.
- Talamadupula, K., Benton, J., Kambhampati, S., Schermerhorn, P., & Scheutz, M. (2010). Planning for human-robot teaming in open worlds. *ACM Transactions on Intelligent Systems and Technology*, 1(2), Article 14.
- VanLehn, K., Jones, R. M., and Chi, M. T. H. (1992). A model of the self-explanation effect. *Journal of the Learning Sciences*, 2(1), 1-59.
- Veloso, M. M., Pollack, M. E., & Cox, M. T. (1998). Rationale-based monitoring for continuous planning in dynamic environments. In R. Simmons, M. Veloso, & S. Smith (Eds.), *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems* (pp. 171-179). Menlo Park, CA: AAAI Press.
- Vukovic, M., & Robinson, P. (2005). GoalMorph: Partial goal satisfaction for flexible service composition. *International Journal of Web Services Practices*, 1(1-2), 40-56.
- Watson, I. (1997). *Applying case-based reasoning: techniques for enterprise systems*. San Francisco, CA: Morgan Kaufmann.
- Weber, B., Mateas, M., & Jhala, A. (2010). Applying goal-driven autonomy to StarCraft, In *Proceedings of Sixth Artificial Intelligence and Interactive Digital Entertainment*. Stanford, CA: AAAI 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.
- Wilson, M., Molineaux, M., & Aha, D.W. (2013). Domain-independent heuristics for goal formulation. In *Proceedings of the Twenty-Sixth Florida Artificial Intelligence Research Society Conference*. St. Pete Beach, FL: AAAI Press.
- Yoon, S., Fern, A., and Givan, B. (2007). FF-replan: A baseline for probabilistic planning. *Proceedings of Seventeenth International Conference on Automated Planning and Scheduling* (pp. 352-359). Providence, Rhode Island: ACM Press.
- Young, J. and Hawes, N. (2012) Evolutionary Learning of Goal Priorities in a Real-Time Strategy Game. In *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Stanford, CA: AAAI Press.

Towards Applying Goal Autonomy for Vehicle Control

Mark Wilson

MARK.WILSON@NRL.NAVY.MIL

Naval Research Laboratory, Navy Center for Applied Research in AI, Washington, DC 20375

Bryan Auslander

BRYAN.AUSLANDER@KNEXUSRESEARCH.COM

Knexus Research Corporation, 9120 Beachway Lane, Springfield, VA 22153

Benjamin Johnson

BLJ39@CORNELL.EDU

Cornell University, School of Mechanical and Aerospace Engineering, Ithaca, NY 14853

Thomas Apker

THOMAS.APKER@EXELISINC.COM

Exelis Inc., 2560 Huntington Ave, Alexandria, VA 22303

James McMahon

JAMES.MCMAHON@NRL.NAVY.MIL

Naval Research Laboratory, Physical Acoustics, Code 7130, Washington, DC 20375

David W. Aha

DAVID.AHA@NRL.NAVY.MIL

Naval Research Laboratory, Navy Center for Applied Research in AI, Washington, DC 20375

Abstract

Unmanned vehicles have been the focus of active research on autonomous motion planning, both deliberative and reactive. However, they are fundamentally limited in their autonomy by an inability to independently reason about, prioritize, and change the goals they pursue. We describe two new projects in which we are incorporating *goal autonomy* on unmanned vehicle platforms. We will apply the Goal-Driven Autonomy (GDA) model to permit our vehicles to reason about their objectives and discuss how properties of the domains affect the application of GDA.

1. Introduction

Unmanned vehicles are often used to explore and act in regions that are dangerous or otherwise undesirable for humans to visit. Many unmanned vehicles are remotely operated: Rather than acting autonomously using onboard control systems, they act directly on control commands from human operators to execute their missions. Remote operation may be desirable in some circumstances (e.g., to maximize control over the safety of an unusually valuable vehicle, such as a Mars rover). However, in many instances we would prefer that unmanned vehicles operate without human input, which would reduce operator load, avoid human error in operating the vehicles, and allow the vehicles to continue pursuing their missions when out of contact with human operators.

Most efforts to provide greater autonomy for unmanned vehicles have focused on a problem we refer to as *motion autonomy*, the primary example of which is to navigate autonomously to a

desired location or to follow a prescribed route (e.g., Tan, Sutton, & Chudley, 2004; Wooden et al., 2010). Although motion autonomy techniques are broadly adaptable and allow robotic vehicles to autonomously accomplish many desired tasks, they do not allow vehicles to dynamically self-select goals to pursue or to re-prioritize their existing goals. This limits motion autonomy to predictable environments, as changes in the environment or previously unobserved facts may require an agent to select new objectives or mission parameters to act correctly.

To address this, we describe two new efforts to enrich unmanned vehicles’ reasoning with *goal autonomy*: the ability to dynamically formulate, prioritize, and assign goals¹. Enabling the vehicle to decide what goal it should accomplish in any given situation, in addition to existing techniques for achieving those goals autonomously, allows the vehicle to act correctly in a broader range of situations without supervision. This is especially valuable in long-duration missions in dynamic environments, where the vehicle is likely to encounter a variety of situations too complex to enumerate *a priori*. For instance, a maritime vehicle on a long mission may encounter a broad range of underwater hazards and opportunities for investigation in unpredictable configurations. To provide the ability to select appropriate goals in a wide range of situations, we will apply *Goal-Driven Autonomy* (GDA), a model for responding to unexpected occurrences by formulating and reprioritizing goals (Molineaux, Klenk, & Aha, 2010a).

In one project, *Autonomous Behavior Technology for Unmanned Underwater Vehicles*, we will apply the GDA model to an underwater vehicle, providing it the decision-making ability necessary to conduct long duration, independent missions with varying objectives. In another project, *Autonomous Systems Integration*, we will apply the GDA model to the task of plume-tracking, in which ground and air vehicles must cooperate to discover the source of an airborne contaminant, while also collecting and transferring power to avoid disruption of activity from loss of battery reserves.

GDA has previously been applied in several simulated test domains inspired by real-world scenarios (Molineaux et al., 2010a) as well as game environments (Weber, Mateas, & Jhala, 2012; Jaidee, Muñoz-Avila, & Aha, 2013). However, the projects presented here, although currently in simulation, will be our first application of GDA on real-world robots or vehicles.

In this paper, we present an overview of GDA, discuss the parameters of the application domains, present initial architectures for both projects, and discuss aspects of applying goal autonomy to situated agents and integrating goal autonomy with motion autonomy in two very different problem domains.

2. An Overview of Goal-Driven Autonomy

Goal-Driven Autonomy (GDA) (Figure 1) is a model for online planning with reasoning about goal formulation and management (Molineaux et al., 2010a). It extends Nau’s (2007) model of online planning, using the Controller to create and pursue new goals when unexpected events occur in complex environments (e.g., stochastic, partially-observable).

The GDA Controller uses the Planner to create a plan to achieve the current goal g from the current state s_0 . The Planner outputs to the Controller a sequence of actions $\langle a_1, \dots, a_n \rangle$ to execute, and a corresponding sequence of expected states $\langle x_1, \dots, x_n \rangle$, where x_n is a goal state for g .

¹ We use “goal autonomy” rather than “goal reasoning” throughout, to distinguish from “motion autonomy.”

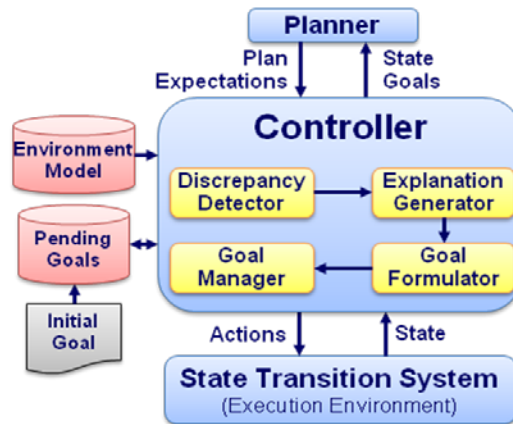


Figure 1: The Goal-Driven Autonomy (GDA) conceptual model.

As the Controller executes the plan in the state transition environment, it performs a four-step cycle to manage goals in response to unexpected events:

1. **Discrepancy detection:** After the Controller executes action a_i , the Discrepancy Detector compares the new observed state s_i to the corresponding expectation x_i . If they differ, a discrepancy has occurred and the GDA model attempts to explain and resolve it.
2. **Discrepancy explanation:** If discrepancies between the new state and the expectation are detected, the Explanation Generator attempts to create an explanation of the discrepancies.
3. **Goal formulation:** The Goal Formulator creates new goals that are appropriate given the explanation.
4. **Goal management:** Finally, the Goal Manager prioritizes and selects among the Pending Goals, including new goals from the Goal Formulator. The selected goal is then given to the Planner to generate a new plan and expectations.

3. Related Work

Related work on autonomy focuses on the areas of goal autonomy, which addresses management of the agent's objectives, and motion autonomy, which addresses tasks such as safely moving a vehicle from one position to another.

Although the projects presented here represent our first efforts to use the GDA model on situated vehicles, GDA has been used in the past to control simulated agents. The ARTUE agent has been used to guide simulated vehicles inspired by Mars rovers (Wilson, Molineaux, & Aha 2013) as well as teams of simulated naval vessels (Molineaux et al., 2010a), but has never been integrated with dynamic motion controllers for real robots. EISBot (Weber et al., 2012), GRL (Jaidee, Muñoz-Avila, & Aha, 2012), and GDA-C (Jaidee et al., 2013) have all been used to successfully control all or part of a player's forces in real-time strategy games, a form of centralized direction for multi-agent systems. We present an architecture for centralized direction, but our system must interface with group control algorithms designed to prevent collisions while allowing several agents to work toward a common goal.

Other types of goal autonomy have also been used to control simulated agents. The ICARUS cognitive architecture (Choi, 2011) has been applied to simulated car-driving domains with a

reactive goal-management component that introduces new goals taken from a long-term goal memory, given general and domain-specific conditions. Coddington’s (2006) MADBot architecture, which can introduce new goals when domain-specific motivational thresholds are exceeded, has been used to control simulated ground-vehicle robots.

Goal autonomy systems have also received attention on robotic platforms. Dora the Explorer (Hawes et al., 2011) is a robot with goal autonomy capabilities, but is limited to goals focused on exploring and categorizing its environment. The SapaReplan planner has been used in the DIARC robotic-control architecture (Schermerhorn et al., 2009) to allow a robotic agent to optionally pursue *soft goals* by taking advantage of ungrounded opportunities in the environment, which it models using simulated objects called *counterfactuals*. However, SapaReplan can pursue such soft goals only temporarily and must not allow them to interfere with its required hard goals. This contrasts with our use of GDA, which permits the indefinite suspension of goals.

An alternative means of encoding multiple objectives onto an autonomous platform is the use of correct-by-construction controller synthesis. Kress-Gazit, Fainekos, and Pappas (2009) present a technique for specifying multiple goals and the conditions required to achieve them as *Linear Temporal Logic* (LTL) formulas. These formulas are used to generate a *Finite-State Automaton* (FSA) controller that is guaranteed to eventually accomplish all specified goals, assuming the required conditions are met and the environment meets defined expectations. However, the computational cost of constructing the FSA grows exponentially with the number of goals and conditions, and requires pre-specification of goals for all situations in which the robot must act. Thus, for large problems this framework requires a goal manager to provide a receding horizon for the controller as in (Wongpiromsarn, Topcu, & Murray, 2009). Livingston, Murray, and Burdick (2012) and Sarid, Xu, and Kress-Gazit (2012) introduce limited forms of goal formulation that respond competently to unexpected states and surprising opportunities, respectively, for synthesized controllers. Using controllers generated from LTL formulas will allow a task planner to plan atomic actions that can be decomposed into multiple LTL-level goals, and ensure that agents that are assigned complex, multi-stage tasks will complete them or provide information about unexpected states in the environment.

Approaches to autonomous control for underwater vehicles can be broadly classed into deliberative and reactive motion planning. Deliberative approaches variously use, among others, genetic algorithms (Alvarez, Caiti, & Onken, 2004), rapidly-exploring random trees (Tan et al., 2004), A* search over discretized environments (Garau, Alvarez, & Oliver, 2005), and gradient-descent optimization over cost functions (Kruiger, Stolkin, Blum, & Briganti, 2007). Plaku and McMahon (2013) address simultaneous task and motion planning for underwater vehicles using LTL task specifications with sampling-based deliberative methods to avoid the complexity of guaranteed correctness. Reactive, or local, planning approaches are particularly useful in regions that are large or not well-mapped. Virtual potential fields (Khatib, 1985) are a common reactive system. Antonelli et al. (2001) alleviate the risk of this approach “trapping” a vehicle in local minima by adding a supervisor module to modify the vehicle’s behavior based on the environment’s geometry. While most of these approaches assume holonomic vehicle models, Apker and Potter (2012) describe a means of encoding a vehicle’s dynamic constraints to improve performance and reliability. However, unlike our work, these systems address motion autonomy rather than the problem of goal autonomy.

The IvP Helm (Benjamin et al., 2010) provides a reactive UUV controller based on multi-objective optimization rather than potential fields, and exhibits limited goal autonomy by

changing *modes* based on the state. However, it does not reason about goals the vehicle should accomplish in the environment.

Research on autonomy for individual air and ground vehicles is more mature than for underwater vehicles, and recent work has focused on guiding groups of vehicles to accomplish given tasks. Several authors have explored combining potential fields with FSAs to allow their systems to react to state changes by changing agent objectives. Mather and Hsieh (2012) apply this approach to robots engaged in surveillance tasks. Worcester, Rogoff, and Hsieh (2011) develop a finite state representation of a construction task, and use a centralized system to partition its components among a team of robots. Martinson and Apker (2012) describe a physics-inspired FSA that operates in the robots' behavior space, changing the way they generate motion commands from potential fields depending on their proximity to a target and navigation quality. In contrast to this body of work, we instead focus on goal autonomy, and discuss applications of these methods to teams of unmanned vehicles in Section 5.

4. Application Domains

4.1 Long-Duration Underwater Autonomy

Autonomously-controlled unmanned underwater vehicles (UUVs) have been used for underwater exploration (Antonelli et al., 2001), observation and inspection of underwater structures (Antonelli et al., 2001), scientific observation (Binney, Krause, & Sukhatme, 2010), and mine countermeasures (LePage & Schmidt, 2002). However, these missions typically are of short duration (at most eight to sixteen hours) and operate over a small region.

In our first project we will apply GDA to autonomously direct a UUV on unsupervised long-duration missions. These missions could eventually last weeks or months. Long-term missions may require the vehicle to pursue different goals at different times, such as goals related to transiting to a region, avoiding other vessels, surveying oceanic geography, detecting mines and other manufactured obstacles, and taking oceanographic measurements. The ocean environment is highly unpredictable, and a UUV on a long-duration mission must be able to react intelligently to unexpected events and objects. Throughout the course of a mission a UUV may need to change its objectives, or even abort its mission, due to unforeseen environmental hazards, underwater barriers, encounters with other vehicles, or failures of onboard systems.

These missions may motivate goal autonomy. Although motion autonomy could correctly guide the vehicle on any task selected in response to such anomalies, goal autonomy provides the ability to select goals generally and dynamically without reference to a human operator. Because an at-sea UUV has very limited communication with human operators, the vehicle must make goal decisions autonomously.

For example, consider a UUV taking oceanographic measurements (e.g., water salinity) over a region, when a surface vessel enters its area and stops. If the measurements are being taken near the ocean surface, attempting to take them at or near the new vessel's position may risk collision. While motion autonomy systems can likely minimize risk and maximize data quality, they cannot consider the broader implications of the vessel's arrival and how best to respond. If it is a friendly vessel, it may be appropriate for the UUV to surface, broadcast that scientific measurements are being taken, and request that the vessel vacate the area. If the UUV is a military vehicle operating in contested or unfriendly waters, and the vessel is not friendly, it may

be appropriate to halt and silence the UUV to avoid detection. If in open waters, the UUV may be correct to abort the data-collection mission and notify its operator of the surface vessel's approach. Goal-driven autonomy is a general model for generating appropriate responses to unplanned situations, and is therefore well-suited to the control of unmanned vehicles at sea.

Key challenges in this domain include:

- **Unpredictable environments:** Existing deliberative motion autonomy techniques for UUVs require advance knowledge of the environment in which the path will be planned while existing reactive motion autonomy techniques respond to unknown environments unpredictably. Both present challenges in long duration missions where a UUV may venture into waters that are not well-charted or for which there are no reliable data on currents. Furthermore, deliberative techniques have difficulty planning for dynamic obstacles whose motion may not be well understood, while reactive techniques can complicate the task of detecting discrepancies that occur during motion plan execution.
- **Computational constraints:** The CPUs that our agent will use to control the UUV are not powerful, and necessitate an emphasis on computationally efficient solutions.
- **Uncertain environment state:** The lack of many sensors often found on ground vehicles and other robots (e.g., for localization, visual inspection, range-finding), combined with noisy readings from sensors that are available, presents unique challenges.

4.2 Airborne Contaminant Detection

Unmanned air vehicles (UAVs) are used in remote sensing, scientific research, and search-and-rescue applications. Unmanned ground vehicles (UGVs) can be used to explore and act in situations that are dangerous to humans, such as in contaminated waste cleanup and explosive ordnance disposal missions, and to provide logistics support, such as carrying equipment.

In our second project, we will apply GDA to direct a team of UAVs equipped with aerosol sensors and UGVs with support equipment that includes landing pads, UAV rechargers, and solar panels. We know that the environment is bounded and that autonomous navigation is possible, but make no assumptions about initial plume locations, availability of traversable paths for the UGVs, or locations of brightly lit areas for solar recharging. This problem combines motion planning, task scheduling, and resource allocation in an unknown environment.

Conventional motion autonomy methods require a complete output specification for each vehicle given possible sensor inputs. In our scenario this is computationally intractable given the potential number of vehicles, sensors, and actions. Using GDA to make goal and task level decisions permits the synthesis of controllers that encode a limited number of relevant responses given the current goal, thus making the motion autonomy problem tractable.

Unlike the UUV domain, in the UAV/UGV domain we must control several vehicles to cooperatively achieve goals. However, if goal decisions are decentralized among vehicles, each vehicle would need to model all its teammates' possible goals and plans, or risk interference with teammates pursuing different goals. By centralizing GDA to coordinate the vehicles, we can guarantee all vehicles will pursue the same goal at any given time, and that the goal will be achieved based on guarantees offered by lower-layer controllers. This leads to the key challenges for GDA implementation in this domain:

- **Motion abstraction:** The GDA Controller must direct multiple autonomous vehicles to accomplish tasks requiring solutions to continuous-motion problems. Multiple vehicles must

autonomously carry out these tasks without interfering with each other, a problem too computationally intensive to solve at the GDA level. Hence, we require abstract representations of the continuous motion problems that are suitable for computation at the goal autonomy layer, while supporting goal decisions that can be used as a basis for planning and controller synthesis for individual vehicles.

- **Individual discrepancies:** Although vehicles are directed in coordinating teams to achieve goals, discrepancies can still occur on the individual level (e.g., one vehicle's battery may run low due to malfunction). Our solution must manage goals and vehicle task assignments to permit responses to each vehicle's discrepancies, while using abstracted representations of goals as team activities that can be continued in spite of individual discrepancies.

5. Applying Goal-Driven Autonomy

GDA is well-equipped for its usual role in providing goal autonomy in task-planning domains. However, applying GDA in robotic vehicle domains requires appropriate abstractions from motion guidance to task-level actions. In this section we describe different approaches to this multi-layered abstraction in our underwater autonomy and airborne contaminant domains.

Factors such as environment predictability and the need for cooperation affect how GDA should be implemented and applied in a given domain. For single vehicles operating in dynamic or poorly specified environments (e.g., Mars rovers or singleton UUVs), each sense-act cycle represents an opportunity to reevaluate and adjust the agent's goals with respect to the most recent state. Loosely coordinated teams, particularly those working closely with humans, benefit from a concurrent control and planning architecture in which the system's goals are drawn from a limited set of easily interrupted goals whose supporting tasks can be learned offline (Talamadupula et al., 2011). In contrast, tightly coordinated teams require team members to behave in a predictable manner so that their teammates can respond appropriately. In this context, each individual's behaviors for achieving goals should be guaranteed; hence, such systems can benefit from correct-by-construction controller synthesis (per team member). In this case, goal interruption must occur safely, which requires extra time to make sure that each team member can safely interrupt its current goal and start another. This delay decreases the reactivity of the goal autonomy layer.

The granularity of atomic actions available to the GDA Controller can vary from simple (e.g., "go to x, y, z ") to complex (e.g., "supply landing sites for the UAVs and recharge their batteries"). This granularity depends on properties of the underlying control layers, which in turn depend on environment predictability and team coordination required. We present examples at opposite extremes of these domain properties, and note how these impact the granularity of the goals used.

5.1 Autonomous Behavior Technology for UUVs

While there is a large body of work on UUV motion autonomy, current approaches do not have the ability to reason about goals. In our planned approach, GDA will allow a UUV to respond with appropriate actions to unexpected situations whenever the vehicle's current set of goals is no longer satisfactory.

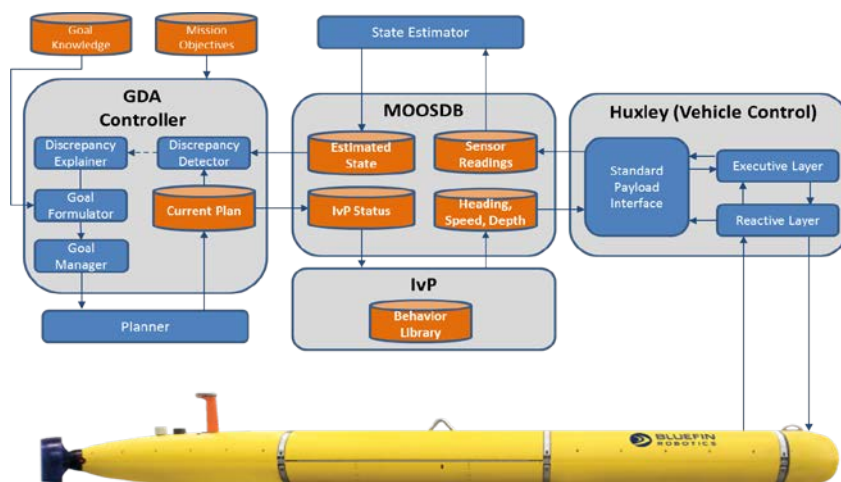


Figure 2: The GDA agent architecture for controlling a UUV with MOOS-IvP.

5.1.1 Integration with Motion Autonomy Systems

Deliberative motion autonomy techniques for UUVs require advance knowledge of the environment in which the path will be planned, any currents that must be taken into account, and the future motion of dynamic obstacles. In a long-duration mission, a UUV may venture into waters that are not well-charted or for which there are no reliable data on currents. Dynamic obstacles may include other vessels that are engaged in unpredictable maneuvering, or whose motion is not well-understood at the time of planning because sensor data are not conclusive. Without such useful constraints on the guidance problem, deliberative path planning alone may not be appropriate for a UUV on a long-duration mission.

We will apply the MOOS-IvP autonomy architecture (Benjamin et al., 2010) to provide suitable path guidance. MOOS is a message-passing middleware system with a centralized publish-subscribe model. IvP Helm is a behavior-based MOOS application that chooses a desired heading, speed, and depth for the vehicle in a reactive manner to generate collision-free trajectories. Unlike potential field methods, IvP Helm uses an interval programming technique that optimizes over an arbitrary number of objective functions to generate desired heading, speed, and depth values and activate or deactivate sensor payloads.

We developed a new GDA agent architecture based on ARTUE (Molineaux et al., 2010a), are using it to control a UUV in simulation, and will later apply it to control our UUV. The GDA Controller will direct the vehicle to perform various tasks (e.g., sensing, navigation) while preserving its ability to navigate partially unknown or poorly mapped environments. It will accomplish this by activating and deactivating specified IvP Helm behaviors and altering the parameters of active behaviors. While IvP Helm can make these decisions independently, it is a reactive mechanism and cannot deliberate about what goal the vehicle should pursue, which is the focus of GDA. Figure 2 depicts our agent architecture, where GDA will direct goal autonomy, IvP Helm will provide motion guidance, and Bluefin’s Huxley control architecture will execute low-level control.

The UUV domain has few constraints on the environment, which distinguishes it from the contaminant detection domain, where we will use a constrained environment and abstractions to

provide guarantees of motion controller correctness. The ocean is large, sparsely mapped, and dynamic. Therefore, it is not possible to provide guaranteed-correct motion control (Kress-Gazit et al., 2009). Furthermore, unlike the controllers we use on the UAVs, IvP Helm cannot independently recognize that a navigational failure has taken place.

To allow IvP Helm independent control over motion while preserving the GDA Controller's ability to recognize anomalous situations, we are developing an abstraction that replaces expected states in our Discrepancy Detector with semantically richer expectations. This will allow our agent to ignore certain values or expect values in some range between actions, and to resolve intervals between actions by checking conditions during execution rather than computing the expected duration of a process from a domain model. This would allow the goal reasoner to, for example, expect position values to fall within some range until a motion is completed or some other unexpected event (e.g., a barrier) triggers a discrepancy. Using this technique affords better separation of responsibilities between the goal autonomy layer and the motion autonomy layer. It also offers improved performance by eliminating discrepancies caused by allowing the motion autonomy layer to independently execute motion tasks and by obviating precise modeling of vehicle motion and other lower-level processes during planning.

5.1.2 Modeling Uncertainty

Our current model of discrepancies assumes that observations are not noisy. This assumption does not hold in real-world environments, where sensors are noisy and sometimes faulty, which can cause uncertainty in observations and the estimated state. The discrepancy model also assumes that observations occur at precise times relative to actions taken (i.e., either immediately after one action or immediately after the amount of time necessary for an event to occur as predicted by the domain model). This second assumption is also unrealistic: the sampling rate of the sensors may not correspond precisely to the timeline of the expected states, and the transmission and reception of the data by asynchronous processes that lack maximum-update-time guarantees may interfere with the timely delivery of the state observation. Hence, when detecting discrepancies, observations may not correspond exactly to expected states as generated by a planner, though they may be closely correlated.

To address these issues, we intend to improve our new expectations model by introducing a probabilistic model that assigns a distribution to each value or range in an expectation. This will allow for computing a likelihood value for each observed state, which can be used to detect discrepancies (i.e., under some conditions, a low likelihood for an observation may indicate a high probability that it is anomalous).

5.2 Autonomous Systems Integration

In this project, we will apply GDA to the problem of controlling a team of UAVs and UGVs to locate the source for a plume of airborne particles. While the maneuvering of sensors for plume source location has been previously studied (Spears, Thayer, & Zarzhitsky, 2009), little work has been done on providing autonomous support for such a team. We will apply goal autonomy to simultaneously coordinate search operations and logistics support, including safe landing zones and recharging stations.

5.2.1 *Integration with Motion Autonomy Systems*

We use a hierarchical approach for implementing team motion autonomy that involves three decision layers. The highest layer uses GDA to select mission goals. The GDA Controller uses a SHOP2_{PDDL+} planner (Molineaux, Klenk, & Aha, 2010b) to produce a sequence of actions and associated safety conditions. The bounded nature of the UAVs' flight envelope guarantees that this planner will generate achievable plans, which are executed by an FSA on each vehicle to allow local trajectory planning, execution, and discrepancy detection. To increase robustness to agent failure and reduce the size of the FSA, we are employing the Physicomimetics swarm control algorithm (Apker & Potter, 2012) to reactively generate vehicle trajectories.

To bridge between high-level goals and low-level tasks in the GDA Controller, we will use LTL as a translation mechanism between decision layers. LTL controller synthesis has been used to automatically produce verifiable FSA controllers to accomplish complex tasks on autonomous robots (Kress-Gazit et al., 2009). In this approach, the GDA Controller will generate a set of complex actions and constraints for each agent's motion autonomy system, and the LTL Controller will generate simpler actions (e.g., "go to (x, y, z) ") for the agent's guidance system. This contrasts with previous approaches, which required LTL tasks to be pre-specified, or required pre-specified templates that can assign newly-discovered areas of interest as new destination goals (Sarid et al., 2012).

For a group of collaborating robots, the LTL controller synthesis problem quickly becomes infeasible. We are addressing this by using goal autonomy to alleviate this state-space explosion problem by supplementing the mission goal with smaller, short term goals with mission constraints. That is, we will use it to decompose the complete task specification into smaller, local specifications for individual or small teams of UxVs, thus limiting the goals that are within the scope of the task. This could reduce an infeasible task into smaller, more computationally efficient tasks for the LTL synthesizer.

The FSA that LTL synthesis creates can be used by the GDA Controller to detect unexpected events during operation. Discrepancies can be detected by comparing the FSA's expected state with the agent's observed state.

Finally, the FSA is guaranteed to satisfy its underlying task specification, which provides a valuable check to ensure that the goals selected by the GDA Controller do not conflict with each other or with the mission's safety constraints. This guarantee on the FSA's behavior assumes that the environment acts as expected, and that the robot's sensors and actuators operate without error. We can relax these assumptions by using Johnson and Kress-Gazit's (2012; 2013) method for analyzing the behavior of an LTL-synthesized controller, which tolerates errors in the sensing and actuation of the robot. After creating a probabilistic model of the robot's interaction with the environment, their method uses model checking to find the probability that the robot exhibits a particular behavior (defined by an LTL formula). This will be used by the Discrepancy Explainer to diagnose the perceived discrepancy.

5.2.2 *Controlling a Team of Vehicles*

In the contaminant detection domain, several UAVs and UGVs must coordinate to locate the contaminant's source. While the vehicles are expected to execute maneuvers independently, their efforts should be centrally coordinated to complete the mission quickly and with minimal mutual interference. Therefore, the GDA Controller must coordinate the vehicles' efforts. Our strategy

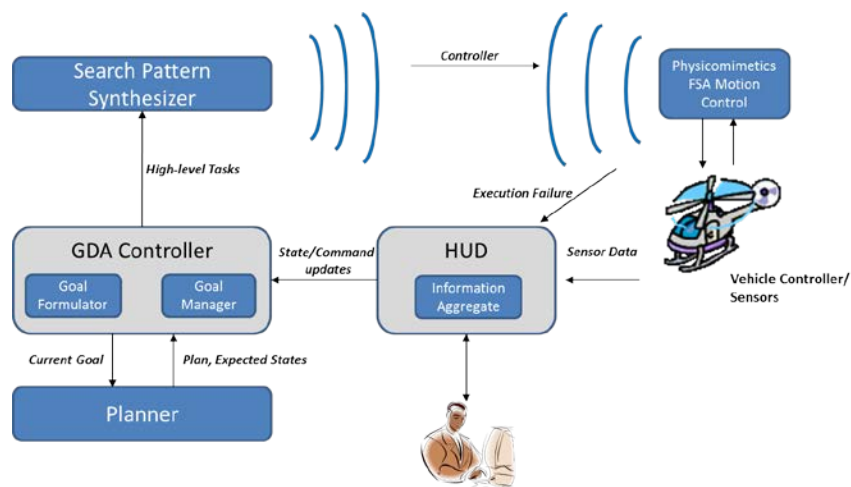


Figure 3. The GDA architecture for controlling the UAVs.

for solving this problem assigns the UAVs to follow plumes of contaminants to their source and uses UGVs in a support role.

Figure 3 depicts our prototype architecture, which uses the MASON simulation toolkit (Luke, 2005) to simulate vehicle motion and chemical-plume dynamics. The mission goal is to detect possible plume locations. Initially, the planner assigns all UAVs to small groups and directs each group to investigate a possible plume, or remain in reserve. Each group's plume assignment is passed to a separate intermediate level planner, which creates a lawnmower search pattern to follow. (In our future work, we will replace this with LTL-synthesized controllers.) All of the UAVs use Physicomimetics motion planning to jointly investigate each location in the pattern for evidence of a plume.

The discrepancies that we currently model concern unexpectedly low UAV battery states, suspected plume locations, and task completion signals from groups or individual agents. When a discrepancy is encountered, the GDA Controller reassesses its goals and forms new plans. For instance, if an agent's battery charge becomes crucially low, then the GDA Controller will assign a new goal for the agent to recharge its battery, and will change the group's composition by tasking other vehicles to continue searching for plumes. Later, we will model anomalies such as opportunities to deploy solar panels, which may interfere with UAV transport or landing operations, and winds that interfere with UAV flight and aerosol sensor performance.

We will integrate UGVs in this domain. They will transport UAVs to contaminated regions, harvest energy for battery power, and recharge the UAVs' batteries during operations. Launch, landing, search patterns, and battery charging involve precise, coordinated motion control that can be achieved only in favorable conditions. This requires guarantees on the agents' behavior throughout a maneuver, which is an ideal application of LTL control. The GDA Controller complements this by managing higher level goals, scheduling these operations, and determining their locations.

6. Discussion

We based our implementation decisions on the degree of predictability in each environment and the need for agent cooperation. These vary substantially between our two projects.

6.1 Environment predictability

Ocean currents, ship traffic, and underwater features are generally unknown in advance of deployment. As a result, any motion autonomy algorithm that makes specific guarantees is bound to fail in the UUV domain. There is little benefit in the UUV domain to synthesizing a guidance system more complex than a MOOS-IvP behavior, as the GDA Controller may frequently select new goals when more accurate states become available.

In contrast, the plume detection environment can be observed and accurately predicted over short time scales, allowing synthesis of controllers that are guaranteed to perform well in those conditions. At longer time scales, much of the environment is static or repetitive (e.g., areas of sun vs. shade), allowing a planner to schedule complex tasks with a high probability of success. The GDA Controller will detect fewer discrepancies in this environment and will be more focused on managing the team's resources.

The plume detection mission benefits from abstractions of the environment and agent behavior that are possible in predictable environments. These abstractions allow goal autonomy to largely ignore issues of motion autonomy.

6.2 Need for cooperation

The UUV domain involves a single vehicle that has little or no interaction with other agents, and reasons about only a few constraints (e.g., to avoid goal oscillations). This frees GDA to make highly independent decisions about the vehicle's activity by selecting the best available goal for its current state. This level of independence permits a direct connection between GDA and the guidance systems, with no need for a controller-synthesis step.

Cooperation is the defining feature of the plume detection domain. As a result, no individual agent can be allowed to replan its actions in a way that interferes with its peers. This forces goal autonomy to a central node whose role is restricted to issuing clearly defined instructions that will be used to synthesize low-level controllers (FSAs) for each team member. These extra layers of abstraction will allow goal autonomy to coordinate the team's behaviors to ensure that no hardware will be lost unexpectedly, although it will introduce delays between selecting and implementing new goals.

Architecture decisions involving cooperative agents need to balance closeness of cooperation with the agents' ability to respond to new information quickly. A continuum of cooperation options exists, varying from agents that cluster closely (to form coherent arrays) to fully independent agents. With less cooperation, fewer abstractions are required between GDA and low-level control, while close cooperation requires more abstractions and, implicitly, a more predictable environment to allow those abstractions.

7. Conclusion

In this paper we described initial architectures and proposed models for projects in which goal autonomy (i.e., the GDA model) will be used to control unmanned vehicles. We identified different modeling requirements in the application of GDA to situated agents depending on certain domain properties, which affect the capabilities afforded to GDA by lower level layers in the autonomy architecture. In particular, the granularity of actions that are atomic for the GDA Controller varies widely according to the computational complexity of motion and the guarantees provided by lower level systems.

In the contaminant detection domain, the motion of a team of vehicles toward a location where sensing will take place must be carefully coordinated so as to avoid collisions or other interference. Solving this guidance problem (i.e., finding waypoints that each individual should follow) in the goal autonomy layer would be computationally infeasible. However, specialized guidance techniques combined with domain-specific controllers, can reduce computational complexity. Hence, in the contaminant detection domain, the abstraction level of the GDA Controller's actions must be at least as high as instructions for each team of vehicles to follow.

In contrast, we do not require coordination of many individual agents in the UUV domain. Therefore, the GDA Controller's plans can be more concrete (e.g., specify a sequence of waypoints for the vehicle to follow). Furthermore, the unpredictability of the ocean environment requires that GDA detect discrepancies without the aid of guarantees as provided by the LTL controllers in the contaminant detection domain. To support GDA discrepancy detection, behaviors implemented by lower level systems should be as predictable as possible. This reinforces our belief that the GDA Controller's actions should be simpler in this domain.

Thus, when designing a goal autonomy robotic controller, the required granularity of the actions will be dictated by the available reactive and abstraction layers. Highly granular actions improve predictability but impose a higher computational burden on the GDA Controller. More abstract actions reduce this computational burden, but generally require more time to safely coordinate goal changes, reducing system reactivity. They also require more predictable environments for low level controllers.

As we progress to more complex tasks and control of non-simulated vehicles, we will develop and implement new models for GDA that address the issues of real-world situated agents. We have argued these models are needed (e.g., probabilistic expectation models for discrepancy detection). We expect to create compelling demonstrations of goal autonomy for controlling unmanned robotic vehicles after these models are in place.

Acknowledgements

Thanks to NRL for supporting this research. Thanks also to NRL's Brian Houston and Ben Dzikowicz for their insightful comments and corrections. 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 the DoD.

References

- Alvarez, A., Caiti, A., & Onken, R. (2004). Evolutionary path planning for autonomous underwater vehicles in a variable ocean. *IEEE Journal of Oceanic Engineering*, 29(2), 418-429.
- Antonelli, G., Chiaverini, S., Finotello, R., & Schiavon, R. (2001). Real-time path planning and obstacle avoidance for RAIS: an autonomous underwater vehicle. *IEEE Journal of Oceanic Engineering*, 26(2), 216-227.
- Apker, T., & Potter, M. (2012). Physicomimetic motion control of physically constrained agents. In W. Spears & D. Spears (Eds.), *Physicomimetics: Physics-based swarm intelligence*. New York: Springer.
- Benjamin, M., Schmidt, H., Newman, P., & Leonard, J. (2010). Nested autonomy for unmanned marine vehicles with MOOS-IvP. *Journal of Field Robotics*, 27(6), 834-875.
- Binney, J., Krause, A., & Sukhatme, G. S. (2010). Informative path planning for an autonomous underwater vehicle. *Proceedings of the 2010 IEEE International Conference on Robotics and Automation* (pp. 4791-4796). Anchorage, AK: IEEE Press.
- Choi, D. (2011). Reactive goal management in a cognitive architecture. *Cognitive Systems Research*, 12(3), 293-308.
- Coddington, A. (2006). Motivations for MADBot: A motivated and goal directed robot. *Proceedings of the Twenty-Fifth Workshop of the UK Planning and Scheduling Special Interest Group* (pp. 39-46). Nottingham, UK: University of Nottingham.
- Garau, B., Alvarez, A., & Oliver, G. (2005). Path planning of autonomous underwater vehicles in current fields with complex spatial variability: An A* approach. *Proceedings of the International Conference on Robotics and Automation* (pp. 194-198). Barcelona, Spain: IEEE Press.
- Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated planning: Theory and practice*. San Francisco, CA: Morgan Kaufmann.
- Hawes, N., Hanheide, M., Hargreaves, J., Page, B., Zender, H., & Jensfelt, P. (2011). Home alone: Autonomous extension and correction of spatial representations. *Proceedings of the International Conference on Robotics and Automation* (pp. 3907-3914). Shanghai, China: IEEE Press.
- Ho, C., Mora, A., & Saripalli, S. (2012). An evaluation of sampling path strategies for an autonomous underwater vehicle. *Proceedings of the International Conference on Robotics and Automation* (pp. 5328-5333). St. Paul, MN: IEEE Press.
- Jaidee, U., Muñoz-Avila, H., & Aha, D.W. (2013). Case-based goal-driven coordination of multiple learning agents. *Proceedings of the Twenty-first International Conference on Case-Based Reasoning* (pp. 164-178). Saratoga Springs, NY: Springer.
- Johnson, B., & Kress-Gazit, H. (2012). Probabilistic guarantees for high-level robot behavior in the presence of sensor error. *Autonomous Robots*, 33(3), 309-321.
- Johnson, B., & Kress-Gazit, H. (2013). Analyzing and revising high-level robot behaviors under actuator uncertainty. To appear in *Proceedings of the International Conference on Intelligent Robots and Systems*. Tokyo, Japan: IEEE Press.

- Khatib, O. (1985). Real-time obstacle avoidance for manipulators and mobile robots. *Proceedings of the International Conference on Robotics and Automation* (pp. 500-505). St. Louis, MO: IEEE Press.
- Kress-Gazit, H., Fainekos, G.E., & Pappas, G.J. (2009). Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6), 1370-1381.
- LePage, K.D., & Schmidt, H. (2002). Bistatic synthetic aperture imaging of proud and buried targets from an AUV. *Journal of Ocean Engineering*, 27(3), 471-483.
- Livingston, S.C., Murray, R.M., & Burdick, J.W. (2012). Backtracking temporal logic synthesis for uncertain environments. *Proceedings of the International Conference on Robotics and Automation* (pp. 5163-5170). St Paul, MN: IEEE Press.
- Mather, T., & Hsieh, M. (2012). Ensemble synthesis of distributed control and communication strategies. *International Conference on Robotics and Automation* (pp. 4248 –4253). St. Paul, MN: IEEE Press.
- Molineaux, M., Klenk, M., & Aha, D.W. (2010a). Goal-driven autonomy in a Navy strategy simulator. 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.
- Nau, D. (2007). Current trends in automated planning. *AI Magazine*, 28(4), 43–58.
- Nau, D., Cao, Y., Lotem, A., & Muñoz-Avila, H. (1999). SHOP: Simple hierarchical ordered planner. *Proceedings of the Sixteenth International Joint Conferences on Artificial Intelligence* (pp. 968-973). Stockholm, Sweden: Morgan Kaufmann.
- Plaku, E., & McMahon, J. (2013). Motion planning and decision making for underwater vehicles operating in constrained environments in the littoral. To appear in *Towards Autonomous Robotic Systems*. Oxford, UK: Springer.
- Powell, J., Molineaux, M., & Aha, D.W. (2011). Active and interactive learning for goal selection knowledge. In *Proceedings of the Twenty-Fourth Florida Artificial Intelligence Research Society Conference*. West Palm Beach, FL: AAAI Press.
- Sarid, S., Xu, B., & Kress-Gazit, H. (2012). Guaranteeing high-level behaviors while exploring partially known maps. In *Proceedings of Robotics: Science and Systems*. Sydney, Australia: MIT Press.
- Schermerhorn, P., Benton, J., Scheutz, M., Talamadupula, K., & Kambhampati, S. (2009). Finding and exploiting goal opportunities in real-time during plan execution. *Proceedings of the International Conference on Intelligent Robots and Systems* (pp. 3912-3917). St. Louis, MO: IEEE Press.
- Spears, D., Thayer, D. and Zarzhitsky, D. (2009). Foundations of swarm robotic chemical plume tracing from a fluid dynamics perspective. *International Journal of Intelligent Computing and Cybernetics*, 2(4), 745–785
- Talamadupula, K., Kambhampati, S., Schermerhorn, P., Benton, J., & Scheutz, M. (2011). Planning for human-robot teaming. In G. Cortellessa, M. Do, R. Rasconi, & N. Yorke-Smith

- (Eds.) *Scheduling and Planning Applications: Papers from the ICAPS Workshop*. Freiburg, Germany: AAAI Press.
- Tan, C.S., Sutton, R., & Chudley, J. (2004). An incremental stochastic motion planning technique for autonomous underwater vehicles. *Proceedings of IFAC Control Applications in Marine Systems Conference* (pp. 483-488). Ancona, Italy: Elsevier.
- Warren, C.W. (1990). A technique for autonomous underwater vehicle route planning. *IEEE Journal of Oceanic Engineering*, 15(3), 199-204.
- Weber, B. G., Mateas, M., & Jhala, A. (2012). Learning from demonstration for goal-driven autonomy. In *Proceedings of the Twenty-sixth AAAI Conference on Artificial Intelligence*. Toronto, Ontario, Canada: AAAI Press.
- Wilson, M., Molineaux, M., & Aha, D.W. (2013). Domain-independent heuristics for goal formulation. In *Proceedings of the Twenty-sixth International FLAIRS Conference*. St. Pete Beach, FL: AAAI Press.
- Wongpiromsarn, T., Topcu, U., & Murray, R. M. (2012). Receding horizon temporal logic planning. *IEEE Transactions on Automatic Control*, 57(11), 2817-2830.
- Wooden, D., Malchano, M., Blankespoor, K., Howard, A., Rizzi, A., & Raibert, M. (2010). Autonomous navigation for BigDog. *Proceedings of the International Conference on Robotics and Automation* (pp. 4736-4741). Anchorage, AK: IEEE Press.
- Worcester, J., Rogoff, J., & Hsieh M. (2011). Constrained task partitioning for distributed assembly. *Proceedings of the International Conference on Intelligent Robots and Systems* (pp. 4790-4796). Vilamoura, Algarve, Portugal: IEEE Press.

Author Index

Aha, David W.	64, 111, 127
Alford, Ron	95
Apker, Thomas	127
Auslander, Bryan	127
Bobrow, Robert	1
Brinn, Marshall	1
Burstein, Mark	1
Cordier, Amélie	26
Cox, Michael T.	10, 79
Forbus, Kenneth D.	34
Garnier, Joseph P.	26
Georgeon, Olivier L.	26
Hinrichs, Thomas R.	34
Johnson, Benjamin	127
Karg, Michael	43
Kirsch, Alexandra	43
Klenk, Matthew	111
Kuter, Ugur	53, 95
Laddaga, Robert	1
Maynard, Michael	79
McMahon, James	127
Molineaux, Matthew	64, 111
Muñoz-Avila, Héctor	53
Nau, Dana	95
Paisner, Matt	79
Perlis, Don	79
Shivashankar, Vikas	95
Vattam, Swaroop	111
Wilson, Mark	127