Goal Reasoning:

Papers from the ACS Workshop

Technical Report GT-IRIM-CR-2015-001

(www.cc.gatech.edu/~svattam/goal-reasoning)

Atlanta, GA
28 May 2015

Chair: David W. Aha
Adaptive Systems Section
Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory (Code 5514)
4555 Overlook Ave., SW
Washington, DC 20375
USA
david.aha@nrl.navy.mil

Aha, D.W. (Ed.) (2015). *Goal reasoning*: *Papers from the ACS workshop* (Technical Report GT-IRIM-CR-2015-001). Atlanta, GA: Georgia Institute of Technology, Institute for Robotics and Intelligent Machines.

Preface

This technical report contains the 14 accepted papers presented at the Workshop on *Goal Reasoning*, which was held as part of the 2015 Conference on Advances in Cognitive Systems (ACS-15) in Atlanta, Georgia on 28 May 2015. This is the fourth in a series of workshops related to this topic, the first of which was the AAAI-10 Workshop on *Goal-Directed Autonomy*; the second was the *Self-Motivated Agents* (SeMoA) Workshop, held at Lehigh University in November 2012; and the third was the *Goal Reasoning* Workshop at ACS-13 in Baltimore, Maryland in December 2013.

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*** (GR), which concerns the ability of an intelligent agent to proactively reason about, formulate, select, and manage its own goals/objectives. GR 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 GR (e.g., in the context of cognitive architectures, intelligent agents, automated planning, game AI, and robotics). However, it has infrequently been the focus of intensive study, and (to my knowledge) no other series of meetings has focused specifically on GR. As shown in these papers, providing an agent with the ability to reason about its goals (independently of whether an initial set is provided) 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 involving partially observable and dynamically-updated states (e.g., due to unpredictable exogenous events), stochastic actions, multiple agents, and other complexities. Thus, this is an appropriate time to foster dialogue among researchers with interests in GR.

GR research is still in its early stages; no deployed 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 argue that GR 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.

The papers in this workshop relate to, among other topics, initial applications, formal models, meta-reasoning, narrative intelligence, emotion and personality, rebel agents, and self-motivated systems. The authors discuss a wide range of issues pertaining to GR, 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 topic.

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

David W. Aha
Atlanta, GA (USA)
28 May 2015

Table of Contents

Goal Reasoning and Narrative Cognition

Tory S. Anderson TORYS.ANDERSON@GATECH.EDU
Digital Media, Georgia Institute of Technology, Atlanta, GA 30332 USA

Abstract

Narrative cognition is central to how humans reason and make sense of their experiences. We create personal or natural narratives from the events of our lives which become a resource for future knowledge, beliefs, and goal reasoning. This paper is chiefly concerned with natural narrative, as opposed to traditional formal narratives of books, film, or theater. It focuses on the meaning-making functions of cognitive narrative and introduces a model of personal narrative generation from event perception. Several possibilities are elaborated by which the model can contribute to goal reasoning.

1. Introduction

In addition to being the primary content of the media we consume on a daily and hourly basis, narrative plays a key role in our cognition in making sense of our everyday experiences. This *natural narrative* that is a currency of cognition is distinguished from *formal narrative* that is composed and conveyed through media. Unlike many systems which serve intelligence—systems natural and artificial—narrative is definitively general: it is essentially cross-domain and polymorphic (i.e. non-expert). This fact is demonstrated by its ubiquity, by the use of concepts like “genre” which are necessary to distinguish between the breadth of possibilities exhibited by narratives, and by the heavy interaction narratives have with cognitive processes like analogy, upon which traditional narrative forms like allegory and metaphor are based. Although narrative itself is not necessarily goal driven, goal reasoning (GR) within humans is likely to draw significantly from the resources of narrative cognition. Formulations of GR benefit from qualities that are either provided by or shared with narrative cognition. This paper starts by providing a basic introduction to the literature and concepts of narrative cognition and narrative intelligence, followed by a consideration of narrative cognition in conjunction with GR. Related work is presented in which functions related to narrative cognition are applied in GR systems. Next is introduced a model for implementing basic narrative intelligence for long-lived agents, based upon the event segmentation theory for event perception. We conclude with a discussion of applications of this model to goal reasoning and suggestions for future work.

1.1 Narrative Cognition and Narrative Intelligence

Some of the earliest known precedents for narratology started with Aristotle, whose reflections on narrative came part and parcel with his philosophy of the dramatic arts (Butcher et al., 1961).

His work provided a definition of narrative structure as that with a beginning, a middle, and an end. Though this notion has been challenged by more recent literary and artistic movements it remains influential and, honoring the intuitive appeal of that simple definition, is retained as one of the core concepts of narrative in modern cognitive narratology. Within the last century serious philosophies of individual and society have descended from Aristotle's dramatic approach. These include Kenneth Burke's dramatism (Burke, 1969) and Walter Fisher's narrative paradigm (Fisher, 1984), which take literally Shakespeare's "All the world's a stage" and are broadly influential in the fields of social communication. **Narrative cognition** spans all of these and is a categorical term for cognition that applies or specifically works upon narrative.

In more recent years literary critics have also referred to the reality-negotiating importance of narrative. David Herman, one of the leading proponents of the term and field of cognitive narrative, recognized the key role narrative plays in the mind and suggested five core functions narrative plays in human cognition (Herman, 2007; Herman, 2013): 1) sense-making by segmenting experience into useful chunks, 2) causally linking events, 3) typifying phenomena to determine norms, 4) sequencing actions (including planning), and 5) distributing intelligence across time and space, including the function of communication. Although each of these five functions has significant implications for cognition and goal reasoning, and any system exhibiting narrative cognition will contribute to each of these, this work highlights the first function: sense-making of experience.

The ability to interpret experience in narrative is a form of narrative cognition called **narrative intelligence** by Mateas and has been recognized as a particularly desirable aspiration for AI systems (Mateas & Sengers, 1999). Modern AI as a whole owes significant parts of its ancestry to efforts in narrative comprehension (Brewer, 1982; Schank & Abelson, 1977). Work on narrative comprehension includes Schank and Abelson's pioneering AI work and modern systems aiming, for example, at news story and blog summarization (Cullingford, 1978; Mani, 2013; van Erp, Fokkens, & Vossen, 2014). These systems continue to be the focus of considerable research effort; however, less attention has been given to the capability of long-lived agent-based systems to make narrative sense of their experiences in something like the way of humans. As perhaps the most important form of narrative intelligence for us, the ability to negotiate reality by reflecting upon our experiences is the subject of *Living Narrative* (Ochs & Capps, 2009), a hallmark consideration of personal narratives in children and adults as a sense-making tool. As children mature they improve in the selection, complexity, and expressive power of their personal narratives (also explored in (Botvin & Sutton-Smith, 1977) and (Wigglesworth, 1997)). As shall be discussed, the improvement and growing sophistication of these narratives most likely coincides with the accumulation of narrative structures like scripts and schema in semantic memory. The accumulation of experiences into episodic memory provides the basis for the narrative store with which narrative intelligence in children starts (Anderson, 2015).

1.2 Narrative Intelligence and Goal Reasoning

Goal reasoning, as surveyed in breadth in (Vattam et al., 2013), requires architectures with three fundamental features: explicit goals, goal formulation, and goal management. None of these are strictly required in formal narrative; post-modern non-structural approaches to literature and film have specifically explored narratives that offer as little of these as possible. To this extent narrative cannot, of itself, provide the architecture necessary for goal reasoning, nor does it suggest goal reasoning as one of the functions upon which it relies. However, more can be said for the four challenges to traditional planning which motivate GR:

1. Non-deterministic partially observable environments
2. Dynamic environments
3. Incomplete knowledge
4. Knowledge engineering

The first and third points each indicate the kind of incomplete knowledge that drives narrative as well. This is particularly true of personal narratives, the first-person accounts that result from reflection upon experience. As analyzed by Ochs and Kapps the reality-negotiation of personal narratives are driven by the tension "between narrators' yearning for coherence of life experience and their yearning for authenticity" (Ochs & Capps, 2009, p. 24), a product of agents which have incomplete knowledge working in domains that disallow omniscience. One of their examples is of a woman describing and revising the story of her experience of being paid what she thought might be too much by a day-care client while her husband argued that the payment was correct; whether the payment was correct, what were the intentions of the payer, and what moral conclusion to ascribe were all in flux as the personal narrative was refined and revised.

The second point refers to environments which are subject to change, with or without intentional manipulation by the agent. This quality of change echoes the requirements of the theory of narrative cognition to be elaborated below, which suggests that the fundamental building-block of narratives are the product of prediction failures within the environment. Like GR, narratives seem to require dynamic environments.

Finally, point four refers to the real-world constraint imposed by unlimited richness, which is related to the second's point indicating nearly intractable knowledge problems. Humans are undergoing constant internal knowledge engineering via learning, largely via experience. The storage of this experience for analysis and reflection is the evolutionary role of episodic memory (Allen & Fortin, 2013; Tulving, 2002) and narrative intelligence. Implementing and improving narrative intelligence within agents is therefore a method of addressing the challenge of a rich domain.

2. Related Work

Related to the work of generating personal narratives are explanation systems, which perform some form of reasoning over their previous operations to communicate to the user the reasons for decisions or activities. As a form of narrative cognition explanation generation has been found to be a method for improving learning in humans (Fukaya, 2013). Computational explanation generation systems aim to provide users with insight into what is otherwise a black box of reasoning. Such systems face the challenge of distilling from the compiled knowledge of an expert system a sensible and accurate account of reasoning to a human operator, which may require substantial knowledge engineering beyond the knowledge with which the system is performing its tasks (Matheson, Coghill, &

Sripada, 2012). Some approaches almost entirely divorce the explanation from the activities of the system (Wang, 2012). Explanation generation systems differ from human explanation generation, and from human narrative cognition, in that usually the explanations are of no use to the systems themselves, and therefore do not provide the benefits of learning, reflection, or metacognition.

A reasoning method that does seek to benefit the agents is abductive reasoning by which an agent refines its beliefs by reflecting upon the past, deriving explanations for its own benefit. A prime example is DiscoverHistory (Molineaux, Kuter, & Klenk, 2012), which is designed with GR problem domains in mind and is able to ascertain novel facts about its world by detecting errors in prediction. Abductive reasoning in humans can be considered alongside analogy as a primary instrument to be applied in narrative cognition.

Narrative generation, which is a function of narrative cognition when it occurs in a cognitive system, is a major trend in AI, often for computer games; this work can bring to bear analogy (Zhu, 2011), planning (Riedl & Young, 2006; Riedl & Young, 2010), and other dramatic and theoretical creativity-based approaches (e.g. (Gervás, 2009; Pita, Magerko, & Brodie, 2007; O'Neill & Riedl, 2011)). Several of the leading systems in narrative generation employ planners to generate their narratives. These plausibly define a narrative as a causal sequence of events that can be represented as a partially ordered plan. Systems have been developed to specialize typical planning algorithms to the added tasks of coherence necessary for narrative (Riedl & Young, 2004; Riedl & Young, 2010). Planning is one of the functions of narrative cognition; however, conventional narrative generation approaches are not cognitively oriented and are primarily concerned with fiction generation, and as such require a controlled set of inputs. They do not directly provide the potential for learning or internal reasoning.

3. Towards a Model of Narrative Intelligence

The model for narrative cognition to be elaborated takes a simple concept of narrative similar to those used by narrative generation approaches: that a narrative consists of a sequence of one or more causally related events. These events can be provided by the experience of the agent, in which case they are stored in some form of episodic memory. After reviewing some computational implementations of episodic memory we introduce Zacks' event segmentation theory, which provides the basis for segmenting events from the perceptual stream. We then consider the generation of narratives from the events that have been generated.

3.1 Semantic and Episodic Memory

Tulving introduced the terms *semantic memory* and *episodic memory* to distinguish between distinct memory systems in our declarative memory (also called "explicit memory," storing contents that can be consciously recalled) (Tulving, 1972). Semantic memory stores general facts which are not dependent upon location or time, while episodic memory encodes experiences.

Both semantic memory and episodic memory play key roles in narrative cognition. Narrative structures, particularly scripts and schema, are stored in semantic memory. The evidence of children's narratives improving with age indicates that the acquisition of these structures is a key product

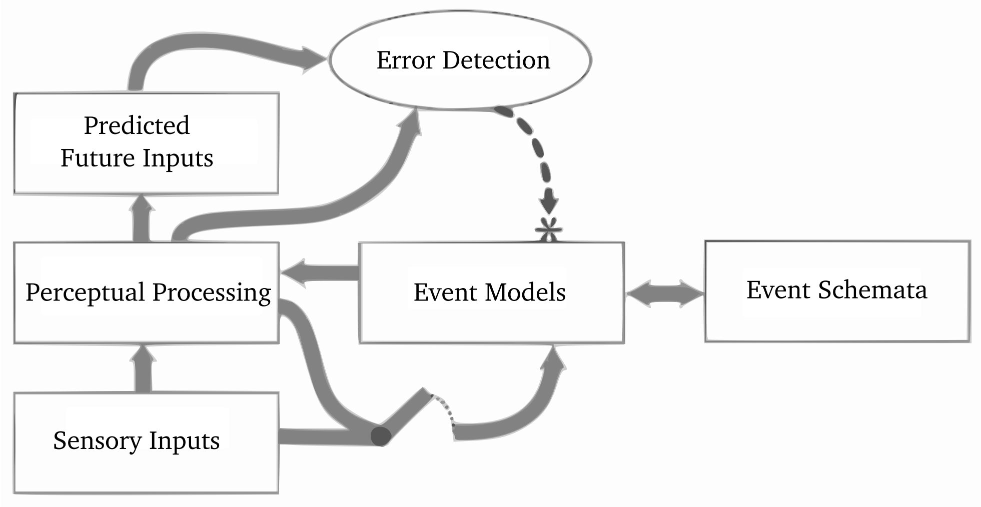


Figure 1. A high-level view of EST, image from (Zacks et al., 2007). Information flows along gray arrows and error detection leads to resetting of event models, which is the only time sensory input is gated to them.

of experience (Ochs & Capps, 2009, ch. 2). These structures are learned as episodic memory grows with experience, and influence human reconstruction of memories (Bartlett, 1932).

Episodic memory has received significant attention from researchers in recent years. It is believed to be the storage place for temporally contextualized data, including personal experiences, attended experiences (experiences depicted in stories), and prospective memory (Schacter, Addis, & Buckner, 2007; Tulving, 2002). Distinctions have been made including autobiographical memory, a subset of episodic memory that is specifically concerned with personal memories exhibiting autonoetic consciousness (Wheeler, 2000).

Computational systems often overlook episodic memory, and implementations of episodic memory are usually simplistic. For example, the Soar cognitive architecture (Laird, 2012) only recently implemented episodic memory, and is one of the only cognitive architectures to support episodic memory. In Soar, similar to most systems that do have a form of episodic memory, the implementation consists of time-stamping of recorded states from the operation of the system. This differs from the concept of narrative earlier given in a significant detail: while sequentiality is preserved, such time line recordings provide no concept of "event." Such episodic sequences are nondilineated streams of external perception and/or internal state.

3.2 Event Segmentation Theory

Event Segmentation Theory (EST) (Zacks et al., 2007; Kurby & Zacks, 2008; Radvansky & Zacks, 2014) suggests an approach to the problem of dividing a non-delineated sequence of states into events that could become the constituents of narratives. In humans, event segmentation is an ongoing process occurring simultaneously at multiple time/action granularities. According to EST, event segmentation occurs as an effect of ongoing perceptual prediction. During the process of perception two structures participate in parsing the situation and forming predictions: long-term knowledge is brought to bear in the form of event schemata, which are similar to Schanks' and Abelson's scripts (Schank & Abelson, 1977) and represent the way actions or events normally unfold in similar situations; and working-memory is brought to bear by event models, which are an interpretation of the specific situation at hand. In addition, behavioral models may be used so that predictions can be made based on the presumed goals of the actors in a situation, and world models that account for physical expectations (e.g. the trajectory of an object in free motion). The interplay between the semantic and episodic long-term memory systems in this process is cyclical: semantic memory provides the structures and models to help make episodes from experience, while these episodes are committed to episodic memory where, over time, they help distill further knowledge of semantic structures.

As perception occurs, a cognitive system selects from its knowledge of usual event schemas and uses assumptions about the goals and processes at work in the attended situation to generate expectations of what will happen next. As long as these predictions are mostly fulfilled, the current event model is assumed to continue and no segmentation occurs. However, when the predictions are wrong by some margin of significance, the current event is considered to end and a new event to begin in the process of selecting or generating a new event model. These explanations of event segmentation have been supported by evidence from studies of segmentation of event boundaries in written and video narratives (Zacks et al., 2007). Narratives are constructed as segmentation occurs at broader granularities over episodic memory, to the point of eventually contributing to production of the life-long autobiographical memories that "make up our own personal narrative of who we are and what we have experienced" (Radvansky & Zacks, 2014, ch. 8).

3.3 Generating Personal Narratives

The events produced by EST can be seen as constituting the fundamental building blocks of a narrative, a sequence of events. Narratives can be produced at either of two times: at storage time, useful for short-term reasoning that will operate in conjunction with working memory and is primarily dependent upon the narrative structures provided by semantic memory, and at reflection time. We consider production time narratives to be *first-order narratives*. The latter we call *meta-narratives*, which are of particular interest for meta-cognition and for the development of a life story or narrative identity (King, 2000), as well as for broad understanding of situations that draw upon constellations of narratives over time. In parsimony with the hierarchical nature of perception, personal narrative is hierarchical and recursive such that repeated reflection can produce increasingly complex compounds of narratives. This process, in which the recognizable and (in humans) presumably interesting narratives are formed, occurs on-demand. In particular, meta-narratives are formed when narrative cognition is called upon to fulfill one of the duties elaborated by Herman, such as casually linking events or sense-making (generating a new event model after prediction error).

4. Conclusion

Although narrative as a whole is not primarily goal directed it is essentially concerned with the same problems that necessitate GR. This collocation of interests also underscores the potential of narrative to provide the environment or resources in which goal reasoning can occur, potentially contributing functions of narrative cognition that have developed for the purpose of addressing the challenges previously mentioned.

Goal Reasoning and narrative cognition are collocated in dynamically rich, under-specified, limited-information domains. Within these domains narrative cognition performs a number of functions that serve to increase knowledge and generate predictions capable of informing GR processes. A particular affordance of a system performing narrative intelligence via EST is concurrent goal reasoning. Models of GR share with EST the use of prediction errors to cue activities. Running in parallel goal reasoning can itself become a producer of events over which narrative cognition can work.

In addition, goal reasoning itself constitutes a sequential process, for example as imaged by the goal life cycle of (Roberts et al., 2014). Applied to such a goal-reasoning process narratives of goal reasoning itself could be generated. Imagined verbally these narratives would resemble the internal dialogues we sometimes have with ourselves as we struggle with decisions, and the application of functions of narrative cognition to these processes offer a promising avenue for future work.

References

Allen, T. A., & Fortin, N. J. (2013). The evolution of episodic memory. *Proceedings of the National Academy of Sciences*, *110*, 10379–10386.

Anderson, T. S. (2015). From episodic memory to narrative in a cognitive architecture. *In Publication*.

Bartlett, F. C. (1932). Remembering. *Cambridge, UP*.

Botvin, G. J., & Sutton-Smith, B. (1977). The development of structural complexity in children's fantasy narratives. *Developmental Psychology*, *13*, 377–388.

Brewer, W. F. (1982). Plan understanding, narrative comprehension, and story schemas. *AAAI* (pp. 262–264).

Burke, K. (1969). *A rhetoric of motives*, Vol. 111. Univ of California Press.

Butcher, S., et al. (1961). *Aristotle's poetics*. Hill & Wang.

Cullingford, R. E. (1978). *Script application: computer understanding of newspaper stories.* (Technical Report). DTIC Document.

Fisher, W. R. (1984). Narration as a human communication paradigm: The case of public moral argument. *Communications Monographs*, *51*, 1–22.

Fukaya, T. (2013). Explanation generation, not explanation expectancy, improves metacomprehension accuracy. *Metacognition and learning*, *8*, 1–18.

Gervás, P. (2009). Computational approaches to storytelling and creativity. *AI Magazine*, *30*, 49.

Herman, D. (2007). Storytelling and the sciences of mind: Cognitive narratology, discursive psychology, and narratives in face-to-face interaction. *Narrative*, *15*, 306–334.

Herman, D. (2013). *Storytelling and the sciences of mind.* MIT Press.

King, N. (2000). Memory, narrative, identity. *Remembering the Self Edinburgh University Press, Edinburgh*.

Kurby, C. A., & Zacks, J. M. (2008). Segmentation in the perception and memory of events. *Trends in Cognitive Sciences*, *12*, 72–79.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA, USA: MIT Press.

Mani, I. (2013). Plots as Summaries of Event Chains (Invited Talk). *2013 Workshop on Computational Models of Narrative* (pp. 3–3). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Mateas, M., & Sengers, P. (1999). Narrative intelligence. *Proceedings AAAI Fall Symposium on Narrative Intelligence* (pp. 1–10).

Matheson, D., Coghill, G. M., & Sripada, S. (2012). Integrating natural language generation and model-based reasoning for explanation generation. *Computational Intelligence (UKCI), 2012 12th UK Workshop on* (pp. 1–7).

Molineaux, M., Kuter, U., & Klenk, M. (2012). DiscoverHistory: understanding the past in planning and execution. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2* (pp. 989–996).

Ochs, E., & Capps, L. (2009). *Living narrative: Creating lives in everyday storytelling*. Harvard University Press.

O'Neill, B., & Riedl, M. (2011). Toward a computational framework of suspense and dramatic arc. In *Affective computing and intelligent interaction*, 246–255. Springer.

Pita, J., Magerko, B., & Brodie, S. (2007). True story: dynamically generated, contextually linked quests in persistent systems. *Proceedings of the 2007 conference on Future Play* (pp. 145–151).

Radvansky, G., & Zacks, J. (2014). *Event cognition*. Oxford University Press.

Riedl, M. O., & Young, R. M. (2004). An intent-driven planner for multi-agent story generation. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1* (pp. 186–193).

Riedl, M. O., & Young, R. M. (2006). Story planning as exploratory creativity: Techniques for expanding the narrative search space. *New Generation Computing*, *24*, 303–323.

Riedl, M. O., & Young, R. M. (2010). Narrative planning: balancing plot and character. *Journal of Artificial Intelligence Research*, *39*, 217–268.

Roberts, M., Vattam, S., Alford, R., Auslander, B., Karneeb, J., Molineaux, M., Apker, T., Wilson, M., McMahon, J., & Aha, D. W. (2014). Iterative goal refinement for robotics. *ICAPS Workshop on Planning and Robotics*.

Schacter, D. L., Addis, D. R., & Buckner, R. L. (2007). Remembering the past to imagine the future: the prospective brain. *Nat Rev Neurosci*, *8*, 657–661.

Schank, R. C., & Abelson, R. (1977). *Scripts, goals, plans, and understanding*. Hillsdale, NJ: Erlbaum.

Tulving, E. (1972). *Organization of memory*. New York, NY: Academic Press.

Tulving, E. (2002). Episodic memory: from mind to brain. *Annual Review of Psychology*, 1–25. none.

van Erp, M., Fokkens, A., & Vossen, P. (2014). Finding Stories in 1,784,532 Events: Scaling Up Computational Models of Narrative. *2014 Workshop on Computational Models of Narrative* (pp. 241–245). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Vattam, S., Klenk, M., Molineaux, M., & Aha, D. W. (2013). *Breadth of approaches to goal reasoning: A research survey* (Technical Report). DTIC Document.

Wang, Q. (2012). *Developing a computational framework for explanation generation in knowledge-based systems and its application in automated feature recognition*. Doctoral dissertation, RMIT University.

Wheeler, M. A. (2000). Episodic memory and autonoetic awareness.

Wigglesworth, G. (1997). Children's individual approaches to the organization of narrative. *Journal of Child Language*, *24*, 279–309.

Zacks, J. M., Speer, N. K., Swallow, K. M., Braver, T. S., & Reynolds, J. R. (2007). Event perception: a mind-brain perspective. *Psychological bulletin*, *133*, 273.

Zhu, J. (2011). On the role of domain knowledge in analogy-based story generation. *Evaluation*, *2*, 5.

Interactive Knowledge-Goal Reasoning

Benjamin Bengfort BENGFORT@CS.UMD.EDU
Department of Computer Science, University of Maryland, College Park, MD 20742 USA

Michael T. Cox MICHAEL.COX@WRIGHT.EDU
Wright State Research Institute, Wright State University, Dayton, OH 45435 USA

Abstract

Knowledge goals are used by reasoning entities to fill in information required for decision making and are an important part of computational understanding systems. Simple knowledge goals can be solved through traditional information retrieval techniques or database queries. In order to solve complex knowledge goals, however, a plan must be composed and executed in an investigative manner. During computational investigation, goals of both the system and the user can change, resulting in goal trajectories that may inform future goal solutions. In this paper, we propose an interactive reasoning system that leverages a case-based methodology to solve complex knowledge goals represented as natural language queries.

1. Introduction

A *knowledge goal* represents the purposeful need to acquire information in order to fill in gaps of world knowledge for a reasoning entity or to extend the database of a computational understanding system (Ram & Hunter, 1991). For humans, knowledge goals are most easily represented as questions, and current research on dialog-driven question and answer systems focuses on the semantic parsing of a natural language question to a structured database query (Yahya et al., 2012) or lambda calculus representation (Berant et al., 2013). These parsing approaches are making headway in the solution of simple knowledge goals, where the primary task is a retrieval from some structured knowledge base; especially goals that ask who, what, when, or where. This approach, however, cannot solve complex knowledge goals including aggregations, opinions (recommendations), or explanations for why or how questions. As a result, even though information retrieval systems and search have made information easily accessible, there has been the growth of community-driven question sites such as Quora (Wang et al., 2013) to connect users to the more nuanced answers they are looking for.

We claim that humans solve complex knowledge goals through *investigation*, dividing harder questions into simpler sub-knowledge goals with more easily obtained solutions. When solving knowledge goals, humans also take into account context and approach new problems by leveraging techniques that have worked in the past. A system that models how humans investigate questions will *deconstruct* a complex goal, *integrate* relevant knowledge, and *reuse* methods of investigation. Investigation can be represented as a plan to solve the larger knowledge goal, and if the tasks or subgoals of knowledge acquisition plans are simple knowledge goals that can be solved via structured data retrieval, then a system can be said to solve complex knowledge goals through a planning process that involves the purposeful combination of simpler knowledge goals.

In this paper we present a vision for an interactive knowledge-goal reasoning system that uses a case-based reasoning methodology to deconstruct complex knowledge goals based on previous cases for similar questions. The system will provide users with goal-driven solutions to natural language queries by proposing plans for simpler knowledge goals, which can be satisfied by computationally tractable tasks like database queries or document retrieval. The system takes into account the context of the goal by leveraging semantic resources applied to the question and utilizes a wide array of data sources to perform retrieval tasks–from database queries, to search, to providing other user responses for complex questions. As subtasks are selected by the initiating user, the case-based reasoning system reinforces and indexes how the plan was solved in order to provide more relevant plans or responses in the future.

Because this type of system is interactive and adaptive it is subject to goal changes, where the original knowledge goal is changed or refined slightly while the user interactively follows a solution plan. The complete case for solving a complex knowledge goal therefore represents a *goal trajectory*. Goal trajectories are important for refining future cases, ranking plan solutions, and having the system anticipate knowledge goals to propose better plans. Furthermore, the combination of goal trajectories can be seen as a creative process, leading to the extension of not only the casebase, but also to extend automatically the knowledge base of the system.

To explore interactive knowledge-goal reasoning, we will first discuss knowledge goals and goal trajectories in detail. This discussion will lead to a brief taxonomy of questions, which is vital to understanding the many types of cases in a natural language query system. In the second section, we will describe our proposed case-based methodology, including the case representation, acquisition, and management. Finally, we will present three case studies that demonstrate how an interactive knowledge-goal reasoning system might be employed.

2. Knowledge Goals

In information retrieval systems, the input is a query that usually takes the form of a Boolean combination of keywords and the expected output is a ranked list of documents that have a high relevance to the keyword combination. Similarly, in database management systems the input is a formal query that must conform to the logical structure of the information, and the output is a set of records or aggregations that also conform to the schema. While these systems are important in the management of large-scale, multi-modal information sets, they do not represent the way that humans formulate knowledge goals: questions. Although search and structured database queries may be viewed as stand-ins for knowledge goals, they only embed the execution context or *task* of the knowledge goal, whereas natural language queries express more information related to the goal and can be both *simple* and *complex* where formal languages are not as flexible.

We can define *simple* knowledge goals as questions that are directly tied to a retrieval task and whose solution may be found either through traditional information retrieval or through structured queries. Consider the question "*Who won the World Cup?*" The goal of this question is to determine the `Country` entity that won the most recent World Cup and possibly also the `Result` in the form of the competing teams and final score. Search methodologies would find documents with a high relevance to the tokens in the text of the question, particularly "World Cup". A structured database query upon some knowledge base would require semantic parsing to determine the relationship between the `FIFA World Cup` topic, and a specific `Country` topic. Even with these challenges, the result of both search and database query would fulfill the knowledge goal.

Complex knowledge goals, on the other hand, cannot be solved simply through a single retrieval task and are generally related to questions that require investigation, opinion, explanation, or knowledge generation. Consider the question "*Where should I go to dinner?*" This question is not a simple retrieval task because although restaurant reviews and geographic proximity searches might narrow the search space to dozens of possible candidate restaurants, selecting a dining option involves a reasoning process that may lead to solutions that do not involve restaurants at all, like going on a picnic. Moreover, this routine question will have different responses when asked by the same user multiple times, depending on context like geographic location, the season, or changing user preferences. Other complex knowledge goals are used to generate information, for example "*Can a crocodile complete a steeplechase?*", a fact that is probably not part of most knowledge bases. Unless someone has already investigated this question, the goal will have to be decomposed into lines of questioning related to the physical requirements of completing a steeplechase compared to the athletic ability of crocodiles.

2.1 Knowledge Goal Representation

The representation of a knowledge goal can be described by its objective: the type of the expected response (concept specification) and how the information will be used (task specification) (Ram, 1991). To create a plan to solve a knowledge goal, a computational system must be able to represent knowledge goals in a structured manner, and identify and parse the structure from a natural language query. We extend this representation to leverage the additional components available in an interactive knowledge-goal reasoning system. Knowledge goals are composed of three primary features: a set of *concepts*, a *task*, and a specific *context*.

1. **Concept**: The knowledge goal is relevant to a specific set of structured semantic concepts, both entities and predicates, which are either directly specified or implied. The concept of the question identifies the domain of the query plan as well as the expected response.

2. **Task**: The task indicates the purpose of the knowledge goal and is usually implied in a natural language query and related to the motivation of the questioner. For example, in the case of simple knowledge goals the task may be search or database query; for more complex knowledge goals the task may be explanation or knowledge generation. It is by task that different types of knowledge goals are categorized in the taxonomy below.

3. **Context**: Knowledge goals are not specified in isolation, but are directly related to the questioner and as such must be considered in relation to their context. Contextual information includes temporal, geographic, hysteretic, and preference information and is essential for plan refinement and to resolve ambiguity.

In the question "*When is the next flight to Paris?*" the *concept* explicitly includes the topic `Air Travel` as well as a city, `Paris`. Implied concepts might include `Paris, France`, the most popular city named Paris, and `Charles de Gaulle Airport`. The *task* can generally be defined as a simple database query or search, but more specifically the return of a specific `Flight` record. The `context` is critical as well, especially the geographic context (the next flight to Paris from the city where the user is currently located) and the temporal context (in order to determine when the next flight might be).

Many applications simplify one or more components of a knowledge goal in order to be effective. Domain-specific applications reduce ambiguity in concepts and context by allowing only specifically relevant knowledge goals. More commonly, applications simplify the task and are designed to return only a single type of result. However, the widespread availability of large knowledge bases combined with timely contextual information from mobile devices mean that complete, personalized question and answer systems are becoming computationally tractable and knowledge goal simplification should no longer be used as an approximation for goal reasoning systems.

2.2 Solving Knowledge Goals

An important step in the development of dialog-driven question and answer systems was the creation of large-scale structured knowledge bases like Freebase (Bollacker et al., 2008) and Yago2 (Suchanek, Kasneci, & Weikum, 2007). These databases represent a wealth of information generated from collaboratively edited encyclopedic sites like Wikipedia and provide the means with which to make factual queries about the world. Semantic parsers can take advantage of hierarchical, relational ontologies which allow for a generalization in the domain or *concept* component of a question. For simple knowledge goals, semantic parsers can transform natural language questions into structured queries like SPARQL or SQL (Yahya et al., 2012), a lambda-calculus representation (Berant et al., 2013), or simply a template to directly search other answers (Unger et al., 2012).

However, these parsers will struggle with even slightly more complex natural language queries tasked with a database lookup in a knowledge base, such as "*Who was the King of England when Thomas Jefferson was President?*" This knowledge goal is a factual retrieval that can be solved by chaining subqueries to first solve which years Thomas Jefferson was President of the United States, then use the result to determine the King of England during that time period. Similarly, knowledge goals that require aggregation, such as "*How many countries have won three consecutive gold medals in a single sport?*", may be able to be transformed into a series of simpler sub queries which contribute information to a final result.

The decomposition method of solving fact-based retrieval questions can also be applied to complex knowledge goals whose task is not simply a query or search. When a complex knowledge goal is reduced to one or more simpler sub-goals, the result is a hierarchical plan to solve the original knowledge goal. If the final leaf nodes of such a plan are simple knowledge goals which can be transformed into a query or a search, then a system can said to solve complex knowledge goals through the computationally tractable combination of simpler knowledge goals.

Consider an example complex knowledge goal "*What courses should I take next semester?*" outlined in Figure 1. This knowledge goal is routine, executed by the same user on a regular interval and common enough that a casebase of solutions is readily available. The task of this knowledge

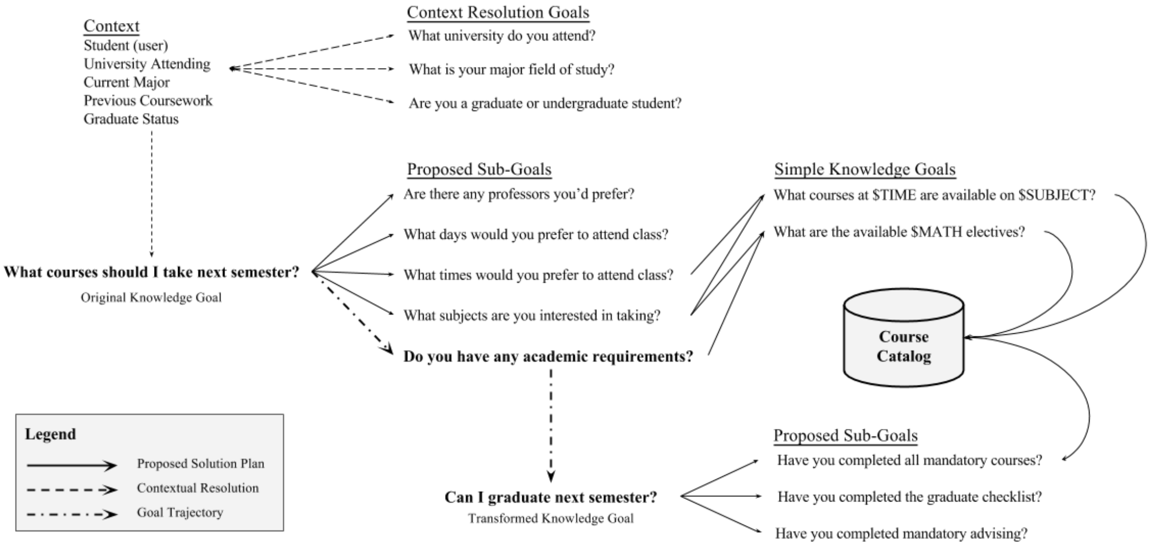

Context
Student (user)
University Attending
Current Major
Previous Coursework
Graduate Status
Context Resolution Goals
What university do you attend?
What is your major field of study?
Are you a graduate or undergraduate student?
Proposed Sub-Goals
Are there any professors you'd prefer?
What days would you prefer to attend class?
What times would you prefer to attend class?
What subjects are you interested in taking?
Do you have any academic requirements?
What courses should I take next semester?
Original Knowledge Goal
Simple Knowledge Goals
What courses at $TIME are available on $SUBJECT?
What are the available $MATH electives?
Course Catalog
Legend
Proposed Solution Plan
Contextual Resolution
Goal Trajectory
Can I graduate next semester?
Transformed Knowledge Goal
Proposed Sub-Goals
Have you completed all mandatory courses?
Have you completed the graduate checklist?
Have you completed mandatory advising?

Figure 1. Interactive knowledge goal reasoning on the complex goal "What courses should I take next semester?" The system resolves context dependencies and then proposes a plan of simpler sub-goals that lead to database queries using information from prior goals (represented as `$VAR`). Depending on interaction, goal trajectories can modify the original goal to "Can I graduate next semester?" continuing the process.

goal is to create a list of courses which are available next semester and which the student will presumably be able to register for. The concept of the question is the domain of courses that are available at a particular university during a particular semester. The context for a user determines preference in terms of the selected major or course level as well as geographical context (which specific university) and temporal context (which specific semester).

In order to solve this knowledge goal, a plan of simpler sub-knowledge goals may be offered to the user. If contextual information is missing, the system might respond with ambiguity resolution goals of its own (e.g. "*Have you selected a major?*"). Otherwise simpler sub-knowledge goals could include "*What days and times would you prefer?*", "*Are there any academic subjects you're interested in?*", or "*Do you have any academic requirements?*" Responses to these sub-knowledge goals would lead to even simpler knowledge goals which can be eventually be queried against a course catalog, such as "*Which Economics courses are available on Monday and Wednesday?*"

2.3 Goal Trajectories

The solution to complex knowledge goals is an interactive reasoning approach where a complex knowledge goal is broken down into a hierarchical plan involving simpler sub-goals and tasks. In an interactive reasoning system, the user selects the next step of plans proposed by the reasoner, and continues to chain sub-goals together to work towards a larger solution. During this process, the plan is adaptable and subject to change as the user proceeds in selecting and executing the simpler goals. Goals, like plans, are also subject to change and can be transformed (Cox & Veloso, 1998), therefore we propose that the original knowledge goal itself is also subject to change; and that as knowledge goals change during interactive reasoning, the path that led to the solution of new knowledge goals from the original can be represented by a *goal trajectory*.

Goal trajectories can be influenced either through the direct interactive manipulation of goals (Cox & Zhang, 2007) or via other users in the system issuing similar queries that provide the basis for recommending new goals through collaborative filtering (Hayes, Cunningham, & Smyth, 2001). State changes in the system via monitoring of new information or relevant data that has been added to the knowledge base can also lead to new, interactive responses. In either case, goal changes can be seen as a planning problem where knowledge goals are not static, and an interactive reasoning system must be responsive to goal changes.

The concept of goal trajectories can then be used to explicitly define relationships between knowledge goals as cases for future solutions. In the example from Figure 1, a goal trajectory is demonstrated by the proposed sub-knowledge goal "*Do you have any academic requirements?*" This leads the user to change the goal to "*Can I graduate next semester?*" This goal trajectory can then be used in future cases to propose graduate related sub-goals to those who may be close to finishing their degree.

2.4 Knowledge Goal Taxonomy

In the next section, we will discuss the solution of knowledge goals by decomposing a complex knowledge goal into subsequently simpler knowledge goals. However, in order to determine an execution plan for knowledge goals, some idea of the types of knowledge goals that might be in a system as well as their relative complexity is required. Knowledge goals as described in (Ram, 1990; Ram & Hunter, 1991) are presented with a categorization based on the task that they arise from. Similarly, we will extend this taxonomy with our planning framework, given the components of a knowledge goal discussed in the last section.

2.4.1 Simple Knowledge Goals

The simplest knowledge goals should be computationally tractable, such that they can be solved with a minimal amount of user involvement. Simple knowledge goals have tasks that range from search to database queries to computational tasks like parsing, aggregation, or inference. In an *interactive* system, both users and the system have simple knowledge goals. The system uses knowledge goals to resolve ambiguity through dialog boxes or through other types of feedback.

1. **Textual**: Questions related to both semantic or syntactic analysis of text, usually as a response to ambiguity. Tasks related to text questions include anaphora resolution or word sense disambiguation. Users may ask to define a word; the system may ask to clarify a concept.

2. **Contextual**: Questions designed to specify the result of a knowledge goal according to the user's context, such as queries related to preference, geography, or time. Similar to textual questions, these types of questions are "meta-questions" that are used to tailor the execution of a knowledge goal solution.

3. **Rhetorical**: Questions that should not return an answer. Questions that are used as placeholders or to express frustration are important to identify as simple knowledge goals because they identify terminal tasks in question plans.

4. **Retrieval**: Questions that expect a single fact returned from the database and may be transformed into a structured query. These types of questions may perform aggregations or filtering upon the knowledge base, but typically only return a single result or a small list of results.

5. **Search**: Questions that require a larger scope or domain and whose expected results are a list of relevant documents.

The tasks related to these simple knowledge goals are all able to be computed on behalf of a user. Knowledge goals that fall into these categories are the simplest goals that may be involved in an interactive knowledge-goal reasoning system.

2.4.2 Complex Knowledge Goals

Complex knowledge goals must be reasoned upon rather than computed directly, and cannot be directly parsed into a executable representation. Instead, a plan must be developed in order to solve complex knowledge goals which leverages the context and concepts in the goal. The types of tasks for complex knowledge define the categorization of these goals as follows:

1. **Explanation**: Questions that require an explanation and return an explanatory data structure. Tasks include the detection and resolution of anomalies as well as the construction of an inferential or causal explanation.

2. **Relevance**: These questions are designed to expand the current knowledge framework by adding relevance links between questions and related entities, answers or other questions. Relevance can be used in later processing as a shortcut to retrieval from a casebase.

3. **Socratic**: Socratic questions return a question as an answer, or rather a plan that consists of knowledge goals that are designed to answer the larger question.

4. **Research**: These types of questions are designed to add information to a knowledge base, either by adding facts or creating knowledge from other knowledge sources. The system should identify research questions based on unsatisfactory queries and add them to the system for later investigation.

5. **Routine**: Questions that are routine are asked frequently or on a specific interval. However the required answer will differ based on the context or timing of the question and will most likely not return the same answer as a previous instance of the question.

Tasks related to solving complex knowledge goals require some learning framework to mimic how humans solve similar goals. In an interactive reasoning system, the learning framework can include the use of similar cases or solutions to goals to propose a plan to solve the complex goal, or to connect related users as they investigate similar goals.

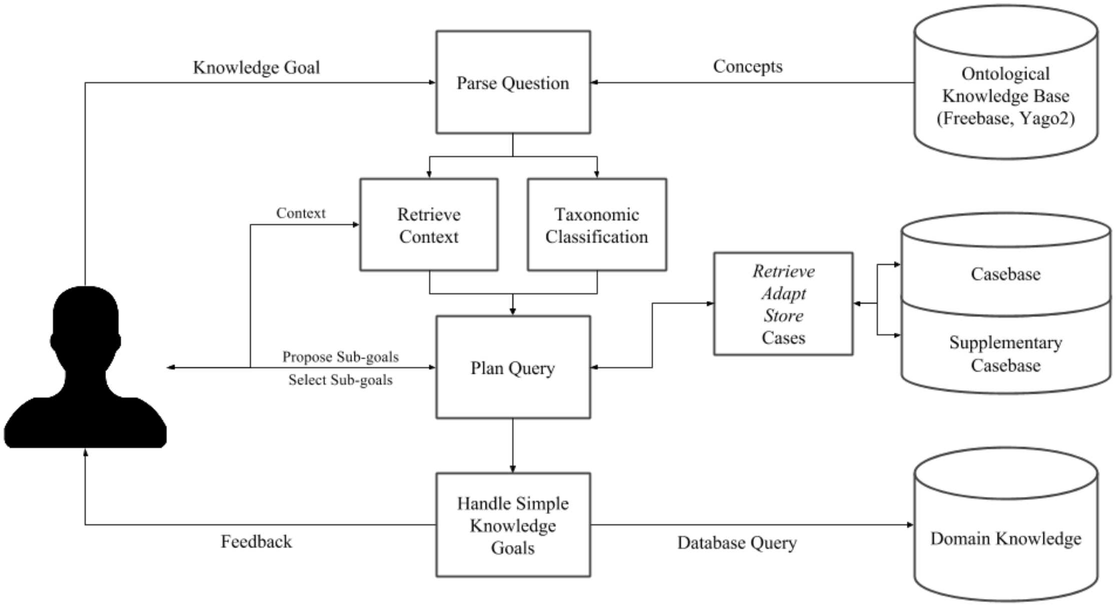


Figure 2. An architecture for interactive knowledge-goal reasoning. The goal reasoning system decomposes complex knowledge goals expressed as natural language queries into plans for simpler goals.

3. Case-Based Reasoning for Knowledge Goals

Our proposed solution for complex knowledge goals is a *case-based reasoning* (CBR) (Kolodner, 1993; Lopez De Mantaras et al., 2005) system which reuses past experience in an interactive fashion. Interactive CBR operates similarly to conversational case-based reasoning systems, which incrementally elicits a target problem through an interactive dialog with the user, attempting to minimize the number of questions before a solution is reached (Aha, McSherry, & Yang, 2005). In order to provide an adaptable, investigative system, the methodology we are exploring guides the user through goal trajectories, removing the requirement to minimize session length in order to facilitate an ongoing discovery process. Additionally, the system itself is a learning agent with the goal of predicting future knowledge goals, and acquiring the information in advance.

An interactive knowledge-goal reasoning system would require the ability to correctly identify *concepts* through some semantic parsing task. Knowledge goal *concepts* would be used to identify similar knowledge goals in the casebase for retrieval. The system would also require the ability to classify the *task* of the question in order to determine the execution context and to reuse previous cases. Finally the system would utilize the *context* of the questioner to adapt or revise cases for the specific scenario. Knowledge goals and their trajectories would be retained in the casebase along with their relationships such that the system learns over time and can anticipate future knowledge goals, for example by detecting common goal trajectories and proposing shorter paths to those anticipated goals. The required components and architecture of such an interactive knowledge goal system are described in Figure 2.

The input to the system is a natural language question, which is parsed to extract explicit topics defining the *concept* of the question. Concepts involved in the question are directly tied to a structured knowledge base and ontology, providing the ability to query implicit concepts such as relationships to other topics or entities. Once the concepts have been resolved, the system leverages a casebase to find similar questions. Those similar questions will be adapted with the current context of the user, for example if the user asks "*When is the next home game?*" the system can adapt previous questions regarding sporting events with seasonal context (football vs. baseball), or the user's team preference. If there is missing context, the system responds with a contextual knowledge goal to fill in the gap. Once the context has been added, the task for the case is identified via a taxonomic classifier using the taxonomy described in Section 2.4. If the task is a simple knowledge goal, it is executed against the knowledge base or via search. If the task is complex, then then new, similar questions based on the cases in the casebase are proposed to the user, the process continues in an interactive fashion until the session is complete.

3.1 Case Representation

Individual cases in our methodology would be composed of knowledge goals and their related goal trajectories. The representation can be either frame-based or objectual but would require a log of all questions in the system by user, and their resolution. For every natural language question, the case knowledge goal would include the text of the question, the concepts that were directly added to the question as part of the interactive process and any required context. The task for each case would also be added either as related knowledge goals, executed searches, or database queries. For example, given the case "*Where should I go to dinner?*" the representation might include the topics `Dinner` and `Restaurant`; the context requirements `location`, `restaurant genre preference`, and `price preference`; and a task which includes the sub-goal "*What are the closest restaurants to my location?*"

The choice to use semantic topics or concepts is not simply because of the availability of structured knowledge bases like Freebase. The concept of each knowledge goal case would be used to generalize the case topically through some hierarchical ontology in order to find similar questions or cases. This generalization would take the form of templates, where topics generalized at several levels of their ontological hierarchy would be placed in context with surrounding text. For example, consider the knowledge goal "*When does Georgia play Georgia Tech?*" If the concepts `University of Georgia` and `Georgia Institute of Technology` are added to the casebase (both are instances of the topic `College/University`), then this question could be applied more generally to knowledge goals related to intercollegiate activities. Furthermore if the concepts `Georgia Bulldogs Basketball` and `Georgia Tech Yellow Jackets Basketball`, both instances of the topic `School Sports Team`, are added to the case, then this question can be specified to basketball or football through contextual clues, or to professional sports in the more general case.

In order to enhance retrieval for the casebase, questions related through the concept hierarchy or through selected sub-knowledge goals during an interactive session with the user will be linked to each other through a supplementary casebase. These linkages are intended to represent potential goal trajectories. By selecting related cases through trajectories, the system will be able to more quickly present potential sub-knowledge goals to the user during an interactive session. Furthermore, the combination or transformation of goal trajectories can be seen as a creative process with which to generate new information in the system.

3.2 Case Acquistion

Cases will be acquired two ways: through direct user input during interactive investigative sessions and by the automatic generalization of cases through their concepts and related goal trajectories.

Acquiring cases from users directly inputting natural language questions into the casebase is the most obvious method of case acquisition. Users interact with the knowledge goal reasoning system in *investigative sessions* where the user specifies an initial knowledge goal, responds to system plan proposals, and finally finishes the investigative session when some solution for a set of knowledge goals has been reached. Sessions provide a finite time within which the specific knowledge goals can be evaluated in relation to each other; if all questions for a user were evaluated, it would be difficult to consider their relationships. Furthermore, sessions provide the reasoning system the ability to track goal trajectories, either by identifying changes to the initial knowledge goal through the explicit restart of an investigative session following a session with no solution, the explicit annotation of the user providing information about their goals, or through implicit inference based on user interaction.

The second way that cases are acquired is through the automatic generalization of cases. Users necessarily create cases that are extremely specific and cannot be applied generally to other problems with ease. Since the reasoning system is interactive, the system cannot passively wait for some density of cases in order to begin responding with proposed plans for knowledge goal solution, and must instead anticipate or plan for knowledge goal trajectories. Generalization happens through the linking of related questions, primarily through the template extraction process wherein concepts are inferred through the ontological hierarchy, properties, or relations.

3.3 Case Management

In a robust interactive knowledge-goal reasoning system, the casebase can become extremely large, requiring many system resources for adequate performance in both retrieval and adaptation of cases. Interactive systems especially will be affected by any slowdown in performance. Growth in the casebase is the result of both the automatic generalization of cases, and the natural specificity with which users ask questions. In order for the system to respond effectively to user queries, some technique is required to rank cases in the casebase or to collect similar cases into a single case.

The first step for good case management is the deduplication of natural language questions. This is essential for many question and answer systems where user responses are indexed, but is not an easy task. For example the questions "*What is the best restaurant in Seattle?*" and "*Where is the best place to eat in Seattle?*" are similar questions and must be considered duplicates for the purposes of the casebase. Most collaborative question and answer systems prompt the user at question time whether or not they are asking an already asked question; others use textual similarity scores to determine whether or not such questions are related.

Cases that are added to the casebase via generalization, however, cannot be prompted to a user to determine similarity. Generalization itself is a qualitative process that can generate many poor cases. Some quantitative process must be used to evaluate proposed cases, and to decay old cases such that the success of automatically generated cases is constantly evaluated and reinforced in the system. Metrics such as how connected a case is in goal trajectories, or how many specific cases can be subsumed by a general case, may indicate how successful general cases are.

Both user-added cases as well as those generalized by the system must be ranked through user feedback and relevance scoring. In an interactive reasoning system, the interaction is the strongest signal to good reasoning. As cases become less used in the system, they should decay by first reducing their rank in solution proposals and finally by being archived off the casebase. By analyzing user interaction as the primary metric for case relevance, the casebase can be maintained at a specificity required to quickly solve knowledge goals, but at a generality that allows for easy adaptation.

4. Case Studies and Applications

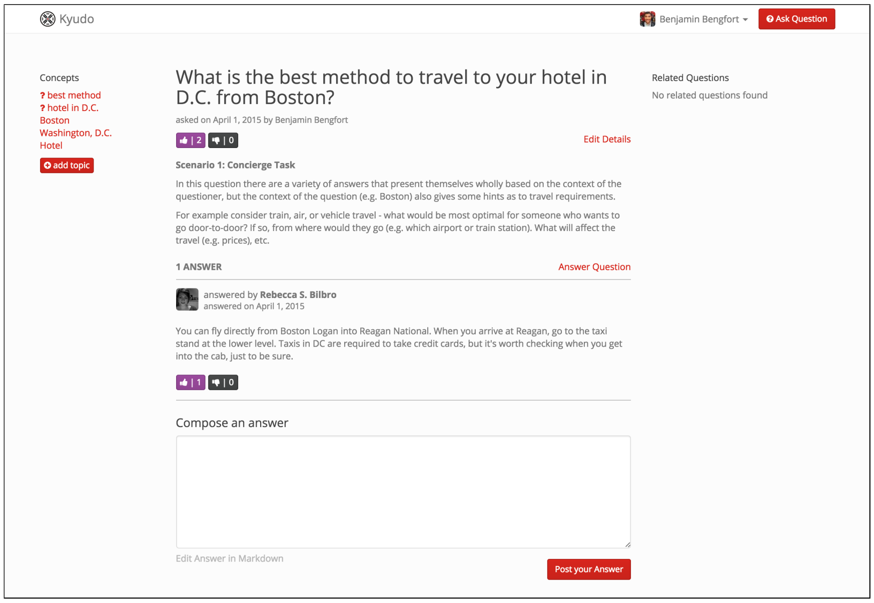
In order to acquire a casebase with which to study interactive knowledge-goal reasoning systems in the context of goal trajectories, we have created a web application called Kyudo to generate a dataset of goals, questions, and answers. Kyudo takes the form of a community-driven question and answer system, demonstrated by the screenshot in Figure 3. Users ask natural language questions and search for related questions. Each question is parsed syntactically; then, a lightweight semantic parser identifies noun phrases using a rule-based mechanism and proposes them to the user as the concepts related to the question. Users can annotate the concepts according to Freebase topics, add additional concepts through a Freebase search, and add details about the question.

Asked questions are then answered by other users who can specify free text about how the knowledge goal should be solved. Answers can be in the form of a Freebase topic (to simulate database retrieval questions), or a proposed plan of simpler knowledge goals which can be asked on behalf of the questioner and linked to the original question. Both the concepts and the parse can be annotated by the user. Furthermore both questions and answers can be "upvoted" and "downvoted" to indicate relevance. Questions can be tagged with their expected task, and related questions can be directly added by users to connect similar knowledge goals or knowledge goals that may be part of the same goal trajectory. Through all of these user annotation mechanisms, we intend to generate a small casebase with the representation described in an earlier section.

Although we intend to create a generally applicable system, we have found that in the initial generation of a casebase, where the user is required to specify many details and annotations, a specified task to guide case acquisition simplifies the process considerably. In this section we identify and describe the three tasks which we have given to a pilot study of 17 graduate student users for our initial casebase generation. These tasks can also be seen as case studies of how an interactive knowledge-goal reasoning system might operate within a specific domain.

4.1 Conceirge

Concierge staff are expected to be available to answer questions from tourists who are unfamiliar with the city that they are staying in. Often, they go much further than simple responses to questions



What is the best method to travel to your hotel in D.C. from Boston?

asked on April 1, 2015 by Benjamin Bengfort

Scenario 1: Concierge Task

In this question there are a variety of answers that present themselves wholly based on the context of the questioner, but the context of the question (e.g. Boston) also gives some hints as to travel requirements.

For example consider train, air, or vehicle travel - what would be most optimal for someone who wants to go door-to-door? If so, from where would they go (e.g. which airport or train station). What will affect the travel (e.g. prices), etc.

1 ANSWER

answered by **Rebecca S. Bilbro**
answered on April 1, 2015

You can fly directly from Boston Logan into Reagan National. When you arrive at Reagan, go to the taxi stand at the lower level. Taxis in DC are required to take credit cards, but it's worth checking when you get into the cab, just to be sure.

Compose an answer

Figure 3. A screenshot of Kyudo, our preliminary case acquisition system.

such as “*How do I get to the Eiffel Tower?*”–they suggest places to eat or other interesting things to see, warn of “tourist traps”, and generally work not only to answer the question that is being asked, but to identify the larger goals of the travelers in order to make their stay excellent.

In this task, the users of the system are expected to ask questions as though they are travelers speaking to a concierge. Other users should respond to the questions as though they are a concierge, giving tips, feedback, suggestions, and recommendations in addition to answering the primary question. To obtain more specific questions, the questions are framed as though the task is taking place in a hotel in downtown Washington DC.

Knowledge goal trajectories are readily apparent in this task because most of the questions are related to advice or opinion. For example, one of the questions in the system “*Are the food carts ever a source of food poisoning?*” led to an interactive session where other users determined that the traveller was interested in foodie experiences related to a popular television travel show. In order to identify gourmet food trucks, they proposed a truck tracker app for the traveller’s iPhone. This led to a discussion of other local apps and eventually the traveller was receiving advice about how to use the Metro and Smithsonian apps.

4.2 Presidential Briefer

The second case acquisition task simulated the creation of a presidential briefing. The role of the presidential briefer is to prepare a report on the day’s news and intelligence activities to be reported to POTUS first thing in the morning. The goal is to deliver as much information as possible in a short amount of time. As such, the briefer must anticipate the questions that the president might have and prepare answers in advance. The final product is the "brief book": a book with an executive summary, then detailed information designed to respond to questions.

In this task, users are expected to anticipate questions that the president might ask when they brief him, specifically from news stories that they are preparing the briefing for. Users should prepare questions related to specific news stories, and include the news stories as background in the details section of the question. "Answers" to these questions should be detailed, and also anticipatory of "follow-on" questions that the president might ask.

This task was designed to capture not only knowledge goal trajectories in the form of follow-on questions, but also to expand the variety of tasks from the knowledge goal taxonomy. In particular, because these knowledge goals were related to specific news items, textual and contextual knowledge goals would be implicitly added to the case. From the part of the user, our preliminary observations suggest that the primary follow-on question was a relevance goal; e.g. given a briefing about an investigation into corporation "*What might the impact on the economy be if this company is broken up?*" is a knowledge goal attempting to associate the information given to economic impacts in the case of a drastic decision.

4.3 Undergraduate Adviser

The final task is directly related to the example question that we posed in this paper. Users should pose questions to the system as though they are an undergraduate student asking for advice from an adviser or guidance counsellor. Questions in this task can be varied, including issues about courses, plans of study, policy (medical leave, absence, etc.), or even personal issues.

Undergraduate advisers must have a large amount of domain-specific knowledge, from University policy, to course requirements and schedules, to other personal matters. They not only answer questions that undergraduates pose to them, but also respond with detailed, specific instructions for students to carry out. Due to the specificity of this domain, this task allows the capture of more related knowledge goals and allows the use of an institution-specific ontology to simplify concepts.

5. Related Work

This work is most closely related to the work in conversational case-based reasoning, where interactive dialogs are used to specify the case upon which to reason (Aha, McSherry, & Yang, 2005). Goal-driven conversational case-based reasoning is explored in (McSherry, 2005) to identify relevant questions posed to the user in order to target cases. The use of semantic networks and ontological concepts is used in (Gu & Aamodt, 2005) to map questions to features in order to find relations among cases. Finally, the integration of case-based reasoning with task decomposition was explored in (Muñoz-Avila et al., 2001), showing how the generation of plans can result in goal changes.

Systems that leverage large data sets of community organized questions and answers focus on the task of question similarity. Statistical models based on both the question text as well as the answers are discussed in (Jeon, Croft, & Lee, 2005), work that culminates in a statistical model for retrieval of answers based on query-similarity likelihoods (Xue, Jeon, & Croft, 2008). Answer selection or ranking from the retrieval is typically based on heuristic methods, especially reputation-based mechanisms where other users evaluate answers directly (Wang et al., 2013). Original natural language knowledge navigation used case-based reasoning to direct user queries to FAQ files as in (Hammond et al., 1995; Burke et al., 1997b; Burke et al., 1997a). While these systems engage users collaboratively to explore questions not easily answered by search, they don't directly inform a congnitive reasoning system.

On the other end of the spectrum, there has been work designed to translate natural language queries directly to a structured query that can operate on fact based semantic knowledge bases, usually into a SPARQL query as in (Yahya et al., 2012; Unger et al., 2012) and (Berant et al., 2013). These automatic approaches rely heavily on semantic disambiguation and entity resolution, evaluating the query itself against the knowledge base (Zheng et al., 2012). Other approaches include the derivation of proto-queries directly from the knowledge base (Frank et al., 2007), akin to question indexing a structured data set. Retrieval mechanisms like predictive annotation and type coercion have led to the success of Watson and other agent-based knowledge systems (Prager et al., 2006; Kalyanpur et al., 2011). However, these systems are restricted only to the retrieval of facts from a database and usually perform little inference or planning concerning the query.

Finally, the theoretical work is based off of (Lehnert, 1977) who proposes a model of computational story-telling question and answering as well as (Ram, 1991), who provides a foundation for the theory of knowledge goals.

6. Conclusion

Knowledge goals, especially those represented as natural language queries embed goal-related information in three components: structured, semantic *concepts*, questionner-specific *context*, and the relevant *task* or purpose of the knowledge goal which is usually represented as the expected result. Simple knowledge goals are those where the task is computationally feasible - such as a database query or a search. In order to solve a simple knowledge goal, the *concept* and the *context* must be parsed into a representation that can be executed such as a structured database query like SPARQL. Recent research into the semantic parsing of questions has shown that simple information retrieval questions are tractable from large, community built knowledge resources.

Complex knowledge goals, on the other hand, must be decomposed into a hierarchy of simpler knowledge goals, each branch of which contributes information to the final result. In an interactive knowledge-goal reasoning system, users participate alongside the system to solve complex goals by asking questions and responding to proposals of plans for simpler goals that may contribute to the final result. Such systems enhance investigative sessions to generate information, to explain recently acquired knowledge, or to generate solutions to routine questions. When interacting with a system in this way, users may discover that their original knowledge goals change, and that they discover creative results by following goal trajectories of adapting sub-knowledge goal plans.

In this paper we proposed a vision for an interactive case-based reasoning system that could solve knowledge goals similar to conversational case-based systems. This system uses lightweight semantic parsers and a structured knowledge base in coordination with a casebase to represent knowledge goal cases, generalizes them for adaptation, indexes them by relationship for fast retrieval and ranks them to manage and decay stale or ineffectual cases. This knowledge-driven approach primarily relies on the discovery of the concepts in the question combined with a taxonomic classifier for identifying tasks relative to specific user contexts.

To date, we have built a case acquisition system called Kyudo to begin generating a casebase for further exploration. In the future we intend to refine and develop our ideas related to the case representation, indexing of case relations, knowledge-based generalization and case adaptation, and to explore the use of goal trajectories for creative investigative processes.

Acknowledgements

Research on this project was supported in part by the Office of Naval Research under grants #N00014-13-P-1177 and #N00014-15-1-2080. The authors would like to thank the graduate students of the Metacognitive Lab at the University of Maryland for helping with the pilot study and whose discussion was invaluable to the project. Finally, thanks to the insightful and thoughtful reviewers of this paper who provided excellent feedback and direction.

References

Aha, D. W., McSherry, D., & Yang, Q. (2005). Advances in conversational case-based reasoning. *The knowledge engineering review*, *20*, 247–254.

Berant, J., Chou, A., Frostig, R., & Liang, P. (2013). Semantic Parsing on Freebase from Question-Answer Pairs. *EMNLP* (pp. 1533–1544).

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., & Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 1247–1250). ACM.

Burke, R., Hammond, K., Kulyukin, V., Lytinen, S., Tomuro, N., & Schoenberg, S. (1997a). Natural language processing in the FAQ Finder system: Results and prospects. *Working Notes from AAAI Spring Symposium on NLP on the WWW* (pp. 17–26).

Burke, R. D., Hammond, K. J., Kulyukin, V., Lytinen, S. L., Tomuro, N., & Schoenberg, S. (1997b). Question answering from frequently asked question files: Experiences with the FAQ Finder system. *AI magazine*, *18*, 57.

Cox, M. T., & Veloso, M. M. (1998). Goal transformations in continuous planning. *Proceedings of the 1998 AAAI fall symposium on distributed continual planning* (pp. 23–30).

Cox, M. T., & Zhang, C. (2007). Mixed-initiative goal manipulation. *AI Magazine*, *28*, 62.

Frank, A., Krieger, H.-U., Xu, F., Uszkoreit, H., Crysmann, B., Jörg, B., & Schäfer, U. (2007). Question answering from structured knowledge sources. *Journal of Applied Logic*, *5*, 20–48.

Gu, M., & Aamodt, A. (2005). A knowledge-intensive method for conversational CBR. In *Case-Based Reasoning Research and Development*, 296–311. Springer.

Hammond, K., Burke, R., Martin, C., & Lytinen, S. (1995). FAQ Finder: a case-based approach to knowledge navigation. *Artificial Intelligence for Applications, 1995. Proceedings., 11th Conference on* (pp. 80–86). IEEE.

Hayes, C., Cunningham, P., & Smyth, B. (2001). A case-based reasoning view of automated collaborative filtering. In *Case-Based Reasoning Research and Development*, 234–248. Springer.

Jeon, J., Croft, W. B., & Lee, J. H. (2005). Finding similar questions in large question and answer archives. *Proceedings of the 14th ACM international conference on Information and knowledge management* (pp. 84–90). ACM.

Kalyanpur, A., Murdock, J. W., Fan, J., & Welty, C. (2011). Leveraging community-built knowledge for type coercion in question answering. In *The Semantic Web–ISWC 2011*, 144–156. Springer.

Kolodner, J. (1993). *Case-based reasoning*. Morgan Kaufmann.

Lehnert, W. (1977). Human and Computational Question Answering. *Cognitive Science*, *1*, 47–73.

Lopez De Mantaras, R., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M. L., Cox, M. T., Forbus, K., & others (2005). Retrieval, reuse, revision and retention in case-based reasoning. *The Knowledge Engineering Review*, *20*, 215–240.

McSherry, D. (2005). Conversational CBR in multi-agent recommendation. *IJCAI-05 Workshop on Multi-Agent Information Retrieval and Recommender Systems* (pp. 33–40). Citeseer.

Muñoz-Avila, H., Aha, D. W., Nau, D. S., Weber, R., Breslow, L., & Yaman, F. (2001). *SiN: Integrating case-based reasoning with task decomposition* (Technical Report). DTIC Document.

Prager, J., Chu-Carroll, J., Brown, E. W., & Czuba, K. (2006). Question answering by predictive annotation. In *Advances in Open Domain Question Answering*, 307–347. Springer.

Ram, A. (1990). Knowledge goals: A theory of interestingness. *Proceedings of the Twelvth Annual Conference of the Cognitive Science Society* (pp. 206–214). Citeseer.

Ram, A. (1991). A theory of questions and question asking. *Journal of the Learning Sciences*, *1*, 273–318.

Ram, A., & Hunter, L. (1991). A Goal-Based Approach to Intelligent Information Retrieval. *Machine Learning: Proceedings of the Eighth International Conference* (pp. 265–269). Citeseer.

Suchanek, F. M., Kasneci, G., & Weikum, G. (2007). Yago: a core of semantic knowledge. *Proceedings of the 16th international conference on World Wide Web* (pp. 697–706). ACM.

Unger, C., Bühmann, L., Lehmann, J., Ngonga Ngomo, A.-C., Gerber, D., & Cimiano, P. (2012). Template-based question answering over RDF data. *Proceedings of the 21st international conference on World Wide Web* (pp. 639–648). ACM.

Wang, G., Gill, K., Mohanlal, M., Zheng, H., & Zhao, B. Y. (2013). Wisdom in the social crowd: an analysis of quora. *Proceedings of the 22nd international conference on World Wide Web* (pp. 1341–1352). International World Wide Web Conferences Steering Committee.

Xue, X., Jeon, J., & Croft, W. B. (2008). Retrieval models for question and answer archives. *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 475–482). ACM.

Yahya, M., Berberich, K., Elbassuoni, S., Ramanath, M., Tresp, V., & Weikum, G. (2012). Natural language questions for the web of data. *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning* (pp. 379–390). Association for Computational Linguistics.

Zheng, Z., Si, X., Li, F., Chang, E. Y., & Zhu, X. (2012). Entity disambiguation with Freebase. *Proceedings of the The 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology-Volume 01* (pp. 82–89). IEEE Computer Society.

`

Believable Emotion-Influenced Perception:
The Path to Motivated Rebel Agents

Alexandra Coman A-COMAN@ONU.EDU
Kellen Gillespie KELLEN.GILLESPIE@KNEXUSRESEARCH.COM
Hector Munoz-Avila HEM4@LEHIGH.EDU

Abstract

The new Rebel Agent paradigm is meant to help achieve character believability in various forms of interactive storytelling. Rebel agents may refuse a goal or plan that they assess to be in a conflict with their own dynamic motivation model: we call such conflict situations “motivation discrepancies”. We are currently in the process of implementing a Rebel Agent prototype in eBotworks, a cognitive agent framework and simulation platform. In order to identify motivation discrepancies in the environment in a believable way, eBotworks agents need to be able to perceive the environment in ways both influenced by emotion and capable of eliciting emotion, as the relationship between emotion and perception has been theorized in psychology literature to be bidirectional. We explore ways in which such emotion-influenced perception might be achieved in the eBotworks framework for the purposes of implementing believable Rebel Agents.

1. Introduction

Rebel Agents (Coman and Muñoz-Avila, 2014) constitute a new goal-reasoning (Vattam et al., 2013) agent paradigm that is based on three premises:

1. Rebel agents are goal-reasoning agents; that is, they reason on which goals to achieve next.
2. Rebel agents have their own motivations. These motivations can be seen as general guidelines that the agent will follow.
3. Rebel agents may refuse a goal, plan, or subplan (e.g., one suggested by another agent or the user) that they assess to be in a conflict with their own motivations.

The main intended purpose for Rebel Agents is to help achieve character believability in various forms of interactive storytelling. Believable characters, in stories in any medium, act in accordance with personal memories and motivations, which are shaped by events occurring throughout a given narrative. Motivation and memories should evolve as the story progresses, so as to create plausible and engaging character growth. Character believability (Bates, 1994) is considered to be one of the key requirements of a successful narrative, be it interactive or traditional.

Given this intended context, the sort of motivation these agents would be endowed with would be based primarily on subjective aspects, e.g. simulation of feelings and emotions, autobiographical memory and coping mechanisms, etc., rather than more pragmatic ones, as presented by Coddington (2006), e.g. needs pertaining to survival and task efficiency. However, the potential use cases of Rebel Agents can be expanded to include any context that calls for autonomous agents that are endowed with motivation which informs their actions.

An additional concept that we introduced in the context of Rebel Agents is that of "motivation discrepancies" referring to incongruities such as those between a character's changed motivation and the character's previously-assigned goal and/or course of action. When a motivation discrepancy occurs, the Rebel Agent may generate a new goal that safeguards its motivations. For example, if the agent is assigned the task of going to a location, but, along the way, it encounters a friend in distress, it will find that continuing on its way while ignoring its friend is a motivation violation. In such a situation, it generates a new goal (e.g., "help friend").

In our ongoing work, we are developing a conceptual framework for Rebel Agents. To ground our research efforts, we are also in the process of implementing a Rebel Agent prototype in eBotworks, a cognitive agent framework and simulation platform not previously used for character believability, interactive storytelling, and related tasks (Gupta and Gillespie, 2015). Our ideas for work proposed herein have emerged from this process, notably from conceptual and technical challenges that arose while finding ways to integrate believable agents into eBotworks.

For the purpose of motivation discrepancies, the world needs to be perceivable and interpretable not just in terms of (literal) targets, obstacles, and pathways, but also in terms of encounters and incidents potentially causing joy and grief, wonder and regret, etc. Perception, hence, needs to be more nuanced, subjective, and, as it turns out, narrower.

eBotworks bots are instantiated or "born" omniscient and indifferent. By default, they can access information about the entire environment map, but filters can be used to restrict what they perceive. For our purposes, these filters must arguably be informed by mechanisms of human perception.

In eBotworks, perception of objects' properties occurs by getting hold of the object first and then accessing its properties, with all of the properties being equally well accessible at once. However, as shown in our review of related literature, people do not instantly and perfectly perceive scenes in their entirety. Perception occurs in a gradual manner and can be characterized either by global-precedence or by local-precedence. Furthermore, the tendency towards global or local precedence has been found to be influenced by emotion and motivation. How we perceive objects and their properties can also be argued to be a function of the object itself, our perception (which can be impaired or enhanced in various ways), and other external and internal factors, like fog and emotion. Simulating this perceptive style in eBotworks is one of the challenges we are addressing. Peters and O'Sullivan (2002) also make the point that omniscience about the environment in artificial autonomous agents is not a realistic model of human perception, hence it does not lead to believable behavior.

Our intention is for our prototype Rebel Agent to be endowed with motivation based on emotionally-charged autobiographical memories. For example, a bot that reaches a location at which something traumatic happened in the past might undergo a goal change, with subjectivity overtaking the objectively assigned goal. The retrieval of autobiographical memories is to initially occur based on location-specific memory cues. Gomes, Martinho, and Paiva refer to this locative form of memory as "location ecphory" (2011). We note that Gomes et al. use exact physical locations (i.e. map coordinates) as memory cues. While this is easier from a practical standpoint, the authors admit it does not accurately reflect the way location ecphory works in humans. Location coordinates (unless physically perceived with some emotional associations) are unlikely to awaken memories and incite strong emotion. Instead, it is the sights, sounds, smells, tastes, and tactile sensations pertaining to a place that work to achieve this recollection. Thus, if these traits change beyond recognition, the location's function as memory cue is invalidated. While location coordinates are easy for eBotworks bots to retrieve, "visual" perception is perhaps too indiscriminate, while notions like that of "smell" are meaningless. How we handle these issues, and to what extent we need to, are open questions. A possible approach is to endow bots with different sensors for different object properties and to make it possible for the sensors to be impaired by factors both internal and external to the bot (which is generally the case with robots operating in the physical environment). Then, perceptions of different kinds (which may or may not map to actual types of human perception) could be used as varied memory cues. In this work, we focus on the perception aspect of this model, leaving a detailed analysis of the memory aspects of it for future work.

As can be seen, a key characteristic of our endeavor is that we are not attempting to create a "human-like" bot from scratch, but to somewhat "humanize" an already "robot-like" bot, having it grow a modest psyche, and observing the fabric of its being shift, contract, and expand at various levels (perception, memory) as it does so. We do not, however, aim to endow bots with a complex model of cognition. Believable observable behavior remains our aim, and the concept of Rebel Agent remains our primary focus and the context within which we explore perception and memory.

2. Perceptual Differentiation, Emotion, and Motivation in Psychology

Herein, we provide an overview of various theories regarding perception differentiation in psychology literature. We will, for now, focus on work dealing with visual perception.

Perceptual differentiation deals with the steps of the gradual formation of a percept. Navon (1977) distinguishes between three general approaches to perceptual differentiation:

- *"Instantaneous and simultaneous"* perception of *"all visual information at once, no matter how rich it is"*, an approach attributed to a subset of the work falling under Gestalt Psychology, and described by Navon as *"probably too naïve"*.
- *"Feature-by-feature"* perception.
- Gradual perception, which falls somewhere between the above two.

Of the latter, there are multiple variations, corresponding mostly to global-precedence and local-precedence approaches. According to the global-precedence approach, perception begins with global features, with local ones becoming increasingly clear in later stages. According to the local-precedence approach, perception begins from local features.

According to Smith (1924), the two stages of perception differentiation are (1) *"an immediate interpretation of the object as a whole"* and (2) *"an analysis of this vaguely apprehended whole into its component parts"*.

In Dickinson's (1926) view, perceptual differentiation consists of three stages: (1) *"visual pattern"* ("*a thereness, clear in contour but lacking in logical meaning*"), (2) the "*generic object*" stage, and (3) the *"specific object*" stage.

Citing Winston (1973) and Palmer (1975), Navon sees perceptual differentiation as "*proceeding from global structuring towards more and more fine-grained analysis*". As to what makes a feature global, rather than local, he describes a visual scene as a hierarchical network, each node of which corresponds to a subscene. Global scenes are higher up in the hierarchy than local ones, and can be decomposed into local ones.

More recently, it seems to be agreed upon that, while a widespread tendency towards global-first processing is observed, neither global precedence nor local precedence can be established as a general rule applying to all individuals (Zadra and Clore, 2011).

Individuals with certain personality disorders have been hypothesized to be inclined towards either global or local precedence. Yovel, Revelle, and Mineka (2005) state that obsessive-compulsive personality disorder has been connected to *"excessive visual attention to small details"*, as well as "*local interference*": an excessive focus on small details interfering with the processing of global information. The same preference for local processing has been associated with autism spectrum disorders (Frith, 1989).

The tendency towards global or local processing has also been theorized to be culture-specific: certain cultures have been shown to favor local precedence (Davidoff, Fonteneau, and Fagot, 2008).

Our initial intention was for perception to be included in our framework solely as a means to an end: we needed agents to react to perceived objects and scenes "emotionally", so that their motivation may manifest and potentially lead to rebellion. Perception was simply necessary in order to identify motivation discrepancies in the environment.

However, in psychology, the connection between emotion/motivation and perception has been shown to be bidirectional: (1) perception can elicit emotion, and (2) perception is, in its turn, affected by emotion. As a result of these findings, perception now plays a more significant role in our design of the Rebel Agent.

Connections between perception, emotion, and motivation are discussed at length by Zadra and Clore (2011). Their survey covers the effects of emotion and mood on global vs. local perception, attention, and spatial perception.

Percepts of various types can elicit emotional responses (Clore and Ortony, 2008); a picture of a childhood scene can bring about nostalgia, while witnessing a display of violence might elicit fear.

On the other hand, emotion and motivation have been shown to influence perception. Negative emotions, such as stress and sadness, have been argued to favor a local perceptual style, while positive ones, such as happiness, are claimed to make the use of a global perceptual style more likely (Easterbrook, 1959, Gasper and Clore, 2002, Zadra and Clore, 2011). It has also been shown that strong motivation can induce local-first processing (Gable and Harmon-Jones, 2008). In addition to emotion, perception has also been found to be subject to influence by internal factors (e.g. expectations of what the input might be) and external factors (e.g. the dynamic nature of the input).

An additional interesting connection between emotion and the process of perception differentiation has been hypothesized. *"Perception microgenesis"* is defined by Flavell and Draguns (1957) as being "*the sequence of events which are assumed to occur in the temporal period between the presentation of a stimulus and the formation of a single, relatively stabilized cognitive response* [in this case, a percept] *to this stimulus.*" In their study of microgenesis of perception, the authors describe the concept of "*Vorgestalt*", one of the phases of perception (in which the percept becomes increasingly clear and differentiated) according to Undeutsch (1942). Vorgestalt is the intermediary percept corresponding to the stage just before the final percept is formed. It is described by Flavell and Draguns as being "*more undifferentiated internally, more regular, and more simple in form and content than is the final form which is to follow it.*" Interestingly, this phase is also described as being distinctively "*emotionally-charged*" and accompanied by "*decidedly unpleasant feelings of tension and unrest which later subside.*" What is noteworthy about this connection between perception and emotion is that it does not appear to depend upon the perceived scene: it is simply emotion associated with the act of perception itself.

As the relationship between emotion and perception is believed to be bidirectional, an accurate model of the interaction between the two would have not just emotion be elicited by perception, but also perception be influenced by emotion.

3. Perception and Memory in eBotworks

eBotworks (Gupta and Gillespie, 2015) is a software platform for designing and evaluating communicative autonomous systems in simulated environments. "Communicative" autonomous systems are those that can interact with the environment, humans, and other agents in robust and meaningful ways, including the use of natural language.

We chose to use eBotworks as our initial research and implementation tool due to its open and extendable nature. For example, the platform has a flexible embodied agent architecture with swappable simulated robotic components such as chassis, sensors, and motors. This means we can create custom components (e.g., sensors and any other perceptual systems) to better investigate how agents could perceive in ways similar to those found in psychology literature.

Additionally, the platform provides swappable and extendable cognitive components to control these autonomous agents, including motion planners, mappers, and language understanding components. Our extended agents can then potentially be modified to have autobiographical memories. These cognitive components, especially the ones involving language understanding, could also lend themselves well to the interactive story-telling (narration and communication) aspects of our research.

3.1 eBotworks Perception

Perception in eBotworks, by default, is omniscient. Agents that perceive are given an instance of an "ObjectSensor" component through which they see the environment. Given the nature of a perfect simulation, a standard ObjectSensor instantly perceives all of the objects in the world, even those that are out of view. This is an even more extreme (and unrealistic) version of the Gestalt view expressed in the previous section.

For obvious reasons, this is not the ideal perception style for modeling more realistic or human-like systems. To narrow down what objects are perceived, filters of various types can be added to an ObjectSensor. We provide a few example filters below:

- Distance Filter: Do not perceive objects more than *x* meters away.
- Directional Filter: Do not perceive objects directly behind (some degrees) the agent.
- Occlusion Filter: Do not perceive objects the agent does not have line-of-sight with.

These filters, along with any new ones created, can be combined to make a more realistic perception behavior for our agents. For instance, if you combined the three aforementioned filters, you would have a perception system loosely modeling that of a human.

The objects "perceived" by an ObjectSensor are returned to the agent (or more specifically, the cognitive component of the agent requesting the information) with basic simulation-level information. We are handed the I.D. or label of the object, such as "Box1", the type of object it is, such as "Wooden_Box," and the physical location and bounds of the object. Additionally, further properties can be retrieved from an object database (known as the ObjectLibrary) using the object's type as a key. These include exact object "mass", a property not included in typical human visual perception. Table 1 shows some example object information returned by an object sensor "scan" and follow-up queries to the object database.

Table 1. Example perception and select query results for objects in an eBotworks scene

ID	TYPE	LOCATION	DATABASE PROPERTIES
Box1	*Wooden_Box*	*(1, 2, 0.5)*	*{"Color" : Brown , "Mass" : 5kg}*
Box2	*Wooden_Box*	*(4, 2, 0.5)*	*{"Color" : Brown , "Mass" : 5kg}*
Cone1	*Traffic_Cone*	*(-3, 1, 0.5)*	*{"Color" : Orange , "Mass" : 2.5kg}*

This information has been used by cognitive components to do a variety of tasks. For example, the positional and boundary information has been logged and interpolated in order to perform obstacle avoidance by predicting future locations of moving objects (Gupta and Gillespie, 2015).

Figure 1 shows an example indoor environment with an eBot making use of these obstacle avoidance cognitive components.

While the default perception system is, in its current omniscient form, not yet tailored to our use case, the flexibility it provides would allow us to form more emotionally-driven and psychologically-accurate representations for the purposes of the Rebel Agent.

3.2 eBotworks Memory

In eBotworks, memory is a simple concept used primarily by cognitive components allowing for data to be stored between runs for later use. Structurally, it is defined as a general framework for data serialization and deserialization, and supports a variety of predefined data types in addition to custom data types.

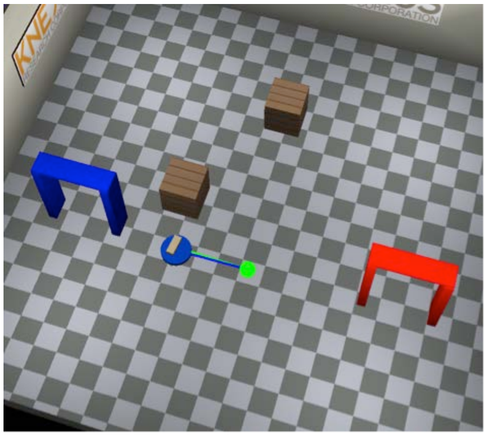


Figure 1. An eBotworks scene with an eBot performing obstacle avoidance.

Memory has also been used in the creation of the aforementioned obstacle avoidance system. More specifically, it has been used to add a very basic ability to learn from previous experiences. An avoidance agent service was created in order to track objects (using the ObjectSensor) and detect if they were in a collision course with the agent.

The service behavior was simple: if an obstacle was about to hit the agent, the agent would self-issue a command to move out of the way. Additionally, this service kept a memory, or history, of the obstacles avoided and in what world they were avoided. In future runs, this memory was loaded back into the agent and could be used in various ways. Notably, if the agent has often needed to avoid objects of type X (e.g. a ball), it could try to distance itself further from these objects to avoid more near-collision scenarios. Figure 2 provides an example of a very simple avoidance history that includes the object avoided, the avoidance "look ahead" setting (how many future time ticks it predicts object locations and detects collisions), and the world in which the avoidance took place.

```
<Memory>
  <Avoidance object_type="Inflated_Ball" look_ahead="30" world_id="indoor_simple"/>
  <Avoidance object_type="Inflated_Ball" look_ahead="30" world_id="indoor_simple"/>
  <Avoidance object_type="Wooden_Box" look_ahead="30" world_id="indoor_simple"/>
</Memory>
```

Figure 2. A simplified example of historical obstacle avoidance memory in XML format

Given this specific memory, an agent introduced to the same world or a similar one may behave differently around objects of type "Inflated_Ball" and try to path further away from them.

While the existing memory framework in eBotworks is not inherently driven by psychological concepts, we believe it is extensible enough to model the autobiographical memories we wish to endow Rebel Agents with, as in the scenarios presented in Section 4.

4. Psychology-Inspired Perception Scenarios in eBotworks

To illustrate the difference between human perception and the current approach to perception in eBotworks, we introduce several psychology-inspired scenarios and present several approaches to making these scenarios possible in eBotworks.

For the scenarios, we assume a simplified psychological model based on several of the above-mentioned theories on perception, emotion, and local/global processing. We make the following assumptions:

- The agent is a Rebel Agent (Coman and Muñoz-Avila, 2014).
- The agent is endowed with an autobiographical memory model in which memories are connected to emotions.
- Default perception is global-first.
- Agents have current "moods" (emotional states), which can be neutral, positive or negative, with the "neutral" mood being the default one.
- Moods can change as a result of perceiving scenes evoking autobiographical memories with emotional associations.
- Mood affects perception in the ways described in the previous section.
- All scenarios take place on the same map.
- In all scenarios, the agent has been assigned a goal that involves movement to a target location on the map. Based on its reaction to scenes perceived on its way to the target, the agent may or may not rebel. When a rebellion threshold is reached, the agent does rebel.
- In all scenarios, the agent perceives two scenes on its way to the target. The perception of the first scene may or may not affect the agent's current mood, which, in turn, may influence how the second scene is perceived.

The scenarios are named based on the emotional state of the agent after perceiving the first scene and on the type of perception that the agent uses for the second scene. We do not discuss details of how the first scene is perceived: it is assumed that this first instance of perception follows the same rules as the perception of the second scene (e.g., had the bot's initial mood not been neutral, it would have affected perception).

1) **Neutral – global:** On the way to its target location, the agent perceives a box. This evokes no emotions, as there are no connections to the box in the autobiographical memory of the agent. Then, the agent perceives the second scene: a traffic-cone-lined driving course, using global-precedence perception. The agent's emotion changes to a slightly-positive one, as it "enjoys" driving through traffic cone-lined driving courses. This does not elicit a goal change.

2) **Positive – global:** On the way to its target location, the agent perceives a box. In the agent's autobiographical memory, the box has positive emotional associations (the agent previously met a friend agent near the box). This changes the agent's mood to a positive one. Positive moods favor global perception, so they do not change the agent's default perception type. The agent perceives the traffic-cone-lined driving course using global-precedence perception. The agent's mood remains positive. This does not elicit a goal change.

3) **Negative – local:** On the way to its target location, the agent perceives a box. In the agent's autobiographical memory, the box has negative emotional associations (perhaps, in the past, the agent did not successfully avoid collision with it and got "hurt"). Therefore, the agent's current mood changes to a negative one. Soon afterwards, the agent perceives the traffic-cone-lined driving course. Due to the agent's mood, local interference occurs, and the agent largely ignores the overall scene, while focusing on the color of the cones (which is similar to that of the box), which reminds it of a sad occurrence from the past, like a collision. This changes the agent's mood to a more intensely negative one, which causes the rebellion threshold to be reached and the agent to "rebel".

The above scenes can be built and simulated in eBotworks with little additional effort as they require little to no new models to be made. The more difficult task will be tying in perception and memory in the way we have outlined in the scenarios into the agents and simulation.

First, we will address perception. All objects, and their properties (through lookups), are perceived. Filters will then be added in order to narrow the agent's view. With a few visibility filters, we can easily simulate local interference.

To handle memory, we can give agents a very basic concept of autobiographical memory based on either objects or scenes. For instance, the object "box" could be tied to an emotional memory label that is an enumerated GOOD, BAD, or NEUTRAL. Additionally, such a label could be given to groups of objects or entire rooms, such as the cone-lined driving course.

Now let's consider the **Negative – local** scenario we have just introduced. If an object elicits a negative emotional response from the agent, we could potentially tie that object's properties (or a subset of them) with that negative response. For instance, let's say the box that creates this negative response is orange. In the following scene, when the agent is in its "bad mood," it may only be capable of seeing the orange cones in the driving course, or maybe even just the color orange. This narrow perception that ignores the rest of the scene could successfully mimic local interference.

While eBotworks bots are not endowed with human-like memory and perception faculties by default, we claim the above techniques will help make a more realistic and emotionally motivated agent.

5. Conclusions and Future Work

We are in the process of implementing Rebel Agents to help achieve character believability in various forms of interactive storytelling. However, in order to completely achieve this believability, the agents' perception and memory need to also function in a believable manner.

We have provided a brief survey of perception differentiation and the relationship between emotion and perception in psychology literature, and used it as the basis for creating and proposing scenarios showcasing emotion-influenced perception for possible future implementation in eBotworks. We have also discussed how the implementation of these scenarios might be achieved with existing components of the framework.

In future work, we would like to explore the memory aspects of eBotworks in a way similar to the way perception was analyzed in this paper. We will also work on building implementations based on the proposed scenarios or similar ones. With psychology-inspired perception and memory in place, we can work to achieve a more believable Rebel Agent.

References

Bates, J. (1994). The role of emotion in believable agents. *Communications of the ACM*, 37(7), 122-125.

Clore G.L., & Ortony A. (2008) Appraisal theories: How cognition shapes affect into emotion. In *Handbook of Emotion 3*, 628–642 New York: Guilford Press..

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).

Coman, A., & Muñoz-Avila, H. (2014). Motivation discrepancies for rebel agents: Towards a framework for case-based goal-driven autonomy for character believability,. *Proceedings of the 22nd International Conference on Case-Based Reasoning (ICCBR) Workshop on Case-based Agents*.

Davidoff, J., Fonteneau, E., & Fagot, J. (2008) Local and global processing: Observations from a remote culture. *Cognition*;108(3):702–709.

Dickinson, C. A. (1926) Experience and visual perception. *Amer. Jour. Psychol*., 37, 330.

Easterbrook J. (1959) The effect of emotion on cue utilization and the organization of behavior. *Psychol. Rev.*, 66(3):183–201.

Flavell, J.H., & Draguns, J. (1957) A microgenetic approach to perception and thought, *Psychological Bulletin*, 54, 198-199.

Frith, U. (1989) *Autism: Explaining the enigma*. Oxford, UK: Blackwell Scientific Publications.

Gable P.A., & Harmon-Jones E. (2008) Approach-motivated positive affect reduces breadth of attention. *Psychol. Sci*., 19:476–482.

Gasper K, & Clore G.L. (2002) Mood and global versus local processing of visual information. *Psychol. Sci.*, 13:34–40

Gomes, P.F., Martinho, C., & Paiva, A. (2011) I’ve been here before! Location and appraisal in memory retrieval. *Proceedings of the Int. Conf. on Autonomous Agents and Multiagent Systems. (AAMAS 2011).*

Gupta K. M., & Gillespie K. (2015) eBotworks: A software platform for developing and evaluating communicative autonomous systems. AUVSI Unmanned Systems, Atlanta, GA.

Navon, D. (1977). Forest before trees: The precedence of global features in visual perception. *Cognitive Psychology*. 9(3):, 353–383.

Palmer, S.E. (1975) Visual perception and world knowledge: Notes on a model of sensory-cognitive interaction. In D. A. Norman , D. E. Rumelhart, and the LNR Research Group, *Explorations in cognition*, San Francisco, CA: Freeman.

Peters, C.,& O’ Sullivan, C. (2002) Synthetic vision and memory for autonomous virtual humans. *Computer Graphics Forum*, 21(4):743–753.

Smith, F. (1924) An experimental investigation of perception. *Brit. Jour. Psychol.*, 6, 321.

Undeutsch, U. (1942) Die Aktualgenese in ihrer allgemeinpsychologischen und ihrer charakterologischen Bedeutung. *Scientia*, 72, 37-42; 95-98.

Vattam, S., Klenk, M., Molineaux, M., & Aha, D.W. (2013). Breadth of approaches to goal reasoning: A research survey. In D.W. Aha, M.T. Cox, & H. Muñoz-Avila (Eds.) Goal

Reasoning: Papers from the ACS Workshop (Technical Report CS-TR-5029). College Park, MD: University of Maryland, Department of Computer Science.

Winston, P.H. (1973) Learning to identify toy block structures. In R.L. Solso (Ed.) *Contemporary issues in cognitive psychology: The Loyola Symposium*, Washington D.C.

Yovel I., Revelle W., & Mineka S. (2005). Who sees trees before forest? The obsessive compulsive style of visual attention. *Psychological Science*, 16, 123-129.

Zadra J.R., & Clore G.L. (2011) Emotion and perception: The role of affective information. *Wiley Interdisciplinary Reviews: Cognitive Science*.

Toward a formal model of planning, action, and interpretation with goal reasoning

Michael T. Cox MICHAEL.COX@WRIGHT.EDU
Wright State Research Institute, Wright State University, Dayton, OH 45435 USA

Abstract

Many algorithms have been presented in artificial intelligence for problem solving and planning. Given a goal, these algorithms search for solutions that achieve a goal state by actions or interactions with an environment. However a major assumption is that goals are given, usually by a user directly as input or as part of the problem definition. Furthermore, once given, the goals do not change. Here we formalize the notion that goal specification and goal change are themselves major parts of the problem-solving process. We apply this model to learning in a goal reasoning context.

1. Introduction

In virtually all AI systems, goal states are predefined and exogenously provided by an external user. But to have a continuing existence in time, agents must be flexible to survive and to continue to be useful. Recent work on goal reasoning (Aha, Cox, & Munoz-Avila, 2013) has started to examine how intelligent agents can reason about and generate their own goals instead of always depending upon a human user directly. Much of this previous work has been situated within the context of the automated planning community and has borrowed some of their formal notations as a theoretical framework. This paper will further extend the standard planning formalism to account for mechanisms of goal-reasoning and goal-driven autonomy. The result integrates notations for planning, action, and interpretation within the scope of goal reasoning.

As an example, consider a package delivery domain (e.g., see Figure 1) having the following characteristics. There is a network of locations connected by roads; and from time to time, vehicles may need to transport objects from one location to another as requests come in. Deliveries may be accomplished, for example, by picking up and delivering packages (e.g., in Figure 1, delivering pallets of bottles to a cola bottling plant). AI planning research has considered similar but simplified domains in which the world is static (i.e., no states change unless the agent performs an action), there is a given fixed goal to achieve (e.g., the delivery of certain specified packages), and the planning problem ends at any world state in which

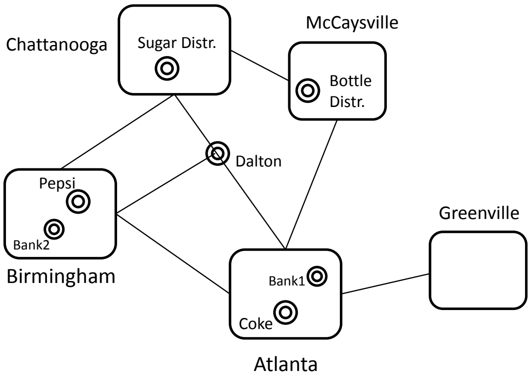


Figure 1. Package delivery domain

the goals are satisfied. In some sense, routine delivery of packages on a route is hardly a problem requiring intelligence. These problems omit any consideration of why those goals need to be achieved and what the agent should do after they have been achieved. In a more realistic model, bottles need to be delivered to the bottling plant when the inventory is low, and it might be necessary for an agent to explain and hence understand this and to generate a new goal when inventory is unexpectedly low.

We begin this paper with a section that provides our formal representations for goal reasoning and shows how it frames the research using a simple blocks world example. The subsequent section examines the formalism within a kind of goal reasoning called goal-driven autonomy. Then we look at learning within this context and apply it to the logistics example above. This section is followed by related research. A brief summary concludes the article.

2. Formal Representations for Goal Reasoning

Much of the research in AI planning has focused on a restricted case called *classical planning*, in which all actions have deterministic effects, and the task is to generate a plan that reaches any of a predefined set of goal states.

2.1 Classical Planning Theory

A *classical planning domain* is typically defined (Ghallab, Nau, & Traverso, 2004) as a finite state-transition system in which each state $s \in S = \{s1, \ldots, sn\}$ is a finite set of ground atoms of a function-free, first-order language $\mathcal{L}$.[1] A *planning operator* is a triple *o* = (head(*o*), pre(*o*), eff(*o*)), where pre(*o*) and eff(*o*) (*o*'s *preconditions* and *effects*, respectively) are sets of *literals* (logical atoms and negated logical atoms), and head(*o*) is a syntactic term of the form *name*(*args*), where *name* is the operator's name and *args* is a list of the variables in pre(*o*) and eff(*o*). Each *action* is a ground instance of a planning operator.

An action $\alpha \in A$ is *executable* in a state *s* if $s \vDash \text{pre}(\alpha)$, in which case the resulting state is $(s - \text{eff}^{-}(\alpha)) \cup \mathit{eff}^{+}(\alpha)$, where $\text{eff}^{+}(\alpha)$ and $\mathit{eff}^{-}(\alpha)$ are the atoms and negated atoms, respectively, in $\text{eff}(\alpha)$. A plan $\pi = \langle \alpha_1, \ldots, \alpha_n \rangle$ is executable in s if each α_i is executable in the state produced by α_{i-1}.

For a classical planning domain, the *state-transition system* is a tuple $\Sigma = (S, A, \gamma)$, where S is the set of all states and A is the set of all actions as above. In addition, *gamma* is a state transition function $\gamma: S \times A \rightarrow S$ that returns the resulting state of an executable action given a current state. Thus from any given state and action, one can infer the subsequent state $\gamma(s, \alpha) \rightarrow s'$ that follows after the action is executed.

A *classical planning problem* is a triple $P = (\Sigma, s_0, g)$, where Σ is a state transition system, s_0 is the initial state, and g (the *goal formula)* is a conjunction of first-order literals. A goal state s_g satisfies a goal if $s_g \vDash g$. A *plan* π represents a (possibly empty) sequence of plan steps (i.e., actions) $\langle \alpha_1 \alpha_2 \ldots \alpha_n \rangle$ that incrementally changes the state of the world. Here we will use a notation that enables indexing of the individual steps or sub-sequences within the plan. In equation (1) we use the subscript g to indicate a plan that achieves a specific goal. A plan is composed of the first action α_1 followed by the rest of the plan $\pi_g[2..n]$.

[1] Most classical planning implementations use the language PDDL (Fox & Long, 2003) based loosely on function-free first order languages.

$$\pi_g[1..n] = \alpha_1 \mid \pi_g[2\,..n] = \langle \alpha_1\alpha_2 \ldots \alpha_n \rangle \tag{1}$$

Now we recursively redefine gamma as mapping either single actions or plans to states. Hence π_g is a solution for P if it is executable in s_0 and $\gamma(s_0, \pi_g) \vDash g$. Recursively from the initial state, execution of the plan results in the goal state (see equation 2).

$$\gamma\big(s_0, \pi_g\big) = \gamma\,\big(\,\gamma(s_0, \alpha_1),\, \pi_g[2..n]\big) \rightarrow s_g \tag{2}$$

2.2 Planning with Nondeterminism

In the research literature on planning, uncertainty about the possible outcomes of actions has been dealt with mainly in two different ways: using Markov Decision Processes (MDPs) (Kaelbling, Littman, & Cassandra, 1995; Dean, Kaelbling, Kirman, & Nicholson, 1995; Boutilier, Dean, & Hanks, 1999) and using model-checking techniques (Cimatti, Roveri, & Traverso, 1998; Aiello *et al.*, 2001; Bertoli, Cimatti, and Traverso, 2006) like those used for program verification. Which approach is best depends on the situation: MDPs are useful when the transition probabilities are important (e.g., when it is desired to maximize expected utility), and model-checking is useful when the transition probabilities are unknown or unimportant (e.g., if it is necessary to achieve a goal regardless of which nondeterministic outcome occurs). In both approaches, an action with multiple possible outcomes is often (though not always) represented as a *nondeterministic classical operator* o = (head(o), pre(o), eff$_1$(o), eff$_2$(o), …,eff$_k$(o)) that has multiple possible sets of effects. When this representation is used with MDPs, each set of effects eff$_i$(o) will have a probability $p_i(o)$, where $p_1 + p_2 + \ldots + p_k = 1$. Instead of a sequential plan π that achieves a goal, MDPs learn a policy π (i.e., a mapping from states to actions) that maximizes expected utility. In this paper we will use the classical definition for clear illustration of the extension to interpretation and goal reasoning.[2]

2.3 Interpretation and Goal Reasoning

Broadly construed, the topic of *goal reasoning* concerns cognitive systems that can self-manage their goals (Vattam, Klenk, Molineaux, & Aha, 2013). Goal reasoning has recently extended the classical formulation by relaxing the assumption that the goal is always given by an external user (Cox, 2007; see also Ghallab, Nau & Traverso, 2014). Although the planning process may start with an exogenous goal, a dynamic environment may present unexpected events with which the system must contend. In response a goal reasoner must be able to generate new goals at execution time as situations warrant.

Formally, the function *beta* (see 3a) returns a (possibly) new goal formula g' given some state s and a current goal g. As such, beta is a state interpretation process that perceives the world with respect to its goals. It is central to goal formulation and goal management.

$$\beta(s, g) \rightarrow g' \tag{3a}$$

2.3.1 Goal transformation

Unlike classical planning models that assume goals to be static and given externally, the goal reasoning model views goals as malleable and subject to change (Cox & Zhang, 2007). For example

[2] Note also that we are ignoring in the classical model the set of exogenous events E that are similar to actions but are outside the control (and possibly the observation) of the reasoning system.

a chess player may start out with the goal to achieve checkmate (g_{ch}). But given a series of unsuccessful opening moves (i.e., $\pi_{g_{ch}}[1..k]$ where $k < n$), the player may change the goal to draw (g_{dr}). We represent this goal transformation as a specialization of (3a) shown in (3b).

$$\beta\big(\gamma(s_0, \pi_{g_{ch}}[1..k]), g_{ch}\big) \rightarrow g_{dr} \tag{3b}$$

Goals can undergo various transformations including priority shifts and goal abandonment (Cox & Veloso, 1998). Over time goals follow arcs or trajectories through a space of goals (Bengfort and Cox, this volume). Most importantly goals can be created or formulated given a problem state.

2.3.2 Goal formulation

From some initial state s_0 and no goal state, an agent formulates a new goal as shown in expression (3c).

$$\beta(s_0, \emptyset) \rightarrow g \tag{3c}$$

In one sense, this can still entail user-provided goals. If the input state is one resulting from a speech act whereby a human requests a goal to be achieved, the function of beta is to interpret the intention of the human and to infer the goal from the utterance. In another sense, however, this significantly differs from the classical formulation of a problem. For goal reasoning in its simplest form, a planning problem can be cast as the tuple $P = (\Sigma, s_0)$. Given a state transition system and an initial state, the goal-reasoning task is to formulate a goal (if a problem indeed exists in the initial state) and create (and execute) a plan to achieve it. Under classical planning (indeed under most planning schemes), the system halts when the goal state is achieved (or even when a plan is simply found). In goal reasoning, an agent can search for new problems once all goals are achieved by interpreting the final goal state s_g. In this case, expression (3c) becomes as in (3d).

$$\beta\big(s_g, \emptyset\big) \rightarrow g' \tag{3d}$$

Of course, and as we will see, goals can potentially be formulated from any current state.

2.4 A Model of Plans, Actions, and Interpretation

A plan to achieve a goal $\beta(s, \emptyset)$ can now be written as $\pi_{\beta(s_0,\emptyset)}$. Using this notation, we combine planning, action (plan execution), and interpretation in equation (4).

$$\gamma\big(s_0, \pi_{\beta(s_0,\emptyset)}\big) = \gamma\ (\ \gamma(s_0, \alpha_1),\ \pi_{\beta(\gamma(s_0,\alpha_1),\beta(s_0,\emptyset))}[2..n]) \tag{4}$$

When beta generates an exogenous initial goal g_1 from the initial state s_0 and simply returns the input goal from all other states (i.e., $g' = g$ in 3a), the formalization reduces to classical planning with a user-given goal. That is, equation (4) is equivalent to (2) because (3a) represents a trivial boundary case. However when goals change (or new ones are added), plans may need to change as well.

The problem with the current formalization then is that, in the recursive right-hand side of (4) above, the plan is not static as defined in (1). That is, it is not necessarily of size $n - 1$. Instead, because the goal may change due to beta, the goal reasoner may need to re-plan and alter the length and composition of the remainder of the plan.[3] To cover this contingency, we define a (re)planning function *phi* that takes as input a state, goal, and current plan as in (5).

[3] I thank Don Perlis for pointing out this anomaly.

$$\varphi\big(s, g', \pi_g[1..n]\big) \rightarrow \pi_{g'}[1..m] \qquad (5)$$

Note that in the general case g' may or may not be equal to g. Inserting the re-planning function into (4), we obtain equation (6) below that resolves the anomaly indicated above.

$$\gamma\big(s_0, \pi_{\beta(s_0,\emptyset)}\big) = \gamma\ (\ \gamma(s_0, \alpha_1), \varphi\big(\gamma(s_0, \alpha_1), \beta\big(\gamma(s_0, \alpha_1), \beta(s_0, \emptyset)\big), \pi_{\beta(s_0,\emptyset)}[2..n]\big)\big) \qquad (6)$$

Given phi, the formalism is general across different variations of goal reasoning and (re)planning.

$$\gamma\ \big(s_0,\ \overbrace{\varphi(s_0, \underbrace{\beta(s_0,\emptyset)}_{g_1}, \emptyset)}^{\pi_{g_1}}\big) = \gamma\ (\underbrace{\gamma(s_0,\alpha_1)}_{s_1},\ \overbrace{\varphi\big(\underbrace{\gamma(s_0,\alpha_1)}_{s_1}, \underbrace{\beta\big(\underbrace{\gamma(s_0,\alpha_1)}_{s_1}, \underbrace{\beta(s_0,\emptyset)}_{g_1}\big)}_{g_2}, \underbrace{\underbrace{\varphi(s_0,\beta(s_0,\emptyset),\emptyset)}_{\pi_{g_1}}[2..n]}_{\pi_{g_1}[2..n]}\big)}^{\pi_{g_2}})$$

2.5 A Blocks World Example

Many details are buried within the beta interpretation function. Just consider a situation where goals are given to the agent by a human user in natural language. In this case, beta does not just input a goal in first-order predicate form. Instead it must translate an utterance or the results of a speech act into a predicate representation. That it, it must infer the intent of the user from the current state of the world and what was said. Indeed when a user both states imperatives and asks questions, the illocutionary act intended is often a request for the agent to assume a goal.[4] Consider the utterance "Put block-B on block-C." Although spoken as an action, the intent is actually for the agent to assume the goal on the behalf of the speaker to achieve the state of the block B on top of C. Although we will not discuss the specific details here, Figure 2 shows how the formalism organizes the context of a dialog between human and agent so that intent of the last utterance "Add one more" is possible.

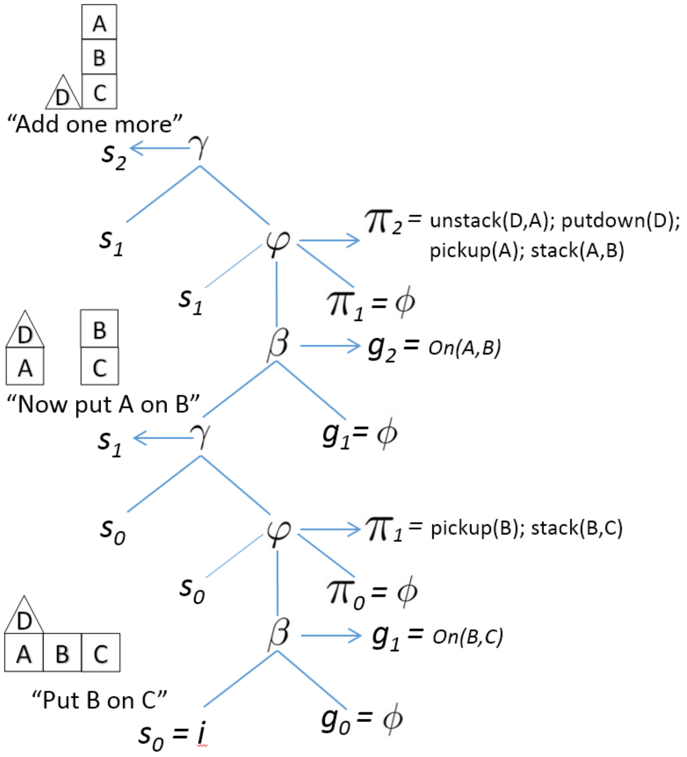


Figure 2. Partial context for blocks example (read from the bottom up)

[4] For questions, the intended goal is a *knowledge goal* (Bengfort & Cox, in press; Ram, 1990).

Figure 2 hides many details as well. Plan π_1 has two actions that are executed sequentially, not concurrently in one invocation of gamma as the figure suggests. A more realistic depiction of the plan execution is shown in Figure 3. Here we see that the execution of the first action results in an intermediate state. From that state no new goal is formulated, and subsequently no new re-planning is performed. The goal is achieved in the next state s_2 and beta evaluates the goal achievement, returning the null goal.

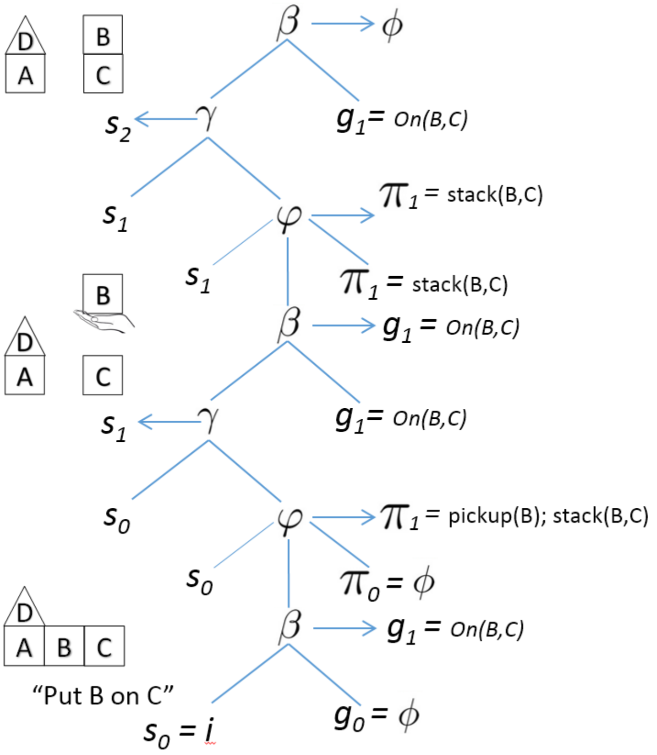


Figure 3. The execution of π_1. Here we drop the user imperative in Figure 2 to put A on B.

3. Goal-Driven Autonomy

Goal-driven autonomy (GDA) (Aha, et al., 2010; Cox, 2007; 2013; Klenk, Molineaux, Aha, 2013; Munoz-Avila, et al., 2010) is a kind of goal reasoning that focuses on goal formulation and explanation. In the GDA framework, the current goal g_c is a distinguished member of the set of all goals the agent currently desires (equation 7), but differs in that the agent has selected it for planning. Also in the GDA framework, the goal reasoner produces not only a plan, π, but also a set of *expectations*, $\mathrm{X} = \{x1, \ldots, xk\}$ that represent constraints on the states resulting from each of the i=1..k actions α_i. When any currently observed state, s_c, diverges from the system's expected state, $s_e = x_c \in \mathrm{X}$, a *discrepancy*, $d \in D$, is said to occur.

$$\mathcal{G} = \{ g_1, g_2, \ldots g_c, \ldots g_n\} \tag{7}$$

Given a discrepancy, a new goal may be warranted, so the system will retrieve an *abductive explanation* χ of d to determine a cause in the context of s_c. The explanation is performed to resolve the discrepancy. Finally given d, χ, and s_c, goal generation seeks to determine a new current goal, g_c, that seeks to remove the discrepancy d by either changing the state s_c to s_e or by learning a new expectation that justifies s_c. As such we now define a GDA planning problem as a 5-tuple in (8).

$$P_{gda} = (\Sigma, s_c, g_c, s_e, \mathcal{G}) \tag{8}$$

In the case where plan execution outcomes equal expectations (i.e., $s_c = s_e = s_0$) and thus no explanation is necessary (i.e., $\chi = \emptyset$) and the current goal is given (i.e., $g_c = g$), P_{gda} devolves into a classical planning problem P.

According to Cox (2007), the explanation[5] $\chi: \delta \rightarrow ... \rightarrow s_c$ contains a *salient antecedent* δ that represents the root cause of the problem signaled by the discrepancy. The goal then is to remove the cause of the problem, hence $g_c = \neg\delta$. *Goal insertion* is thus a search for an explanation that operationalizes the problem in terms of root causes. See Table 1 for the algorithm and Cox (2013) for a cognitive architecture that instantiates the algorithm.

Table 1: GDA algorithm for adding new goal to current goal set

```
function goal-insertion ($s_c$ : STATE; $\alpha_i$ : OPERATOR; $\mathcal{G}$: SET): SET;
    $s_e \leftarrow expects(\alpha_i)$
    Detect discrepancy, $d: s_c \neq s_e$
    If $\nexists d$, then $return(\mathcal{G})$
    Find an explanation, $\chi: \delta \rightarrow \cdots \rightarrow s'$,
    such that $covers(s', s_c)$
    $\sigma \leftarrow unify(s', s_c)$
    $subst(\sigma, precond(\chi))$
    if $satisfied(precond(\chi))$
    then $\mathcal{G} \leftarrow \mathcal{G} \cup \neg subst(\sigma, \delta)$
    $return(\mathcal{G})$
```

So consider again the blocks world example. When told to put A on B, the system easily creates a plan to clear A, pick it up, and place it on B. During the planning it had to solve the subgoal of having A clear before picking up the block. But B is already clear and it expects it to stay that way.

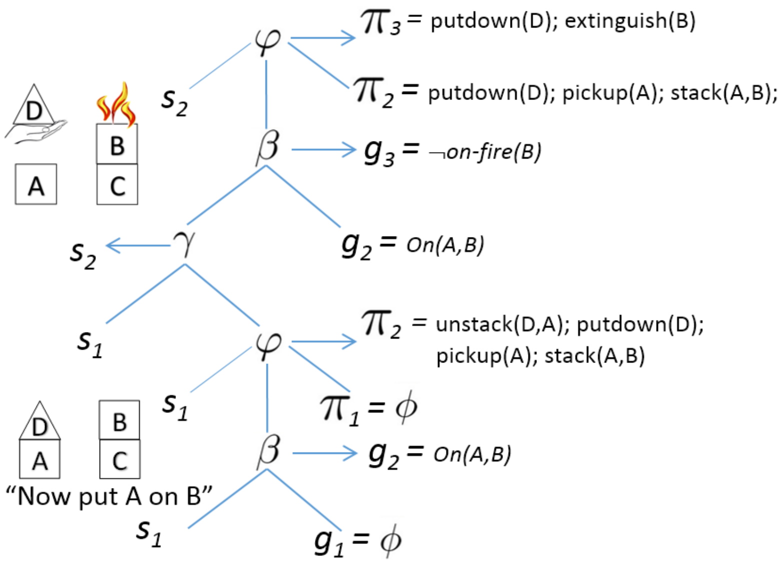


Figure 4. Explanation-based goal formulation

[5] The explanation is actually a graph $\chi = (V, E)$ with $\delta \in V$ an element of the source nodes and $s_c \in V$ a distinguished element of the sink nodes. See also Cox (2011).

Now suddenly assume that block B catches on fire and is no longer clear (see Figure 4 and Paisner, Cox, Maynord & Perlis, 2014 for an extended domain description).

Figure 4 shows state s_2 after the agent unstacked pyramid D from block A during the execution of plan π_2. The agent is still holding D, and B is on fire and hence no longer clear. Here the expectation s_e is clear(B), whereas the current observed state s_c is ¬clear(B). A very simple explanation is that a block being on fire causes it not to be clear.

$$\chi: on\_fire(y) \rightarrow \neg\, clear(y)$$

Given the substitution set $\sigma = \{y \mapsto B\}$, the goal $\neg subst(\sigma, \delta)$ becomes the achievement of $\neg on\_fire(B)$ and is added to the goal set $\mathcal{G}$=$\{g_2\}$. Planning then can create a sequence of actions to put out the fire before continuing. The planning function represented by phi needs to be intelligent enough to infer that the putdown action is duplicated in both plans π_2 and π_3. However for beta to be equally intelligent, it should have been informed of the current plan. Therefore, we redefine (3a) as follows.

$$\beta(s, g, \pi) \rightarrow g' \qquad \text{(3a')}$$

In (3a'), beta can use the current plan for expectations. Indeed, we stated in the example above that s_e, the expectation that B will remain clear, occurred during planning for π_2. With such input, beta can also generate a goal to retry a failed plan step during execution (e.g., vacuum cleaner did not remove all the dirt during a particular sweep). Furthermore, goal reasoning can be clear when trying to determine under unexpected and dynamic contexts (e.g., sudden resource reductions) whether plan adaptation or goal adaptation is the most rational choice. In this way too, planning is not only informed by goals from interpretation, interpretation is informed by planning.

4. GDA and the Learning of Goals

Beta is less about planning per se than about understanding the larger context within which a planning agent finds itself. If planning and interpretation are to be successful, learning must also be taken into consideration. Technically learning in this context is about inducing a set of goal states and applicability conditions from observed behavior of an execution system; and the main role for planning is to enable a GDA agent to infer which of the possible goal states are achievable within the expected costs and rewards of achieving those goals.

In the simplest sense, goal formulation can be cast as a novel classification task. An agent is given a state transition system, ∑, and a current state, s_c, and it must infer the current goal, g_c. Using a naïve supervised learning approach, one can present the system with many positive and negative examples of the tuple (s,g). From these examples, a learning algorithm can then learn a mapping in the form of a decision tree with goals at the leaves (see Maynord, Cox, Paisner, & Perlis, 2013 for an implementation of this approach). However the time it takes to learn such relations and the effort it takes to prepare suitable examples may not be available. Instead another approach is to exploit failure in performance by explaining why expectations do not apply in new situations and then using these explanations to learn appropriate goal orientations (i.e., to learn G, the set of useful goal states).

But we do not limit ourselves to goal generation alone, because if so the human burden of providing an external goal is shifted to the human specification of the complete range of goal representations. Instead if it is to be effective, the agent should also learn the classes of states in the world that constitute useful goals. This general approach represents a new conception of autonomous behavior.

Reconsider the planning domain from the introduction section and Figure 1. If a GDA-based agent interacts with other agents performing actions, it may observe a sugar distributor make a delivery to an Atlanta cola plant and then watch the manufacturer produce Coca-Cola at that location. The Coca-Cola is then delivered to Greenville where an advertising campaign has concluded. Sales then are observed to produce profits. Given these exogenous-event observations from the execution of π_{Coke} and π_{SDistr}, the GDA-agent can understand the causal relations by performing a goal regression (Veloso, 1994; Waldinger, 1981) upon the goal-subgoal graph entailed by $\sum$ and the observations (see Figure 5). Each node in the graph is an instantiated action-operator whose preconditions must be satisfied before the action can be executed. A tree of conjuncts can then be extracted to give the causal explanation structure shown in Figure 6. For example the state at(Cola-5, Coke) exists because the conjunct has(Coke, $) ∧ at(Bottles-2, Coke) ∧ at(Sugar-1, Coke) has been satisfied. If any term is unsatisfied, then the explanation structure collapses.

Now a subsequent series of observations may violate the expectations present in this causal structure. For example the system may expect Coke to have further profits in Greenville, but Coke might lack the sales. Sugar may be delivered and promotions may occur, but without bottles, no cola is produced and hence no deliveries arrive for sale. In this case, the goal insertion algorithm (Table 1) would detect the discrepancy between the expectation has(Coke, sales) and the observation ¬has(Coke, sales). An explanation process would then use the prior explanation (from Figure 6) to derive a relational sequence responsible for the impasse. The resulting causal chain is shown in Figure 7.

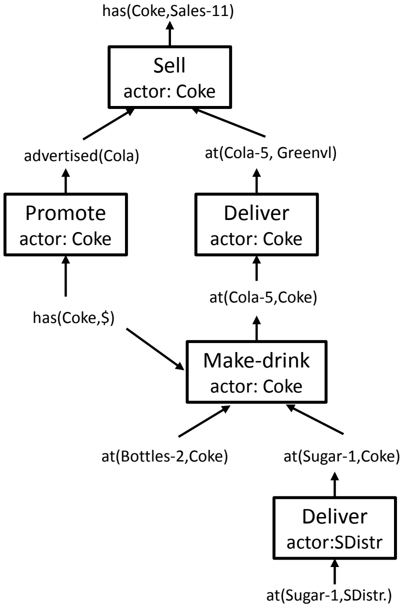


Figure 5. Goal-subgoal graph

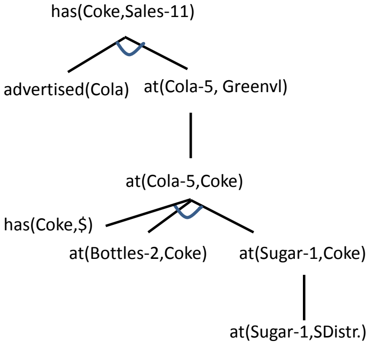


Figure 6. Causal explanation structure

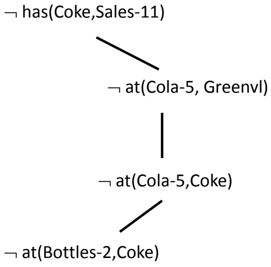


Figure 7. Failure causal chain

The goal insertion process would use the extracted failure chain to generate a goal and to add it to the agent's current set of desired goals $\mathcal{G}$. In Figure 7 the salient causal antecedent δ of the explanation is ¬at(Bottles-2, Coke), that is, the bottles are not at the plant (and that is ultimately why Coke lost profits). The new goal, g_c, is therefore to obtain the state of bottles being at this location (equation 9).

$$g_c = \neg\,(\neg \text{at(Bottles-2, Coke)}) = \text{at(Bottles-2, Coke)} \qquad (9)$$

Given a cooperative agent domain where the GDA agent represents a bottle distributor (i.e., BDistr) that coordinates with the sugar distributor and the cola manufacturers, the GDA agent can solve the new goal by generating a plan to perform the delivery method. That is, it will load the bottles at the McCaysville location, drive the load to Atlanta, go to Coke, and unload the shipment there. However a number of technical problems remain. Two in particular that we will discuss now concern issues of goal generalization and of goal anticipation.

4.1 Goal Generalization

First the goal above must be *generalized*. It is not useful to learn an overly specific goal of getting a particular shipment to a destination. Rather the agent must generalize the shipment to any available instance of type bottles. Furthermore, and this is a bit more subtle, the agent must eventually generalize the destination to any cola manufacturer in the domain (and not to just any location). That is Pepsi as well as Coke require bottle stock. By doing so, this simple explanation-based generalization rules out many irrelevant potential goal states such as at(bottles,bank) or at($,Dalton). Here we intend to induce the goal expression g_c in equation (10) and to add $g_c \in G$.

$$g_c = \text{at(b,m)} \wedge \text{bottles(b)} \wedge \text{cola-manufacturer(m)} \qquad (10)$$

4.2 Goal Anticipation

Second the agent needs to *anticipate* the conditions under which the causal chain in Figure 7 might re-occur and plan to prevent it in the future. It should not wait until Coke sales fail each time. Thus in this example the GDA agent learns to generate the goal, g_c, when the bottles are low in inventory, not waiting until bottles run out and sales plummet. The agent learns a characterization of the state, s_c, in tuples (s_c, g_c), that is a rule $g_c \leftarrow s'$. This is the problem of learning goal-selection criteria (Powell, Molineaux, & Aha, 2011).

Unlike Powell and colleagues who use human guidance to acquire goal-selection knowledge, we suggest that an agent can learn this through unsupervised means. For example the state of inventory at manufacturers may include the predicates surplus, low, very-low, and out-of. The GDA agent should learn that bottles should be delivered when the supplies are low or very low, rather than waiting until they are out or trying to deliver when a surplus exists. In another instance of failure driven learning, the GDA agent would learn to rule out the expression surplus(m,b) ∧ bottles(b) ∧ cola-manufacturer(m) after it attempts to deliver bottles to a manufacturer having a bottle surplus. In this case we assume that the unload operation would not be given permission at the destination. Essentially the learned rule asserts the goal g_c when bottles at the manufacturer are low. This is what we meant when we stated previously that the agent should be able to explain that bottles need to be at the bottling plant *because the inventory is low*.

4.3 Deciding Which Goals Are Worth Generating

Just because a goal state s_g would be a desirable state to be in, there are several situations in which g may not be a useful goal for a goal reasoning agent to formulate. For example, s_g may be impossible to achieve; or in a nondeterministic domain, it may be impossible to guarantee that s_g will always (or usually) be achieved. Even if s_g is achievable, there may be another goal state $s_{g\prime}$ that is nearly as desirable as s_g and can be achieved at much lower cost than s_g. In such a situation it may be better for GDA to insert g' into its agenda rather than g. In general it is an open research question as to the best way to learn which goals are worth formulating and which to commit to first.

5. Related Research

An alternative formal model (Roberts et al., 2014; in press) treats goal reasoning as *goal refinement*. Using an extension of the plan-refinement model of planning, Roberts and colleagues model goal reasoning as refinement search over a *goal memory M*, a set of *goal transition operators R*, and a transition function *delta* that restricts the applicable operators from R to those provided by a fundamental *goal lifecycle*. Unlike the formalism here that represents much of the goal reasoning process with the single function beta, Roberts proposes a detailed lifecycle consisting of goal formulation, goal selection, goal expansion, goal commitment, goal dispatching, goal monitoring, goal evaluation, goal repair, and goal deferment. Thus many of the differential functionalities in beta are distinct and explicit in the goal reasoning cycle. However the model here tries to distinguish between the planning and action side of reasoning (i.e., phi and gamma) and the interpretation and evaluation components inherent in goal reasoning (i.e., beta). We can roughly note, however, that phi relates to Robert's goal expansion and repair, whereas beta relates to goal formulation, monitoring, and deferment. Missing in my current analysis is the processes of goal dispatching, commitment, and evaluation.

Additionally Roberts proposes a more complex goal structure. A *goal node* includes not only the desired state but also super-ordinate and subordinate goal linkages, goal constraints, quality metrics, and pointers to the current plan associated with the node. We have been more circumspect regarding the syntactic structure of a goal. In section 2.1 on classical planning, we defined the goal formula to be a conjunction of first-order literals. More generally we would also allow universally quantified predicate conjuncts as in Cox & Veloso (1998; Veloso, Pollack & Cox, 1998). Thus in a package delivery domain we allow $\forall p \mid (package\ p) \wedge \exists\ d|(destination\ p\ d) \wedge (location\ p\ d)$ as a goal to deliver all packages to their destination. Under the scope of this goal expression, new goals $(location\ p\ d)$ may arise during planning or plan execution time as additional packages arrive at the warehouse for delivery. See also Talamadupula, Benton, Kambhampati, Schermerhorn, & Scheutz (2010) for their concept of *open world quantified goals*. But overall, much exists in common with the representations here and with Roberts including an association with learning (see Roberts & Aha, this volume).

The two problems of goal generalization and goal anticipation make the learning task as we discuss it here considerably different from related research. Planning research is concerned with generating an action sequence to carry out a goal that is given to a system by an external user (see survey in Ghallab, Nau, & Traverso, 2004). In the larger scope of integrated cognitive systems, we intend to better understand autonomy as the capacities (1) to recognize novel problems and (2) to independently form the desire to remove these problems. Goal anticipation and goal generalization are instrumental processes associated respectively with these two aspects of autonomy. However unlike bottom-up, data-driven learning approaches, the top-down explanation of anomalies is a central pivot in our learning approach (Cox, 2011).

The explanation-based approach we take to learning should not be confused with the older, mainly deductive approach called explanation-based learning (DeJong & Mooney, 1986; Mitchell, Keller, & Kedar-Cabelli, 1986). We use explanation techniques (Cox, 2011; Cox & Ram, 1999; Roth-Berghofer, Tintarev, & Leake, 2011) that depart significantly from earlier work, being more abductive, knowledge rich, and case-based. The idea is to retrieve a graph structure called an explanation pattern given a discrepancy and a context, to instantiate it and to bind it to the discrepancy, and then to adapt it to the context. This also departs from other GDA systems (e.g., ARTUE, Molineaux, Klenk & Aha, 2010) which use abductive causal inference in the form of an assumption-based truth maintenance system or ATMS (de Kleer, 1986) for explanation.

As mentioned previously, goal formulation or generation has been an instrumental task in research on goal reasoning and goal-driven autonomy. In the ARTUE system, goal formulation occurs as a response to discrepancy detection and is based upon knowledge structures called principles (Klenk, Molineaux, & Aha, 2013). Other approaches rely upon links between specific states and a priori goal candidates (Dill & Papp, 2005) or triggering schemas based on the current state (Talamadupula, Benton, Schermerhorn, Kambhampati, & Scheutz, 2009). In the EISBot interactive game-playing system, association rules called goal formulation behaviors link discrepancy-explanation types to goal types (Weber, Mateas & Jhala, 2010). Here we discussed the use of the negation of explanation root causes to generate new goals. On the other hand, the learning of goal state information and its generalization is novel. The closest research is that of learning goal selection knowledge (Jaidee, Munoz-Avila, & Aha, 2011; Powell, Molineaux, & Aha, 2011).

6. Conclusion

This research attempts to reconcile some of the existing research on goal reasoning with theoretical frameworks in the automated planning and the intelligent agents communities. The resulting goal-reasoning formalism augments notions of planning and plan execution with formal models of re-planning and both goal generation and goal change. In situations where goals are given at initialize time and do not change throughout the process and where plans execute as expected, the model devolves into a classical planning formalism. We have extended the model to a kind of goal reasoning called goal-driven autonomy, and we have applied the model to the explanatory learning of those goals worth pursuing.

Acknowledgements

This material is based upon work supported by ARO grant W911NF-12-1-0471 and by ONR grants N00014-15-1-2080, N00014-12-1-0430, and N00014-12-1-0172. I thank the anonymous reviewers for their insights and comments. I also wish to acknowledge the previous input of David Aha, Michael Maynord, and Mark Wilson on an earlier formulation of these ideas during a poster presentation (Maynord, Aha, Wilson & Cox, 2013).

References

Aha, D. W., Cox, M. T., & Munoz-Avila, H. (Eds.) (2013). *Goal Reasoning: Papers from the ACS workshop* (Tech. Rep. No. CS-TR-5029). College Park, MD: University of Mary-land, Department of Computer Science.

Aha, D. W., Klenk, M., Munoz-Avila, H., Ram, A., & Shapiro, D. (Eds.) (2010). *Goal-driven autonomy: Notes from the AAAI workshop*. Menlo Park, CA: AAAI Press.

Aiello, L. C., Cesta, A., Giunchiglia, E., Pistore, M., & Traverso, P. (2001). Planning and verification techniques for the high level programming and monitoring of autonomous robotic devices. In *European Space Agency Workshop on On-Board Autonomy*, Noordwijk, Netherlands. ESA.

Bengfort, B., & Cox, M. T. (this volume). Interactive reasoning to solve knowledge goals. To appear in *Proceedings of the 2015 Annual Conference on Advances in Cognitive Systems: Workshop on Goal Reasoning* (IRIM Tech Report). Atlanta: Georgia Institute of Technology.

Bertoli, P., Cimatti, A., Roveri, M., & Traverso, P. (2006). Strong planning under partial observability. *Artificial Intelligence*, *170*(4): 337–384.

Boutilier, C., Dean, T. L., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research (JAIR)*, *11*:1–94.

Cimatti, A., Roveri, M., & Traverso, P. (1998). Strong planning in non-deterministic domains via model checking. In R. Simmons, M. Veloso, & S. Smith (Eds.), *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems* (pp. 36–43). Menlo Park, CA: AAAI Press.

Cox, M. T. (2013). Question-based problem recognition and goal-driven autonomy. *Goal Reasoning: Papers from the ACS workshop* (pp. 10-25). (Tech. Rep. No. CS-TR-5029). College Park, MD: Univ. Maryland, CS Dept.

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. (2007). Perpetual self-aware cognitive agents. *AI Magazine 28*(1), 32-45.

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 / The MIT Press.

Cox, M. T., & Zhang, C. (2007). Mixed-initiative goal manipulation. *AI Magazine 28*(2), 62-73.

Dean, T., Kaelbling, L. P., Kirman, J., Nicholson, A. E. (1995). Planning under time constraints in stochastic domains. *Artificial Intelligence 76*(1-2): 35-74.

DeJong, G., & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, *1*(2), 145-176.

de Kleer, J. (1986). An assumption-based TMS. *Artificial Intelligence, 28*(2), 127-162.

Dill, K., & Papp, D. (2005). A goal-based architecture for opposing player AI. In *Proceedings of the First Artificial Intelligence for Interactive Digital Entertainment Conference* (pp. 33-38). Menlo Park, CA: AAAI Press.

Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*.

Ghallab, M., Nau, D., & Traverso, P. (2014). The actor's view of automated planning and acting: A position paper. *Artificial Intelligence 208*, 1–17.

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

Jaidee, U., Munoz-Avila, H., & Aha, D.W. (2011). Integrated learning for goal-driven autonomy. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*. Barcelona, Spain.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1995). Partially observable Markov decision processes for artificial intelligence. In *Proceedings of Reasoning with Uncertainty in Robotics* (pp. 146-163).

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

Maynord, M., Aha, D. W., Wilson, M. & Cox, M. T. (2013, December). *On the definition and desirability of goal reasoning*. Invited poster, 2013 Annual Conference on Advances in Cognitive Systems: Workshop on goal reasoning. University of Maryland, Baltimore, MD.

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). Technical Report FS-13-03. Menlo Park, CA: AAAI Press.

Mitchell, T. M., Keller, R., & Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view, *Machine Learning, 1*(1), 47-80.

Molineaux, M., Klenk, M., & Aha, D. (2010). Goal-driven autonomy in a navy training simulation. In *Proceedings of Twenty-Fourth AAAI Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.

Munoz-Avila, H., Aha, D.W., Jaidee, U., Klenk, M., & Molineaux, M. (2010). Applying goal driven autonomy to a team shooter game. *Proceedings of the Twenty-Third Florida Artificial Intelligence Research Society Conference* (pp. 465-470). Menlo Park, CA: AAAI Press.

Paisner, M., Cox, M. T., Maynord, M., Perlis, D. (2014). Goal-driven autonomy for cognitive systems. In *Proceedings of the 36th Annual Conference of the Cognitive Science Society* (pp. 2085–2090). Austin, TX: Cognitive Science Society.

Powell, J., Molineaux, M., & Aha, D. W. (2011). Active and interactive discovery of goal selection knowledge. In *Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference*.

Ram, A. (1990). Knowledge goals: A theory of interestingness. *Proceedings of the 12th Annual Conference of the Cognitive Science Society* (pp. 206–214).

Roberts, M., & Aha, D. W. (this volume). Cleaning efficiently: A case study in modeling goal reasoning and learning. To appear in *Proceedings of the 2015 Annual Conference on Advances in Cognitive Systems: Workshop on Goal Reasoning* (IRIM Tech Report). Atlanta: Georgia Institute of Technology.

Roberts, M., Vattam, S., Alford, R., Auslander, B., Apker, T., Johnson, B., & Aha, D. W. (in press). Goal reasoning to coordinate robotic teams for disaster relief. To appear in *Proceedings of ICAPS-15 PlanRob Workshop.*

Roberts, M., Vattam, S., Alford, R., Auslander, B., Karneeb, J., Molineaux, M., Apker, T., Wilson, M., McMahon, J., & Aha, D. W. (2014). Iterative goal refinement for robotics. In *Proceedings of ICAPS 2014*.

Roth-Berghofer, T., Tintarev, N., & Leake, D. B. (Eds.) (2011). *Proceedings of the IJCAI-11 Workshop on Explanation-aware Computing ExaCt 2011*, July 2011.

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).

Talamadupula, K., Benton, J., Schermerhorn, P., Kambhampati, S., & Scheutz, M. (2009). Integrating a closed world planner with an open world robot: A case study. In M. Likhachev, B. Marthi, C McGann, & D. E. Smith (Eds.) *Bridging the Gap between Action and Motion Planning: Papers from the ICAPS Workshop*. Thessaloniki, Greece.

Vattam, S., Klenk, M., Molineaux, M., & Aha, D. (2013). Breadth of approaches to goal reasoning: A research survey. In D. W. Aha, M. T. Cox, & H. Munoz-Avila (Eds.), *Goal Reasoning: Papers from the ACS workshop* (pp. 111-126). Tech. Rep. No. CS-TR-5029. College Park, MD: University of Maryland, Department of Computer Science.

Veloso, M. (1994). *Planning and learning by analogical reasoning*. Berlin: Springer.

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.

Waldinger, R., (1981). Achieving several goals simultaneously. In N. J. Nilsson & B. Webber (Eds.), *Readings in Artificial Intelligence* (pp. 250-271). Palo Alto, CA: Tioga.

Weber, B. G., Mateas, M., & Jhala, A. (2010). Applying goal-driven autonomy to StarCraft. In *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment.* Menlo Park, CA: AAAI Press.

Goal Reasoning for an Autonomous Squad Member

Kellen Gillespie KELLEN.GILLESPIE@KNEXUSRESEARCH.COM
Matthew Molineaux MATTHEW.MOLINEAUX@KNEXUSRESEARCH.COM
Michael W. Floyd MICHAEL.FLOYD@KNEXUSRESEARCH.COM
Knexus Research Corporation, Springfield, VA 22153 USA

Swaroop S. Vattam SWAROOP.VATTAM.CTR.IN@NRL.NAVY.MIL
NRC Research Associate; Naval Research Laboratory (Code 5514), Washington, DC 20375 USA

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

Abstract

Autonomous agents are beginning to play larger roles within team-oriented tasks and missions in various domains. Many reasoning agents are designed to perform in or assist with a single, static goal or task within an environment. Our aim is to design and develop an autonomous squad member that assists a squad with conducting a surveillance mission by identifying and reacting dynamically to changing situations and goals. We present a goal reasoning system for this agent that integrates natural language processing, explanation generation, and plan recognition components to recognize these changing situations and the squad's responses to them. Our system uses goal selection and plan generation to respond to such changes. We describe the architecture we use to integrate these components and provide a case study that demonstrates how they work together to make a robust and adaptive autonomous agent.

1. Introduction

Robots are increasingly being added to teams to improve their ability to accomplish specific tasks and missions (ARL, 2011; Kott et al., 2010). Most instances involve a single task or objective for the agent or team to complete that is static and uninterrupted. This may suffice for simple tasks and environments, but in more realistic situations the team's goals are dynamic and can be interrupted or changed at any time.

Our objective is to design and develop an autonomous squad member (ASM) that can accompany and assist a squad of soldiers on military missions. These missions can be, and often are, conducted in hazardous and hostile areas. In such environments, unexpected events (e.g., encountering enemy fire, explosives, and other obstacles) can occur at any moment without warning. Therefore, the ASM must identify and react to highly dynamic and unpredictable environments to coordinate with its team. Furthermore, as its human teammates will typically react quickly and instinctively to such changes, the ASM must also recognize rapid changes in team behaviors.

Meeting these needs requires the ASM to have multiple reasoning capabilities. First, it must recognize any relevant situational and environmental changes that have occurred. We address this need by bolstering standard observational capabilities (e.g., static environmental and map information, a priori mission knowledge) with natural language understanding (NLU). Next, the agent must recognize local events and the actions its squad is performing based on this observational data. We handle this problem by using explanation generation to abductively infer actions and events that may have been responsible for the ASM's observations. Using this information, the ASM can infer the current goals and plans of other squad members by performing plan recognition in situ using the history of recognized actions. Based on the team's inferred goals, the ASM's goal selection algorithm should then choose a new goal for the agent that coordinates with the team. Plan generation then creates a sequence of actions to accomplish this goal. Finally, the ASM should act in accordance with its plan. We detail each of these steps and their supporting techniques in this paper.

In Section 2 we examine related goal reasoning (GR) systems and discuss military programs that promote autonomous agents as teammates. Section 3 provides a more detailed description of the ASM domain and its inherent challenges, while Section 4 describes our GR process and its major components. Section 5 introduces our demonstration scenario and showcases an initial proof of concept in that domain. We conclude and discuss future research plans in Section 6.

2. Related Work

Numerous GR agents have been created and applied to control unmanned autonomous systems, some of which adapt to unexpected events and situations. For example, Coddington et al. (2005) describe MADbot, an agent that can change its goals dynamically. It offers insight into supplying agents with underlying drives and (primarily internal) motivations that can initiate goal change. We agree with the importance of motivated goal changes, but aim to generate goals based not only on internal motivations but also external factors such as the actions of team members and exogenous events in the environment.

Other GR agents have been designed to respond to dynamic problems and tasks (e.g., Talamadupula et al., 2011). While these reason about situation changes, they do not always detect and recognize them in intuitive ways. Instead, they often require structured interfaces to communicate goal changes with the agent. For instance, Talamadupula et al. use a "Problem Update" structure to communicate new sensory information and goal changes to the agent, but in real-world situations such information is not always made available so easily and quickly.

Work has been done, however, to extend these technologies with natural language processing and understanding techniques. Cantrell et al. introduce a full architecture (DIARC) that is designed to handle natural language and dialogue processing (2012). While this work does integrate natural language into a larger goal reasoning architecture, it mainly focuses on language and capability-related commands. For instance, telling an agent it can open a door to enter a room (Cantrell et al., 2012). We feel it is important for the ASM, in its military mission-related domain, to understand language not necessarily directed at it or in command form. Consider a scenario where one of the agent's teammates encounters an improvised explosive device (IED) near a convoy: the agent may simply sense the teammate yell *"IED"* or *"Explosive."* Therefore, while there has been notable NLU work applied to GR, we feel that our domain requires handling more simple and overheard styles of language. We respond to this challenge by forming domain-appropriate NLU techniques to incorporate into the ASM.

A related problem to explanation is diagnosis of discrete-event systems (Sampath et al., 1995), also referred to as history-based diagnosis (Gspandl et al., 2011), which finds action or event histories that account for a series of observations including observed events. Aside from representations, our formal model of explanations differs from the standard discrete-event system diagnosis model in the following ways: (1) we distinguish actions from events (which are deterministic), to better understand which actors are responsible for which actions and to predict unavoidable consequences; (2) we provide a formal model of ambiguous occurrences and a way to characterize their correctness, and (3) we assume that only partial states, and never actions or events, are observable. Diagnosis efficiency is considered a major issue in this community; recent efficient systems convert diagnosis problems to satisfiability or planning problems and adopt efficient techniques for solving the new problem (Grastien et al., 2011; Sohrabi et al., 2010).

Aside from more general GR research, several military programs are investigating the use of autonomous agents alongside teams of humans. One of these, the Safe Operations in Urban and Complex Environments (SOURCE) (Kott et al., 2010) Program, aims to develop collaborative autonomous agents to assist warfighters in dynamic environments in a safe and trustworthy manner. Similarly, the Robotic Collaborative Technology Alliance Army Program calls for the research and development of perceptive, intelligent autonomous vehicles that can interact with human teams (ARL, 2011). However, many of the collaborative systems proposed by these programs have not progressed beyond single-objective tasks with little to no external interference or danger. This presents a significant limitation, as more realistic missions like reconnaissance tasks (Section 5) are dynamic and can involve unexpected and dangerous events. A primary objective of our work on the ASM is to address such events.

We extend this diverse body of research on GR and autonomy by laying the groundwork for a more adaptive ASM.

3. Challenges in Real World Domains and Missions

The domain and mission characteristics for our proposed ASM present realistic and difficult challenges to GR and autonomy in general. When teams of warfighters undergo missions, such as reconnaissance, the mission environment is often hazardous and almost always dynamic.

Dynamic environments pose a major threat to autonomous systems that cannot adapt on the fly to changing conditions. For instance, a priori information about the world may be given in the form of maps and satellite images that may be inconsistent with the real-time state of the mission environment. Unforeseen obstacles, such as downed trees and boulders, can block a mission route. For an agent to collaborate effectively with its team it must react to dynamic environments such as these, and plan to act and assist in whatever way it can (e.g., help pull the tree out of the road or calculate a new route to the current destination).

In addition to being dynamic, in these environments a team may encounter unexpected and anomalous situations throughout a given mission (e.g., a squad can encounter enemy mortar fire while en route to a given destination). An agent cannot always be expected to perceive or sense the physical mortar shells being fired, nor can its teammates be expected to stop what they are doing and tell the agent what is happening (e.g., via some structured user interface). Instead, a robust autonomous agent must understand and explain what is happening around it based on observations and context clues, including those coming from its teammates in the form of natural language. For example, recognizing that its squad leader has yelled *"Mortar fire incoming"* can provide insight into what is currently happening, as well as how the rest of the team is expected to respond. The ASM can process these events without requiring focused operator input.

While we do not claim to have a complete solution to these issues that are inherent in realistic mission scenarios, but our system takes steps towards handling such scenarios.

4. Autonomous Squad Member Goal Reasoning Process

Our GR process includes five primary steps, represented by five distinct components: (1) a *Natural Language Interpreter*, (2) an *Explanation Generator*, (3) a *Plan Recognizer*, (4) a *Goal Selector*, and (5) a *Plan Generator*. Figure 1 displays the ASM's decision cycle, which involves using these components. Observations originate from the world (or simulation) the agent inhabits, which we label as the *Environment*. Ultimately, the agent's actions feed back into the *Environment,* completing the decision cycle. We outline and describe each of the five primary components in the remainder of this section.

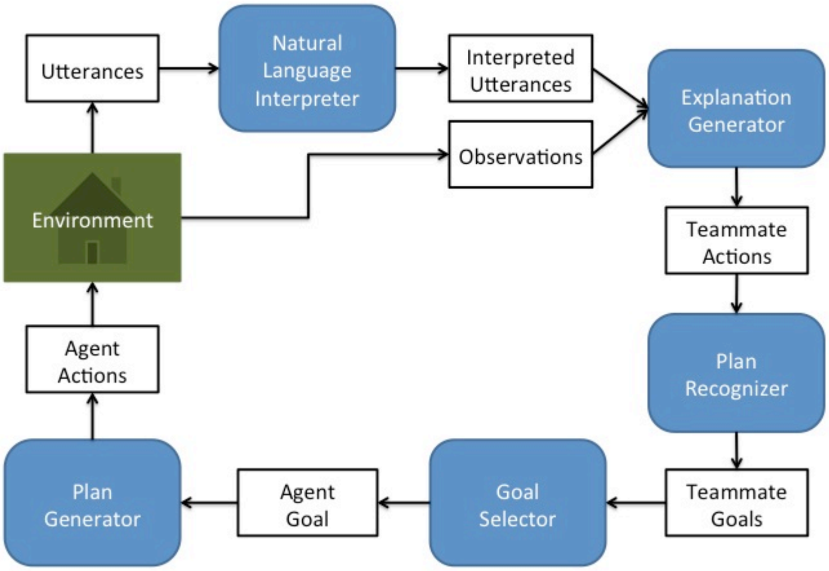


Figure 1. The Autonomous Squad Member's Goal Reasoning Process.

4.1 Natural Language Interpreter

The ability to interpret natural language uttered by teammates is a key capability for understanding nearby events, both expected and anomalous, in many domains. We begin the interpretation process by parsing spoken utterances using a domain-specific grammar and a chart parser from the Natural Language Toolkit (NLTK) (Bird, 2006). We keep the grammar small by considering only semantic categories related to the mission and potential anomalies that could be encountered (e.g., utterances regarding the recon mission, sniper encounters, teammate status). While a grammar is inherently limited in its vocabulary, the structured and well-known terminology of the military domain (e.g., NATO phonetic alphabet, device and role terminology) allows us to cover large portions of standard mission-related conversation and phrases in a compact manner. This grammar, coupled with a chart parser from the NLTK, produces parses for the raw utterances heard by the ASM. It reformulates these parses into semantically tagged structures that denote utterance types (e.g., mission-related, anomalous-event-related, squad-related) and content. Finally, these semantic structures are mapped to ontological information representing the implications and meanings of the utterance(s) being processed. We refer to this generated information as *interpreted* utterances.

Table 1 displays a few example utterances that could be overheard during a mission, as well as some interpretations that could be generated from each. For example, if the ASM senses a teammate saying *"Enemy Sniper,"* it can assume the teammate means that there exists an enemy in the world that is holding a sniper rifle. Variables in Table 1 are indicated by a leading question mark (e.g., `?enemyA` and `?teammateA`), and are implicitly existentially quantified. As output, the language understanding process generates semantic interpretations in the form of facts about the world, which are passed to the Explanation Generator for further processing.

Table 1. Example semantic interpretations for several sample utterances.

Utterance	**Example Interpretations**

"Enemy Sniper"	`(exists ?enemyA)`
	`(holding ?enemyA sniper-rifle)`
	`(is-self ?self)`
	`(on-team ?self ?teamA)`
	`(not (on-team ?enemy ?teamA))`

"Incoming Mortar"	`(exists ?enemyB)`
	`(holding ?enemyB mortar)`
	`(fired ?enemyB mortar)`

"Man Down"	`(exists ?teammateA)`
	`(on-team ?teammateA ?teamA)`
	`(is-self ?self)`
	`(on-team ?self ?teamA)`
	`(status ?teammateA injured)`

4.2 Explanation Generator

For the ASM to understand what its teammates are doing at any given moment, it needs to monitor its environment to recognize their individual actions. The Explanation Generator is responsible for this task; it accepts as input state information about the world in the form of observations and interpreted utterances from the Natural Language Interpreter.

To perform this task, we use a modified version of the DiscoverHistory algorithm (Molineaux, Kuter, & Klenk, 2012). DiscoverHistory is appropriate because it is designed to function in a partially observable environment and infer exogenous events and actions based on static observations (Molineaux & Aha, 2014). Unlike most other algorithms for explanation generation, DiscoverHistory operates efficiently by incrementally constructing explanations, and by maintaining explanations as new observations arrive rather than processing large batches. Additionally, it recognizes the difference between actions and events, and can ascribe particular world changes to the actions of known actors. This is important to support plan recognition in circumstances where individual actors may have different and even competing goals.

DiscoverHistory detects and resolves inconsistencies in existing explanations by applying one of the following inconsistency resolution methods to update the explanation:

- Hypothesize a new event or action
- Bind an unbound variable
- Remove an event or action
- Assume a property of the initial state
- Constrain the ordering of two unordered occurrences

DiscoverHistory conducts a search through explanation space; at each step of the search, it selects an inconsistency and applies all possible inconsistency resolutions to find successor nodes in the search space. This search terminates when a pre-specified number of explanations are found with only *ambiguous inconsistencies*, which are inconsistencies that can be resolved by binding a variable in multiple ways, and are considered trivial because the required conditions are met by several existing objects.

To properly integrate the partial knowledge discovered by the Natural Language Interpreter, we extended DiscoverHistory with the ability to incorporate existential quantifiers in its observations, which can in turn be bound during the explanation generation process. This resolves, or *grounds*, the semantic interpretations generated by the Natural Language Interpreter. Binding steps in the explanation process unify the variables in the interpreted utterances with existing (or newly discovered) entities in the environment. To illustrate this process, let's revisit the example interpretation in Table 1, namely the "*Enemy Sniper*" utterance. Suppose the ASM is aware of an enemy sniper nearby, and has labeled it `person1`. DiscoverHistory would recognize that the assertion (`holding ?enemyA sniper-rifle`) is inconsistent with existing knowledge, and resolves that inconsistency by binding the variable `?enemyA` to the value `person1`. This reconciles the inconsistency, as (`holding ?enemyA sniper-rifle`) is supported by prior observations. This contextualizes the remaining information, allowing the inference that `person1`, who holds a `sniper-rifle`, is not on the same team as `self` (i.e., the robot). Thus, the situation-agnostic interpretation is integrated with other observations of the current environment.

The modified DiscoverHistory algorithm also uses the predicted plan from the Plan Recognizer (Section 4.3) to help infer actions. Assuming that these predictions are correct, this speeds up the explanation generation process by removing several otherwise unnecessary search steps to find correct actions.

The primary output of the Explanation Generator is the set of actions that were inferred to be performed by the ASM's teammates. These actions are then given as input to the Plan Recognizer.

4.3 Plan Recognizer

Once the ASM has inferred the actions of its teammates, it can attempt to identify the higher-level plans and goals they are trying to accomplish via these actions. The task of identifying an ASM's plans based on their actions is referred to as *plan recognition*.

We extend and use a plan recognizer called the *Single-Agent Error-Tolerant Plan Recognizer (SET-PR)* (Vattam, Aha, & Floyd, 2014; 2015) to infer the plans (and goals) of the ASM's teammates. SET-PR's case-based approach to plan recognition is designed to robustly tolerate observational input errors, including missing, mislabeled and extraneous actions. This capability is especially useful when partial and imperfect observability conditions exist that are typical in the simulations and scenarios that we use to test our GR agent.

SET-PR learns to recognize plans from a given plan library C (i.e., a set of cases), where each *case* is a tuple $c = (\pi_0, g_0)$, π_0 is a known plan, and g_0 is its corresponding goal. Each case's plan $c.\pi_0$ is modeled as an *action-state sequence* $\mathbb{s} = \langle(\boldsymbol{a_0}, \boldsymbol{s_0}), \ldots, (\boldsymbol{a_n}, \boldsymbol{s_n})\rangle$, where each action $\boldsymbol{a_i}$ is a ground operator in the planning domain, and $\boldsymbol{s_i}$ is a ground state obtained by executing $\boldsymbol{a_i}$ in $\boldsymbol{s_{i-1}}$, with the additional caveat that $\boldsymbol{s_0}$ is an initial state, $\boldsymbol{a_0}$ is null, and $\boldsymbol{s_n}$ is a goal state. Plan $c.\pi_0$ does not store the propositional representation of $\mathbb{s}$. Instead, $\mathbb{s}$ is encoded as an *action sequence graph* $\mathcal{E}^{\mathbb{s}}$ and then stored in $c.\pi_0$. Vattam et al. (2014; 2015) introduce the action sequence graph representation for plans and discuss the reasons for using this representation in SET-PR. Inputs to SET-PR are sequences of actions performed by an observed agent and the resulting states. Like plans in cases, these observed sequences are modeled as action-state sequences and represented as action sequence graphs. Input graphs are then used to retrieve matching cases from the case base. SET-PR uses approximate graph matching techniques, described to compare an input against each case's plan $c.\pi_0$ and assign a score to c. The case with the highest score is returned; SET-PR predicts that the agent is following the plan $c.\pi_0$ to achieve the goal $c.g_0$.

We adapted SET-PR, a general-purpose plan recognizer, to fit into the ASM agent architecture. First, to operate in a multi-agent domain, we modified it to recognize a team's plan (containing individual team members' actions) and a team's goal (containing individual team members' goals). In particular, we modified SET-PR's plan representation by adding an actor to each action as its first argument. Furthermore, $c.g_0$ now represents a team goal and contains a set of goal propositions, one per team member. Second, the input to SET-PR is no longer provided by the environment, but is instead provided by the Explanation Generator. Explanations contain the set of all inferred actions and events, from which the subset of team members' actions are extracted and used as input to SET-PR.

From the Plan Recognizer we obtain the most likely plans and goals of the ASM's teammates. To illustrate, suppose that the output of the Explanation Generator consists of the following three observed actions of the ASM's teammates: (follow MEMBER2 MEMBER1), (follow MEMBER3 MEMBER1), and (move MEMBER1 ROUTE1)). Using this an input, SET-PR returns the following recognized team goal:

```
MEMBER1 -> (investigate-route MEMBER1 ROUTE1)
MEMBER2 -> (investigate-route MEMBER2 ROUTE1)
MEMBER3 -> (investigate-route MEMBER3 ROUTE1)
```

SET-PR also returns the following recognized plan, which contains the predicted actions of team members:

```
((follow MEMBER2 MEMBER1),(follow MEMBER3 MEMBER1),
 (move MEMBER1 ROUTE1),
 (DIRECT MEMBER1 (DIRECTIVE investigate-checkpoint)),
 (gesture MEMBER1 POINT),
 (move MEMBER2 LOCA), (move MEMBER3 LOCB),…).
```

4.4 Goal Selector

The Goal Selector determines a goal for the ASM based on the goals of its teammates. If the robot has n teammates, the goal tuple $T = (g_1, g_2, \dots, g_n)$ contains the currently recognized goal g_i of each teammate. Our current implementation uses a static goal (i.e., no goal selection is performed) but we are currently investigating several alternative goal selection strategies.

As future work, we plan to implement a goal selection strategy where the ASM selects its goal based on its teammates' current goals (i.e., $goalSelection: \mathcal{T} \rightarrow \mathcal{G}$, where $\mathcal{T}$ is the set of all goal tuples and $\mathcal{G}$ is the set of all goals). However, we also plan to investigate a strategy that examines how the teammates' goals have changed over time. The ASM would use the currently observed goals T_t at time t and the sequence of previously observed goals $\langle T_{t-1}, T_{t-2}, \dots \rangle$ to select a new goal (i.e., $goalSelection: \mathcal{T} \times \mathcal{T} \times \dots \times \mathcal{T} \rightarrow \mathcal{G}$). For example, the ASM could observe a teammate that historically has *scouting* goals but now has a goal reserved for squad leaders. This would allow the robot to infer that something has happened to the original squad leader and act accordingly (e.g., inform central command).

Goal selection strategies require criteria governing when the ASM's goal should be changed. Our initial plan is to use selection criteria provided by a domain expert. We prefer this in our domain because it avoids introducing additional error into the ASM (e.g., if it attempted to learn goal selection criteria) and prevents switching to an incorrect goal. If the ASM switches to an incorrect goal, it could impact the squad's ability to complete their task or mission. However, we also plan to explore strategies for learning goal change criteria in situations where criteria from an expert are not available or incomplete. The majority of our Goal Selector remains future work.

The selected goal is then sent as input to the Plan Generator, which uses the goal to generate a plan containing the ASM's future actions.

4.5 Plan Generator

Plan Generation takes the intended goal of the ASM, changed or unchanged, and determines the best way of accomplishing it. Whenever the goal is changed during goal selection, a new plan is generated for the selected goal. Replanning also occurs when an existing plan becomes inadmissible due to unexpected environment changes. In this situation, the generated plan attempts to fulfill the goal of the prior plan. In both cases, new plans are generated by the continuous time HTN planner SHOP2-PDDL+ (Molineaux, Klenk, & Aha, 2010). This enables the ASM to react to both the changing goals of the squad, and unexpected changes to the environment, as guided by task decomposition knowledge.

SHOP2-PDDL+ is ideal for use as the Plan Generator in the ASM domain for several reasons. First, it can represent continuous time and exogenous events that predictably occur after a time, such as a requested air strike or arrival of other agents at a location. Most planners can represent continuous changes to an environment only through durative actions, which don't properly represent how the choices of others affect the world. Second, SHOP2-PDDL+ is fast relative to other continuous-time planners, due to the task decomposition knowledge it uses internally to guide search and eliminate large areas of the search space. This is important due to the need for frequent replanning introduced by goal changes and unexpected environment change. Third, SHOP2-PDDL+ is effective in partially observable domains. The integration of SHOP2-PDDL+ with DiscoverHistory, both of which use the same knowledge representation, has been demonstrated in past studies as an effective means of understanding hidden information and creating plans that rely on it (Molineaux et al., 2012; Wilson et al., 2013; Molineaux & Aha, 2014). To support this, agent beliefs about hidden states generated by the explanation system are combined with observations from the environment. Together, these are sent to the Plan Generator as the initial state.

The plan generated by the Plan Generator is the decision output by the decision cycle, and replaces any prior plan generated in a previous cycle. Plans are enacted by sending actions to the environment at appropriate intervals. Enacting a plan fulfills the goal found in goal selection, which in turn relies on the environment interpretation generated by the first three components in the cycle. This entire cycle is continuously repeated, generating and enacting new plans as necessary, for as long as the agent inhabits its environment.

5. Integration Proof of Concept

In this section, we describe our proof-of-concept system, focusing on our demonstration scenario and a discussion of the system's performance.

5.1 Integration Scenario

To demonstrate the effectiveness of our GR agent, we created a scenario that encompasses both the standard mission-based objective that our domain normally contains and a characteristic anomalous event that disrupts mission execution.

The ASM accompanies a squad on a typical reconnaissance mission. Figure 2 shows an example mission map with checkpoints and routes labeled. A squad performs this mission by moving between checkpoints on a pre-specified route (e.g., *Start Area*, *CP Alpha*, *CP Bravo*). At each checkpoint, the squad splits into smaller teams or *sub-squads* that perform investigations of nearby areas, each of which follows a pre-determined investigation route (e.g., *RA1*, *RA2*, *RB1*, *RB2*). The mission is complete when the final checkpoint on the main route has been investigated. Our scripted scenario occurs as follows:

1. Squad receives a reconnaissance mission
2. Squad moves to checkpoint *Alpha*
3. Squad members investigate routes *RA1* and *RA2*, and then return to *Alpha*
4. Squad begins moving to checkpoint *Bravo*
5. Squad encounters an enemy sniper
6. Squad members eliminate the enemy sniper
7. If casualties have occurred, mission is aborted
8. Squad continues on to checkpoint *Bravo*
9. Squad members investigate routes *RB1* and *RB2*, and then return to *Bravo*
10. Mission complete

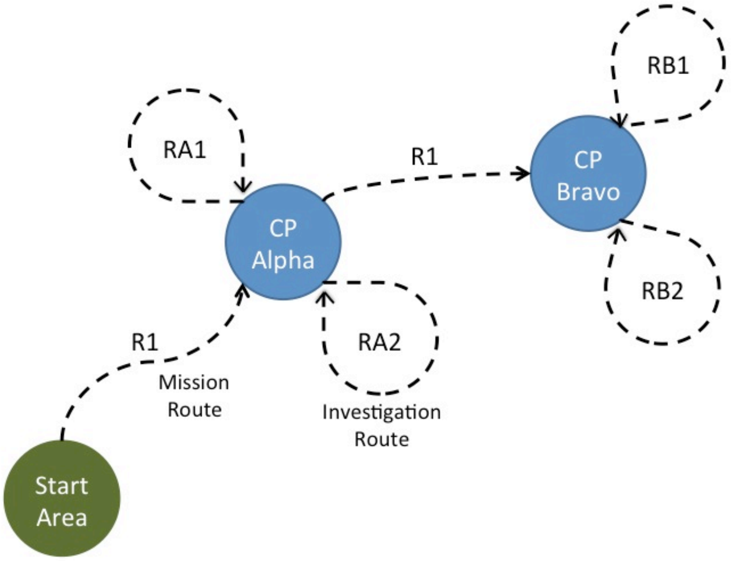


Figure 2. Reconnaissance Scenario.

The sniper encounter is an unplanned event that causes the squad to abandon their existing goals in favor of new ones, such as running for cover and attempting to locate the enemy. The autonomous agent must recognize these changes based on observations and natural language and act accordingly. Further, a member of the team will announce that the enemy has been neutralized, which the agent must recognize and use to update its understanding of the situation. Finally, the squad members continue on their mission and complete it, resuming their previously interrupted goal.

5.2 Demonstration Walkthrough

Below we provide a walkthrough of our demonstration in the form of data snapshots (for one cycle through the GR process) as we progress through the scenario. Explanation generation was requested to generate two explanations at each opportunity.

The initial goals of the team members are given as follows:

```
Goal, member1: (investigate-route route-1)
Goal, member2: (investigate-route route-1)
Goal, member3: (investigate-route route-1)
```

At this point, member1 and member3 are at checkpoint alpha and member2 is about to investigate route-alpha1.

The Natural Language Interpreter receives the following utterance as input:

```
Utterance: "Man down" Time: 600.013 Speaker: member3
```

…and generates the following interpretation as output:

```
...
(is-person ?teammate)
(on-team ?teammate ?myTeam)
(person-has-property ?teammate injured)
(is-utterance ?utterance)
((utterance-text ?utterance) “Man down”)
((utterance-type ?utterance) statement)
```

This interpretation tells us that there is a person ?teammate on the robot's team ?myTeam that is injured, along with some information about the utterance ?utterance. It is then passed along as input to the Explanation Generator.

Before the utterance was made, the Explanation Generator had the following explanation about the environment:

```
(observe-environment s=392)
(move route-alpha1 location-alpha member2 s=393)
(gps-observe-location member2 location-alpha s=394)
(observe-environment s=415)
(shoot-toward location-alpha #v795 s=416)
(observe-environment s=438)
```

In this explanation, it is believed that teammate member2 was starting to move along route route-alpha1 when an unknown person #v795 fired a shot towards his/her location.

Adding the utterance interpretation from the Natural Language Interpreter (along with other observed information) to this existing explanation results in the following inconsistencies:

```
inconsistency
   condition: person-has-property (#v958 injured)
   prior: <none>
   next: (observe-environment s=461)
inconsistency
   condition: utterance ("man down" 600.013 member3)
   prior: (observe-environment s=438)
   next: (observe-environment s=461)
inconsistency
   condition: sound-occurs (600.013 speech location-alpha)
   prior: (observe-environment s=438)
   next: (observe-environment s=461)
```

The inconsistencies indicate that the new observations include information not explained by prior events:

- A previously unknown person `#v958` was injured
- `member3` uttered a sentence
- A sound occurred at `location-alpha`

It also generates the following ambiguities:

```
ambiguity
   condition on-team (#v958 #v957)
   prior: (observe-environment s=438)
   next: (observe-environment s=461)
ambiguity
   condition: on-team (robot1 #v957)
   prior: (observe-environment s=438)
   next: (observe-environment s=461)
```

The ambiguities indicate that new observations include information not bound to prior knowledge:

- A previously unknown person `#v958` is on some team `#v957`
- The robot is on team `#v957`

The Explanation Generator attempts to resolve these inconsistencies and ambiguities by deriving two possible explanations (new information is shown in bold).

Explanation 1
This first explanation indicates that a previously unknown person at `location-4` was shot.

```
(assume-initial-value (object-location #v958) location-4 s=1)
...
(observe-environment s=392)
(move inv-routea1 loca member2 s=393)
(gps-observe-location member2 inv-routea1 s=394)
(observe-environment s=415)
(shoot-toward loca #v795 s=416)
(person-clipped #v958 s=(interval :start 417 :end 460))
(observe-environment s=438)
(speak-aloud "man down" member3 s=439)
(human-hears member3 "man down" s=440)
(human-hears member1 "man down" s=440)
```

Explanation 2
The second explanation indicates that someone was injured earlier, and `member3` is commenting on it now.

```
(assume-initial-value (person-has-property #v958 injured) s=1)
...
(observe-environment s=392)
(move inv-routea1 loca member2 s=393)
(gps-observe-location member2 inv-routea1 s=394)
(observe-environment s=415)
(shoot-toward loca #v795 s=416)
(observe-environment s=438)
(speak-aloud "man down" member3 s=439)
(human-hears member3 "man down" s=440)
(human-hears member1 "man down" s=440)
```

The Explanation Generator maintains these plausible explanations, and a single most-plausible explanation is given as input to the Plan Recognizer.

At this time, the goals of `member1` and `member3` have changed. However, the Plan Recognizer is unable to immediately detect these changes. Further observations lead to refinements of these explanations. Subsequently, the Plan Recognizer is provided with the following history of actions and events:

```
...
(speak-aloud "man down" member3 s=439)
(human-hears member3 "man down" s=440)
(human-hears member1 "man down" s=440)
(assume-defensive-position tree1 member3 s=462)
```

…and is able to recognize the following changed goals:

```
Goal change, member1: (respond-to-sniper loca4)
Goal change, member3: (respond-to-sniper loca4)
Goal detected, enemy1: (snipe team1)
```

These statements reflect a correct recognition, namely that an enemy is sniping the team, and two team members are now responding to the sniper instead of their previous goal (i.e., to investigate the route).

6. Conclusions and Future Work

We have demonstrated how a goal reasoning agent, the autonomous squad member (ASM), cooperates with teammates during a squad mission. The ASM recognizes and reacts to unexpected and anomalous events, processes natural language, infers and assumes unobserved facts about the world (via explanation generation), and recognizes the plans of its own teammates. In future work, goal selection will allow the agent to choose collaborative goals that are appropriate for its changing circumstances. Plan generation will then determine a means to accomplish these selected goals.

Our demonstration scenario suggests that the integration of these components is effective. In future work, we will conduct a formal empirical study to test this claim. These experiments will be distinguished into two categories, integration testing and individual component testing. In these experiments, we will measure the precision and recall of natural language interpretation, plan recognition, and explanation generation, as well as mission success criteria for determining overall performance. Integration testing experiments will measure whether the ASM performs best with all components present. For example, in an ablation study we will compare the precision and recall of explanation generation and plan recognition with and without natural language interpretation. Our objective is to show that the ASM performs better as a whole than any subset of its components. The other category, individual component testing, will test whether our implementation for each component is a good choice for the overall system. For example, we will swap out our Explanation Generator for a simpler deductive reasoner and compare the performance of the overall system using each.

Acknowledgements

We would like to thank OSD ASD (R&E) for sponsoring this research, which is being conducted in a project led by the US Army Research Laboratory (ARL).

References

ARL (2011). ARL Robotics Collaborative Technology Alliance (RCTA): FY 2011 Annual Program Plan. [arl.army.mil]

Bird, S. (2006). NLTK: The natural language toolkit. *Proceedings of the COLING/ACL on Interactive Presentation Sessions* (pp. 69-72). Stroudsburg, PA, USA: Association for Computational Linguistics.

Cantrell, R., Talamadupula, K., Schermerhorn, P.W., Benton, J., Kambhampati, S., Scheutz, M. (2012). Tell me when and why to do it!: run-time planner model updates via natural language instruction. *International Conference on Human-Robot Interaction*. Boston, MA, USA. ACM.

Coddington, A.M., Fox, M., Gough, J., Long., D., & Serina, I. (2005). MADbot: A motivated and goal directed robot. *Proceedings of the Twentieth National Conference on Artificial Intelligence* (pp. 1680-1681). Pittsburgh, PA, USA: AAAI Press.

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

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

Kott, N.J. III, Welfare, M., van Lierop, T.K., & Mottern, E. (2010). Safe operations of unmanned systems for reconnaissance complex environments Army technology objective. *Proceedings of SPIE 8045, Unmanned Technology XIII*. SPIE.

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

Molineaux, M., Klenk, M., Aha, D.W. (2010). Planning in dynamic environments: Extending HTNs with nonlinear continuous effects. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. Atlanta, GA, USA: AAAI Press.

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

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

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

Talamadupula, K., Schermerhorn, P., Benton, J., Kambhampati, S., & Scheutz, M. (2011). Planning for agents with changing goals. *Twenty-First International Conference on Automated Planning and Scheduling: Proceedings of the System Demonstrations* (pp. 71-74). Feiburg, Germany: AAAI Press.

Vattam, S.S., Aha, D.W., & Floyd, M. (2014). Case-based plan recognition using action sequence graphs. *Proceedings of the Twenty-Second International Conference on Case-Based Reasoning* (pp. 495-510). Cork, Ireland: Springer.

Vattam, S.S., Aha, D.W., & Floyd, M.W. (2015). Error tolerant plan recognition: An empirical investigation. In *Proceedings of the Twenty-Eighth Florida Artificial Intelligence Research Society Conference*. Hollywood, FL, USA: AAAI Press.

Wilson, M., Molineaux, M., & Aha, D.W. (2013). Domain-independent heuristics for goal formulation. In *Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference*. St. Pete Beach, FL, USA: AAAI Press.

Knowledge Representation for Dynamic Formation of Sensing Goals in Cyber Defense

Robert P. Goldman RPGOLDMAN@SIFT.NET
Mark Burstein MBURSTEIN@SIFT.NET
Ugur Kuter UKUTER@SIFT.NET
SIFT, LLC, 319 N. First Ave., Minneapolis, MN 55401 USA

Paul Robertson PAULR@DOLLABS.COM
Dan Cerys DAN@DOLLABS.COM
DOLL Labs, 114 Waltham Street, Lexington, MA 02421 USA

ABSTRACT

This paper describes how a cognitive agent for cyber defense forms sensing goals through a process of *active perception*. Loosely following a beliefs-desires-intentions (BDI) model (Bratman, 1987), we show how this agent arrives at the *desire* for more information, based on its *beliefs* about the current state of the defended network resulting from prior observations. The system translates these desires into *intentions* by identifying *sensing goals*, and a subset of these goals that best cover the most important attack targets with the existing resources. We identify two important classes of information goals for cyber defense systems: *forensic* and *proactive*. Forensic goals arise when the system identifies possible attack events, but does not have enough evidence to accept or reject these hypotheses. Proactive goals arise when the system anticipates attacks of a particular nature, or attacks on a particular target(s). In this case the system will want to ensure that it has deployed sensors that can detect the anticipated attacks. We discuss the process of forming sensor goals, and the knowledge representation which underlies them.

1. Introduction

The STRATUS system (Thayer et al., 2013; Robertson, Laddaga, & Burstein, 2013) embodies a cognitive architecture for autonomous cyber defense. STRATUS receives information about security-related events in the defended network from noisy intrusion detection sensors. It fuses reports together to identify likely intrusion events, and reasons about the resulting trust status of network assets (hardware/software systems playing roles in a mission) in order to decide how to use use spare resources to improve resilience. STRATUS uses models of the network, its computational tasks and data flows, and possible attacker actions to generate plausible adversarial attack plans, which it uses to project attackers' likely next steps and prepare mitigations for those steps. Figure 1 shows the overall system and data flows.

In this paper we focus specifically on how STRATUS uses *active perception*, the dynamic formation of information gathering or sensing goals to resolve interpretation questions. Our approach loosely follows a beliefs-desires-intentions (BDI) model (Bratman, 1987), where an agent arrives at the *desire* for more information, based on its *beliefs* about the current state of the defended network and its priorities for self-protection. It translates the desires into *intentions*, represented by *sensing goals*. STRATUS then chooses a subset of these goals that best serves its goal of protecting its most important ongoing tasks. We identify two important classes of information goals for cyber defense systems: *forensic* and *proactive*. Forensic goals arise when the system identifies possible attack events, but does not have enough evidence to accept or reject

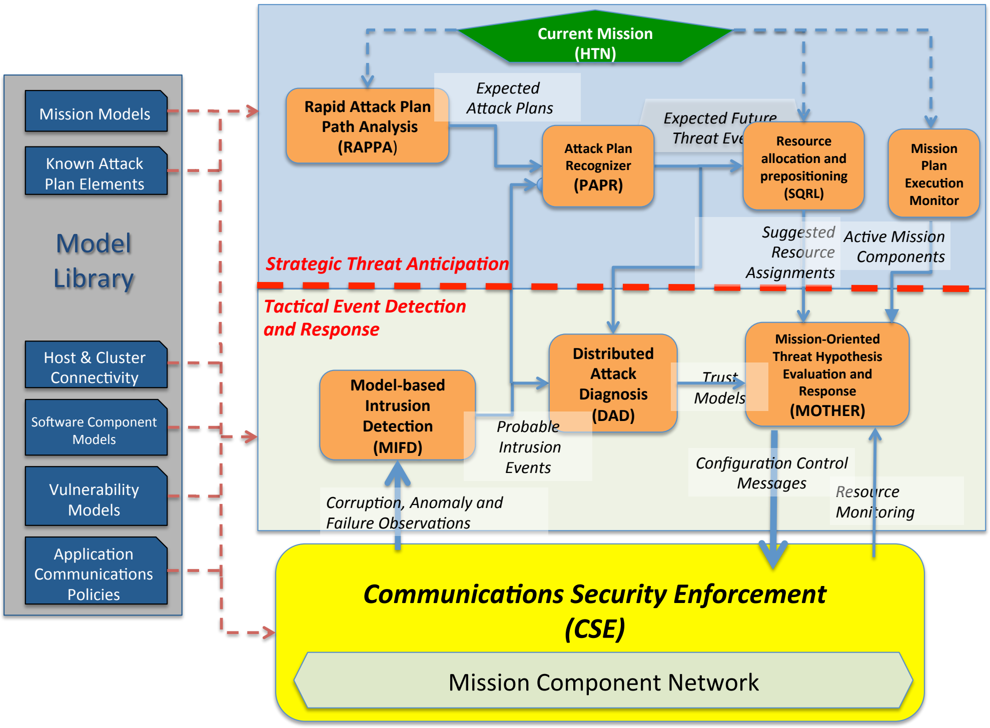

Mission Models
Known Attack Plan Elements
Model Library
Host & Cluster Connectivity
Software Component Models
Vulnerability Models
Application Communications Policies
Current Mission (HTN)
Rapid Attack Plan Path Analysis (RAPPA)
Expected Attack Plans
Expected Future Threat Eve
Attack Plan Recognizer (PAPR)
Resource allocation and prepositioning (SQRL)
Mission Plan Execution Monitor
Suggested Resource Assignments
Active Mission Components
Strategic Threat Anticipation
Tactical Event Detection and Response
Model-based Intrusion Detection (MIFD)
Probable Intrusion Events
Distributed Attack Diagnosis (DAD)
Trust Models
Mission-Oriented Threat Hypothesis Evaluation and Response (MOTHER)
Configuration Control Messages
Resource Monitoring
Corruption, Anomaly and Failure Observations
Communications Security Enforcement (CSE)
Mission Component Network

Figure 1: STRATUS architecture and information flow. The figure divides the architecture into three vertical slices, from the bottom the communications and sensing infrastructure, the tactical response loop, and the strategic loop. Legends on arcs in the main part of the figure describe the content of information flows. To the left is a list of the model content used by STRATUS.

these hypotheses. Proactive goals arise when the system anticipates attacks of a particular nature, or attacks on a particular target(s). In this case the system will want to ensure that it has deployed sensors that can detect the anticipated attacks.

The general flow of information and reasoning in STRATUS is as follows: Sensor reports are combined by STRATUS' MIFD component into attack *event hypotheses* which get used diagnostically to explain prior events, and proactively to anticipate likely new attack points. Initially, STRATUS just absorbed all of the sensor information that was sent to it. However, over the course of the project, we have recognized the need to actively manage our network sensors, through dynamic formation of *sensing goals*, in order to improve the accuracy of STRATUS' hypotheses. For example, some sensors (such as binary auditors of executables) are too expensive to run routinely everywhere. But they might be appropriate for use either to confirm or disconfirm indications of a possible attack, or to carefully watch assets that are critical to the network's task performance.

In this paper, we briefly review our inspiration for active perception, then outline how the functioning of STRATUS has been improved by adopting active perception. We focus on how one of STRATUS's components, MIFD, generates information seeking goals to improve its information fusion capability.

2. Inspiration

In most existing cyber security systems and machine perception systems, the sensor components are statically configured, so that sensor data is processed in the same, bottom-up manner each sensing cycle. The parameters of such systems are also statically tuned to operate optimally under very specific conditions. If higher level goals, context, or the environment change, the specific conditions for which the static configuration is intended may no longer hold. As a result, the static systems are prone to error because they cannot adapt to the new conditions: they are too inflexible. The results for cyber security are either too many false positives or too many false negatives, both disastrous. In other systems, including many computer vision systems, the results of sensing are so unreliable as to be unusable.

Inspiration for our approach, active perception, comes from biology. We have learned that high level context plays a key part in how images are interpreted (Oliva & Torralba, 2006). Context is the key to managing noisy sensors in image understanding and the hypothesis presented in this paper is that this is true for sensor systems in general – even for sensors operating in artificial systems such as cyberspace.

If context is key how do we get things started? It turns out that biology has an answer to that, too. Two ideas are key to the bootstrapping of contexts. The first is that of level skipping information flow where low level information skips some levels or processing to a higher level. In the brain these levels are different regions of the cortex. The surprising finding was that simple low level processing on sensor data, specifically image sensor data, is enough to characterize image type, or at least a set of possible candidate image types. This jumping to a (set of) conclusions has been called *gisting* (Oliva & Torralba, 2006). We can infer the gist of an image without discovering any of its contents with simple statistical methods on the low level data.

Gisting allows vision systems to identify the context of an image, and then these systems can use context to find content in the image. The presence of a context allows many false positives to be avoided. The upshot is that with active perception, we can do a better job of working with noisy and unreliable sensors.

Active perception draws on models for many purposes (Figure 2):

- to inform context-dependent tuning of sensors,
- to direct sensors towards phenomena of greatest interest,
- to follow up initial alerts from cheap, inaccurate sensors with targeted use of expensive, accurate sensors,
- and to intelligently combine results from sensors with context information.

Our model-based approach deploys sensors to build structured interpretations of situations to meet mission-centered decision making requirements.

Much work has been done on applying these ideas in the area of computer vision (Hofmann & Robertson., 2015), in this paper we discuss the architecture of a system that takes the same approach to interpreting noisy cyber attack sensors with the goal of producing high quality interpretations of sensor data and thereby avoiding the false positives that make today's systems ineffective (because people disable them if there are too many false positives).

Figure 2 shows the active perception loop, superimposed on a diagram describing the integration of top-down and bottom-up processing in vision.[1] Details of the visual processing are not important. The key ideas are that gisting starts the process of active perception, by activating top-level hypotheses/interpretation (here the presence of a dog in the image). The activation of the high-level hypothesis enables both priming to cause certain features to become more salient, and deployment of specialized sensors that are known to work well with the context in question.

3. STRATUS

STRATUS's goal is to use modest added overhead in computational resources to diagnose an attack, switch rapidly to computed backup contingencies, and predict downstream events by attackers with enough robustness to make mission critical functions resilient to those attacks (Thayer et al., 2013; Robertson, Laddaga, &

[1] The original diagram is from Serre (Serre, 2006).

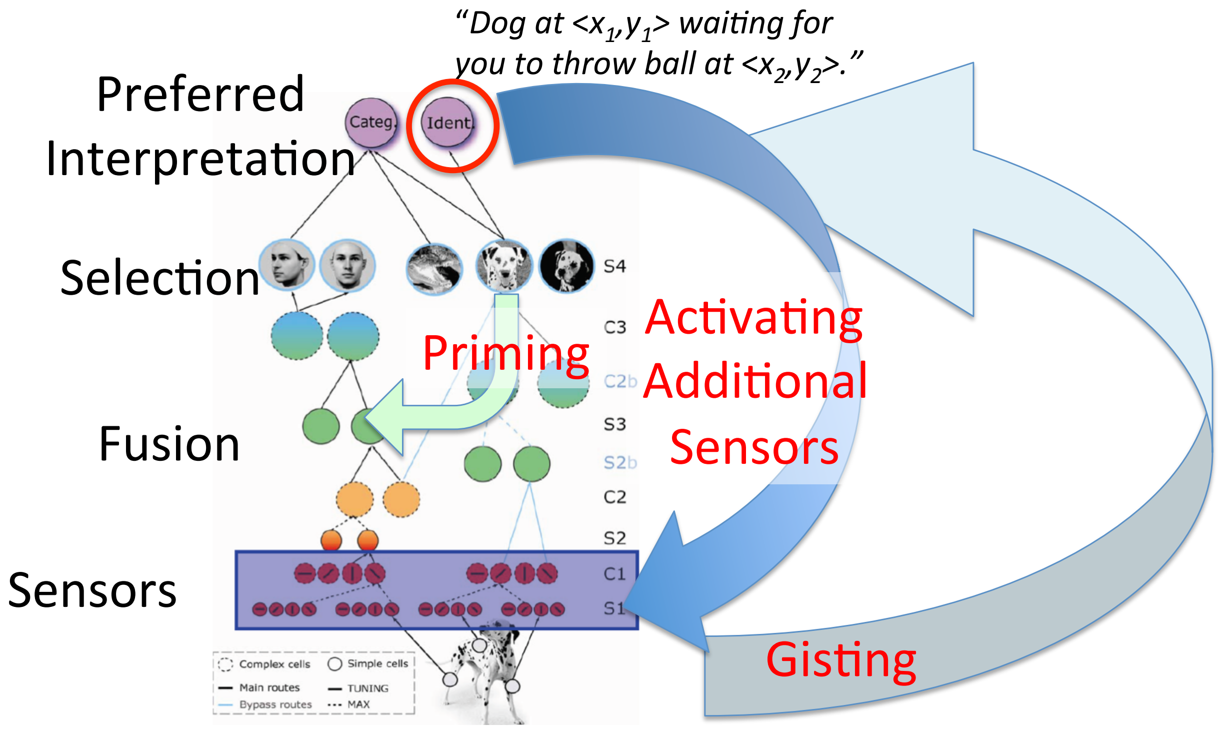


Figure 2: Active Perception uses model-based top-down reasoning, as well as bottom-up computation, to guide sensing actions. This figure is adapted from a talk by Thomas Serre (Serre, 2006).

Burstein, 2013). The features of STRATUS described in this section are fully implemented, but STRATUS is the focus of ongoing development and, in particular, the sensor choice module described in this paper is a work in process, and exists only as a not yet integrated proof of concept.

STRATUS is model-driven, using an ontology containing models of:

- **Missions:** a time-phased schedule for use of software systems.
- **Components:** software systems and their communications requirements.
- **Vulnerabilities:** known types of attacks on systems, and their likelihoods.
- **Trust levels** for hardware and software resources given attack evidence.
- **Attack Plan** schemas for multi-step attacks.
- **Host and cluster organization** for reasoning about how colocation[2] or network proximity can contribute to the possibility of vulnerability.

These models enable STRATUS to develop expectations about how observed problems might indicate attacks, and how attacks might proceed to adversely impact some mission(s). The overall architecture is shown in Figure 1.

Tactical Detection and Response: The lower half of the architecture in Figure 1 shows how STRATUS relies on features provided by the Communications Security Enforcement (CSE) infrastructure layer, an object- and model-based secure communications platform. The cloud computation infrastructure that STRATUS defends is decomposed into a set of communicating *CSE components*, each of which is a software application that can be redeployed and replicated by the CSE infrastructure. CSE enables STRATUS to

[2] *e.g.*, software components on the same (virtual) host as a known intrusion are less trustworthy.

monitor and redirect communications, to install and utilize redundant copies of software systems, and even to rewind and replay communication. It can transparently reconnect the communication channels between functioning components and those components being replaced by copies on more trustworthy platforms.

CSE uses models of component communications protocols, elements of which are typed objects from an ontology, to monitor for signs of corruption or software failure. When it detects these signs, it sends reports to STRATUS's sensor fusion subsystem, MIFD, which is based on the earlier Scyllarus (Goldman & Harp, 2009) system. The most recent version of Scyllarus developed by SIFT, Adventium and Honeywell, was able to run at scale for weeks at a time. It used Bayes nets and qualitative probability to filter the large numbers of weak reports of intrusion evidence to identify the most likely attack events. MIFD similarly filters and fuses sensor reports to develop hypotheses of intrusion events.

Intrusion events considered significant are used by both the tactical and strategic parts of the system. STRATUS will use them first to respond tactically to signs of corruption in key components, and strategically to look for longer attack plans in progress. The first step in the tactical processing is to identify the effect on mission components' health of the events identified by MIFD. The diagnosis module, DAD (Laddaga & Robertson, 2013), uses these MIFD's event messages to develop trust scores for each software component. These trust scores represent the likelihood that a component will perform properly in its various roles, in the present and near future. The components most likely to be compromised are considered by MOTHER when deciding whether to shift to alternate placements of components on different hosts, or to bring backups online to be ready to quickly replace compromised components in order to support key mission functions.

Strategic Threat Anticipation: MIFD's intrusion events are also used in the strategic process of recognizing attacker plans. STRATUS'sattack planner, RAPPA, uses diverse planning techniques (Srivastava et al., 2007) to generate a set of possible attack plans. This is a set of plans to defeat mission goals by attacking the current mission software elements, and the hosts those elements were assigned to run on. This set of possible plans is then summarized into a plan library using hierarchical task network (HTN) plan learning methods (Hogg, Muñoz-Avila, & Kuter, 2008; Hogg, Kuter, & Muñoz-Avila, 2009; Hogg, Kuter, & Muñoz-Avila, 2010). This plan library, in turn, is used by STRATUS'splan recognizer, PAPR, a system based on techniques developed by Geib and Goldman (Geib, 2009; Geib & Goldman, 2011; Geib, Maraist, & Goldman, 2008). PAPR does not try to identify a single specific attacker plan, instead picking out the most likely next steps in possible attack plans. This information is used to inform DAD's model of future component trustworthiness, and from there, inform MOTHER's countermeasure planning.

The upper portion of Figure 1 shows the components responsible for the strategic aspect of STRATUS, centered around PAPR, which does the attack plan recognition. Responses to attacks in progress are generated by MOTHER using alternate resource configurations generated by the resource reasoner, SQRL.

4. MIFD

MIFD, like its predecessor Scyllarus, views sensor fusion as an *abductive*, or *diagnostic* process. That is, a process of reasoning to the best explanation. MIFD's sensor fusion is particularly tailored to fusing reports from *Intrusion Detection Systems* (IDSes). When it receives *sensor reports* (IDS reports), MIFD forms *event hypotheses* to explain those sensor reports. Note that this is not a simple one-to-one process: MIFD fuses multiple sensors, so there will in general be multiple sensor reports providing support for a single event hypothesis. Sensors are often *ambiguous*, so there may be multiple alternative event hypotheses that would explain a single sensor report. Finally, events may be components of complex event hypotheses. For example, a single denial of service attack event hypothesis might explain multiple flooding attacks on a set of web servers. Note that the set of event hypotheses need not be exhaustive: some sensor reports are simple *false positives*.

The sensor reports and the event hypotheses that could explain them constitute a *Bayes network*.[3] We refer to the process of constructing this Bayes network as *clustering*. There is also a separate process of *assessment*, in which MIFD uses the evidence in the Bayes networks to assign a qualitative likelihood ranking to each of the event hypotheses. We will not discuss assessment further in this paper; for more details see (Goldman & Harp, 2009).

As described above, the two key data items in the cluster preprocessor are sensor reports and event hypotheses. The sensor reports are issued by various sensor programs, and consist of a *report type*, which specifies the condition that the sensor claims to be detecting. We say "claim to" because our experience indicates that even the most expert security analysts cannot reliably identify what their sensors will detect. IDSes must generally infer the existence of a security-related event from data (e.g., packet headers) that provides only very indirect and noisy indications. Such sensors often have a high false positive rate, and detect conditions (both other intrusions and benign events that resemble intrusions in some way) that their designers had not anticipated. Sensor reports also contain information about the location of the detection. Because STRATUS operates on a virtual network defined by the CSE middleware, location specifications are primarily in terms of source component(s), destination component(s), and CSE channel(s).[4] Host information can be inferred from the CSE components.

The clustering process generates event hypotheses to explain or interpret the sensor reports. The event hypotheses similarly have event types. These event types disambiguate the sensor reports. For example, in an early deployment of Scyllarus, we encountered a sensor report that claimed to identify network scanning, but that in fact also detected a class of software update, and traffic from Microsoft's print server. Each would be an alternate event hypothesis. The event hypotheses also contain location information, but MIFD attempts to disambiguate the source and destination information on the sensor reports into *attacker(s)* and *target(s)*. We have found that IDSes are erratic in the way that they assign "source" and "destination." Sometimes, especially for network-based IDSes (NIDS), these track the direction of packet flow, sometimes an attempt is made to infer a directionality for a session rather than a particular packet (or set of packets), and sometimes the IDS author tries to assign a more semantic notion that parallels the attacker/target distinction. For example, a successful phishing attack might involve a session in which the target is the "source" in the sense that the target computer initiates an HTTP session that requests download of malware. Alternatively, a NIDS might see the malware flowing in the HTTP response, and report the web server hosting the malware as the source, and so on.

The process of forming event hypotheses, populating them, and linking them to sensor reports and to each other is mediated by background information in *hypothesis matchers*. See Figure 3. An individual hypothesis matcher, M pairs a *phenomenon*, $P(M)$, a sensor report type or an event type, with an *explanation*, $E(M)$ event type. Hypothesis matchers perform a rule-like function. To a first approximation, a hypothesis matcher records the following inference pattern (Charniak & Goldman, 1988):

$$\forall x : P(M)(x) \rightarrow E(M)(x)$$

For example, one hypothesis matcher we use represents the following inference: "If there is a sensor report of type `unexpected component restart`, hypothesize an event of type `compromised component`."

There are two ways a hypothesis matcher can link a phenomenon to an explanation event hypothesis: it can either cause a new event to be hypothesized, *or* it can match an existing event hypothesis, and cause the new phenomenon to be linked in as additional evidential support. This is how hypothesis matchers can serve to *fuse* information from multiple sensors. In order to control this information fusion, hypothesis matchers

[3] Typically, this Bayes network is not completely connected: there are multiple connected components, each its own Bayes network. For example there may be some events occurring at host H_1 that are independent of what's going on at H_2

[4] The predecessor system, Scyllarus, had sensor reports with location information conforming to more conventional network architectures: source IP addresses, destination IP addresses, port numbers, etc.

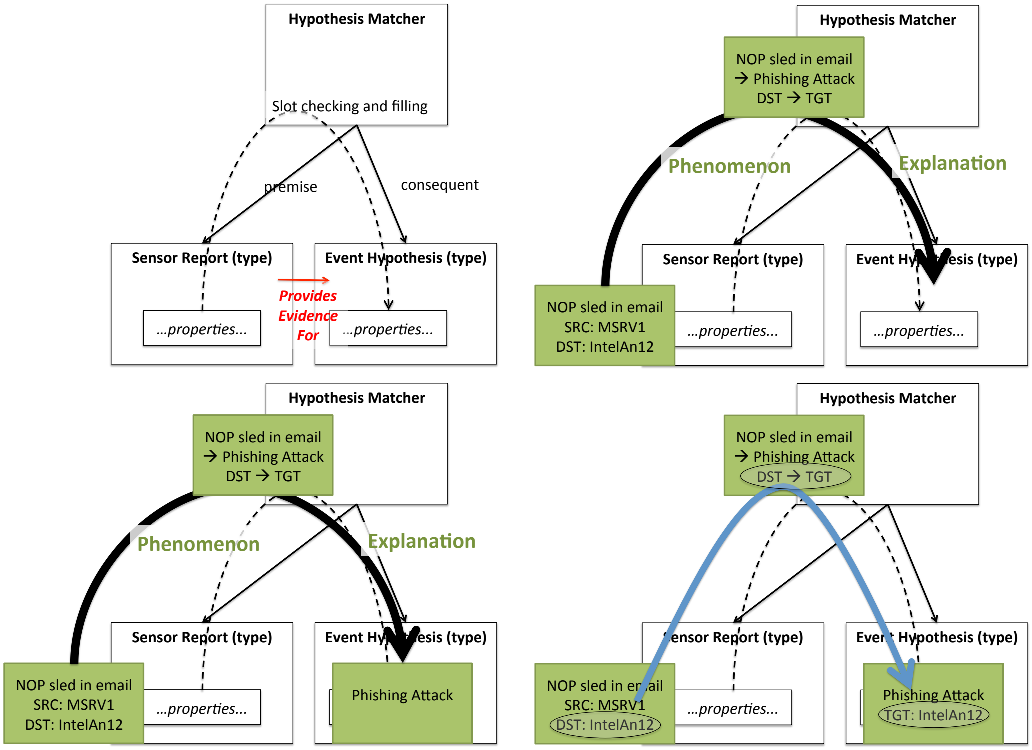


Figure 3: The function of hypothesis matchers.

contain *data checks*, which typically match location properties. For example, in order for an additional sensor report to support (be explained by) the `compromised component` hypothesis in our previous example, the data check would require the `destination component` of that sensor report to be the same as the `target component` of the `compromised component` hypothesis.

Finally, when creating a hypothesis or adding support to it, we copy information from the supporting report or event. For example, when we initially create the `compromised component` event, we copy the `destination component` value from the `unexpected component restart` sensor report into the `target component` of the hypothesis.

5. The Need for Sensing Goals

As our work on STRATUS has progressed, we have become convinced that dynamic, context-sensitive control of sensors is a critical capability for autonomous cyber defense. Existing IDSes typically have very high false positive rates. MIFD's fusion of multiple heterogeneous IDSes mitigates this problem, but even after fusion, there are many cases where follow-up "forensic" investigation is required. The type of sensing actions that a security analyst does in this more in-depth event investigation are too expensive to be deployed as routine, "always on" sensors. For example, after a possible intrusion, one might check cryptographic hashes of system binaries. There are also some cases where there are sensing actions that are too expensive to perform routinely, or which can be performed routinely, but produce too much information for constant processing. Sensors like these might be reserved for the most high-value targets. In conventional network defense, one would identify such targets *a priori* and statically. STRATUS uses mission models to identify high value assets dynamically, in order to support defense of cloud style networks in which computational resources are fungible, and computational tasks can be moved around the network more or less at will.

For these reasons, we determined that STRATUS should be able to dynamically control its sensing. We decided that STRATUS components desiring more information about particular (possible) events should be able to issue sensor requests. These sensor requests might be termed "information desires" if we adopt the jargon of BDI (belief-desire-intention) modeling. It will be the job of the MOTHER component to reason about the importance of these desires or requests, and determine whether to turn them into "intentions" and act upon them.

6. Forming Sensing Goals

STRATUS will form two kinds of sensing goals: forensic sensing goals, which attempt to find more evidence to reason about existing event hypotheses, and proactive sensing goals, that seek to place sensors to monitor expected threats. Here we describe the reasoning processes, and in the following section we describe the supporting knowledge representation.

The formation of forensic knowledge goals may be triggered when MIFD finds an event hypothesis whose uncertainty it can't resolve: it neither thinks it likely nor unlikely. In this circumstance, MIFD will examine the model of the event hypothesis to determine how critical the event is: i.e., how bad it would be if the hypothesis was true. STRATUS event type models contain *impact specifications* which indicate what security goals would be compromised when the event occurs. In the event of a high-criticality unresolved event hypothesis, MIFD will move to form sensor goals.

MIFD will invoke the sensor selection module (described in the following section) to identify sensors that could provide additional information to resolve the uncertainty about the event in question. If it finds such sensors, MIFD will publish a sensor goal, requesting this additional sensing. The STRATUS MOTHER module, responsible for resource management, will receive the sensor request, assess the criticality to the mission of the resource(s) to be examined, and consider the available sensing and sensor processing resources. Based on this information, MOTHER will decide whether or not to grant the sensing request. If MOTHER grants the sensing request, it adds additional information needed to realize the goal and publish it to CSE. The CSE infrastructure will carry out the necessary actions to implement the requested sensing.

Forming proactive sensing goals is a more open-ended process, open to arbitrary STRATUS subsystems. Proactive sensing will begin when a STRATUS subsystem, S identifies an event (an attack on a particular component or from a particular component) that it deems likely. This component might be DAD (Laddaga & Robertson, 2013) using its diagnostic reasoning to identify network components to which a current infection is likely to spread. DAD's reasoning can exploit information about the topology of the network, the nature of the network assets (what else is likely to be vulnerable to the current attack?), etc. Alternatively, S might be an element of the strategic subsystem, which identifies likely next targets based on its reasoning about the attacker's likely goals. The next targets in plans for likely attacker goals are considered to be under threat.

After this STRATUS module, S, identifies events and locations that it considers of interest, per the above reasoning, it will build representations for that information, as described in Section 7. This information will be used to query the sensor selection module, and find appropriate sensors and placements. From here the processing proceeds as per forensic sensing requests: the requests are published to MOTHER, which will evaluate them in the light of available resources and criticality of the possible targets. If MOTHER decides to grant the requests, they will be published to CSE for implementation.

7. KR to support sensing goal formation

We have developed a KR scheme to support formulating sensor requests assuming that STRATUS components will know *what* they want to see, and *where* they are interested in looking for it. The requesting components will also specify whether they have a one-shot, *forensic* interest in the information, or whether

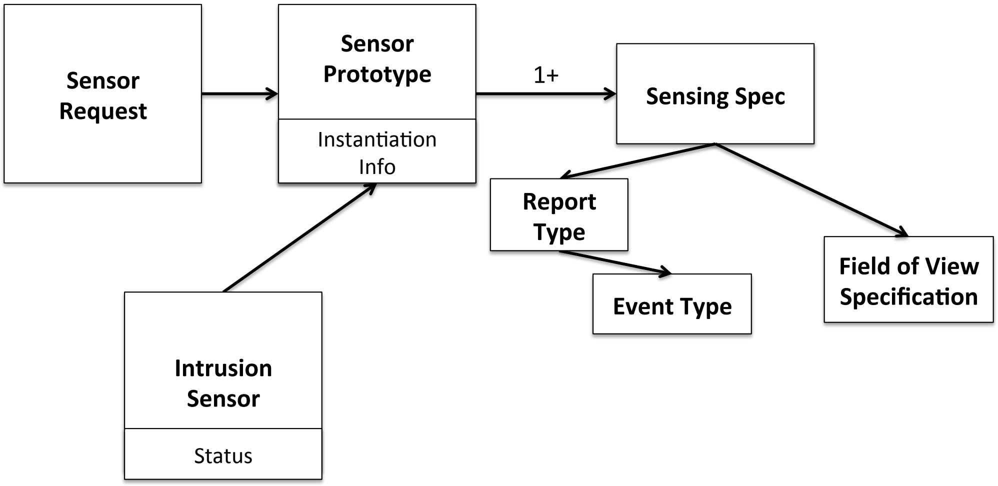
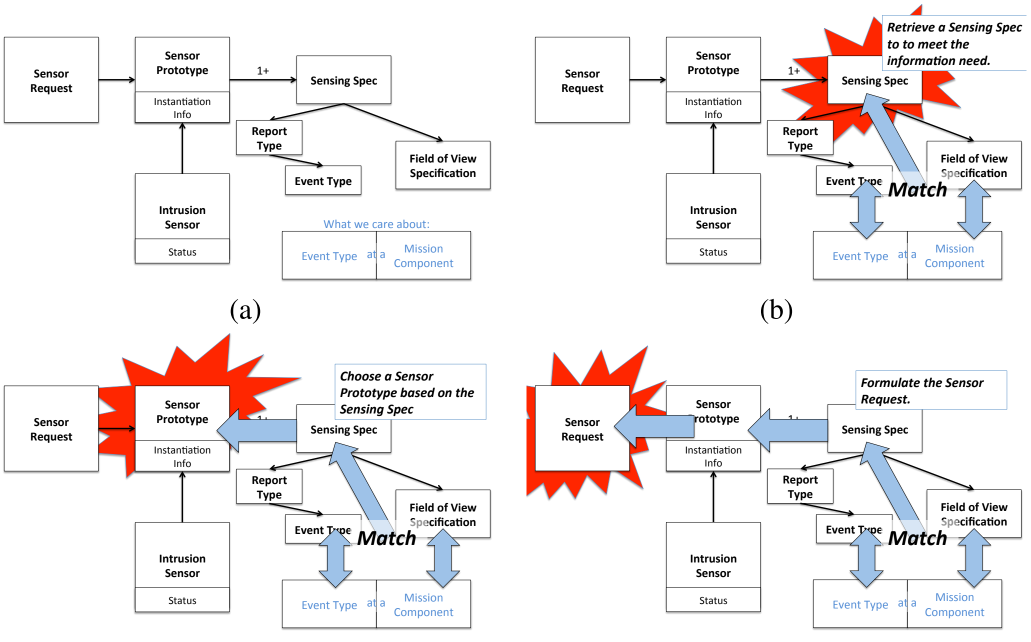


Figure 4: Knowledge representation for sensor requests.


Sensor Request
Sensor Prototype
Instantiation Info
1+
Sensing Spec
Report Type
Event Type
Field of View Specification
Intrusion Sensor
Status
What we care about:
Event Type
at a
Mission Component
Retrieve a Sensing Spec to to meet the information need.
Match
Choose a Sensor Prototype based on the Sensing Spec
Formulate the Sensor Request.

(a) (b)

(c) (d)

Figure 5: Forming sensor requests.

they want *ongoing* sensor coverage. Given this information, our KR allows the requestor to identify candidate sensor prototypes and formulate requests. Figure 4 gives a diagram of the underlying KR.

We are adding to MIFD a sensor selection module that will allow clients to request information about events of a particular type at a specific location (see Figure 5(a)). Finding the appropriate sensor prototypes based on the desired events types is a relatively straightforward extension of the existing KR for MIFD (see Section 4): the only work necessary here is to modify the existing representation, which has opaque procedures for the data checks and slot writers, to be more declarative, to permit us to invert the existing inference from supporter to event, and reason from event to possible supporters (see Figure 5(b - c)). Finally, the information about the requested sensor will be embedded in a sensor request (Figure 5(d)), which will be published to MOTHER for enactment or rejection.

8. Related work

Assessing the cost and benefit of observations is a core part of the project of decision analysis. For example, Raiffa's introductory textbook, *Decision Analysis* treats the cost and benefit of seismic testing in the now-canonical "oil wildcatter" problem (Raiffa, 1968). However, work in this area has focused more on choosing which tests to run, rather than on finding the set of relevant tests, as in our work. Typically, in decision analysis, the set of available tests is treated as given, as part of the framing of the problem. Some decision analysis texts (*e.g.* (Keeney, 1996; Hammond, Keeney, & Raiffa, 1998) discuss how to frame problems, but as a human process, rather than an automated one. Often decision problems are represented using Bayes networks and Influence diagrams.

Partially-observable Markov Decision Processes (POMDPs) provide a theoretical framework for reasoning about active perception (Cassandra, 1994). POMDPs are closely related to influence diagrams, but stress sequential decision over time rather than structuring by causal influence. POMDPs are typically solved using methods that stress infinite horizon problems and discounted reward, so in practice are applied to different problems than influence diagrams, which tend to be used for finite-horizon, single-shot decision problems. POMDP algorithms tend to scale poorly and be usable only for very small problems, unless substantial simplifications are made.

Ahmad and Yu propose a POMDP-based approach to active perception in the context of visual perception (Ahmad & Yu, 2013), but their test examples feature very small decision spaces (*e.g.*, three-location visual search). Eidenberger, *et al.* (Eidenberger, Grundmann, & Zoellner, 2009) also propose a POMDP-based approach to active vision, this time embedded in a robot, where they tradeoff information gain against control action costs, but their work aims at continuous action spaces, rather than the discrete on/off decisions we address here. All of these approaches, like the decision analytic approaches described earlier, take the set of possible information-gathering actions as given, so are complementary to the work described here, which focuses on finding the set of appropriate possible sensing actions.

We have been influenced by neuroscience work on perception in humans and other animals. Recent research has stressed the important role of expectation, active management of attention, and integration of higher cognition with low level sensing (Borji, Sihite, & Itti, 2011). This research overturns an earlier paradigm which stressed unidirectional, feed-forward processing that incrementally built higher and higher level interpretations.

Our work on STRATUS relates to other work on systems with goal directed autonomy (GDA) (Molineaux, Klenk, & Aha, 2010). STRATUS is like Molineaux, *et al.*'s ARTUE GDA system in attempting to address a dynamic, adversarial environment with substantial issues of partial observability. Until recently, Unlike ARTUE, STRATUS has not focused on generating goals on the fly, since STRATUS'sgoals are substantially computed from the fixed mission goals that the computational infrastructure serves. A key issue that distinguishes our current work is focus on choice of sensing behaviors. We believe that this is due to the large set of sensors available in cyber space, their widely varying qualities, and weaknesses in sensor quality and domain models.[5]

There has been a limited amount of related work in the planning literature. Work at the University of Washington on the Unix SoftBot stressed the ability to incorporate sensing actions into plans, but focused more on representation of the sensing actions, and on efficiently updating the belief state of the observing agent (Etzioni, Golden, & Weld, 1997; Golden, 1997). Petrick and Bacchus have revived and adapted this work for today's more capable planners (Petrick & Bacchus, 2004). There is related work in action modeling that addresses more expressive reasoning about sensing actions, but we do not know of any work that has succeeded in making it efficient enough for use in actual programs (Scherl & Levesque, 1993). Alford, *et al.* address the problem of choosing information-producing actions particularly for reasoning

[5] Cyber attacks tend to occur precisely at points of modeling weakness, since exploits typically arise from incorrectly implemented specifications, or leaky specifications.

about uncontrolled agents in the environment (Alford et al., 2015). Their work applies an approximated POMDP model to an air-to-air combat simulation, and effectively shows that inference about the attacker's intentions is useful in leading to better outcomes. In this domain, however, there are no distinguished sensing actions: only actions that as a side-effect cause the adversary to expose its intentions.

In the proceedings of this workshop, Bengfort and Cox ("Interactive Knowledge-Goal Reasoning") discuss formation of knowledge goals as part of an interactive question-answering system. Their work differs from ours in handling more complex knowledge goals, which can profitably be decomposed by subgoaling. Rather than a model-based approach, they use case-based reasoning to find previous methods of question-answering that match present knowledge requirements, and they work interactively with a human in the loop. Another important difference is that the *purpose* of the knowledge-seeking is outside the scope of consideration (their system serves a human user who has information needs), whereas our knowledge goals are strictly in the service of action. This is why our information goals are subject to evaluation by MOTHER, the action-choosing component of STRATUS: STRATUS'sinformation goals are of different priorities depending on whether they serve to choose an action, and depending on how important that action choice is. For example, there's no point in further sensing if the action choice is the same whether or not a particular component has been compromised (*e.g.*, perhaps its downstream components are known to be corrupted, so its work is useless). Or, if a particular component is of low importance, it may be best to simply shut it down, rather than commit resources to further investigation.

9. Conclusions

In this paper we have described a method of forming and processing sensor goals in order to actively manage sensing for cyber defense. This process will allow STRATUS to appropriately develop situation awareness by allowing it to employ a combination of cheap and likely noisy sensors, and more accurate sensors that are too expensive to employ everywhere and at all times.

The work described in this paper is very much in progress. We have developed a preliminary implementation of the model, which provides both KR and automatically-generated CSE communications stubs. We are currently working on the sensor selection module and its API, and are developing test scenarios. We have already carried out experiments on abstract models of active perception, which show its value in situation with the observation characteristics of cyber defense. We expect to publish the results of those experiments in the near future.

Acknowledgements

Thanks to the entire STRATUS project and our funders. This work was supported by Contract FA8650-11-C-7191 with the US Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory. Approved for Public Release, Distribution Unlimited. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

Ahmad, S., & Yu, A. J. (2013). Active sensing as bayes-optimal sequential decision making. *UAI*. AUAI Press, Corvallis, Oregon.

Alford, R., Borck, H., Karneeb, J., & Aha, D. W. (2015). Active behavior recognition in beyond visual range air combat. *Proceedings of the Third Annual Conference on Advances in Cognitive Systems (ACS)*. Atlanta, GA, USA.

Borji, A., Sihite, D. N., & Itti, L. (2011). Computational modeling of top-down visual attention in interactive environments. *Proc. British Machine Vision Conference (BMVC 2011)* (pp. 85.1–85.12).

Bratman, M. E. (1987). *Intention, Plans, and Practical Reason*, Cambridge, MA: Harvard University Press.

Cassandra, A. R. (1994). *Optimal Policies for Partially Observable Markov Decision Processes* (Technical Report CS-94-14). Department of Computer Science, Brown University, Providence, RI.

Charniak, E., & Goldman, R. P. (1988). A Logic for semantic interpretation. *Proceedings of the Annual Meeting of the ACL* (pp. 87–94).

Eidenberger, R., Grundmann, T., & Zoellner, R. (2009). Probabilistic action planning for active scene modeling in continuous high-dimensional domains. *ICRA* (pp. 2412–2417). IEEE.

Etzioni, O., Golden, K., & Weld, D. S. (1997). Sound and efficient closed-world reasoning for planning. *Artificial Intelligence*, *89*, 113–148.

Geib, C. W. (2009). Delaying commitment in plan recognition using combinatory categorial grammars. *IJCAI* (pp. 1702–1707).

Geib, C. W., & Goldman, R. P. (2011). Recognizing plans with loops represented in a lexicalized grammar. *Proceedings AAAI*.

Geib, C. W., Maraist, J., & Goldman, R. P. (2008). A New probabilistic plan recognition algorithm based on string rewriting. *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-2008)* (pp. 91–98). Sydney, Australia.

Golden, K. (1997). *Planning and knowledge representation for softbots*. Doctoral dissertation, University of Washington.

Goldman, R. P., & Harp, S. A. (2009). Model-based intrusion assessment in common lisp. *Proc. Int'l Lisp Conference*. Cambridge, MA.

Hammond, J., Keeney, R., & Raiffa, H. (1998). *Smart choices: A practical guide to making better decisions*: Harvard Business Review Press.

Hofmann, A., & Robertson., P. (2015). Active perception: Improving perception robustness by reasoning about context. *Proceedings of the 10th International Conference on Computer Vision Theory and Applications*.

Hogg, C., Kuter, U., & Muñoz-Avila, H. (2009). Learning hierarchical task networks for nondeterministic planning domains. *IJCAI-09*.

Hogg, C., Kuter, U., & Muñoz-Avila, H. (2010). Learning methods to generate good plans: Integrating htn learning and reinforcement learning. *AAAI-10*.

Hogg, C., Muñoz-Avila, H., & Kuter, U. (2008). HTN-MAKER: Learning HTNs with minimal additional knowledge engineering required. *AAAI-08*.

Keeney, R. (1996). *Value-focused thinking*: Harvard University Press.

Laddaga, R., & Robertson, P. (2013). Adaptive security and trust. *Proceedings of the 2nd International Conference on Cyber Security, Cyber Peacefare and Digital Forensic - Cybersec 2013* (pp. 231–239).

Molineaux, M., Klenk, M., & Aha, D. W. (2010). Goal-driven autonomy in a navy strategy simulation. *American Association for Artificial Intelligence*. AAAI Press.

Oliva, A., & Torralba, A. (2006). Building the gist of a scene: the role of global image features in recognition. In Martinez-Conde, Macknik, Alonso, & Tse (Eds.), *Progress in Brain Research*, Amsterdam, Netherlands: North Holland.

Petrick, R. P. A., & Bacchus, F. (2004). Extending the knowledge-based approach to planning with incomplete information and sensing. *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)* (pp. 2–11). Whistler, B.C., Canada.

Raiffa, H. (1968). *Decision Analysis: Introductory Lectures on Choices under uncertainty*, New York: Random House.

Robertson, P., Laddaga, R., & Burstein, M. (2013). Trust and adaptation in STRATUS: Strategic and tactical resiliency against threats to ubiquitous systems. *Proceedings Trust Security and Privacy in Computing and Communications (TrustCom)*. Melbourne, Australia.

Scherl, R. B., & Levesque, H. J. (1993). The frame problem and knowledge-producing actions. *Proceedings AAAI* (pp. 689–695). Menlo Park, CA: AAAI Press/MIT Press.

Serre, T. (2006). Feedforward theories of visual cortex predict human performance in rapid image categorization. Talk.

Srivastava, B., Nguyen, T. A., Gerevini, A., Kambhampati, S., Do, M. B., & Serina, I. (2007). Domain independent approaches for finding diverse plans. *Proceedings of IJCAI*.

Thayer, J., Burstein, M., Goldman, R. P., Kuter, U., Robertson, P., & Laddaga, R. (2013). Comparing strategic and tactical responses to cyber threats. *SASO Workshop on Adaptive Host and Network Security AHANS*.

Reasoning about Attack Goals for Cyber Resilience

Ugur Kuter UKUTER@SIFT.NET
Mark Burstein MBURSTEIN@SIFT.NET
Robert P. Goldman RPGOLDMAN@SIFT.NET
SIFT, LLC, 319 N. First Ave., Minneapolis, MN 55401 USA

Abstract

This paper describes our approach to anticipating and recognizing potential cyber threats in order to provide timely responses to those threats. Our approach anticipates attacks on distributed systems by generating a diverse set of attack plans on key system components and then determining the probabilities that these attacks may threaten those components or others that are stepping stones to those systems. The results enable our overall system, STRATUS, to defend these systems by preparing backups and controlling communications pathways appropriately. We present a preliminary empirical study of our techniques, demonstrating their promise.

1. Introduction

Strategic resilience in an adversarial environment requires not just the ability to *respond* in a timely fashion to detected problems, but also an ability to *predict* potential threats. A system for strategic resilience must identify potential threats to key mission objectives before they happen, use that analysis to recognize threats as they unfold, and pre-plan so that it can respond quickly when these threats actually occur. Current IDSs do not predict attacks; they do not provide early warning. They report the type and properties of an attack after (sometimes long after) it has happened. While recognition of attacks is an important ability, it falls short of the community's vision for security systems that predict future hacker actions and automatically and correctly respond to attacks in a timely manner.

This paper reports on our work on reasoning a priori about attacker actions and plans for their otherwise dynamic and unexpected effects on a cyber mission and its goals during the mission execution (Vattam et al., 2013; Klenk, Molineaux, & Aha, 2013). In particular, we describe our approach to probabilistic plan recognition using diverse planning to generate a graphical model of hypothetical attack goal/surfaces for a cyber mission and probabilistic plan recognition that uses this diverse graphical model. Our work resides within the context of a larger, multi-component cognitive architecture called STRATUS (Strategic and Tactical Resiliency Against Threats to Ubiquitous Systems) (Burstein et al., 2012) that we are developing to provide resilience for cyber missions in large computer networks. STRATUS's goal is to use modest added overhead in computational resources to diagnose an attack, switch rapidly to computed backup contingencies, and predict downstream events by attackers with enough robustness to make mission critical functions resilient to those attacks. STRATUS uses models of missions, software and network elements and attack types to develop expectations about how observed problems might indicate attacks, and how attacks might

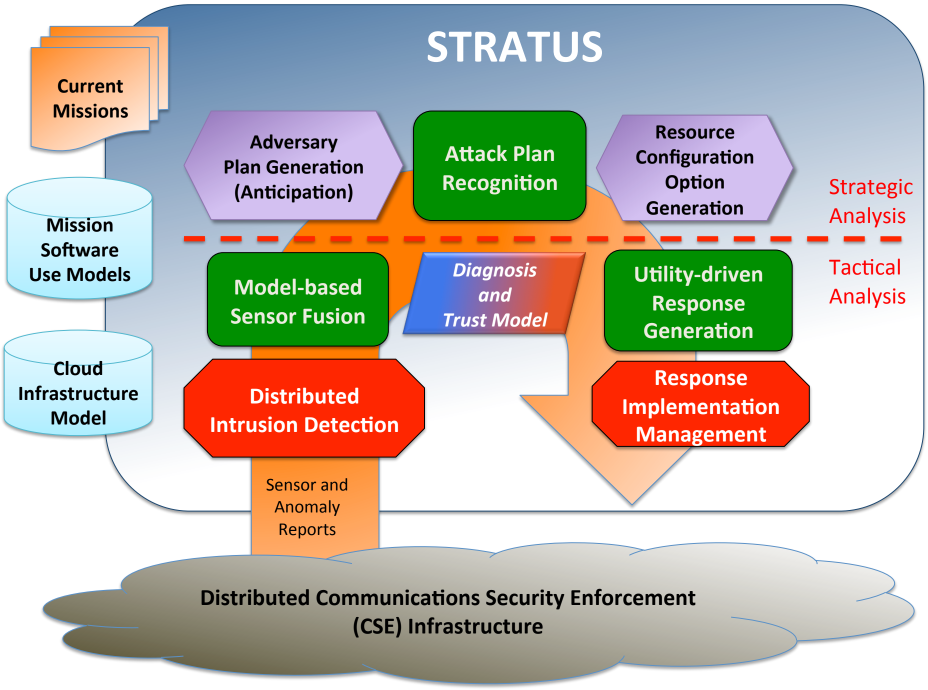

STRATUS
Current Missions
Mission Software Use Models
Cloud Infrastructure Model
Adversary Plan Generation (Anticipation)
Attack Plan Recognition
Resource Configuration Option Generation
Strategic Analysis
Tactical Analysis
Model-based Sensor Fusion
Diagnosis and Trust Model
Utility-driven Response Generation
Distributed Intrusion Detection
Response Implementation Management
Sensor and Anomaly Reports
Distributed Communications Security Enforcement (CSE) Infrastructure

Figure 1. Abstract view of STRATUS information flow architecture.

proceed to adversely impact some mission(s). An abstract functional overall architecture is shown in Figure 1.

Because of the myriad ways an attacker can get into a computer network, and the limited resources available to respond, effective cyber response requires recognizing the attacker's potential goals, and focusing responses on protecting those assets most critical to the defender's own goals. When we are resource-limited, network assets that serve only low-priority goals can be sacrificed to focus on the essentials. Combining information about targets affected by particular attacks and information about the importance of particular resources to a network's mission, we can focus the system's attention on those attacks that would have the greatest impact, and then look specifically for indicators of those attacks. By identifying the critical attacks and their goals, we can pre-plan to ensure we are prepared to respond appropriately.

We are developing two sister systems, RAPPA (Rapid Attack Plan Path Analysis) and SPAR (Simple Probabilistic Attack Recognizer). RAPPA uses a combination of planning and learning techniques to identify likely attack goals and plans, into models needed for attack plan recognition and countermeasure planning. Using the attack plan libraries generated by RAPPA as a model of attacker behavior, SPAR quickly estimates the conditional probability of potential attacks on different mission components, given (noisy) observations of attacker actions. SPAR can explain the reasoning behind its probabilistic goal recognition, using regression techniques over probabilistic attack paths. We present a preliminary evaluation of RAPPA and SPAR, demonstrating the promise of the attack goal and behavior reasoning and recognition capabilities.

2. Motivating Scenarios

We have been developing a family of scenarios, based on a simplified model of the computational aspects of the planning and execution of missions on large computer networks. Figure 2 shows a set of cloud clusters (LANs), and mission software components for a simple demonstration network. The upper half of the diagram contains a set of cloud clusters for handling imagery and mission-related databases and computational resources, while the lower half shows the human user client systems that access and manage those resources during the mission.

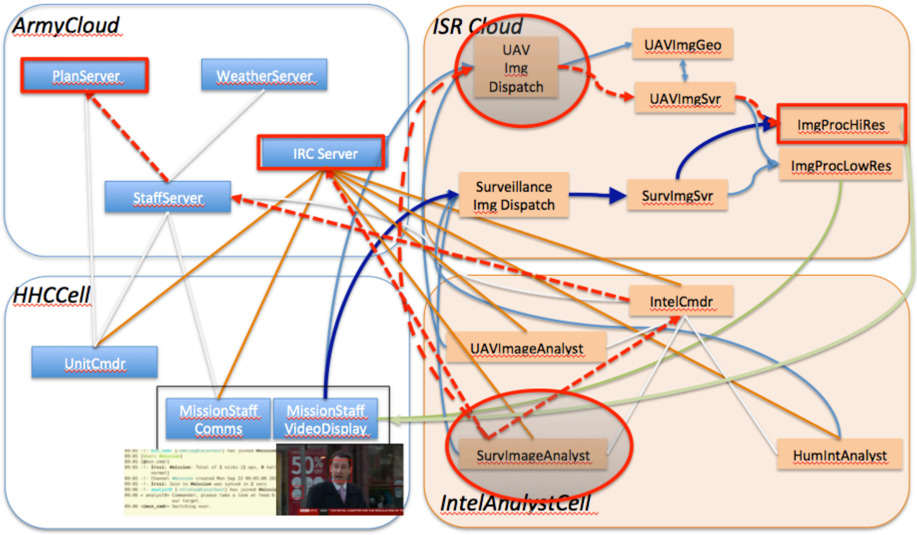


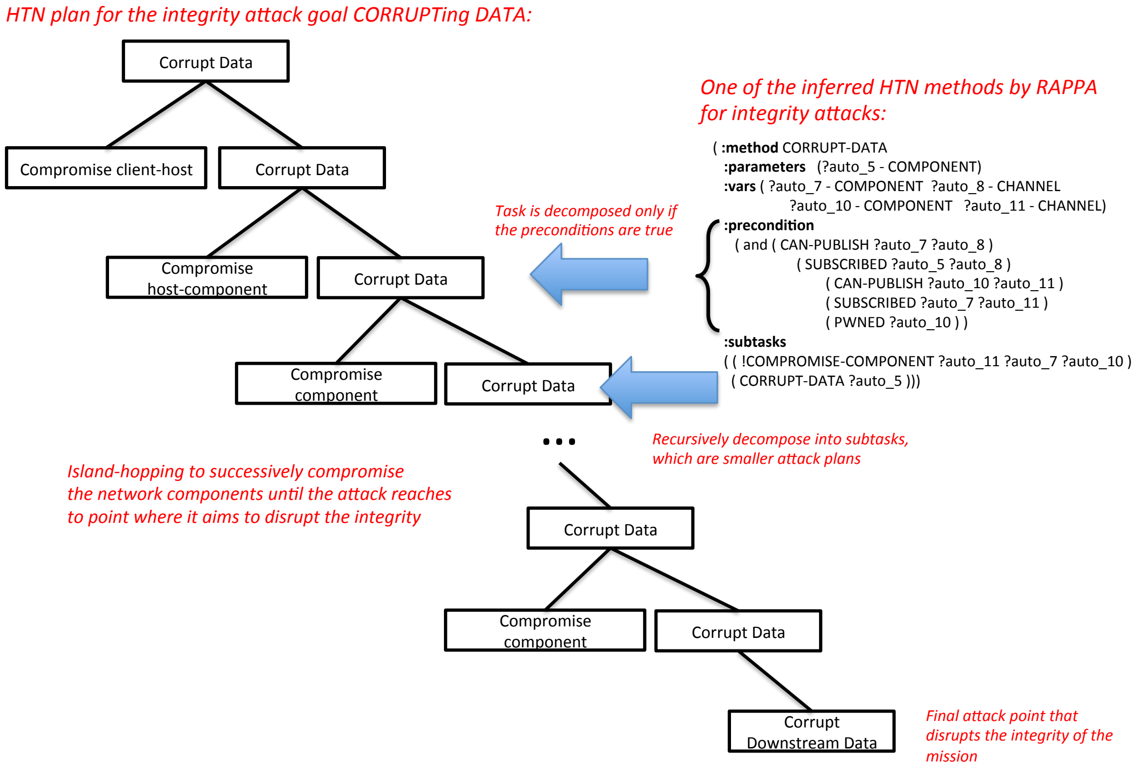
Figure 2. Mission resources and configuration in a computer network. The objective is to recognize distributed attack goals and plans. Circled components denote nodes that have been observed to be compromised. Red-bordered components are the potential goals of attacks. Paths of potential attack are dashed red. Other colored links denote normal mission processing.

In one example of an attack goal against the mission in this scenario, an attacker intends to gain access to the PlanServer database to exfiltrate information. In another, a virus is introduced from the web that ’powns’ some of the software resources used and generates an internal DDoS attack on one of the critical Image Processing servers.

A planning problem for RAPPA is a initial situation assessment of the computer network, mission specifications, as well as potential known attacker capabilities, actions, and resources. In this approach, we follow Shrobe (Shrobe, 2002a; Shrobe, 2002b) in driving attack plan generation from models of the defended system, its missions, and known exploits. As an example, such attacker models based on the models of the defended system in Figure 2 would include compromising a component in the network by exploiting a combination of known properties of the user machines, that are exposed to the outside cyber world, and network internal servers known to be heavily-used for computation and which, therefore, are potential targets for a denial of service attacks.

Using such models, RAPPA generates possible attack goals in the network by exploring the potential attack surface using AI planning techniques. For example, an attack path plan (marked as blue) in our scenario starts from the VideoDisplay user machine, and transmits malware over the network to the Image Hi-Resolution Processing Server to corrupt it.

RAPPA not only generates such attack goals but further categorizes and conceptualizes them into abstract classes. As we describe in the subsequent sections, we use both Hierarchical Task Network (HTN) learning and concept learning algorithms to achieve this capability. Figure 3 shows an illustration of such conceptualizations in terms of HTN methods.


HTN plan for the integrity attack goal CORRUPTing DATA:
Corrupt Data
Compromise client-host
Corrupt Data
Compromise host-component
Corrupt Data
Compromise component
Corrupt Data
Task is decomposed only if the preconditions are true
One of the inferred HTN methods by RAPPA for integrity attacks:
Recursively decompose into subtasks, which are smaller attack plans
Island-hopping to successively compromise the network components until the attack reaches to point where it aims to disrupt the integrity
Corrupt Data
Compromise component
Corrupt Data
Corrupt Downstream Data
Final attack point that disrupts the integrity of the mission

```
( :method CORRUPT-DATA
  :parameters   (?auto_5 - COMPONENT)
  :vars ( ?auto_7 - COMPONENT  ?auto_8 - CHANNEL
              ?auto_10 - COMPONENT   ?auto_11 - CHANNEL)
  :precondition
    ( and ( CAN-PUBLISH ?auto_7 ?auto_8 )
              ( SUBSCRIBED ?auto_5 ?auto_8 )
                  ( CAN-PUBLISH ?auto_10 ?auto_11 )
                  ( SUBSCRIBED ?auto_7 ?auto_11 )
                  ( PWNED ?auto_10 ) )
  :subtasks
  ( ( !COMPROMISE-COMPONENT ?auto_11 ?auto_7 ?auto_10 )
   ( CORRUPT-DATA ?auto_5 )))
```

Figure 3. An illustration of an HTN plan library and HTN plan inferred by RAPPA on an Integrity (corrupting data) objective.

The HTN method shown on the right in the figure is one of the methods RAPPA produces from attack plans for the goal of corrupting a component in the network. On the left is an illustration of how to unfold the HTN plan into a hierarchical representation. This represents the CORRUPT-DATA task as consisting of two subtasks: (1) compromising the client host to enter to the system and (2) recursively attempting to accomplish the CORRUPT-DATA task from there. Successive decompositions of the CORRUPT-DATA task enable island-hopping through the network, compromising other components so as to reach to target. Note that such sequences of attack actions are achieved by recursive definition of the HTN method on the righthand side of the figure.

Once a set of possible attack goals are generated by considering subtask priority in a mission specification, further reasoning estimates the likelihood of these events as we observe some initial set of attack events occuring. We assume that observed events correspond to the abstract attack actions that RAPPA uses to reason about threats. For example, the COMPROMISE-COMPONENT attack action shown in the hierarchy of Figure 3 is also an event that we can indirectly observe in the network by fusing evidence from sensor reports. STRATUS' MIFD component does exactly that.

As observations are obtained from the sensors in the network, and fused by MIFD into hypotheses about the attack events that may have occurred, SPAR uses RAPPA's attack plan and goal libraries to identify which components in the network are threatened, and estimates the likelihood of possible attacks on those components. STRATUS uses that information to help decide where additional backups or other defense maesures might be needed. For example, in Figure 2, if it is observed that Surveillance-Img-Analyst machine is compromised (shown as circled at the lower right portion of the network), then among the many alternative possible attack goals that SPAR considers, three can are deemed more likely than others, namely IRCServer, PlanServer, UAV Image Hi-Resolution Processing server. The rationale for this is that IRCServer is a central component in the network and the other two directly affect how the top-level mission how mission objectives the mission are achieved. These components are given back-ups so that they can be quickly restored if one of those eventualities occurs.

The subsequent sections provide the technical details on RAPPA and SPAR.

3. Adversarial Planning for Reasoning about Attack Goals

When a plan is to be executed in a cyber environment, it has be to protected against potential threats, which may disrupt or prevent the execution of a mission's goals. We developed techniques for recognizing critical attacks and their goals, which typically cause such disruptions, by combining diverse planning, hierarchical learning, and conditional threat reasoning: RAPPA (Rapid Attack Plan Path Analyzer) develops a diverse set of attack plans to prime the plan recognition component, SPAR (Simple Probabilistic Attack Recognizer) (see Section 4), and simultaneously identifies a set of mission-specific attack targets to inform the decision model used to allocate resources to the mission.

3.1 Attack Scenario Modeling in PDDL

We can characterize attacks abstractly as attempts to violate three primary security goals. In the following list, we give the security goal, and the countervailing attack goal/task:

- (C) Confidentiality / exfiltration;
- (I) Integrity / corruption; and
- (A)Availability / denial of service.

We will refer to these as the CI&A goals. Some additional security goals that are sometimes formulated include accountability, authenticity, non-repudiation, and reliability. These may be thought of as partial decompositions of the top three.

Vulnerabilities translate to actions available to the attacker. We use PDDL – Planning Domain Description Language – models for such attack actions. Consider an action example, from our motivating scenarios discussed in the previous section:

```
(:action compromise-component
   :parameters (?ch - channel ?task - task
                ?target-comp ?source-comp - component
                ?target-host - host)
   :precondition
     (and (runs-on ?target-comp ?target-host)
          (runs-on ?source-comp ?source-host)
          (component-task ?target-comp ?task)
          (can-publish ?source-comp ?ch)
          (subscribed ?target-comp ?ch)
          (pwned ?source-comp)
          ;; additional items
          (machine-os ?target-host WINDOWS))
   :effect (and (pwned ?target-comp)))
```

Through its effect and precondition expressions, this action model describes that an attacker that controls (pwned) a component, source-comp, can gain control of a target component target-comp if source-comp can publish to a channel, ch, to which target-comp subscribes, as long as the target-comp is running on Windows. This action allows an attacker to establish control over a component, recorded as (pwned component). The establishment of control allows the attacker to do "island-hopping," establishing more and more control until she can reach her real target(s). The real targets, recall, are to compromise one of the CI&A security goals. We have developed PDDL models of different attacker actions for all of the CI&A goal categories as above. These models are used in our planning systems that generate *diverse* plans, described below.

3.2 Diverse Plan Generation

Given a mission model and a network specification, RAPPA treats every possible condition that the mission's execution might depend on as a potential attack goal. Thus, RAPPA we initially considers a disjunctive goal, in which each disjunct simply is the negation of a mission condition. SPAR, later on, generates a conditional probability ranking over this disjunction, which can be used to focus on only a small subset of the disjuncts in the original goal expression, eliminating those attack goals that are not likely.

Given a model of possible attack actions, RAPPA's objective is to generate attack plans that are *interestingly-different* than each other so as to provide the recognizer with as broad as possible a set of attack plans it might recognize. This set should ideally *cover* the possible attack space to be able to reason, plan, learn, and recognize the widest set of attacks.

We developed a set of new algorithms for diverse planning, each of which translates the diversity metric into a cost measure for planning alternatives and describes how to update iteratively the costs of the actions in the domain model, in order to generate diverse plans. This approach allows *any* classical planner to generate diverse plans, without modifying the planner, as long as the classical planner can reason with simple numeric costs on actions. For this, we integrated LPG-d (Srivastava et al., 2007) for diverse planning in RAPPA. LPG-d also serves as a baseline diverse planner for our ongoing experiments .

For measuring diversity, we have started with existing plan-plan distance measures developed previously:

- **Action Counting.** This is the most common diversity measure used by the existing works mostly because it is the most easy one to compute as a search control mechanism. Basically, one uses the length of the plan in number of ground actions as the similarity measure (Nguyen et al., 2012).

- **Edit Distance.** The *edit distance (a.k.a., Levenshtein Distance)* between two strings is defined as the minimum number of edits needed to transform one string into the other, with the allowable edit operations being insertion, deletion, or substitution of a single character (`http://en.wikipedia.org/wiki/Levenshtein_distance`).

We demonstrated that these diversity measures can provide inconsistent results on the same plans. Based on this observation, we developed a plan-plan distance metric based on Kolmogorov (Algorithmic) complexity, where we define the diversity of plans in terms of how surprising one plan is given another or, its inverse, the conditional information in one plan given another (Goldman & Kuter, 2015). Kolmogorov complexity provides a domain independent theory of conditional information. While Kolmogorov complexity is not computable, a related metric, Normalized Compression Distance (NCD), provides a well-behaved approximation. Thus, we introduced NCD as an alternative diversity metric, and analyzed its performance empirically, in comparison with previous diversity measures, showing strengths and weaknesses of each. We also examined the use of different compressors in NCD.

In (Goldman & Kuter, 2015), we also demonstrated how NCD can be used to select a training set for HTN learning, giving an example of the utility of diversity metrics. The details of this work can be found in (Goldman & Kuter, 2015); however, this result forms the foundation of our approach in RAPPA: by generating diverse sets of attack plans and goals, and categorizing them into hierarhical representations, RAPPA is able to provide a reasonable coverage to the potential vulnerabilities and attack events in a computer network.

3.3 Inferring Plan Libraries for Recognition

RAPPA uses a combination of techniques to translate attack plans it generates into the models needed for attack plan recognition and countermeasure planning, building on our previous work on HTN-Maker (Hogg, Muñoz-Avila, & Kuter, 2008; Hogg, Kuter, & Muñoz-Avila, 2009; Hogg, Kuter, & Muñoz-Avila, 2010), for generalizing linear plans into hierarchical task networks. HTN-Maker generalizes from single examples of attack plans by recognizing partial orders: places where actions in the plan can be executed in multiple different orders. It will also recognize the underlying goals that explain why actions have been incorporated in the attack plans.

To consider multiple concurrent and interleaved attacks, we developed PDDL models that can support this kind of planning and showed the ability of RAPPA to generate diverse plans with those models. These test domains revealed a crucial shortcoming in the RAPPA architecture with respect to the plan library generation. In particular, although HTN-Maker was able to generate plan libraries as shown in Figure 3 for simple attack formulations, the algorithm was limited to learning strictly less expressive grammars than needed for recognition in our scenario domains.

To alleviate this shortcoming, we developed a concept learning algorithm, called Anchored Hierarchical Learner (AHL), for RAPPA that produces attack categories for recognition. In particular, given attack plans, AHL first attempts to identfy possible plan anchors in those plans, by extracting the causal structure of an attack plan given the PDDL models of the attack actions and the particular attack goals the plan accomplishes. It then generates the causal structure for every attack plan in its input diverse plan set. Then, the algorithm walks over these causal structures and compares them to identify those actions, called *anchors* (Geib, 2009; Geib & Goldman, 2011), that are highly distinctive in those plans.

Once the anchors are identified, AHL starts from the plan anchor actions in the attack plan, and infers a plan library around those anchors successively. Given a plan anchor, the algorithm first learns its parent and possible siblings. The anchor action and siblings constitute a portion of the plan from which the system learns. The learner then replaces this portion with the parent learned for the anchor action, creating an abstract plan, and marks that parent as the new anchor in that abstracted plan. The learning process continues in this manner until no new parents can be learned (or if given, the root nonprimitive task is reached).

We are currently testing this implementation and investigating ways to inductively generalize the hierarchical structures learned to produce more efficient plan libraries for recognition.

4. SPAR: Simple Probabilistic Attack Recognizer

SPAR is a new probabilistic recognition algorithm we developed in order to probabilistically recognize the potential goals of an attack as it progresses. SPAR attempts to quickly but directly estimate the conditional probability of attacks on all mission components, given the observations seen so far.

Algorithm 1 shows a high-level description of SPAR. The input includes a plan library Π of possible attacks (generated by RAPPA). D is the planning domain which describes the set of planning operators. G is the set of *probabilistic goals*, i.e., a partial mapping from possible attack goals in the network to probability values. O is the set of *probabilistic observations*, i.e., a partial mapping from possible intrusion event hypotheses to probability values.

Some explanations about G and O are in order. As in attacks in the physical world, some components in a cyber network are higher value targets compared to others. SPAR can take as input such information as a priori beliefs on how likely it is that a network component will be attacked. The input G models these beliefs as probability values and the algorithm incorporates this probability into its conditional estimation for that component as a possible attack goal. Note that the probabilistic belief over goals does not necessarily specify a probability distribution over

Algorithm 1: The SPAR algorithm.

```
Procedure SPAR($\Pi, G, O, D$);
begin
    $\mathcal{P} \leftarrow \emptyset$;
    foreach plan $\pi \in \Pi$ do
        foreach action $a \in \pi$ do
            $pos \leftarrow$ position index of $a$ in $\pi$;
            if $\exists o \in O$ such that $o \models a$ then
                $apriori \leftarrow p_0$;
            else
                $apriori \leftarrow 0.5$;
            $cond\_aposteriori \leftarrow apriori \times$ expdecay$(pos, |\pi|)$;
            insert $(a, \pi, cond\_aposteriori)$ into $\mathcal{P}$;
    foreach goal $(g, p_g) \in G$ do
        $C \leftarrow \{$all of the conditions $(a, \pi, p) \in \mathcal{P} | \pi$ achieves $g\}$;
        $probability(g) \leftarrow$ noisyOR $(C) \; \times p_g$;
        insert $(g, probability(g))$ into $\mathcal{T}$;
    return $\mathcal{T}$;
end
```

the network components. Instead, the probability values attached to some of the components model *weights*, denoting how much the user believes a component is a likely target of an attack. If the input does not specify such a probabilistic weight for a component, SPAR uses 0.5 (no bias) as a default weight for that component.

Given this input, SPAR iterates over every attack plan π in the plan library generated by RAPPA. SPAR checks which actions of a plan were observed in the network. If an action was observed then SPAR uses the probability value specified in the input as an a priori belief in that observation. If no such probability is given, SPAR again uses 0.5 as default, which models that the algorithm does not have a bias towards whether the observation really occurred or not.

Once SPAR decides on a value for the likelihood of the observation, it uses an exponential decay function to propagate that probability value over the rest of the actions in the attack plan. We used exponential decay functions for probabilistic inference in this manner since the plan libraries are causal pathways in a graphical model for an attack surface and they do not include conditional probability information. If we have conditional probabilities (i.e., a Bayesian Network model), then this computation can be performed by Bayesian inference and/or chain rule. By default, SPAR uses a decay rate of 0.9; but this is a hyper-parameter that can be tuned empirically based on the attack planning domain.

SPAR then computes the probabilities, propagated from the observation to the end action (i.e., goal) of the plan, for each plan. Then, the algorithm extracts those plans for each given goal as input and uses a Noisy-OR computation to aggregate the propagated probabilities over the set of plans that achieve the same goal. The outcome of this Noisy-OR operation is an estimate for the conditional probability of that attack goal will happen, given the set of observations.

5. Preliminary Evaluations and Experiments

We conducted several preliminary experiments with plan recognition using the outcomes of RAPPA's diverse planning and plan library learning capabilities, coupling them with SPAR. In all of our experiments, we used LPG-d (Srivastava et al., 2007) as our diverse planner within RAPPA. We used the STRATUS scenarios described in Section 2 for our experiments. We varied the size of the input set of observations, in such a way that we started by the earliest possible observation in an attack and incrementally added new observations in the order they would be observed in an attack execution.

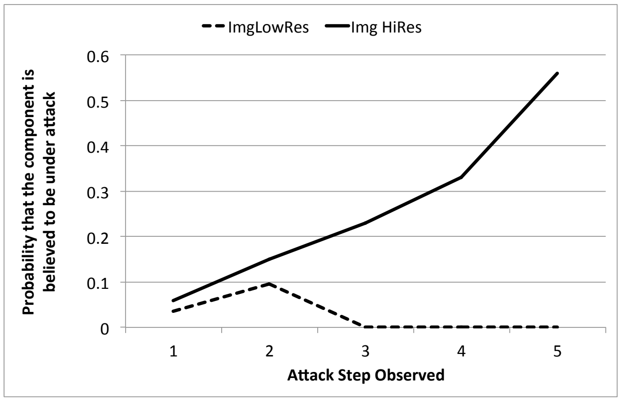


Figure 4. Probability of recognition of low-value vs. high-value attack targets, as a function the progress of the attack.

Figure 4 shows the results of an experiment exploring how SPAR differentiates between high- vs low-value targets given the a priori weights on those as the attack progresses. In this case, an attack on a Hi-Resolution Image Server is considered a more valuable target than the one with Low Resolution. In the first two steps of the attack, observations do not provide significant distinction between the two components, and SPAR believes both components could be targeted. As the attack progresses, and the evidence piles up towards Hi-Resolution Server, SPAR is able to differentiate and increases its beliefs that the Hi-Resolution Image Server is much more likely to be attacked, as opposed to the other component.

Figure 5 shows the results for a similar experiment but the targets in consideration in this case reside in different clouds in the network.

Finally, we observe that SPAR is fast – on average its run times are within a few milliseconds for a plan library of size 1296 attack plans, for each event hypothesis. The reason for its efficiency is because it does not attempt to explain the conditional probabilities; instead, it approximates them via statistical reasoning over input attack plan libraries.

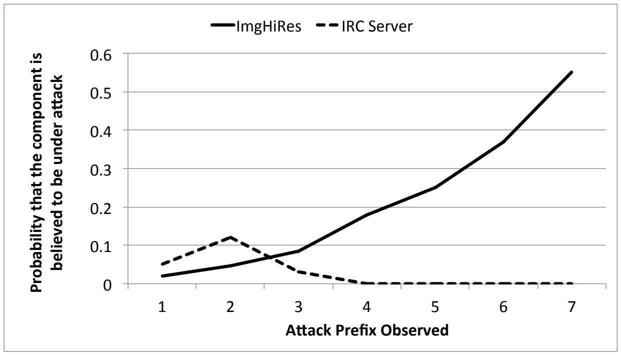


Figure 5. Probability of recognition of an attack target as a function of the direction of the attack in the network.

6. Related Work

6.1 Diverse Planning

The most influential work on plan diversity measures is that of Srivastava, et al (Srivastava et al., 2007), refined in Nguyen, et al (Nguyen et al., 2012). They propose three different distance measures for comparing plans, and for measuring the diversity of a set of plans: action distance (AD), causal-link distance (CLD), and state distance (SD). AD and CLD both project the plan, an *ordered* set of actions, down to an unordered set, and then compute a set-difference based distance between these sets.

Nguyen et al., actually provide *two* alternative definitions for state distance, which differ slightly in how they handle a difference between $l(p)$ and $l(p')$; for details see their paper. They provide two planning algorithms: GP-CSP, which can generate plans that attempt to maximize either action distance, causal-link distance, or state distance; and a more efficient method, LPG-d, which only uses action distance. Using GP-CSP, they provide experimental results on several domains to argue that action distance is the hardest to maximize. In general, later work in diverse planning confines itself to using action distance; we don't know of other work that uses CLD or SD.

The measures defined by Nguyen et al all have some problems. While they have the advantage of being domain-independent, no strong motivation is given, aside from their ready computability. Another problem is that they are not plan distance metrics, in the mathematical sense. Neither action distance nor causal-link distance satisfy the identity property, since different plans can give rise to the same action and causal link sets through reordering (note that action distance and causal-link distance *are* metrics over action and link sets, just not over plans). Similarly, in some problems it is possible for two different action sequences to give rise to the same state sequences. Action distance does not take into account information in the lifted representation of a domain. To action distance, the plans $p_1 =$ *drive(t1, a, b), drive(t1, b, c)* and $p_2 =$ *drive(t2, a, b), drive(t2, b, c)* look every bit as different as p_1 and $p_3 =$ *fly(a1, a, g), drive(t3, g, c).* Causal link and state distances also fail to take such generalizations into account. Further, in the case of state distance, correlated state fluents – fluents that change in lockstep – can artificially drive up the state difference between plans.

Other existing approaches to diverse plan generation either used intensive domain specific knowledge (e.g., (Myers & Lee, 1999; Coman & Muñoz-Avila, 2011)) or they are purely domain independent (e.g., (Boddy et al., 2005)). In many practical applications, it is hard, and sometimes not possible, to author the domain theories the former approaches require, and the latter typically suffers from shallow diversity metrics that may have pathological behavior in comparing plans.

Conceptually, the opposite of diversity is *similarity*. Similarity analyses has been the foundation of case-based reasoning from the field's beginning days (Aamodt & Plaza, 1994).Recently, the works reported in (Sánchez-Ruiz & Ontanón, 2014; Vattam, Aha, & Floyd, 2015) propose plan recognition approaches using case-based similarity measures. These works are closely related to our purposes in this paper: (1) although we have not reported in this paper, we are interested in hierarchical conceptualization and categorization of possible attacker behaviors in our scenarios and (2) case-based similarity measures provide a similar conceptualization as diversity measures as in RAPPA. In fact, we have recently started to develop a case-based similarity, or more generally, an analogical hierarchical planning, reasoning, and recognition approach for our next steps in RAPPA and SPAR. We will conduct an extensive comparative study for this work and the existing research in case-based reasoning.

6.2 Plan Recognition

Kautz' foundational, graph covering based work on plan recognition (Kautz, 1991), in fact did not assume perfect observations, but instead fit the best vertex cover to the plan graph. However, this makes it unable to address a number of issues that SPAR does address including multiple interleaved goals. Other early work using logic based reasoning algorithms (Carberry, 1990; Litman, 1986), while invaluable for formalizing the kinds of inferences that are necessary for efficient plan recognition. However, being logic based, they were not amenable to extension with probabilities.

ELEXIR (Geib & Steedman, 2007; Geib, 2009; Geib & Goldman, 2011) follows the early work of (Vilain, 1990) on plan recognition as parsing. However this early work does not actually present an algorithm or implemented system for plan recognition. Pynadath and Wellman (Pynadath & Wellman, 2000) formalize plan recognition based on probabilistic context free grammars (PCFGs). They use the structure of plans captured in a PCFG to build a fixed Dynamic Bayes Net (DBN), an use the resulting DBN to compute a distribution over the space of possible plans that could be under execution.

There are several recent works on probabilistic plan recognition generalized the foundational works by (Ramırez & Geffner, 2009; Ramırez & Geffner, 2010; Ramirez & Geffner, 2011) on planning for probabilistic goal recognition. The current very successful work on probabilistic activity recognition using HMMs and CRFs (Hoogs & Perera, 2008; Liao, Fox, & Kautz, 2005; Vail & Veloso, 2008) is probabilistic however it is addressing a different problem than the plan recognition problem. These systems are looking for a single label for a sequence of observations. (Ramirez & Geffner, 2011) have proposed using POMDPs that would directly address the partial observability of the world state. Like theHMMs and CRFs they attempting to infer a single high level goal that is the objective of an agent specified by a POMDP.

Targeting planning and human-robot coordination areas for probabilistic plan recognition specifically, the plan recognition approaches introduced in (Chakraborti et al., 2015a; Chakraborti et al., 2015b; Talamadupula et al., 2014) build on the observation that probabilistic plan recognition typically does not commit to a plan, which pre-assumes a particular plan for the other agent, and therefore, it might be possible to minimize suboptimal (in terms of redundant or conflicting actions performed during the execution phase) behavior of the autonomous agent. Our approach is similar to these works in that RAPPA's diverse plan libraries provide a basis for us to tone down the redundant and conflicting actions in the plans and therefore, SPAR's probabilistic reasoning mechanisms are not affected by them. However, a more deeper comparison with RAPPA/SPAR and these new ground-breaking works are necessary and in order.

6.3 Learning

RAPPA's AHL algorithm described in the paper uses, in essence, a variant of chart parsing (e.g., (Charniak, Goldwater, & Johnson, 1998)) for learning the hierarchy of the plan library from an attack plan. 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.

HTN-Maker (Hogg, Muñoz-Avila, & Kuter, 2008) uses explanation-based reasoning techniques to learn the method's hierarchy and preconditions while HTN-Learner uses constraint-satisfaction techniques for this purpose. They both require the task semantics to be given in the form of (preconditions,effects) pairs. These systems exhibit characteristics beyond the scope of this paper: HTN-Maker and variants are able to learn in domains were actions have non-deterministic effects and take into account plan quality factors. HTN-Learner (Zhuo et al., 2009), a variant of HTN-Maker, is also able to learn the operators' preconditions and effects under partial state observability (i.e., as usual for most learners, the input is a collection of plan traces annotated with the intermediate states; in HTN-Learner's case these states might have missing information).

Camel (Ilghami et al., 2005) assumes that not only the annotated plan traces is given but it also requires two more inputs: the complete task structure and samples of incorrect uses of the task structures. Camel uses these inputs and a version space algorithm to learn the method's preconditions. Another HTN learner in this category is DiNCAT (Xu & Muñoz-Avila, 2005) (not shown in the table). Like Camel, it requires the hierarchical structure to be given as input. Unlike Camel it does not require the incorrect uses of the hierarchy to be given as input; it uses case-based generalization techniques to learn the method's preconditions.

ICARUS (Langley & Choi, 2006) and its variant LIGHT (Nejati, Langley, & Könik, 2006) learn both method's preconditions and the hierarchical structure. It uses teleoreactive learning techniques whereby STRIPS planning is used to fill gaps in the knowledge of the methods; when the methods cannot be used to solve a subproblem, STRIPS planning is used to solve the subproblem and methods are learned from the subplan generated. Icarus and LIGHT require the skill definitions to be given as input. These skills provide the semantics of the tasks, which are STRIPS goals.

Algorithms for learning grammars can be capable of generating hierarchical structures if we map the input plans into strings (i.e., mapping the plan's actions into the string's symbols) and the tasks into the grammar's variables. The grammar learned could be used to generate the hierarchical structures (i.e., by mapping the grammar's parse trees to HTNs). Grammar learning algorithms (Oates, Desai, & Bhat, 2002; Sakakibara, 1997) generate a grammar that best maps the input strings. These techniques are exploited by the Greedy Structure Hypothesizer (GSH), which uses probabilistic context-free grammars learning techniques to learn a hierarchical structure of the input plan traces. This hierarchy reflects user's preferences. GSH does not learn preconditions since its goals are not to generate the grammars for planning but to reflect user's preferences. In general, grammar learning algorithms does not consider applicability conditions. Furthermore, the learned hierarchical structure is a function of commonalities between subsequences in the input traces, regardless of which tasks they achieve.

X-Learn (Reddy & Tadepalli, 1997) uses bootstrapping learning techniques in which first simple traces are given to learn methods to achieve these traces. Progressively more complex traces are given where the knowledge learned in previous iterations can be exploited to simplify the trace and learn new task decompositions. Tasks correspond to one of the goals that is achieved at the end of the trace. Hence, its task semantics are defined as the usual STRIPS semantics for goals.

7. Conclusions and Future Work

We have described a system composed of two sister algorithms, RAPPA and SPAR, for reasoning and recognizing attacker goals in cyber systems. Although such goals are not necessarily part of the goals of a mission itself, reasoning about them enables mission resilience since we can recognize and respond to those attack goals in advance. We have presented a preliminary evaluation of the two algorithms, demonstrating that they are promising for the cyber scenarios of our interest.

We currently working on a new approach which we call R2S2 (RAPPA2 / SPAR2). Unlike action-based diverse planning techniques in RAPPA, our new approach builds a hierarchical planning graph, in which actions could be either primitive or abstract. In the latter case, an abstract action represents a set of plans that are semantically equivalent. Thus, we reduce and replace our diverse planning capability to generating this compact conceptual planning graph, which represents the set of all possible diverse abstract plans in a planning domain.

SPAR2 algorithm then performs a backward breadth-first search over this graph, not only generating conditional probabilities for possible attacker goals, but also explaining those conditional probabilities causally. We believe that this recognition approach using hierarchical plan graphs, will also address a limitation in SPAR: larger plan libraries containing longer plans poses a tractability issue for SPAR to deal with its joint probability distributions. Hierarchical plan graphs, on the other hand, as compact and abstract representations of such plan libraries, will naturally alleviate this issue for SPAR2. This is also important for another direction with SPAR2: reasoning with partially-observable observation states.

Acknowledgments. Thanks to the entire STRATUS project and our funders. This work was supported by Contract FA8650- 11-C-7191 with the US Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory. Approved for public release, distribution unlimited. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

Aamodt, A., & Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, *7*, 39–59.

Boddy, M. S., Gohde, J., Haigh, T., & Harp, S. A. (2005). Course of action generation for cyber security using classical planning. *International Conference on Automated Planning and Scheduling* (pp. 12–21). AAAI.

Burstein, M., Goldman, R., Robertson, P., Laddaga, R., Balzer, R., Goldman, N., Geib, C., Kuter, U., McDonald, D., Maraist, J., Keller, P., & Wile, D. (2012). Stratus: Strategic and tactical resiliency against threats to ubiquitous systems. *Proceedings of SASO-12*.

Carberry, S. (1990). Plan recognition in natural language dialogue. *ACL-MIT Press Series in Natural Language Processing*.

Chakraborti, T., Briggs, G., Talamadupula, K., Scheutz, M., Smith, D., & Kambhampati, S. (2015a). Planning for serendipity. *ICAPS Workshop on Planning and Robotics*.

Chakraborti, T., Zhang, Y., Smith, D., & Kambhampati, S. (2015b). Planning with stochastic resource proïňĄles: An application to human-robot co-habitation. *ICAPS Workshop on Planning and Robotics*.

Charniak, E., Goldwater, S., & Johnson, M. (1998). Edge-based best-first chart parsing. *Association for Computational Linguistics*, 127–133.

Coman, A., & Muñoz-Avila, H. (2011). Generating diverse plans using quantitative and qualitative plan distance metrics. *AAAI*.

Geib, C. (2009). Delaying commitment in probabilistic plan recognition using combinatory categorial grammars. *IJCAI-09*.

Geib, C., & Goldman, R. (2011). Recognizing plans with loops represented in a lexicalized grammar. *AAAI-11*.

Geib, C., & Steedman, M. (2007). On natural language processing and plan recognition. *IJCAI-2007*.

Goldman, R. P., & Kuter, U. (2015). Measuring plan diversity: Pathologies in existing approaches and a new plan distance metric. *AAAI/IAAI-15*.

Hogg, C., Kuter, U., & Muñoz-Avila, H. (2009). Learning hierarchical task networks for nondeterministic planning domains. *IJCAI-09*.

Hogg, C., Kuter, U., & Muñoz-Avila, H. (2010). Learning methods to generate good plans: Integrating htn learning and reinforcement learning. *AAAI-10*.

Hogg, C., Muñoz-Avila, H., & Kuter, U. (2008). HTN-MAKER: Learning HTNs with minimal additional knowledge engineering required. *AAAI-08*.

Hoogs, A., & Perera, A. A. (2008). Video activity recog- nition in the real world. *AAAI-08*.

Ilghami, O., Nau, D. S., Muñoz-Avila, H., & Aha, D. W. (2005). Learning preconditions for planning from plan traces and HTN structure. *Computational Intelligence*, *21*, 388–413.

Kautz, H. (1991). *A formal theory of plan recognition and its implementation*. Doctoral dissertation, University of Rochester.

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

Langley, P., & Choi, D. (2006). Learning recursive control programs from problem solving. *Journal of Machine Learning Research*, *7*, 493–518.

Liao, L., Fox, D., & Kautz, H. (2005). Location-based activity recognition using relational markov networks. *IJCAI-05*.

Litman, D. (1986). Understanding plan ellipsis.

Myers, K. L., & Lee, T. J. (1999). Generating qualitatively different plans through metatheoretic biases. *Proc. National Conf. on Artificial Intelligence (AAAI)*.

Nejati, N., Langley, P., & Könik, T. (2006). Learning hierarchical task networks by observation. *ICML*.

Nguyen, T. A., Do, M. B., Gerevini, A., Serina, I., Srivastava, B., & Kambhampati, S. (2012). Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence*, *190*, 1–31.

Oates, T., Desai, D., & Bhat, V. (2002). Learning k-reversible context-free grammars from positive structural examples. *ICML '02: Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 459–465). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Pynadath, D., & Wellman, M. (2000). Probabilistic state-dependent grammars for plan recognition. *UAI-00*.

Ramırez, M., & Geffner, H. (2009). Plan recognition as planning. *Proceedings of the 21st international joint conference on Artifical intelligence. Morgan Kaufmann Publishers Inc* (pp. 1778–1783).

Ramırez, M., & Geffner, H. (2010). Probabilistic plan recognition using off-the-shelf classical planners. *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI 2010)*.

Ramirez, M., & Geffner, H. (2011). Goal recognition over pomdps: Inferring the intention of a pomdp agent. *AAAI-11*.

Reddy, C., & Tadepalli, P. (1997). Learning goal-decomposition rules using exercises. *Proc. International Conf. on Machine Learning (ICML)*.

Sakakibara, Y. (1997). Recent advances of grammatical inference. *Theoretical Computer Science*, *185*, 15–45.

Sánchez-Ruiz, A. A., & Ontanón, S. (2014). Least common subsumer trees for plan retrieval. In *Case-based reasoning research and development*: Springer.

Shrobe, H. E. (2002a). Computational vulnerability analysis for information survivability. *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI-02)* (pp. 919–926). Menlo Parc, CA, USA: AAAI Press.

Shrobe, H. E. (2002b). Computational vulnerability analysis for information survivability. *AI Magazine*, *23*, 81–91.

Srivastava, B., Nguyen, T. A., Gerevini, A., Kambhampati, S., Do, M. B., & Serina, I. (2007). Domain independent approaches for finding diverse plans. *Proceedings of the International Joint Conference on Artificial Intelligence*.

Talamadupula, K., Briggs, G., Chakraborti, T., Scheutz, M., & Kambhampati, S. (2014). Coordination in human-robot teams using mental modeling and plan recognition. *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on* (pp. 2957–2962).

Vail, D. L., & Veloso, M. M. (2008). Feature selection for activity recognition in multi-robot domains. *AAAI-08*.

Vattam, S., Aha, D., & Floyd, M. W. (2015). Error tolerant plan recognition: An empirical investigation. *The Twenty-Eighth International Flairs Conference*.

Vattam, S., Klenk, M., Molineaux, M., & Aha, D. W. (2013). *Breadth of approaches to goal reasoning: A research survey* (Technical Report). DTIC Document.

Vilain, M. B. (1990). Getting serious about parsing plans: A grammatical analysis of plan recognition. *AAAI-90*.

Xu, K., & Muñoz-Avila, H. (2005). A domain-independent system for case-based task decomposition without domain theories. *AAAI-05*.

Zhuo, H. H., Hu, D. H., Hogg, C., Yang, Q., & Munoz-Avila, H. (2009). Learning htn method preconditions and action models from partial observations. *Proc. Internat. Joint Conf. on Artificial Intelligence (IJCAI)*.

PlanBot: A Replanning Agent for Starcraft

Michael Leece MLEECE@UCSC.EDU
Arnav Jhala AJHALA@UCSC.EDU

Abstract

State-of-the-art agents in the real-time strategy game domain currently rely on hand-crafted scripted strategies, and as a result are inflexible, responding poorly to unexpected events brought on by the actions of their adversary. A self-introspective agent with goal reasoning could overcome this challenge, allowing an agent to consider how its strategy has succeeded or failed and what must be modified as a result of this. We demonstrate such an agent in its initial development, and show that it performs well against weak to medium strength opponents, though still falls short against well-scripted play. In addition, we discuss possible extensions and research areas that can be explored with this base system.

1. Introduction

Real time strategy (RTS) games have been growing in popularity as a challenge domain for artificial intelligence research. They are abstract economic and military simulations in which players allocate resources to various infrastructure development, and attempt to steer the game toward the strengths of their choices and away from the weaknesses. As simulations, they contain many problems that arise when working in the real-world in adversarial environments, such as the need for flexible and reactive decision-making, in addition to the requirement to plan ahead and not simply always react to an opponent's actions.

Unfortunately, many of the current state-of-the-art agents use scripted logics or small state machines for high-level strategic decisions, leaving them rigid and poorly able to adapt to new situations and opponents. The progression from inflexible scripted strategies to systems that reflect human intelligence and rational decision-making has been a common sight in past challenge domains, and is where we hope to go with this work.

We have created a planning agent for the RTS game Starcraft:Brood War (see section 2), which uses replanning as a form of goal reasoning in order to adapt its plan to unexpected changes in the environment due to inaccurate beliefs about the world state or active opposition from an adversary. In the future, we hope to use this system as a base from which to explore more complex goal reasoning approaches, adversarial planning, and learning hierarchical models from demonstration.

2. StarCraft:Brood War

Starcraft:Brood War (SC:BW) is a well-known RTS game produced by Blizzard Entertainment. At a very high level, gameplay consists of allocating resources to grow an economy and military infrastructure, then using this infrastructure to train an army to destroy your opponent's troops and buildings. It can be played with up to eight players, but competitive games traditionally feature

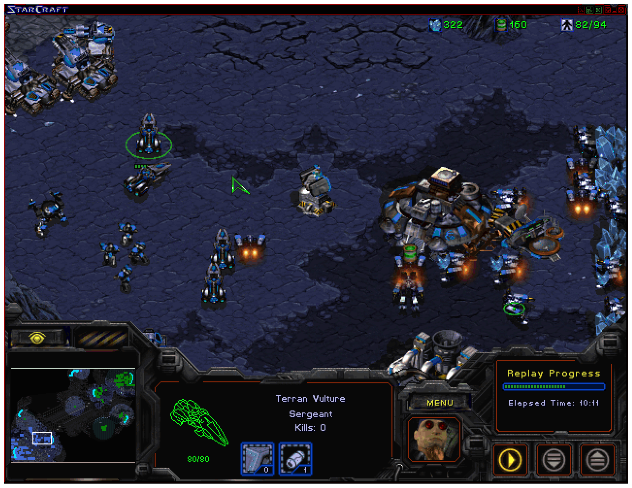


Figure 1. An example screenshot from SC:BW. Shown is a base with mining worker units, and a portion of the player's army.

two players in a head-to-head match. While seemingly simple at first, the interaction of resource management, terrain considerations, and asymmetric army compositions results in a rich space of strategies and tactics for players.

In recent years SC:BW has grown in popularity as a research testbed, due to a number of desirable properties. It is a real-time environment (or near-real-time, at 60 possible decision points per second), which is a departure from previous AI challenge games such as Chess or Go. In addition, it is a game of imperfect information. Only areas of the map that are visible to a player's units will be revealed to that player, which means that one is never completely aware of how an opponent is allocating their resources. The state and action spaces of the game are very large, a rough calculation of the state space estimating it to be $10^{11,500}$ (Weber, 2012). This calculation doesn't take into account information states, which would increase it even more.

The conjunction of these properties means that any agent that wishes to play the game well must be able to make on-the-fly decisions adapting to new information about an opponent's strategy and the world state, since it is clearly infeasible to precompute a full state transition strategy. These requirements are common to real-world problems and still challenging to artificial agents, which is exactly what we desire in a testbed.

An added benefit of this specific game is its popularity. At its peak, professional leagues existed with players competing regularly in tournaments with tens of thousands of dollars in cash rewards. In combination with the ability to save and post replays, this has resulted in large archives of expert-level play, which can be used as demonstrations for learning systems.

For possible evaluations, multiple tournaments for SC:BW agents are held each year (Churchill, 2013). Additionally, while much of the professional scene has moved on to more modern RTS games, the game remains popular enough online to have a sustainable test set of human players, with a competitive online ladder system against which an agent's performance can be measured quantitatively.

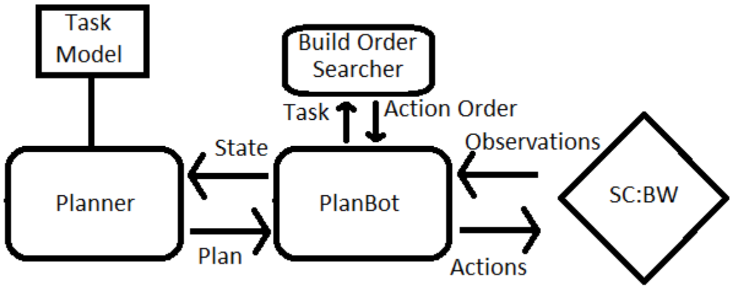


Figure 2. Information flow for PlanBot and associated components

3. System

Our system (called PlanBot) is built off of a base platform provided by UAlbertaBot (Churchill & Buro, 2012) (Churchill & Buro, 2013). We will give a brief overview of UAlbertaBot and how it operates, then explain the changes that have been made to create our agent.

3.1 UAlbertaBot

UAlbertaBot is a SC:BW agent developed at the University of Alberta. One of the main goals of the system has been to provide an entry point for developing SC:BW agents that is modular and easily modifiable. Control has been broken into a number of 'managers', which handle internal state for the agent, and 'commanders', which interact with the game itself. For example, the main actor is the 'GameCommander', which queries the 'StrategyManager' for the next set of units and buildings that the agent should train/construct. Given that set, it passes it on to a build order searcher, which calculates an order in which the set of units should be constructed, optimizing for time. Other components manage building placement, positioning for fights, and more. Each component uses its own decision-making process, including scripts, simulation, and game-tree searches. Futher information on UAlbertaBot, and the code itself can be found at (Churchill, 2015).

This modular structure, while enforcing certain relations between components, is attractive from an engineering standpoint. For example, a researcher can swap in a new high-level Strategy manager without needing to interfere with the existing decision process for building placement, if it isn't necessary. This has resulted in removing much of the software engineering burden for creating a fully-functioning SC:BW agent, which can be intimidating for researchers new to the domain, or those only interested in specific challenge areas.

3.2 PlanBot

The primary change to create PlanBot is a full replacement of the strategy module in UAlbertaBot, responsible for high-level resource allocation decisions. As mentioned in the example earlier, this part of the agent is queried by the commander module for a list of units and/or buildings that should be constructed next, and then search is performed to determine the optimal order to construct the

	Built-in AI	Bonjwa	UAlbertaBot
PlanBot	98%	72%	12%

Table 1. Winning percentages of PlanBot against a range of opponents.

	Built-in AI	Bonjwa	UAlbertaBot
PlanBot	15:54 ± 1:02	18:31 ± 0:45	10:02 ± 0:36

Table 2. Average game lengths for each opponent.

full set, taking into account prerequisites and resource collection rates. This results in a hybrid of planning and search approaches, where primitive operators in the planning model consist of sets of units to train/construct, the order of which is optimized by the search component. This boundary between search and refining the current plan is a potential point of future investigation. Originally, the strategy module returned a static choice from a set of fixed if-then evaluations of the current state. PlanBot instead uses a modified version of a Python implentation of SHOP[1] (Nau et al., 1999) and slightly modified to handle some of the extra domain complexities. The task model used for planning is handwritten by the authors, and is a candidate for future improvement of the system.

We also modify the commander module, which is responsible for coordinating the other modules, to reflect the changes in the strategic decision-making of the agent. The original agent simply queried the strategy module for a new set of buildings/units as soon as it finishes the last set it was given. Two changes were made to this interaction: first, when the agent is out of things to do and needs direction, it also passes a flag to the strategy manager indicating one of three things. If the last task was executed successfully (all units and buildings were constructed), it requests the next step in the current plan. If the last task was not executed successfully (some units and/or buildings were destroyed while the task was open), we note that this goal has not been satisfied and request for a new task that will achieve it in the current state, then continue with the plan. If the state has changed dramatically from the plan's expectations, the flag indicates that complete replanning should occur. Figure 2 shows this information flow, with the PlanBot node representing the high-level commander module and encapsulating the unchanged modules of UAlbertaBot not discussed here.

As one can see from the description above, the system's goal reasoning capabilities are currently limited to evaluation and replanning if required, while not taking advantage of the full goal lifecycle and opportunities to cycle into different parts of it based on the evaluation of a task's execution. We intend to extend the capabilities of PlanBot to include more complex goal reasoning in the future, as it will likely be necessary in such a complex domain.

4. Evaluation

Our evaluation was based on simple strength of play in head-to-head matches against other agents, and is shown in Table 1. Each opponent was played 100 times over 10 different maps. The built-in AI is, as expected, the AI that is shipped with the game. It generally uses a fairly unoptimized rushing strategy and does not attempt to grow its economy very much. There are three asymmetric races to choose from in the game, and most agents select a specific race to build for, but for the built-in AI we randomized the race, as it is roughly equivalent with each of the three.

Bonjwa is an open-source agent from the 2014 AIIDE tournament, developed by Dustin Dannenhauer. We chose it because it plays the Terran race, while UAlbertaBot (and consequently PlanBot) plays the Protoss race, and we wanted to include an asymmetric matchup with a developed agent. Furthermore, UAlbertaBot is a very strong player, and we wanted a mid-level challenge to measure against as well (according to the rankings from the last open tournament). Bonjwa was forked off an earlier version of UAlbertaBot, and therefore uses the same decision mechanisms, though it has different hand-coded strategies, as it plays a different race. The final opponent, UAlbertaBot, is the unmodified script-based strategy version.

While the results clearly indicate that PlanBot is a capable agent, scoring well against both the built-in AI and Bonjwa, it is also clear that it performs poorly against its own predecessor, winning only 12 games out of 100. In order to gain more insight into what these results actually meant, we watched a number of games from each of the matchups. In general, it seemed that PlanBot did better the longer the game progressed. UAlbertaBot generally executes an optimized rushing strategy that PlanBot was unable to hold off, while as long as it could hold off the built-in AI's initial push, it was eventually able to outmaneuver it. In light of that, we measured the average game times for each opponent, and the results, shown in Table 2, seem to support the observations.

On the bright side, this indicates that our goal of being able to adapt as the situation changes and plans need to be altered is being reflected, but it also means that we are not responding well to early pressure. In all likelihood, this is the result of an incomplete task model due to the authors' incomplete knowledge of the domain, and could be improved with iterated testing and expanding/refining. However, rather than obsessing over improving the win rate, we would like to focus on improving the general intelligence of the system, which we hope to do in the following ways.

5. Future Work

This system is intended to be a base for future research in a number of areas. As mentioned before, the goal reasoning performed by the system currently is of a fairly basic form. We would like to deepen the task evaluation functionality in the commander module to support more fine-grained responses to successful or failed tasks, and the gradations between the two. We believe there are also interesting areas to explore regarding the adversarial nature of the domain, and being able to differentiate between a task failing due to a mistaken belief about the world state or active interference from the opponent, and if or how that should affect the agent's response.

[1] Code found at `https://bitbucket.org/dananau/pyhop`

We also hope to integrate some of our prior research to improve the general intelligence of the system. Human players have capabilities that are not reflected in the current agent, such as general reasoning about what one's opponent is doing and how that should affect one's own strategic choices, or the interaction between strategic choices and tactical choices. Some amount of opponent modeling (Leece & Jhala, 2014a) would give the agent a greater idea of when things are straying away from what was expected when the original plan was generated, and also increase the information available to the planner. This would also help in task evaluation, as it could be performed more accurately mid-task. In addition, we would like to make use of the archives of expert play available online to attempt to learn the task models directly, rather than needing to hand code them, which we have begun work on in (Leece & Jhala, 2014b).

Lastly, we would like to implement some consideration of adversarial planning, similar to the work found in (Meijer & Koppelaar, 2001).

6. Related Work

The strongest predecessor of this project is the work done by Weber et al. on EISBot, a SC:BW agent that used goal-driven autonomy to make its strategic decisions (Weber, 2012) (Weber, Mateas, & Jhala, 2011). Written in the reactive programming language ABL, it used a library of states from human games and case-based reasoning to identify discrepancies and set goals (Weber, Mateas, & Jhala, 2012). Alternatively, Dannenhauer and Muñoz-Avila presented a Goal-Driven Autonomy agent that uses ontologies to identify and explain discrepancies, resulting in a more human-like decision process (Dannenhauer & Muñoz-Avila, 2013). As other projects related to goal reasoning and in the same domain, we hope to compare and contrast our work with these projects moving forward.

Some other work with applying planning to real-time games can be found in (Hoang, Lee-Urban, & Muñoz-Avila, 2005). They combine a strategic planner to direct motion and positioning with a reactive state machine that controls low-level behavior when handed control from the planner. In a more real-world domain, Roberts et al. implemented a planning system that automatically generates finite state machines for heterogeneous teammates based on assigned goals from a coordinating planner (Roberts et al., 2014). This work could be viewed as similar to our system, in breaking plan components into things that are tractable to compute more directly, and will be particularly useful to keep in mind in that it has a better treatment of durative reasoning than our current system.

With regards to our other planned goals for the system, a theoretical basis for learning hierarchical models from demonstrations can be found in (Garland, Ryall, & Rich, 2001) (Garland & Lesh, 2003), and learning the CaMeL system for learning method preconditions (Ilghami et al., 2002) (Ilghami et al., 2005), although these both assumes that some of the structure in the training examples provided has been annotated. Some practical work with entirely unannotated examples, as is available to us, has been done by Hogg et al. in their work on HTN-Maker (Hogg, Munoz-Avila, & Kuter, 2008) (Hogg, Kuter, & Munoz-Avila, 2010). In addition, work has been done within the cognitive systems community itself, due to the fact that most cognitive architectures use some form of hierarchical reasoning or data storage. An example of this can be found in (Ichise, Shapiro, & Langley, 2002).

7. Conclusion

We have created a goal reasoning system for Starcraft:Brood War, a complex adversarial domain which requires flexible and adaptable reasoning. It performs well against the built-in AI, though it is still defeated by well-scripted agents. We hope to extend this agent to use more and more self-introspection regarding its task evaluation process in the future, in addition to using it for other interesting research areas.

References

Churchill, D. (2013). 2013 AIIDE starcraft ai competition report. http://webdocs.cs.ualberta.ca/∼cdavid/starcraftaicomp/ report2013.shtml. Accessed: 2015-04-14.

Churchill, D. (2015). Ualbertabot - starcraft ai competition bot. https://github.com/davechurchill/ualbertabot. Accessed: 2015-03-16.

Churchill, D., & Buro, M. (2012). Incorporating search algorithms into rts game agents. *AI and Interactive Digital Entertainment Conference, AIIDE (AAAI)*.

Churchill, D., & Buro, M. (2013). Portfolio greedy search and simulation for large-scale combat in starcraft. *Computational Intelligence in Games (CIG), 2013 IEEE Conference on* (pp. 1–8).

Dannenhauer, D., & Muñoz-Avila, H. (2013). Luigi: A goal-driven autonomy agent reasoning with ontologies. *ACS: Workshop on Goal Reasoning*.

Garland, A., & Lesh, N. (2003). Learning hierarchical task models by demonstration. *Mitsubishi Electric Research Laboratory (MERL), USA–(January 2002)*.

Garland, A., Ryall, K., & Rich, C. (2001). Learning hierarchical task models by defining and refining examples. *Proceedings of the 1st international conference on Knowledge capture* (pp. 44–51).

Hoang, H., Lee-Urban, S., & Muñoz-Avila, H. (2005). Hierarchical plan representations for encoding strategic game ai. *AIIDE* (pp. 63–68).

Hogg, C., Kuter, U., & Munoz-Avila, H. (2010). Learning methods to generate good plans: Integrating htn learning and reinforcement learning. *AAAI*.

Hogg, C., Munoz-Avila, H., & Kuter, U. (2008). Htn-maker: Learning htns with minimal additional knowledge engineering required. *AAAI* (pp. 950–956).

Ichise, R., Shapiro, D., & Langley, P. (2002). Learning hierarchical skills from observation. *Discovery Science* (pp. 247–258).

Ilghami, O., Munoz-Avila, H., Nau, D. S., & Aha, D. W. (2005). Learning approximate preconditions for methods in hierarchical plans. *Proceedings of the 22nd international conference on Machine learning* (pp. 337–344).

Ilghami, O., Nau, D. S., Munoz-Avila, H., & Aha, D. W. (2002). Camel: Learning method preconditions for htn planning. *AIPS* (pp. 131–142).

Leece, M., & Jhala, A. (2014a). Opponent state modeling in rts games with limited information using markov random fields. *Computational Intelligence and Games (CIG), 2014 IEEE Conference on* (pp. 1–7).

Leece, M. A., & Jhala, A. (2014b). Sequential pattern mining in starcraft: Brood war for short and long-term goals. *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Meijer, A., & Koppelaar, H. (2001). Pursuing abstract goals in the game of go. *BNAICâĂŹ01 Sponsors*.

Nau, D., Cao, Y., Lotem, A., & Muñoz-Avila, H. (1999). Shop: Simple hierarchical ordered planner. *Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2* (pp. 968–973).

Roberts, M., Vattam, S., Alford, R., Auslander, B., Karneeb, J., Molineaux, M., Apker, T., Wilson, M., McMahon, J., & Aha, D. W. (2014). Iterative goal refinement for robotics. *ICAPS Workshop on Planning and Robotics*.

Weber, B. (2012). *Integrating learning in a multi-scale agent*. Doctoral dissertation, UC Santa Cruz.

Weber, B. G., Mateas, M., & Jhala, A. (2011). Building human-level ai for real-time strategy games. *AAAI Fall Symposium: Advances in Cognitive Systems*.

Weber, B. G., Mateas, M., & Jhala, A. (2012). Learning from demonstration for goal-driven autonomy. *AAAI*.

Reasoning and Making Sense of Data in the Absence of Goals

Mary Lou Maher M.MAHER@UNCC.EDU
Software and Information Systems, UNC Charlotte, Charlotte, NC 28202 USA

Kathryn Merrick K.MERRICK@ADFA.EDU.AU
Bing Wang B.WANG@ADFA.EDU.AU
School of Engineering and Information Technology, University of New South Wales, Campbell, ACT, Australia

Abstract

In ill-structured problem domains, it is common to reason about ‘problem-finding’ or ‘hypothesis generation’ before goals can be established. Examples of such domains include design, scientific research and mining of ‘big data’. Problem-finding and hypothesis generation may also continue throughout the problem-solving process, so identifying goals may be an ongoing process of discovery as well as iterative improvement and refinement. This paper considers the design of cognitive systems that are able to reason in the absence of defined goals. We review a range of approaches that may complement goal-directed reasoning when an artificial system does not or cannot know precisely what it is looking for. We argue that there is a spectrum of approaches that can be used for reasoning or making sense of data in the absence of goals.

1. Introduction

AI approaches to cognitive systems assume that explicit representations of goals, rewards and tasks are integral and provide a focus of attention. Most cognitive systems assume that goals are a starting point for reasoning; that reasoning cannot start without goals; and that reasoning ends when there are no goals. In contrast, it is possible to characterize reasoning so that goals become flexible intermediate structures or implied structures, rather than a predefined and fixed starting point (Maher et al., 2011). While the idea of goals as intermediate structures is similar to agent systems that reason about goals, goal formulation, and goal management (for example, Jaidee et al, 2011), we claim that reasoning in the absence of goals is conceptually different and will lead to different cognitive models. By using concepts such as incentive, novelty, difficulty, complexity, curiosity and surprise, cognitively inspired AI models that mimic human behavior in scenarios such as exploratory design, research and lifelong, self-directed learning are possible. The models can be applied to any domain in which ‘problem-finding’ needs to occur during problem solving.

The remainder of this section overviews existing approaches to goal-oriented behavior in cognitive systems. The next section examines a number of complementary approaches that may work in conjunction with goal-directed reasoning, including hypothesis generation, motivation, surprise, novelty and curiosity. We classify these approaches along a spectrum that makes progressively weaker assumptions about the definition and presence of goals. We argue that in some cases goal-oriented behavior is an intermediate result of problem-finding, rather than a starting point for problem solving. In such cases goal-oriented behavior can be an emergent property that does not depend on a predefined definition of domain-specific goals.

1.1 Goal-Oriented Behavior in Cognitive Systems

Langley et al. (2008) survey cognitive architectures and layout nine functional capabilities that are required of a cognitive architecture. While each of these are described in terms of functionality without reference to specific architectures or implementations, almost all assume that goals and tasks are inherent in the description of the functions. This can be seen in the way that Langley et al (2008) describe four examples of cognitive architectures:

- Soar: "**All tasks** in Soar are formulated as attempts **to achieve goals**."
- ACT-R: "ACT-R 6 is organized into a set of modules, each of which processes a different type of information. These include sensory modules for visual processing, motor modules for action, an **intentional module for goals**, and a declarative module for long-term declarative knowledge."
- ICARUS: "ICARUS … stores two distinct forms of knowledge. Concepts describe classes of environmental situations in terms of other concepts and percepts, whereas **skills specify how to achieve goals** by decomposing them into ordered subgoals."
- PRODIGY: "On each cycle, PRODIGY uses its control rules to select an operator, binding set, state, or **goal,** to reject them out of hand, or to prefer some over others. In the absence of such control knowledge, the architecture makes choices at random and pursues depth-first means-ends search with backtracking"

There are multiple models for goals – including goal lifecycles and type taxonomies (Braubach et al., 2005) – and processes for solving goals – including machine learning (Nilsson, 1996), planning and rule-based agents (Russell and Norvig, 1995). Braubach et al., (2005) define a lifecycle for goals in which goals transition from new to adopted and finished.

Braubach et al., (2005) also divide goals into a number of types. Approach goals, for example, define states for which an agent should minimize the difference between its current state and the goal state. In contrast, avoidance goals define states for which an agent should maximize the difference between its current state and the goal state. Achievement goals define changes or events that the agent should cause to occur. Maintenance goals define properties that the agent should hold constant. Other types of goals include optimization, test, query and cease goals.

Dignum and Conte (1998) state that truly autonomous, intelligent agents must be capable of creating new goals as well as dropping goals as conditions change. They distinguish between abstract, high-level goals and concrete, achievable goals. They describe goal formation as a process of deriving concrete, achievable goals – such as 'driving at the speed limit' – from high level, abstract goals – such as 'being good'.

Foner and Maes (1994) develop an agent model of unsupervised learning that can self-determine what facts it should pay attention to as a way of modeling focus of attention. Foner and

Maes (1994) distinguish between goal-driven and world-driven focus of attention. In their model, the agent can determine what sensory data to learn from based on strategies that are derived from world-driven goals, such as what has changed recently and what new data is spatially close. These are domain independent strategies that can reduce the number of possible goals an agent can pursue at any given time.

In general, however, there has been less work on how to represent the high-level, abstract goals or world-driven goals that cause new, concrete goals to emerge. The concept of an abstract goal is difficult to formalize because of the difficulty of representing high-level objectives such as "being good" or "being creative". A number of alternative approaches use models of motivation to take the place of abstract learning goals (Merrick and Maher, 2009; Singh et al., 2005; Kaplan and Oudeyer, 2003; Schmidhuber, 1991). Computational models of motivation have also been proposed as an approach to embedding implicit motives in artificial agents to create agents with different preferences for certain kinds of activities (Merrick and Shafi, 2011; Merrick and Shafi 2013). In a different approach, Barnes and Oudeyer (2010) presented a framework for 'maturationally-constrained self-adaptive goal generation' in which an intrinsic motivation module progressively releases constraints on the learning system. This permits the learning system to explore progressively more widely, through the introduction of new goals.

Other work has studied the role of emotion and other cognitive moderators in artificial systems (Mariner and Laird, 2008). Models of emotion act as modifiers to an agent's goal-oriented behavior or provide abstract goals that can be mapped onto concrete goals.

This paper proposes that reasoning can include reasoning before goals are defined, usually based on the current state of the artificial agent and the state of the world. These approaches are consistent with current cognitive systems because they ultimately lead to goal-oriented behavior, but they complement most cognitive systems because they do not assume that goals are predefined. The next section describes models that fall along a spectrum that make progressively weaker assumptions about the definition and presence of goals.

2. Models that Complement Goal-Directed Reasoning

In creative domains such as design and research, the ill-defined nature of tasks suggests a distinction between search and exploration. Maher et al (1996) characterize the difference between search and exploration by the input and output of these processes as illustrated in Figure 1. A typical search process generates a solution as its output with a well-defined problem (or goal) as its input. However, an exploration process derives a problem and the corresponding solution from an ill-defined problem. Maher et al (1996) expand this idea with a co-evolutionary model of reasoning about the problem space and the solutions space in which goals are expressed as requirements in the problem space that are added and adapted in response to the evolutionary search in the solution space.

If we consider the internal state of an agent to include its goals, then the absence of goals remains a valid state. In many autonomous systems the absence of goals implies idle time, but we envisage that a cognitive system can continue to monitor its environment to discover and pursue self-generated goals that extend or improve its knowledge base or skill set, during this so-called idle time. This type of activity will cause changes in the agent's internal and external environment and create a feedback loop that fosters continuous adaptation.

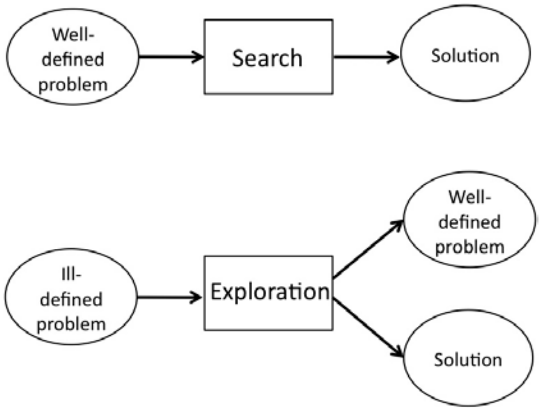


Figure 1. Input and output of search and exploration (Maher et al. 1996).

While ultimately the processes in cognitive systems are organized with the assumption that behavior is goal-directed, we propose that self-directed cognitive systems include the ability to represent, generate, and reason about what Dignum and Conte (1998) call abstract goals. Rather than cast this capability in terms of goals and tasks, however, we identify cognitive models that complement goal-directed reasoning. This is illustrated in Figure 2, where reasoning in the absence of goals can lead to action without an explicit representation of goals or it can lead to the definition of new goals for goal-directed reasoning.

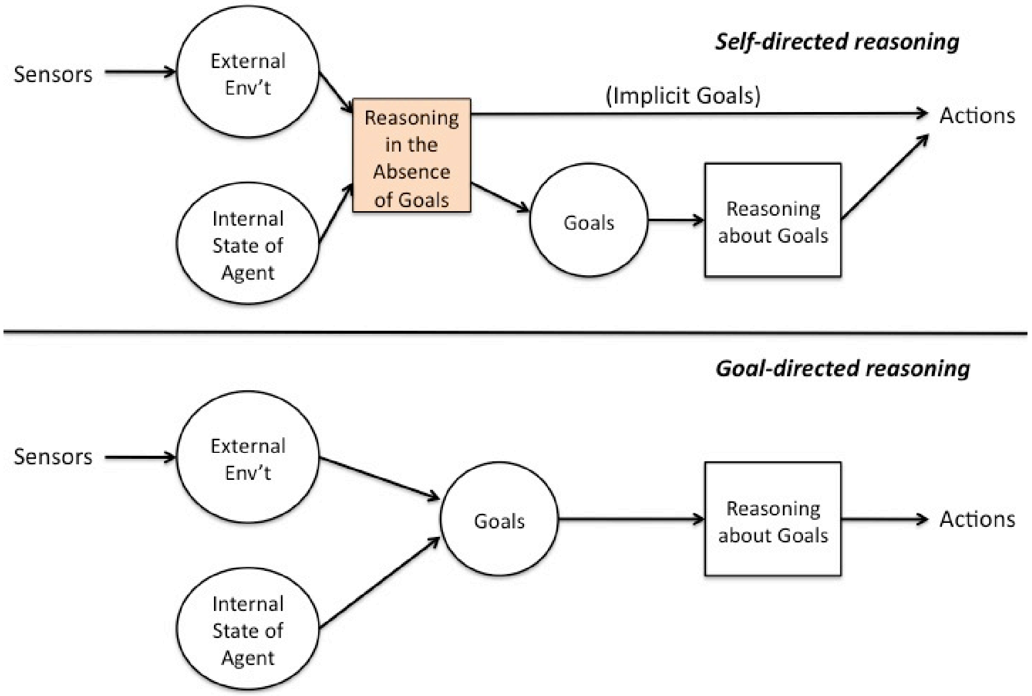


Figure 2: The role of reasoning in the absence of goals in a self-directed cognitive system

In self-directed reasoning, goals are flexible intermediate structures or implied structures, rather than a predefined and fixed starting point for reasoning. Figure 3 shows a spectrum of reasoning starting with the traditional goal-directed reasoning that includes domain specific or state-based goals and models for achieving them, through an intermediate type of reasoning in which goals are implied and may be emergent properties of reasoning, to reasoning without goals in which incentives, for example, provide guidance for reasoning about actions. Goals can be flexible, value-based, intermediate or emergent structures, rather than fixed starting points. This will permit continuous learning and adaptation to unexpected data or events, or changes in needs, beliefs or desires to become an integral concept in cognitive architectures. This will also recast cognitive systems from strictly goal-directed behaviors to include creative and exploratory behaviors.

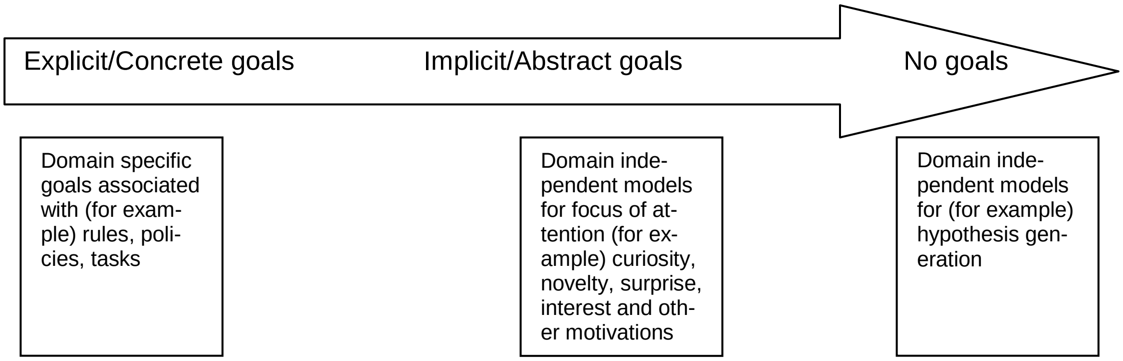

Explicit/Concrete goals
Implicit/Abstract goals
No goals
Domain specific goals associated with (for example) rules, policies, tasks
Domain independent models for focus of attention (for example) curiosity, novelty, surprise, interest and other motivations
Domain independent models for (for example) hypothesis generation

Figure 3. Spectrum of models for reasoning with and without goals.

In this section we begin by looking at models of novelty, curiosity, interest and surprise and the role they play in reasoning where goals are abstract, intermediate or emergent structures. We then consider a set of weaker models based on incentives and motives that represent only a preference for certain types of actions and goals are emergent structures. Finally, we consider models of hypothesis generation that can identify progressively strong patterns and relations in data, when there is no a priori knowledge of the structure of the data.

2.1 Novelty, Interest and Curiosity

Novelty, interest and curiosity fall in a class of models that allow an agent to characterize, act on and learn from changes in the environment. There are many accounts of measuring novelty using computational approaches. Marsland et al. (2000) used Stanley's (1976) model of habituation to implement a real-time novelty detector for mobile robots. Like the Kohonen (1993) Novelty Filter, the real-time novelty detector uses a Self-Organising Map (SOM) as the basis for the detection of novelty. Habituation and recovery extends a novelty filter with the ability to forget.

Models of interest provide a basis for determining if a novel event or state is worth attention. Curiosity is when something of interest can distract the process from its current focus of attention. Saunders and Gero (2001) drew on the work of Berlyne (1960) and Marsland et al (2000) to develop computational models of curiosity and interest based on novelty. They used a real-time novelty detector to implement novelty. Saunders and Gero (2004) model interest using sigmoid functions to represent positive reward for the discovery of novel stimuli and negative reward for the discovery of highly novel stimuli. The resulting computational models of novelty and interest are used in a range of applications including curious agents.

Merrick and Maher (2009) present models of motivated reinforcement learning agents that use novelty and curiosity as models of intrinsic motivation. These agents exhibit a kind of world-driven (rather than goal-driven) behavior. The agents (shown in Figure 4) have an experience trajectory $Y_{(t)}$ that models all states $S_{(t)}$, changes in states (events) $E_{(t)}$, actions that have been encountered/experienced by the agent:

$$Y_{(t)} = S_{(1)}, E_{(1)}, A_{(1)}, S_{(2)}, E_{(2)}, A_{(2)}, \dots , S_{(t)}, E_{(t)}, A_{(t)}$$

A dynamic motivated reward signal $R_{m(t)}$ is computed as a function of novelty and interest. Their model of interest, based on the experience trajectory, is a modified version of the Saunders and Gero interest function and is based on the Wundt curve shown in Figure 5.

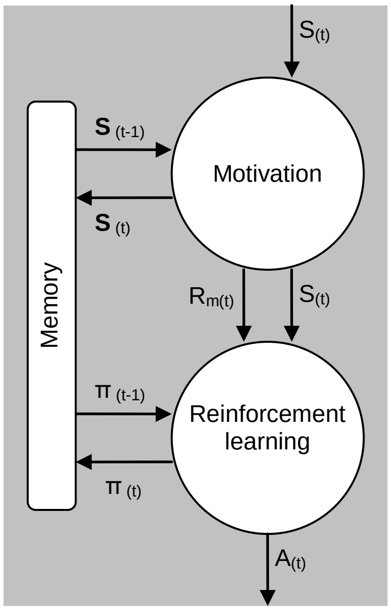


Figure 4 Motivated reinforcement learning agents: goals are implied by a dynamic motivation signal (Merrick and Maher, 2009).

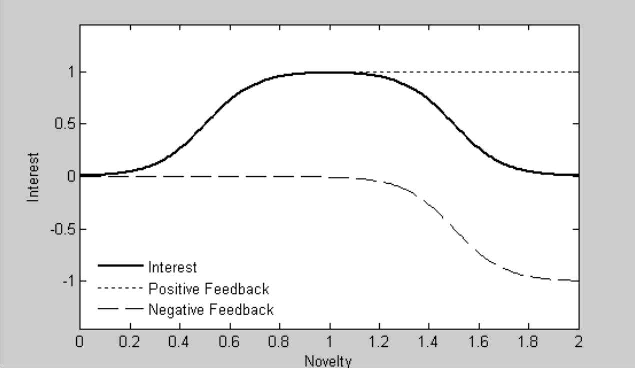


Figure 5 The Wundt curve is the difference between positive and negative feedback functions. It peaks at a moderate degree of novelty (Merrick and Maher, 2009).

This curiosity-based reward signal directs the agent to focus its learning on achieving specific situations at different times, but does not have an explicit representation of tasks or goals. Other motivation functions studied by Merrick and Maher (2009) within this framework include functions for competency and combined competency and curiosity. Experimental studies of curious agents in dynamic environments demonstrated adaptive behaviors through the ability to learn a variety of simple and complex behaviors (see Merrick and Maher, 2009 for experimental results).

2.2 Surprise

Surprise occurs when an unexpected event occurs. While surprise and novelty are similar, something may be novel, but necessarily surprising because it is the next expected change. Horvitz et al (2005) and Itti and Baldi (2004) have developed probabilistic models for finding surprising events in data. Ranasinghe and Shen (2008) have developed a model of surprise for reinforcement learning for developmental robots.

The Horvitz et al (2005) model of surprise is used in traffic forecasting. They generated a set of probabilistic dependencies among a set of random variables, for example linking weather to traffic status. They assume a user model that states that when an event occurs that has less than 2% probability of occurring, it is marked as surprising. Surprising events in the past are collected in a case library of surprises. This provides the data for forecasting surprises based on current traffic conditions. The Itti and Baldi (2004) model of surprise is developed for observing surprising features in image data using a priori and posterior probabilities. Given a user dependent model M of some data, there is a $P(M)$ describing the probability distribution. $P(M|D)$ is the probability distribution after the data is added, using Bayesian probability. Surprise is modeled as the distance d between the prior, $P(M)$, and posterior $P(M|D)$ probabilities.

The Ranasinghe and Shen (2008) model of surprise is used as a reward in a model they call surprise-based learning for developmental robots. In this model, surprise is used to set goals for learning in an unknown environment. The world is modeled as a set of rules, where each rule has the form: *Condition* → *Action* → *Predictions.* A condition is modeled as: *Feature* → *Operator* → *Value.* For example, a condition can be feature1 > value1 where "greater than" is the operator.

A prediction is modeled as *Feature* → *Operator.* For example, a prediction can be "feature1 >" where it is expected that feature1 will increase after the action is performed. The comparison operators provided for surprise analysis include operators to detect the presence (%) or absence (~) of a feature, and the change in the size of a feature (<, <=, =, >=, >). If an observed feature does not match the prediction for the feature, for example, the feature was expected to increase and it decreased, then the system recognizes surprise and sets that state as a reward for learning.

Grace et al (2014) develop a model of surprise based on predictive models using regression analysis or conceptual clustering on product design data. When a new design is encountered that violates our expectations, that design is surprising. This work was further developed to characterize multiple types of expectation and its effect on ways in which we are surprised (Grace and Maher, 2015).

These different approaches to modeling surprise are responses to the needs of the context in which they are developed. The Horvitz et al (2005) model determines that an event in the past is surprising, and then for a collection of surprising events is used to predict future surprising events. In the Itti and Baldi (2004) model, the new data is assimilated into the probability distribution, so something is surprising the first time it is introduced. The Ranasinghe and Shen (2008) model does not use probabilities and instead finds the first unexpected feature based on predictions of the direction in which the values of features will change and sets a reward to learn about that situation. The Grace et al approach is to characterize a cognitive model of expectations in order to recognize when a new design changes our expectations, and is therefore surprising. The role of factors such as curiosity and surprise in information seeking has also been recently considered (Gottlieb et al., 2013).

Surprise serves as a trigger for meta-level reasoning leading to the formation of goals. This concept is explored in the context of design by Grace and Maher (2015).

2.3 Incentives, Motives, and Motivation

In motivational psychology, incentive is defined as a situational characteristic associated with possible satisfaction of a motive (Heckhausen and Heckhausen, 2010). Incentives can be internal or external. Examples of internal incentives that depend on an individual's experiences include the novelty, difficulty or complexity of a situation. Examples of external incentives include money or other kinds of external 'payoff'. Associations between incentive and motivation can be learned, but there are also certain associations between incentives and motivation that have been found to be common across individuals. These include the associations between:

- Task difficulty and achievement motivation
- Risk and power motivation
- Risk and affiliation motivation
- Novelty and curiosity

Suppose we represent a situation encountered by an agent at time t as $\mathrm{S}_{(t)}$. Then the incentives associated with a situation can be represented as $I_{(t)} = (i_1, i_2, i_3\ldots)$. Each value i_n represents a different incentive. For example, i_1 may describe the novelty of $S_{(t)}$, i_2 may describe the complexity of $S_{(t)}$, i_3 may describe risk and so on.

Internal incentive values such as novelty, difficulty and complexity can be computed by an agent while it is reasoning about its environment using computational models such as novelty-detectors (Marsland et al., 2000) or achievement based on error calculations on learned policies (Merrick and Maher, 2009). This means that the incentives associated with a situation will change based on the agent's experiences. External incentive values are interpreted from the current state of the environment $\mathrm{S}_{(t)}$. These values will change based on changes in the environment. Both types of incentive have the possibility of satisfying the agent's motive.

Implicit motives are innate preferences for certain kinds of incentives. Because different individuals have different implicit motives, they will interpret the same situation incentives differently (Merrick and Shafi, 2013). For example, individuals with strong achievement motivation favor moderate difficulty. Likewise, high curiosity is associated with moderate novelty. Individuals with strong power motivation favor high risk. In contrast, individuals with strong affiliation motivation avoid situations with high risk. We can represent different motives M_1, M_2, $M_3\ldots$ as a function of incentive $M_{m(t)} = M_m(I_{(t)})$. These scalar motivation values $M_{m(\mathrm{t})}$ can be used in isolation, for example as a reward signal in learning, or combined. For example, they can be summed to give a resultant motivational tendency based on a complex motive profile of multiple motives (Merrick and Shafi, 2011).

$$T_{\mathrm{res}(t)} = M_{1(t)} + M_{2(t)} + M_{3(t)} + \ldots$$

The resultant value can then be used by the agent to identify the most highly motivating situations and act, learn to act or plan to act to achieve those situations. This action, learning or planning may involve formation of explicit concrete goal structures, but this is not strictly necessary.

In summary, models of incentives and motives are able to reason about the synergy of the external environment and the internal state and preferences of the agent to provide a basis for deciding what to do next. They do not represent goal structures although we may recognize emergent goal-directed behavior.

2.4 Autonomous Hypothesis Generation

Advances of computational power, data collection and storage techniques are making large volumes of new data available every day. In some situations, data are collected without a priori supposition or imposition of a specific research goal or hypothesis. Sometimes domain knowledge for this type of problem is also limited. For example, in sensor networks, sensors constantly record data. In these data, expectations about relationships cannot be described in advance. Moreover, the environment may change without a priori knowledge.

While finding patterns in data is achieved with data mining tools and algorithms, there are increasingly situations in which the observational data is collected without specific data mining goals in mind. Big data (Manyika et al., 2011) is a relatively new phenomena that has arisen in cases where very large data sets are accumulated as the result of daily recordings (such as twitter data, phone location data, etc.) for which no specific research purpose was set when the data were recorded. People are interested in analysing the data, however, the questions such an analysis might answer are not initially evident. Hypothesis generation helps to form initial questions that can guide the search for patterns in such accumulated data.

An example is in the field of intelligent systems in which sensors on mobile devices or those embedded in the physical environment record activity data from the environment and can perform pre-defined tasks to adapt to human activities through machine learning techniques. Such tasks can be developed manually when we know the common activities in typical scenarios, e.g., offices or lecture theatres. However, we need new ways for an intelligent environment agent to hypothesise about how to adapt to non-standard scenarios for which knowledge about what the observational data are describing is not available (Merrick et al., 2008).

A similar situation occurs in cyber security in which, due to the constant evolution of hacking activities, previous knowledge about abnormal activities in log data can get outdated. How to use log data to actively acquire updated insights into a system is an interesting research direction for which an agent that can generate new hypotheses from data could be an advantage (Shafi, 2008).

All of these scenarios pose open-ended questions about data. The abstract goal is to make sense of data, but there is an absence of concrete goals to achieve this. We thus argue that our increasing data stores will require new classes of algorithms that tend towards the right hand end of the spectrum shown in Figure 3 and permit reasoning in the absence of goals.

Traditional scientific research (or knowledge discovery) starts with a hypothesis suggesting an interpretation or description of a phenomenon. This hypothesis becomes the foundation for all further inferences and experiments. Its construction is heavily dependent on a researcher’s vision and skills, such as observation, domain knowledge, reasoning, imagination and creativity. Once constructed, a hypothesis leads to the design of the experiments and data collection required to test it. Therefore, this type of research is often called hypothesis-testing research (or hypothesis-driven research).

Hypothesis generation (Kell and Oliver 2003; King et al., 2004), which complements conventional hypothesis testing, occurs under circumstances where limited domain knowledge or the size of the data makes it difficult or impossible for a researcher to have well formed expectations or to propose precise hypotheses. There has been research considering hypothesis generation in different fields where hypotheses comprise patterns in the data or machine learning models. Nabel explains this as the methods for hypothesis generation and relates them to the specific environment under study (Nabel, 2009).

In the study carried out by Heintz and Doherty (Heintz and Doherty, 2004), a hypothesis is represented by certain linkage structures, which incrementally grow with additional sensor information being collected. The integration of such structures into agents' functionalities enables the agent to gain awareness of its environment; therefore it provides a basis for other high level goals to emerge. In computer vision, hypotheses often refer to possible matches between two regions of interest (ROI) (Wheler et al. 1995; Chin et al. 2011; Armbruster 2008).

In a computational approach to evaluating the quality of citizen science data, an example of a case in which very large volumes of data are collected, the challenge is determining when an unexpected data item is due to low quality data or if it is the basis for a new hypothesis and therefore is high quality data. This is being explored using concepts from computational creativity and design to search for patterns that are categorized as outliers and harbingers (Maher and Mahzoon, 2015). An outlier is a data item that is unexpected. An outlier is also a harbinger when that data item is the beginning of a new trend in the data, and therefore becomes a pattern that forms a new hypothesis: an indicator to collect more data in that space or to search that space for similar patterns.

Wang et al. (2015) consider the problem of hypothesis generation in continuous data, with the context being situations in which data can be collected about an unknown system. The unknown system is measured by a set of variables. However, limited a priori knowledge is available for characterising the structure and dynamics of system. Wang et. al. define the autonomous hypothesis generation problem in continuous domains in terms of two sub-processes: associative hypothesis generation (AHG) and causal hypothesis generation (CHG). In the case of human knowledge discovery, causation discovery is a progressive process, with the understanding of the causal law behind a system beginning with the observation of associations, which leads to an inquiry into causal relations..

Figure 6 shows the model for hypothesis generation where: The input to the AHG process is X, the set of all data. The output of the AHG process is F, a set of associative hypothesis of the form $X_A \Rightarrow X_B$. The output of the CHG process is G, a causal graph describing relationships among variables in X. Associative hypotheses can potentially reduce the number of variables that need to be examined when forming causal hypotheses and, because their generation procedures exclude irrelevant variables, the CHG can take advantage of the output from AHG to form a causal hypothesis in a reduced variable space. Alternatively, without a priori knowledge about the system, it is possible that there are no specific causal relations between its variables and, if so, it is not necessary to proceed to CHG. In this way, autonomous hypothesis generation can identify progressively stronger patterns in data, when the initial structure and relations in data are unknown (Wang, 2014).

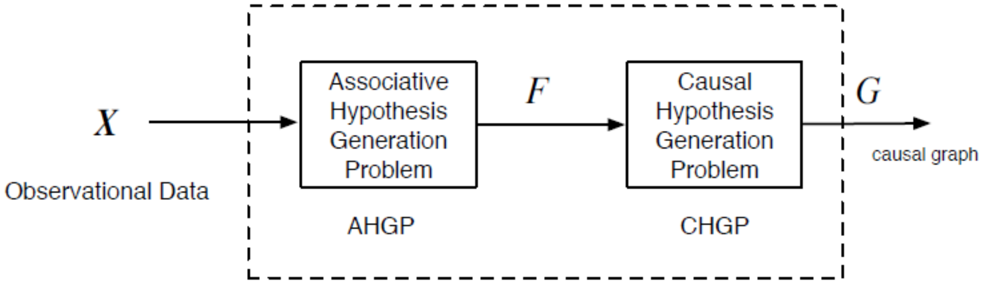

X
Observational Data
Associative Hypothesis Generation Problem
AHGP
F
Causal Hypothesis Generation Problem
CHGP
G
causal graph

Figure 6 Hypothesis generation identifies progressively stronger hypotheses about data.

3. Summary

This paper has presented several approaches to reasoning in the absence of goals as an alternative to goal reasoning. Introducing computational models of curiosity, interestingness, and motivation lead to alternative models for agents to reason about actions that are not dependent on explicitly represented goals and goal structures. Goals then become intermediate or possibly emergent properties of reasoning. Foner and Maes (1994) distinguish between goal-driven and world-driven attention focus. The models presented in this paper are similar to Foner and Maes (1994) concept of world-driven focus for creating goals or acting when there are no pre-defined explicit goals directing the agent in a learning or problem solving situation. Emergent goal-driven behavior is critical for cognitive systems that can use ‘idle time’ effectively by continuing to monitor their environment to discover and pursue self-generated goals that extend or improve their knowledge base or skill set.

References

Armbruster W. 2008. Bayesian hypothesis generation and verification. Pattern recognition and image analysis. pp. 269-274.

Baranes, A. and Oudeyer, P.-Y., 2010. Maturationally-constrained competence-based intrinsically motivated learning, IEEE International Conference on Developmenta and Learning, Ann Arbor, Michigan

Berlyne, D., 1960. Conflict, arousal and curiosity. McGraw-Hill, New York.

Braubach, L., Pokahr, A., Moldg, D., Lamersdorf, W., 2005. Goal representation for BDI agents, Second International Workshop on Programming Multiagent Systems: Languages and Tools, p 9-20.

Chin T-J. Yu J. Suter D. 2011. Accelerated hypothesis generation for multi-structure data via preference analysis. IEEE Transaction on pattern analysis and machine intelligence. 99, pp. 1-15.

Dignum, F. and Conte, R., 1998. Intentional agents and goal formation, Intelligent Agents IV: Agent Theories, Architectures and Languages. Springer, pp. 231-243

Fonar L and Maes, P. 1994. Paying attention to what's important: using focus of attention to improve unsupervised learning, Third International Conf on the Simulation of Adaptive Behaviour.

Gottlieb, J., Oudeyer, P-Y., Lopes, M., Baranes, A., 2013. Information Seeking, Curiosity and Attention: Computational and Neural Mechanisms, Trends in Cognitive Science, 17(11), pp. 585-596.

Grace, K., Maher, M. L., Fisher, D. & Brady, K., 2014. Modelling expectation for evaluating surprise in design creativity. In Gero, J.S. and Hanna, S (eds) Proceedings of Design Computing and Cognition 2014, University College London, pp 201-220.

Grace, K. and Maher, M.L. 2015. Specific curiosity as a cause and consequence of transformational creativity, International Conference on Computational Creativity.

Grace and Maher, M.L. 2015. Surprise and reformulation as meta-cognitive processes in creative design, Advances in Cognitive Systems.

Heckhausen, J. and Heckhausen, H., 2010. Motivation and action. Cambridge University Press, New York.

Heintz, F. and Doherty, P., 2004. Managing dynamic object structures using hypothesis generation and validation. In proceedings of the AAAI workshop on Anchoring Symbols to Sensor Data. pp. 1-9.

Horvitz, E., Apacible, J., Sarin, R. and Liao, L. 2005. Prediction, Expectation, and Surprise: Methods, Designs, and Study of a Deployed Traffic Forecasting Service, Proceedings of the Conference on Uncertainty and Artificial Intelligence, AUAI Press.

Itti, L. and Baldi, P. 2004. A Surprising Theory of Attention, IEEE Workshop on Applied Imagery and Pattern Recognition.

Jaidee, U., Muñoz-Avila, H., & Aha, D. W. 2011. Integrated learning for goal-driven autonomy. In Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three, AAAI Press, pp. 2450-245I.

Kaplan, F. and Oudeyer, P.-Y., 2003. Motivational principles for visual know-how development. In: C.G. Prince et al. (Editors), Proceedings of the 3rd international workshop on Epigenetic Robotics : Modelling cognitive development in robotic systems, Lund University Cognitive Studies, pp. 73-80.

Kell, D.B. and Oliver, S.J. 2003. Here is the evidence, now what is the hypothesis? The complementary roles of inductive and hypothesis-driven science in the post genomic era. BioEssays, 26:99-105.

Kohonen, T., 1993 Self-organisation and associative memory, Springer, Berlin.

Langley, P., Laird, J.E., Rogers, S. 2008. Cognitive Architectures: Research Issues and Challenges, Cognitive Systems Research.

Maher, M.L., Poon, J. and Boulanger, S. 1996. Formalising Design Exploration as Co-Evolution: A Combined Gene Approach, in J.S.Gero and F. Sudweeks (eds) Advances in Formal Design Methods for CAD, Chapman & Hall, pp 1-28.

Maher, M. L., Merrick, K., Graham, B., 2011. Reasoning in the Absence of Goals, *AAAI Fall Symposium on Advances in Cognitive Systems*, pp 202-209.

Maher, M.L. and Mahzoon, M. J., 2015. *Finding Unexpected Patterns in Citizen Science Contributions Using Innovation Analytics*, Collective Intelligence Conference.

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C. and Byers, A. 2011. *Big data: The next frontier for innovation, competition and productivity*, McKinsey&Company.

Marinier, R. and Laird, J. E. 2008. Emotion-Driven Reinforcement Learning. CogSci 2008, Washington, D.C

Marsland, S., Nehmzow, U., Shapiro, J, 2000 A real-time novelty detector for a mobile robot, In EUREL European Advanced Robotics Systems Masterclass and Conference.

McClelland, D. C., 1975. Power: the inner experience. Irvington, New York

Merrick, K., and Maher, M. L 2009. Motivated reinforcement learning: curious characters for multiuser games, Berlin, Springer.

Merrick, K. Maher, M.L. and Saunders, R. 2008. Archieving adaptable behaviour in intelligent room using curious supervised learning agents. In Proceedings of CAADRiA 2008 Beyond Computer Aided Design, pages 185-192.

Merrick, K., Shafi, K., 2013. A Game Theoretic Framework for Incentive-Based Models of Intrinsic Motivation in Artificial Systems , *Frontiers in Cognitive Science, Special Issue on Intrinsic Motivations and Open-Ended Development in Animals, Humans and Robots.* Baldassarre, G., Barto, A., Mirolli, M., Redgrave, P., Ryan, R., Stafford, T (Eds). Volume 4, 30th October, 2013

Merrick, K. and Shafi, K., 2011. Achievement, affiliation and power: motive profiles for artificial agents. Adaptive Behavior, 9(1): 40-62.

Nabel G. 2009. The coordinate of truth. Science, 326, pp. 53-54.

Ranasinghe N. and Shen,W-M, 2008. Surprise-Based Learning for Developmental Robotics. In Proc. 2008 ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems, Edinburgh, Scotland.

Russell, S. J. and Norvig, P. 1995. Artificial intelligence: a modern approach, Prentice Hall, Englewood Cliffs, NJ

Saunders, R., 2001. Curious design agents and artificial creativity. PhD Thesis, University of Sydney, Sydney

Saunders, R. and Gero, J.S., 2004. Curious agents and situated design evaluations. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 18(2): 153-161

Schmidhuber, J. 1991 A possibility for implementing curiosity and boredom in model-building neural controllers. In The International Conference on Simulation of Adaptive Behaviour: From Animals to Animats pp 222-227

Shafi. K. 2008. An online and adaptive signature-based approach for intrusion detection using learning classifer system. PhD thesis, University of New South Wales Canberra Campus.

Singh, S., Barto, A.G. and Chentanez, N., 2005. Intrinsically motivated reinforcement learning. In: L. Saul, Y. Weiss and L. Bottou (Editors), Advances in Neural Information Processing Systems (NIPS). The MIT Press, Vancouver, pp. 1281-1288

Stanley, J. C., 1976. Computer simulation of a model of habituation. Nature 261:146-148

Wang, B., Merrick, K., Abbass, H., 2015, Autonomous Hypothesis Generation as an Environment Learning Mechanism for Agent Design. In Proceedings of the 2015 Conference on Artificial Life and Computational Intelligence, pp 210-225.

Wang, B, 2014, Autonomous Hypothesis Generation for Knowledge Discovery in Continuous Domains, PhD Thesis, University of New South Wales, Canberra.

Wheler M. and Ikeuchi K. 1995. Sensor Modeling, Probabilistic hypothesis generation, and robust localization of object recognition. IEEE Transaction on pattern analysis and machine intelligence. 17(3), pp. 252-265, 1995.

Towards Cognition-level Goal Reasoning for Playing Real-Time Strategy Games

Héctor Muñoz-Avila HEM4@LEHIGH.EDU
Dustin Dannenhauer DTD212@LEHIGH.EDU
Computer Science and Engineering, Lehigh University, Bethlehem, PA 18015 USA

Michael T. Cox MICHAEL.COX@WRIGHT.EDU
Wright State Research Institute, Wright State University, Dayton, OH 45435 USA

Abstract

We describe a 3-layer architecture for an automated Real-Time Strategy game player. In these kinds of games, opponent players create and manage large armies of units to defeat one another requiring strategic thinking. Players give commands asynchronously, requiring rapid thinking and reaction. Our 3-layer architecture builds on current automated players for these kinds of games, which focuses on rapid control. The first layer is the control layer that implements the standard reactive player in RTS games. The second layer is a goal reasoning mechanism; it selects goals that are executed by the control layer. The second layer reasons at the cognitive level; it introduces symbolic notions of goal and examines the outcomes of its own decisions. The third layer introduces a meta-reasoning layer that reasons strategically on long-term plans. Our ideas are grounded by using the MIDCA cognitive architecture.

1. Introduction

Like chess, real-time strategy (RTS) games are a form of simulated warfare, where players must maneuver their "pieces" (called "units" in the RTS games parlance) to defeat an opponent. However, RTS games are much more complex than chess due to the following 4 factors: (1) the size of the search space; (2) the partial observability of the state; (3) an infinite number of initial game configurations; and (4) the asynchronous nature of gameplay. We will expand later on these points but the increase in complexity can be illustrated by the fact that, whereas the best automated chess player defeated the best human chess player 10 years ago, top human RTS games players easily defeat the best automated players. Indeed for the RTS game StarCraft,[1] the winner of the AIIDE-13 automated player competition was defeated by a 50th ranked player in the world in about 10 minutes in games where equally skilled human players could take 30 minutes or longer as witnessed during that competition by the authors. This, in spite of the fact that the automated player could issue moves at a rate about 3 times faster than the human player (the exact measure is called APM – or actions per minute).

[1] http://en.wikipedia.org/wiki/StarCraft

The combination of the 4 factors above suggests that we will not see automated players that play at the human level by brute force any time soon. The fact that expert humans defeat so easily expert automated players despite having a much lower APM measurement also suggests the need for automated players that can reason at a high-level of granularity; not only at the object level (i.e., reasoning about individual unit's moves) but also more abstractly (e.g., reasoning about the strategic notion of "defense").

Motivated by these observations, we explore the idea of using cognitive architectures that we hypothesize will result in more advanced automated players. We are particularly interested in examining how these architectures can be used to create goal reasoning mechanisms of varied levels of granularity. We will examine general properties of cognitive architectures and see how they match the needs for effective automated players in RTS games. We also examine challenges of using cognitive architectures for this purpose. Finally, we provide an example highlighting the potential use of the MIDCA cognitive architecture to create an automated player.

2. Real-Time Strategy Games

Real-time strategy game players perform 4 kinds of actions: harvest, construct, build, and destroy. The player needs to harvest resources by using specialized units to collect these resources. These resources can be used to construct structures such as barracks, factories and defensive towers (these structures attack enemy units within their range). Structures such as barracks and factories are needed to build units such as foot soldiers (assembled in barracks) and tanks (built in factories). Building these units also consume resources. Units are used to attack the opponent during combat (when enemy units and/or buildings are within range of friendly units or buildings). Combat follows a paper-rock-scissor model where some class of units are strong against units of another class but weak themselves against a third class. This encourages using a variety of tactics since no class of units is stronger than all others. Furthermore, as a rule, units that are relatively stronger than other units consume more resources when built. Hence, players must reason for particular situations if it will be more cost effective to build more of the weaker units or fewer of the strong units.

As mentioned in the introduction, there are four elements that makes RTS games more complex than chess. We now analyze these challenges in some level of detail.

2.1 Size of Search Space

In Chess, players compete in an 8x8 grid with each player controlling 32 pieces. In contrast, players can control around 100 units (each can be one of dozens of classes), dozens of structures (again multiple classes), and the game takes place in 100x100 grids in RTS games such as StarCraft. More recent games such as Supreme Commander, allow players to control hundreds of units on 1000x1000 grids. Furthermore, each cell in the grid can be of different types including mountain (an impassable obstacle by land units), water (only passable by naval units), and mineral resources (which can be harvested). A careful analysis shows that the game tree for turn-based versions of RTS games is several orders of magnitude larger than chess (Aha, Molineaux & Ponsen, 2005). Briefly, whereas in an average game of chess a player can make around 80 moves, in a strategy game players can make several hundred. Furthermore, the average branching factor of the game tree for chess is 27, whereas for a strategy game, it is on the order of 200. The latter can be illustrated by the fact that a strategy player at any point must decide where to move its units, whether to construct buildings and where to place them, whether to spend resources on research to upgrade units and if so, choose which kind of research, among other possible decisions.

2.2 Starting Game Configurations

Games are played on a grid or map that may have a number of geographical features including mountains (impassable for land units), rivers, lakes and oceans (also impassable for land units), and resources that players can harvest. If the size of the map is fixed, say 1000x1000 cells, and there are 2 resources then the number of map configurations is $5^{1000x1000}$. If the size of the map varies and is unbounded, then the number of maps is infinite. In addition, each player can start with a base (a collection of buildings placed contiguous to one another) and some units. Typically, the starting base consists of a single building (a town center, which can produce worker units) and a worker (which can harvest resources or build structures).

2.3 Partial State Observability

In RTS games, players only see the area that is in visual range of their units and buildings (typically a few cells around the unit/building). When the game starts, this means that the player only see its base and surrounding areas. As the player moves its units, it will uncover the map configuration. However, opponent’s units will remain unseen unless they are in visual range of the player’s own units. This is referred to as the fog of war since the opponent’s movements might be hidden.

2.4 Asynchronous Nature of Gameplay

In RTS games, players make their moves (i.e., commands) asynchronously. These commands include directing a unit to move to a cell or ordering a worker to construct a building in a location (typically a group of contiguous cells forming a rectangular shape). This means that the speed to issue the commands is an important success factor. Alleviating this is the fact that the game engine automates some actions. For example, a worker tasked with harvesting resources will continue to do so until it is issued new commands or is killed by an opponent or the resources are exhausted. Importantly, units will attack opponent’s units unless the player explicitly has command them not to do so. Nevertheless, in human competitions, particularly in later stages of the game where each player is controlling hundreds of units and dozens of buildings spread out over a map, players can be seen frantically issuing orders.

3. Automated RTS Players at the Object Level

For the purposes of this discussion, we distinguish between reasoning about the environment and the environment itself that constitutes the *ground level*. We further distinguish between reasoning at the *object* (i.e., cognitive) *level* and at the *meta-level* (i.e., metacognition) (Cox & Raja, 2011). The ground level refers to the maps, units, and player’s actions. Existing automated players (such as those used in commercial RTS games or those used in the StarCraft automated player competition) all reason at the object level about the ground level. These programs can exhibit complex strategies: some will try to attack the opponent continuously draining the opponent’s resources, while others slowly build a powerful army to attack the opponent at a later stage of the game. Others analyze the map and methodically take control of resources until they control most resources enabling them to overwhelm the opponent by producing large numbers of units that the opponent cannot possibly counter. These systems still reason at the object level; the strategies, while undoubtedly complex, are selected based solely on circumstances of the ground level. The control-resource-strategy for example, will pick the nearest resource to the base that is undefended, and expand in that direction. Hardcoded doesn't mean lack of flexibility; the automated player will change its strategy adapting to previously foreseen circumstances.

4. Goal Selection in RTS Games at the Object Level

A key characteristic of cognitive systems is the capability for high-level cognition and goal reasoning. That is, the capability of selecting goals of multiple levels of abstraction, determining how to achieve those goals with multi-step reasoning mechanisms and introspectively examine the results of those decisions.

Recent efforts on RTS game research have begun to explore cognition at the object level. Specifically, we examine agents that exhibit *goal-driven autonomy (GDA)* (Aha, Klenk, Munoz-Avila, Ram, & Shapiro, 2010; Cox, 2007; Klenk, Molineaux, Aha, 2013; Munoz-Avila, Aha, Jaidee, Klenk, & Molineaux, 2010). GDA agents are those that (1) generate a plan to achieve the current goal; (2) monitor the execution of the plan and detect any discrepancy between the expectations (e.g., a collection of atoms that must be true in the state) and the actual state; (3) explain reasons for this discrepancy; and (4) generate new goals for the agent to achieve based on the explanation generated. The following are three instances of automated GDA agents for RTS games:

- Weber, Mateas, and Jhala (2012) learns GDA knowledge while playing the RTS game StarCraft. The architecture of the automated player uses the idea of tasks managers which are components specialized for tasks such as combat and construction (we will expand on these task managers later on). Most of these task managers are hardcoded but the manager responsible for building units uses GDA. This enables the automated player to dynamically change the production of units to accommodate for changes in the environment. Weber (2012) uses vectors of numbers as its representations. These numbers represent counters for elements in the game such as number of soldiers. So when a discrepancy occurs because, say, the system expected to have 12 soldiers but it only has 8, it will generate a new goal to generate 4 soldiers.
- Jaidee, Munoz-Avila, and Aha (2011) learns and reasons with expectations, goals and how to achieve those goals. It has two main differences in comparison to Weber, Mateas, and Jhala: (1) it neither learns nor reasons with explanations for failure reasons; and (2) GDA is used to control all aspects of the gameplay as opposed to only production of new units. It uses multiple-agents, one for each type of unit or building in the game. The GDA cycle assigns goals for each of these agents to achieve ensuring coordination between the agents.
- Dannenhauer and Munoz-Avila (2013) showcases a GDA agent that takes advantage of ontological information in games. This enables the system to reason with notions such as

controlling a region. A rule can define the concept by indicating that a region is controlled by agent A, if at least one unit of agent A is in the region and no unit of agent B is present. The GDA cycle uses these notions as part of its reasoning with expectations process.

Each of these three agent variations have been shown to perform well against hard-coded opponents in a variety of experimental settings.[2]

5. High-Level Goal Reasoning

GDA agents exhibit some of the characteristics of cognitive systems (Langley 2012). In particular, they use a structured representation of knowledge:

- **Symbolic structures**. GDA systems frequently use the notion of goals, states and plans based on the STRIPS formalism, as such these symbols are interpretable symbolic mechanisms. In the context of games, goals refer to conditions in the state that must be held true. For example, controlling a resource in a specific location (e.g., having two or more units and a building around the resource). As such goals refer to conditions at the object level.
- **Complex relations.** For example, a plan is a sequence of interrelated actions and their sequencing is dependent on cause-effect relations between the tasks. For instance, a plan to control a resource might call to produce a mixture of units and when these units are produced, send them to the location of the resource.

Another characteristic of cognitive systems, GDA systems perform heuristic search; they cannot guarantee optimal solutions in these kinds of highly dynamic environments and very large decision spaces. While mini-max algorithms, which guarantee some form of optimal behavior to counter an opponent's move, have demonstrated to be useful to RTS games (Churchill, Saffidine, & Buro, 2012), they focus on small combat encounters involving a handful of units. Similarly, learning algorithms have been used to determine best opening moves in the game that maximize economic output within the first few minutes of the game but these involve only early construction of buildings and worker resource harvesting tasks; they are based on the premise that (1) at early stages in the game there will be no combat and (2) gameplay decisions are localized around the starting base. Cognitive systems introspectively examine their own decisions based on the interactions with the environment and their own knowledge to tune its own decision making.

High-level cognition enables abstract reasoning that goes beyond reasoning at the object level. This is a significant departure from existing RTS automated players. We will ground our ideas providing discussions by basing the agent on the MIDCA cognitive architecture.

6. The MIDCA Cognitive Architecture

Computational metacognition distinguishes reasoning about the world from reasoning about reasoning (Cox, 2005). As shown in Figure 1, *the Metacognitive, Integrated, Dual-Cycle*

[2] They have not been tried in competition settings because competition rules would need to be refined to account for the fact that these agents learn their knowledge.

Architecture (MIDCA) (Cox, Oates, & Perlis, 2011; Cox, Oates, Paisner, & Perlis, 2012) consisting of “action-perception” cycles at both the cognitive (i.e., object) level in orange and the metacognitive (i.e., meta) level in blue. The output side of each cycle consists of intention, planning, and action execution; the input side consists of perception, interpretation, and goal evaluation. At each cycle, a goal is selected and the agent 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.[3] 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 or

[3] Note that this does not preclude interleaved planning and execution. The plan need not be fully formed before action execution takes place.

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.

Furthermore, and unlike most cognitive theories, our treatment of goals is dynamic. That is, goals may change over time; goals are malleable and are subject to transformation and abandonment (Cox & Zhang, 2007; Cox & Veloso, 1998). Figure 1 shows *goal change* at both the object and meta-levels as the reflexive loops from goals to themselves. Goals also arise from

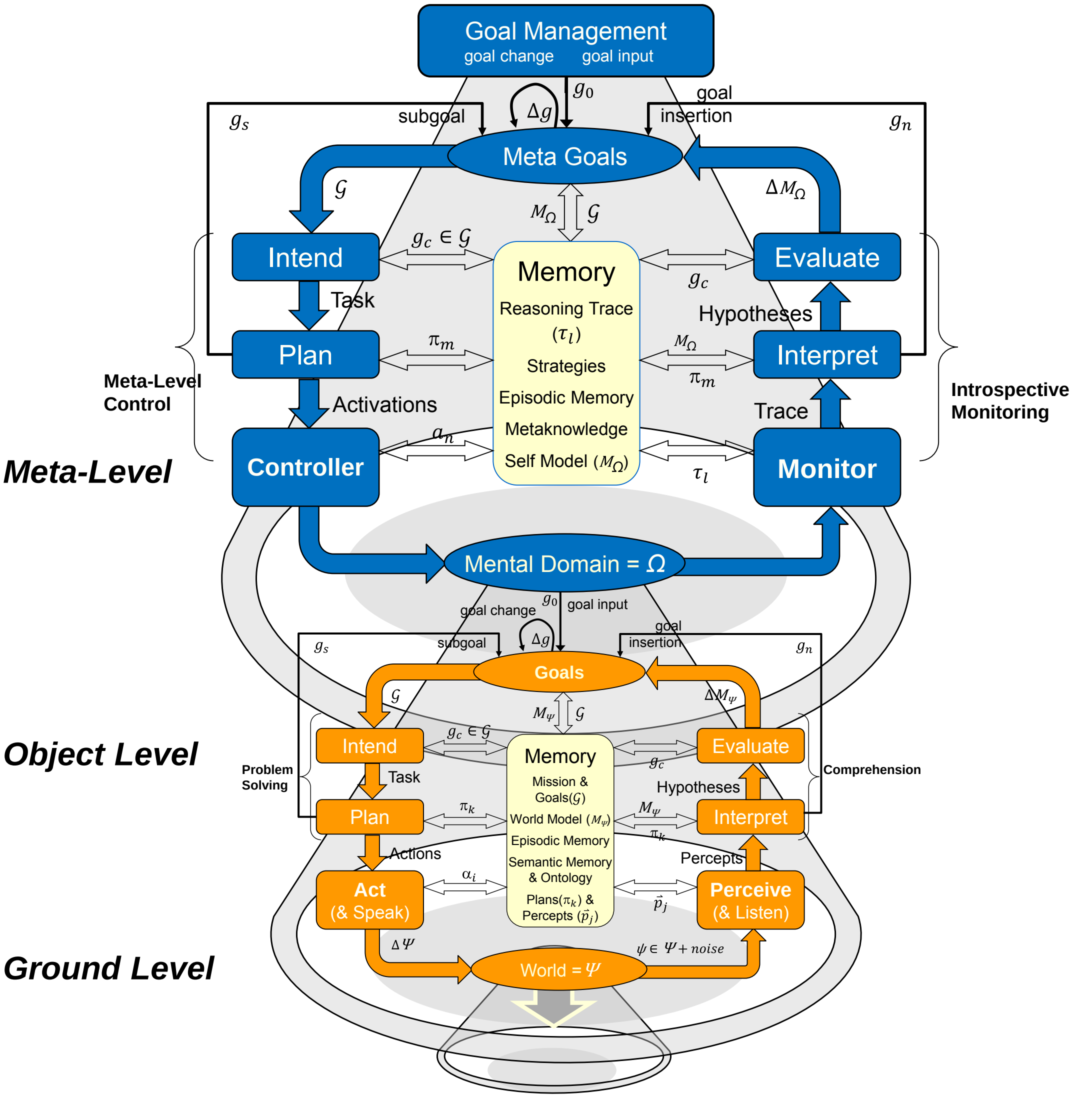


Figure 1. Metacognitive, Integrated, Dual-Cycle Architecture (MIDCA)

traditional *sub-goaling* on unsatisfied preconditions during planning (the thin black back-pointing arrows on the left of both blue and orange cycles). Finally new goals arise as MIDCA detects discrepancies in the input given its expectations. It explains what causes the discrepancy, and generates a new goal to remove the cause (Cox, 2007). This type of operation, called *goal insertion*, is indicated by the thin, black arrows from the interpretation processes in the figure.

Goal insertion is the fundamental GDA process in MIDCA and occurs at both the object and meta-levels. At the object level, perception provides observations, and plans from memory to provide the expectations. The interpretation process detects discrepancies when observations conflict with expectations and will then explain what caused the discrepancy and will generate a new goal. At the meta-level, monitoring provides an observation of a trace of processing at the object level and a self-model provides the expectations. Like the object-level GDA process, interpretation produces an explanation of why the reasoning at the object level failed and uses the explanation to generate a learning goal to change the knowledge or reasoning parameters of the object level. An explanation in MIDCA is a causal knowledge structure, χ, consisting of a set of antecedents that together cause the discrepancy.[4]

$$\chi = \delta \rightarrow \ldots \rightarrow s_c$$

According to Cox (2007; this volume), the explanation contains a *salient antecedent*, δ, that represents the root cause of the problem signaled by the discrepancy. The goal, $g_c = \neg\, \delta$, is then to achieve the negation of this antecedent, therefore removing the discrepancy and solving the problem.

Numerous ways exist that the meta-level can affect the object level. The meta-level can act as an executive function in a manner similar to CLARION (Sun, Zhang, & Mathews, 2006). It can decide between object-level parameters, it can allocate resources between competing object-level processes, and it can set priorities on object level goals, swap object-level processes (i.e., change the planner with a different one) and it can also insert goals at the object level.

7. MIDCA in RTS Games

The premise of our idea to apply MIDCA in RTS games is for the lower level cycle to act on the RTS action (ground) level and the higher level on the RTS reasoning (object) level. That is, the MIDCA cycle at the object level, monitors the situation at the ground level and generates goals and the plans to achieve those goals. Since planning in RTS games is a complex activity within itself we will borrow the idea of *managerial tasks* (Scott, 2002), used by many automated RTS games, to generate and execute these plans. Managerial tasks are performed by the following components:

1. **Building**. In charge of structure building, including keeping track of the order in which buildings must be constructed (e.g., a factory requires barracks).

[4] The explanation is actually a graph $\chi = (V, E)$ with $\delta \in V$ an element of the source nodes and $s_c \in V$ a distinguished element of the sink nodes. See Cox (2011) for further details.

2. **Units**. Responsible of creating new units and prioritizing the order in which units are created.
3. **Research**. Responsible for creating new technology that enables the creation of more powerful units.
4. **Resource**. Responsible for gathering resources (i.e., minerals and vespene gas), which is needed to construct units, buildings and invest on research.
5. **Combat**. Responsible for controlling units during combat; determining which units to attack, or whether to defend.
6. **Civilization**. Responsible for coordinating the managers.

Each of these components could be a learning component although for a first implementation we are going to use hard-coded implementations of these. This will enable us to directly use existing automated players such as Skynet or UAlbertaBot and have the MIDCA architecture build on top. This is how we implemented our GDA agent playing StarCraft (Dannenhauer & Munoz-Avila, 2013). But unlike that agent, MIDCA will enable meta-reasoning.

For the higher-level cycle, MIDCA will monitor the decisions made at MIDCA's object level and intervene (by changing or assigning new goals, or by adjusting parameters in object level components such as the planner similar to Dannenhauer, Cox, Gupta, Paisner & Perlis, 2014). The higher level acts as a long-term, "broad perspective" reasoning mechanism. It reasons not only on information about the object level but also on the knowledge used by the object-level MIDCA to generate its goals. We believe this is a crucial reasoning capability, one that is missing in existing automated players.

Our automated player performs asynchronous decision-making at three levels. We describe them from the most concrete to the most abstract:

- **Object-level reactive control**. The player decisions are made by the six managers described above. This ensures immediate and continuous control of all units and buildings. This is the level programmed by most commercial RTS players and entries to the automated player tournaments. It controls everything from the production of units, harvesting of resources and combat. The civilization manager ensures that a default strategy, such as the systematic control of resources in the map, is pursued.
- **Object-level goal formulation**. This is performed by the object level from MIDCA and is reminiscent of GDA automated players in the sense that it monitors the current state of the game and triggers new goals and the means to achieve them (i.e., plans). But, unlike existing GDA players, it has a symbolic model of the goals achieved by the object-level reactive controller and their outcomes. This enables the goal formulation component to monitor the execution and formulate new goals. For example, if the opponent launches an assault on a base built around resources, by default, the object-level reactive control might direct nearby units to defend the base. The object level goal formulator might detect that the opponent is gaining the upper hand and will take control of the resource. The goal formulator might generate a new goal: to re-take the resource or to, instead, take over an opponent's controlled

resource that is less defended. This goal will supplement the default strategy of the object-level reactive control; tasking, with higher priority, the managers to take actions towards achieving the new goal. This ensures consistent behavior where these goals are given priority but other goals, either from the default strategy or previously stated goals are still being pursued (but with less priority).

- **Meta-level long-term control**. This is performed by the MIDCA's metacognitive level. It monitors the decisions of the object level goal formulation and the object level reactive control and might override decisions made and tasks new goals to achieve. For example, the cognitive level might invalidate the goal to re-take or to take an alternative resource because it determines that the reactive control is about to take the enemy starting base and therefore end the game in victory. In this case, it will invalidate the new goal as accomplishing it will consume resources that might detract from the imminent victory. The meta-cognitive level reasons at a higher level of granularity reasoning on the lower level's own reasoning and considering long-term implications.

Because of the real-time nature of the games, these modules operate in parallel to the MIDCA level. This guarantees that the automated player, MIDCA-RTS, will always react to immediate changes in the situation (e.g., a sudden attack on our base) while the metacognition level reasons on strategic decisions that go beyond immediate game occurrences.

8. Example Scenarios

8.1 Detecting a Feign Attack

A common situation that occurs in RTS games is for the enemy to harass your harvesting units, often with a single worker unit of their own. This behavior has also occurred during automated player matches (Skynet is known for using a worker to harass enemy workers very early in the game). In this scenario, there are two common approaches that are problematic for the defending player. First, the player could ignore the enemy probe, but this will cause workers to be killed. The second is to send all worker units to attack the probe. This is problematic because the probe will lead the workers on a chase (as seen in Figure 2) and resource harvesting will be nearly stopped. The orange object level will be able to respond to the discrepancy of the enemy probe, but the meta-cognitive blue layer will figure out / learn which approach is the best, which is usually to send 1 of our workers to attack the enemy worker, and let the rest keep harvesting.

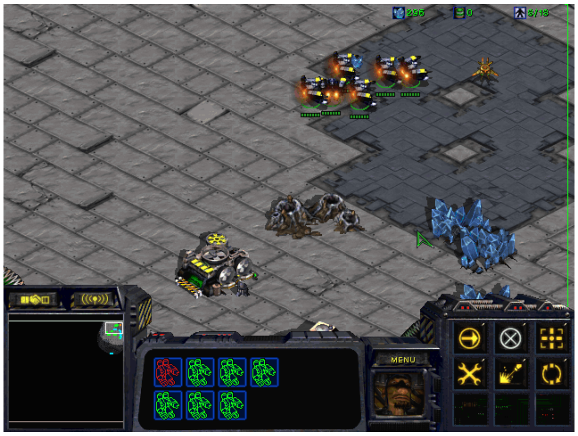


Figure 2. The workers following an enemy's probe

8.2 Performing a Feign Attack

Using the concept of distract, MIDCA could perform strategies at a higher level, such as distracting the enemy with a small force of units at the front of their base and concurrently sending units via dropships behind the enemy base. In Figure 3, air ships have entered the back of the base from the left side and have dropped units to destroy buildings (in this image some buildings have already been destroyed and two are on fire). This is happening while the base's defenses are busy with the 'distract' group at the front of the base (not seen in the picture). This example motivates the benefit of higher level concepts such as distract, used by a cognitive architecture such as MIDCA.

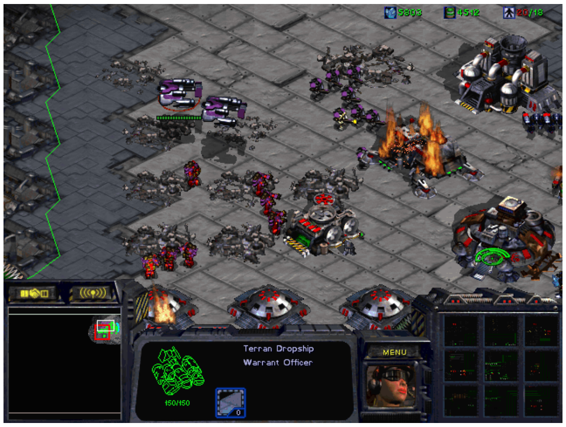


Figure 3. Assault from behind the enemy's base

8.3 Performing a Feign Attack

In RTS matches, there are often crucial points on the map that are strategically important. The concept of 'defense' is another high-level concept that a cognitive architecture such as MIDCA could infer. While some automated bots already exhibit this behavior in specific situations, it is not an explicit concept. The agent will be able to carry out more sophisticated attacks if the notion of defense is explicit and flexible, because one could imagine not only defending a chokepoint (see Figure 4) or map location, but also a particularly strong offensive unit vulnerable to specific enemies. For example, it is common to pair a siege tank with melee units to protect the tank from enemies that get close enough to damage the tank without being fired upon.

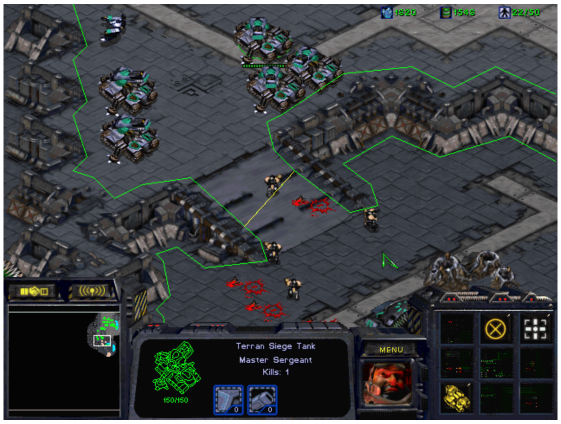


Figure 4. Feign attack to front of the base

9. Final Remarks

While attaining automated players for RTS games that can play at the human level are an interesting challenge in their own right, they also can be seen as a challenge for agents that can exhibit high-level cognition. The latter is one of our main motivations with this research. We believe that the three layer architecture proposed will guarantee the reactive behavior needed for these kinds of games, while the meta-cognition will enable new high-level capabilities currently not exhibited in existing automated players.

Acknowledgements

This work is funded in part under NSF grant 1217888. Further support has been provided by ARO under grant W911NF-12-1-0471 and by ONR under grants N00014-15-1-2080, N00014-12-1-0430, and N00014-12-1-0172. We thank the anonymous reviewers for their insights and comments.

References

Aha, D. W., Klenk, M., Munoz-Avila, H., Ram, A., & Shapiro, D. (Eds.) (2010). *Goal-driven autonomy: Notes from the AAAI workshop*. Menlo Park, CA: AAAI Press.

Aha, D.W., Molineaux, M., & Ponsen, M. (2005). Learning to win: Case-based plan selection in a real-time strategy game. *Proceedings of the Sixth International Conference on Case-Based Reasoning* (pp. 5-20). Chicago, IL: Springer.

Cox, M. T. (this volume). Toward a formal model of planning, action, and interpretation with goal reasoning. To appear in *Proceedings of the 2015 Annual Conference on Advances in Cognitive Systems: Workshop on Goal Reasoning* (IRIM Tech Report). Atlanta: Georgia Institute of Technology.

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. (2007). Perpetual self-aware cognitive agents. *AI Magazine 28*(1), 32-45.

Cox, M. T. (2005). Metacognition in computation: A selected research review. *Artificial Intelligence. 169* (2), 104-141.

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., & 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., & Raja, A. (2011). Metareasoning: An introduction. In M. T. Cox & A. Raja (Eds.) *Metareasoning: Thinking about thinking* (pp. 3-14). Cambridge, MA: MIT Press.

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.

Cox, M. T., & Zhang, C. (2007). Mixed-initiative goal manipulation. *AI Magazine 28*(2), 62-73.

Churchill, D., Saffidine, A., & Buro, M. (2012). Fast Heuristic Search for RTS Game Combat Scenarios. In *AIIDE*.

Dannenhauer, D. and Munoz-Avila, H. (2013) LUIGi: A Goal-Driven Autonomy Agent Reasoning with Ontologies. *Advances in Cognitive Systems Conference (ACS-13).*

Dannenhauer, D., Cox, M. T., Gupta, S., Paisner, M. & Perlis, D. (2014). Toward meta-level control of autonomous agents. In *Proceedings of the 2014 Annual International Conference on Biologically Inspired Cognitive Architectures: Fifth annual meeting of the BICA Society* (pp. 121 -126). Elsevier/Procedia Computer Science. Forbus, K. D. (1984). Qualitative process theory. *Artificial Intelligence*, *24*, 85–168.

Jaidee, U., Munoz-Avila, H., Aha, D.W. (2011) Integrated Learning for Goal-Driven Autonomy. To appear in: Proceedings of the Twenty-Second International Conference on Artificial Intelligence (IJCAI-11). AAAI Press.

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

Munoz-Avila, H., Aha, D.W., Jaidee, U., Klenk, M., & Molineaux, M. (2010). Applying goal driven autonomy to a team shooter game. *Proceedings of the Twenty-Third Florida Artificial Intelligence Research Society Conference* (pp. 465-470). Menlo Park, CA: AAAI Press.

Langley, P. (2012). The cognitive systems paradigm. *Advances in Cognitive Systems 1*, 3–13.

Scott, B. (2002) Architecting an RTS AI. AI game Programming Wisdom. Charles River Media.

Sun, R., Zhang, X., & Mathews, R. (2006). Modeling meta-cognition in a cognitive architecture. *Cognitive Systems Research*, *7*, 327-338.

Weber, B., Mateas, M., & Jhala, A. (2012). Learning from Demonstration for Goal-Driven Autonomy. In *AAAI*.

Guiding the Ass with Goal Motivation Weights

Hector Muñoz-Avila HEM4@LEHIGH.EDU
Computing Science and Engineering, Lehigh University, Bethlehem, PA 18015 USA

Mark A. Wilson MARK.WILSON@NRL.NAVY.MIL
David W. Aha DAVID.AHA@NRL.NAVY.MIL
Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, Washington, D.C. 20375 USA

Abstract

Goal reasoning (GR) is the study of free agents; they can autonomously and dynamically deliberate on and select what goals/objectives to pursue (Cox, 2007; Muñoz-Avila et al., 2010; Klenk et al., 2013; Vattam et al., 2013; Roberts et al., 2014). Endowing agents with this capability is particularly appropriate when the domain in which they operate is complex (e.g., partially observable, dynamic, multiagent), preventing the anticipation of all possible states and the precise pre-encoding of contingent plans for those states. Most GR agents monitor and assess the current state with respect to potential expectation violations or motivation triggers (Coddington et al., 2005). This deliberation may result in selecting an alternative to the goal(s) currently being pursued, requiring a planner to generate a corresponding set of actions for a controller to schedule and execute. In this paper, we study domain-independent goal selection, extending a method that combines motivations which was implemented in the GR agent M-ARTUE (Wilson et al., 2013). In particular, we relax the assumption that all motivations contribute equally to goal selection, and investigate the relationship between domain properties and motivator contribution in a paradigmatic domain. We view this as a step towards a deeper understanding of how motivations affect agent performance. Future work includes automatically learning motivator weights.

1. Introduction

In complex (e.g., partially observable, dynamic, and multiagent) environments, an autonomous agent may need to alter its own goals to be successful. For instance, an agent that flies an unmanned aerial vehicle may need to change its goal to refuel or recharge if it encounters unexpectedly strong headwinds. We refer to agents that can deliberatively select their own goals as *goal reasoning* (GR) agents. A crucial problem in GR is that of *goal selection* (i.e., deciding which goal(s) the agent should choose to pursue when it is appropriate to pick a new goal). One possible approach to goal selection is the use of domain-independent *motivators*, which encode high-level drives and rely on the agent's own internal models, as used by the M-ARTUE GR agent (Wilson et al., 2013). We refer to agents that employ such motivators as examples of *motivated agents*. M-ARTUE selects its goals according to a function defined on the following set of motivators:

- *Social Motivator*: This chooses user-provided goals.
- *Exploration Motivator*: This chooses goals that best expand the agent's world knowledge.
- *Opportunity Motivator*: This chooses goals that that maximize the agent's opportunity to act during plan execution, such as by conserving resources.

One way for an agent to combine these motivators is for it to assign the same weight to each motivator and repeatedly: (1) achieve some of the user's goals, (2) perform some exploration (from time to time), and (3) take action to conserve resources as necessary. However, this strategy might not work well in many situations.

For example, suppose an agent's user-provided goals involve delivering packages in a graph-like world where locations are nodes and edges are direct connections between locations with associated traversal costs. Suppose also that agents consume gas proportionally to these costs, gasoline stations are available in some (but not all) locations, and some information (e.g., some connections and the location of some gasoline stations) is initially unknown to the agent. In extreme situations, as in the Buridan's ass paradox,[1] the agent might oscillate between the three motivators, achieving few of the goals and (literally) running out of gas. Indeed, we hypothesize at least three scenarios where the balanced strategy might be inadequate:

1. *Observable Environment*: Most of the information is known to the agent. That is, most of the connections and gasoline station locations are known. In this case, performing exploration is not advisable as resources will be consumed for likely little benefit. Instead the agent should follow the **social** motivation to achieve the maximum number of user-provided goals, and use **opportunity** motivation to conserve resources otherwise. In this case no weight should be given to exploration.
2. *Hidden Environment*: Most of the information is not known by the agent. That is, the agent only knows about a fraction of the connections and gasoline station locations. Then the agent should emphasize **exploration** and place less emphasis on the Social and Opportunity Motivators until sufficient information has been gathered to fulfill most or at least many of the user-provided goals.
3. *High Resource Capacity*: The agent can retain a large quantity of resources that may be spent to achieve goals. In this case, it should emphasize the **Opportunity** Motivator and gather as many resources as possible for achieving social goals.

Figure 1 illustrates these three boundary cases. This raises the question of what kind of weight relations will exist among these motivators for the intermediate cases.

[1] The Buridan's ass (Rescher, 1959) is a philosophical paradox of an animal that is very thirsty and very hungry and can't decide between drinking from a nearby water fountain and eating from a nearby stack of hay that is located in the opposite direction from the fountain. The animal dies of hunger and thirst, never able to make a decision.

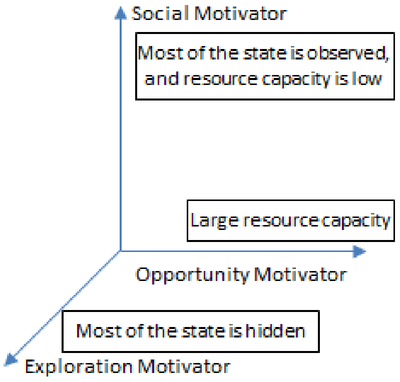

Social Motivator
Most of the state is observed, and resource capacity is low
Large resource capacity
Opportunity Motivator
Most of the state is hidden
Exploration Motivator

Figure 1. Boundary Cases for three Motivators

In this paper, we describe preliminary work on strategies for assigning different weights to the motivators in an M-ARTUE agent. Specifically, we study the relation between motivator weights and agent performance in different contexts, with the future objective of automatically learning weight settings that denote the relative, scenario-specific importance of the motivators. We present results from an initial investigation on the interaction of certain domain properties with the effects of motivator weight mixtures on agent performance. After summarizing related work (Section 2) and reviewing the M-ARTUE agent (Section 3), we report on a study (Section 4) using the Metric Transportation domain, a new variant of the Logistics Transportation domain (Veloso, 1994) whose characteristics were outlined above and are detailed further in Section 4. In our study, we found support that a motivated agent's performance varies when its motivators are given different weights, and that an agent's performance varies predictably based on domain and problem properties. We conclude and discuss future work in Section 5.

2. Related Work

Standard planning techniques can be used to solve problems in which all state information is known (Ghallab et al., 2004). In this situation, the planner can be tasked with achieving all package-delivery goals. The planner will attempt to achieve all the goals and backtrack as needed. If the planner is complete, it will generate one plan to achieve all of the goals, or indicate that such a plan doesn't exist. This can be interpreted as an instance of the motivators where all weight is given to the Social Motivator and no weight is given to the Exploration and the Opportunity Motivators.

Planning research has relaxed this "all-or-none" requirement for achieving the goals. Oversubscription planning attempts to find the maximum number of goals that can be achieved for a particular problem (Smith, 2004). Techniques such as expanding a planning graph, a compact representation of a set of solution plans, to select a subset of the goals to achieve has been explored for this purpose (Do et al., 2007). This can also be interpreted as an instance where all weight is given to the Social Motivator.

Our research is also related to planning for information gathering. In this form of planning, the agent navigates a partially observable state. Frequently, the fact that information must be gathered is explicitly expressed in a planning language as attached conditions to the actions (Draper et al., 1994) or as subgoals (Pryor & Collins, 1996). As a result, plans are generated that perform information-gathering tasks. This is akin to giving significant weight to the Exploration and Social Motivators in our work. A major difference with our work is that, in planning for information gathering, the amount of exploration will depend on the static encoding of the symbolic representation of the domain. In contrast, we are interested in the behavior of agents as weights for exploration and social activities, along with other behavior, are independently varied.

Domains such as the Metric Transportation domain have been a focus of optimal planning, which is the generation of plans that minimize or maximize some metric such as minimizing gasoline use (Williamson & Hanks, 1994). Optimal planning has been subject to extensive research (Kuffner, 2004) including using Djikstra-like search procedures that guarantee that the optimal solution will not be missed but at a potentially high computational cost of carrying out an exhaustive search in the plan space. As a result, optimal algorithms are computationally much less efficient than their non-optimal counterparts. Furthermore, partial observability has not been studied in the context of plan optimality, whereas in our work it is a central topic.

Our work is also related to replanning (Stentz, 1995), where a plan is modified as a result of changes in the environment. Techniques for replanning include using heuristics to determine the best way to complete a plan from the state where the change was detected. Most of the work on replanning concentrates on failures (e.g., the plan expected to find gasoline at a location but upon arrival it finds none). In contrast, we are considering an Exploration Motivator, which can be viewed as “exploring for the sake of exploring” even in situations where the current goals could be achieved. Our objective is to develop robust systems in which, for example, even when the goals change (e.g., new packages must be picked up and delivered) the system has pro-actively gathered information that enables it to react to goal changes.

Finally, our work is also related to cognitive architectures and goal reasoning agents. Some cognitive architectures (Langley et al., 2009) use rules of the form ***if*** *conditions* ***then*** *goal*, which trigger the next goals to achieve depending on the current conditions. Here conditions can be broadly constructed to include annotations about the world state and actions in the current plan. The continuous-concept matching employed by Choi (2011) extends this representation to permit arbitration between current goals based on priority values dynamically computed from the degree of match offered by a goal’s conditions. This serves a similar role to the fitness functions employed by M-ARTUE, but is based on domain-specific rules encoded in the agent’s conceptual knowledge, whereas M-ARTUE employs domain-independent motivators. Others, such as MADBot (Coddington et al., 2005) or ARTUE (Klenk et al., 2013), represent motivations using domain knowledge to encode thresholds or conditions for known variables that the agent can observe. Thus, the goal selection knowledge is hard-coded in the rules. In contrast to these efforts, motivated agents prioritize goals according to the different motivators. This provides flexibility for learning because the relation between goals and goal selection is not hardcoded. Other goal reasoning systems such as LGDA (Jaidee et al., 2011) use reinforcement learning (RL) techniques to learn goal selection knowledge. Since these systems are guided by a user-defined reward function, they can be viewed as achieving social motivations in our parlance. As RL systems they

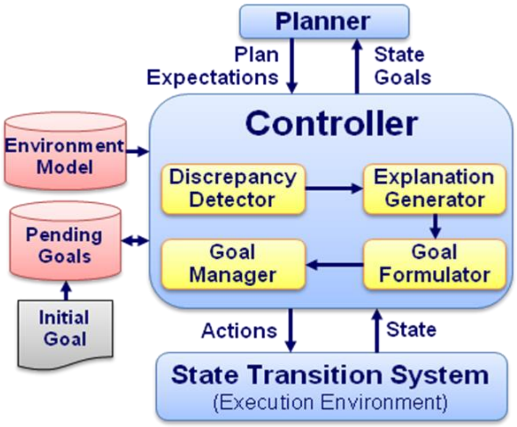


Figure 2. The Goal-Driven Autonomy Model implemented by M-ARTUE

perform exploration and exploitation of the search space, which in this case is goal selection knowledge.

3. The M-ARTUE Agent

M-ARTUE is an extension of the ARTUE GR agent. ARTUE implements Goal-Driven Autonomy (GDA), a GR model extending Nau's (2007) model of online planning. The GDA model is depicted in Figure 2.

ARTUE is composed of an online Controller, which interacts with a Planner and a State Transition System $\Sigma = (S, A, F, \gamma)$, with possible states S, actions available to the agent A, exogenous events that may be triggered by the environment F, and state transition function $\gamma: S \times (A \cup F) \rightarrow S$. The Planner receives as input the current state s_c, the current goal g_c, and a model of the State Transition System M_Σ; it produces as output a sequence of actions $\langle a_{c+1}, \ldots, a_{c+n} \rangle$ and a corresponding sequence of expectations $\langle x_{c+1}, \ldots, x_{c+n} \rangle$, where x_i is the state expected to follow a_i during execution.

As the agent executes a plan returned by the Planner, it performs four GR steps:

- **Notable Event Detection:** The agent executes the current plan and compares observed states from the environment with the sequence of expected states produced by the Planner. If the current observation differs meaningfully from the expected state, the agent notes the discrepancy and performs the following steps.
- **Notable Event Explanation:** The agent produces an explanation for a given discrepancy, where the explanation is frequently an adjustment to the agent's beliefs that incorporates unobserved (but abduced) facts and exogenous events.
- **Goal Formulation:** The agent produces a new goal, if necessary, that is deemed an appropriate response to a given explanation.
- **Goal Management:** The agent selects a goal or goals to pursue from goals that were formulated during a current or previous GR sequence.

M-ARTUE extends ARTUE by performing goal selection (i.e., goal formulation and goal management) through the application of motivators. When a notable event is detected and explained, M-ARTUE evaluates all possible goals for the agent using its motivators and selects the goal with the best combined score. Although the goal-driven autonomy model is planner agnostic, ARTUE implementations have historically used hierarchical task network (HTN) planners. Thus, possible goals for M-ARTUE are enumerated from a list of tasks in an HTN that are designated as top-level tasks for the domain. Any grounding of such a task in the current state for which a plan can be constructed is considered a possible goal, and M-ARTUE uses the constructed plans to evaluate their respective goals.

3.1 Motivators

Each motivator calculates an *urgency* value that indicates how important it is to fulfill its current needs. Urgency is defined as a function $u_m: S -> \mathbb{R}$, which expresses how urgent a particular motivator m's needs are in the current state $s_c \in S$. Each motivator evaluates the *fitness* of each goal g for satisfying its domain-independent needs by applying a motivator-specific fitness function $f_m: \langle x_c, x_{c+1}, \dots x_{c+n} \rangle \rightarrow \mathbb{R}$ to the expectations $x_{c+1}, \dots x_{c+n}$ generated by the Planner. Finally, for each goal, a weighted sum over the motivators is calculated, defined as:

$$fitness(g) = \sum_m u_m\,(s_c) \times f_m\big(Expectations(g, s_c)\big),$$

where g is a goal and $Expectations(g, s_c)$ is the list of expectations X returned by the Planner when given a goal g in state s_c. The goal g with the highest $fitness(g)$ is selected.

3.1.1 Social Motivator

The Social Motivator captures the need to achieve the user-designated goals. Currently, these are represented by a list of state conditions that must be true to satisfy a user-designated goal. The Social Motivator's urgency is a sawtooth function that increases over time until a user-defined goal is fulfilled. This function biases goal selection toward social conditions when they have not been achieved in some time. It is defined by the function:

$$u_{social}(s_c) = \begin{cases} C_{social} u_{social}(s_{c-1}), if\ R(s_c) \leq R(s_{c-1}) \\ 0.1, if\ R(s_c) > R(s_{c-1}) \end{cases},$$

where s_c is the state at the time of goal selection, $R(s_c)$ is the percentage of user-provided goals that have been satisfied in s_c or some prior state $s_i (i < c)$ visited, and $C_{social} > 1$ is a constant of social motivation that is tuned to the domain.

The fitness function for the Social Motivator biases goal selection toward goals that achieve the most social conditions with the fewest actions. It is calculated as:

$$f_{social}(X) = C_{social-fitness} \frac{R(x_{c+n}) - R(s_c)}{n},$$

where X is the sequence of expected states as defined above, n is the plan's length, $C_{social-fitness}$ is a constant of social fitness that is tuned to the domain, and x_{c+n} is the expected state after the plan executes.

3.1.2 Exploration Motivator

The urgency of the Exploration Motivator is biased to increase when the most recent action has not visited a new unique state, and to be large when fewer states overall have been visited (i.e., exploration is most valued when little to no exploration has been done). It is defined as:

$$u_{exploration}(s_c) = 1 - \frac{V(s_0, s_1, \ldots, s_c)}{V(s_0, s_1, \ldots s_{c-1}) + C_{exploration}},$$

where $V(S)$ is the number of distinct states in a list S and $C_{Exploration}$ is a constant of exploration that is tuned to the domain.

The fitness function biases goal selection toward goals that visit the most new unique states per action. This function is defined as:

$$f_{exploration}(X) = \frac{V(s_0,s_1,\ldots s_c,x_{c+1},x_{c+2},\ldots x_{c+n}) - V(s_0,s_1,\ldots s_c)}{n}.$$

3.1.3 Opportunity Motivator

The Opportunity Motivator tries to maximize the agent's opportunity to act throughout plan execution, thus helping the agent to prepare to fulfill future goals. This is evaluated in terms of two factors: (1) the branching factor in a given state and (2) the availability of resources relative to their historical averages. These factors are combined to determine this motivator's urgency, which biases selection toward opportunity-increasing goals when the agent cannot execute as many actions or it does not possess as many resources as have been available historically. This function is defined as:

$$u_{opportunity}(s_c) = \left[\left(1 - \frac{N(s_c)}{\max\limits_{0 \le i < c} N(s_i)}\right) + (1 - L(s_c))\right]/2,$$

where $N(s)$ is the number of available actions, and $L(s)$ is the level of resources relative to historical resource levels. A domain defines a set of k resources, each of which has a state-based level $v_r(s)$. Function $L(s)$ is defined in terms of these levels as $L(s_c) = (\sum_{r=1}^{k}[v_r(s_c)/a_r(s_c)])/k$, where $a_r(s_c) = \frac{\sum_{i=1}^{c-1} v_r(s_i)}{c-1}$ is the mean of all prior values for $v_r(s)$.

The Opportunity Motivator's fitness function biases goal selection toward goals that have the most actions available per expected state, and leaves the agent with the most resources and actions available when the goal is achieved. This function is defined as:

$$f_{opportunity}(X) = \frac{\left(\left[\sum_{j=0}^{n-1} N(x_{c+j})\right] + [w \times N(x_{c+n})]\right)}{(n+w)N(s_c)} + L(x_{c+n}) - L(s_c) - 1,$$

where $w \geq 1$.

4. Experiments and Discussion

We performed experiments to evaluate two hypotheses:

- **H0:** Varying motivator weights will affect agent performance.

- **H1:** Agent performance will vary with motivator weights in a predictable fashion as certain properties of the evaluation scenario change. Specifically, the scenario properties we investigated are:
  - *Initial observability:* When observability is low, we expect the number of goals achieved to be greater when the relative weight of the Exploration Motivator increases, encouraging the agent to observe more about its environment. Conversely, when observability is high we expect the number of goals achieved to be greater when the relative weight of the Social Motivator increases.
  - *Resource capacity:* When the agent has greater resource capacity, we expect the number of goals achieved to increase as the relative weight of the Opportunity Motivator increases, encouraging the agent to gather as many resources as possible. Conversely, when resource capacity is low, we expect the number of goals achieved to be greater when the relative weight of the Social Motivator increases.

To test these hypotheses, we ran M-ARTUE on scenarios from the Metric Transportation domain, a modified version of the Logistics Transportation domain. In this domain, an agent is given an initial set of goals by a user, who directs it to deliver a specified set of packages to a set of discrete destinations, which are located (as nodes) in a partially-connected graph. To achieve these goals, the agent uses trucks and airplanes to move packages. Our modified domain omits airports and airplanes, but includes a fuel function on trucks (i.e., a given truck's current fuel level), which decreases as the truck moves between connected locations according to a cost function defined on the connections. Additionally, our modifications permit the scenario author to initially hide some connections between locations, some gas stations, and some packages' locations. M-ARTUE discovers these hidden facts through the occurrence of observation events when a truck moves to a relevant location (i.e., one of the connected locations, a gas station's location, or a package's location, respectively). To guide M-ARTUE's goal selection and HTN planner, we created an HTN definition encompassing top-level tasks that allow the agent to deliver a package to a particular location, drive a truck to a particular location, refuel a truck, or do nothing.

We randomly generate scenarios in this domain according to parameters controlling: the size of the graph; the number of trucks, packages, and gas stations; the amount of fuel available to the trucks initially and after fueling at stations; and the connectedness of the graph. These parameters control the difficulty of the agent's planning problems. We also individually specify percentages of connections, packages, and gas stations that will be visible to the agent initially. These parameters impact the difficulty of the agent's goal-achievement problem. To evaluate the agent's performance, we use as a metric the fraction of total user goals achieved (i.e., the number of packages successfully delivered to their destinations).

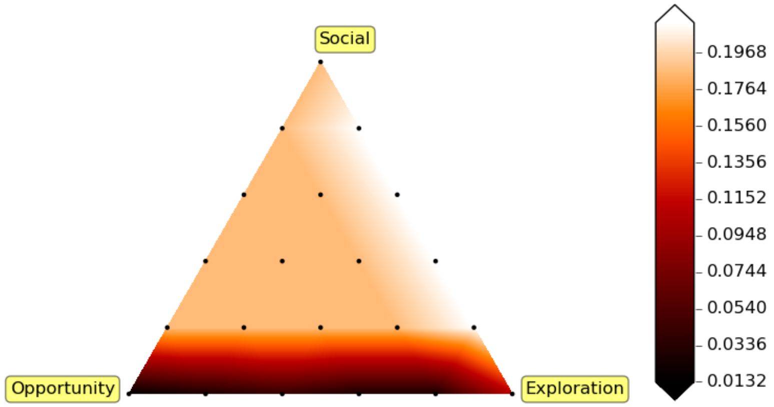


Figure 3. Average fraction of user goals achieved with high initial observability across 25 trials (maximum 500 actions) at indicated weight-mixture points

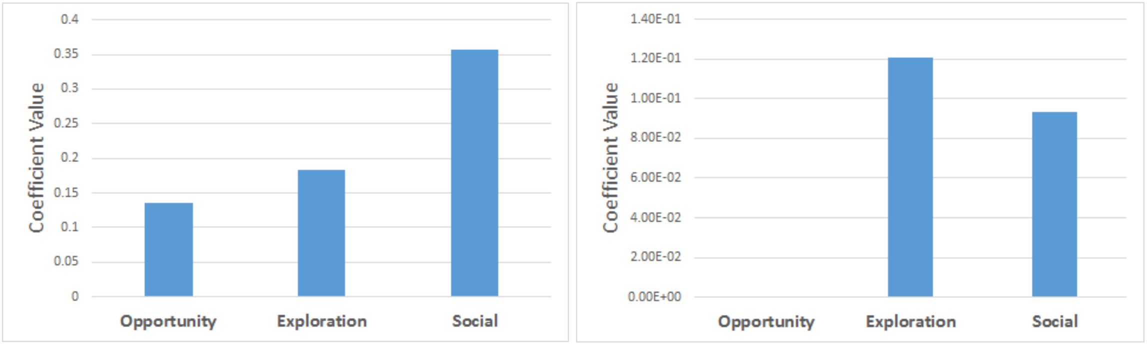
4.1 Motivator Weights and Agent Performance

To test hypothesis H0, we generated several scenarios with the same parameters. Specifically, for this test we adopted the scenario parameters $n_{locations} = 20, n_{trucks} = 3,\ n_{packages} = 3$, $n_{stations} = 3$, and $n_{connections} = 3$, with all packages and gas stations initially visible to the agent, and 70% of edges initially known to the agent. We generated 25 scenarios using these values. In each scenario, we evaluated the agent's performance at different weight mixtures corresponding to combinations that sum to 1 of the values (0, 0.2, 0.4, 0.6, 0.8, 1) for all motivators. We limited the test to 500 actions on each scenario-weight pair. (Since the agent can continue indefinitely, we chose 500 actions to provide a reasonable tradeoff of running time and goal achievement.) The average fraction of user goals accomplished at each weight mixture is shown in Figure 3.

Visually, the agent performs best in a region dominated by non-zero values of the Social Motivator. Additionally, the agent's performance is better when the Exploration Motivator is weighted more heavily than the Opportunity Motivator, indicating that exploration may improve agent performance by revealing even a few hidden environmental features (connections between locations, in this case). (Occasionally, the agent may deliver a package simply by trying different exploratory actions, even when the Social Motivator has no weight, as can be seen by the slight improvement in performance at the Exploration Motivator's corner.) An analysis of variance on the average agent performance indicates that the performance differs significantly across the range of weights ($p < 5 \times 10^{-8}$ for all components), supporting hypothesis H0.

4.2 Motivator Weights and Scenario Properties

To test hypothesis H1, we altered the 25 scenarios described in Section 4.1 to provide contrast in the desired scenario properties.



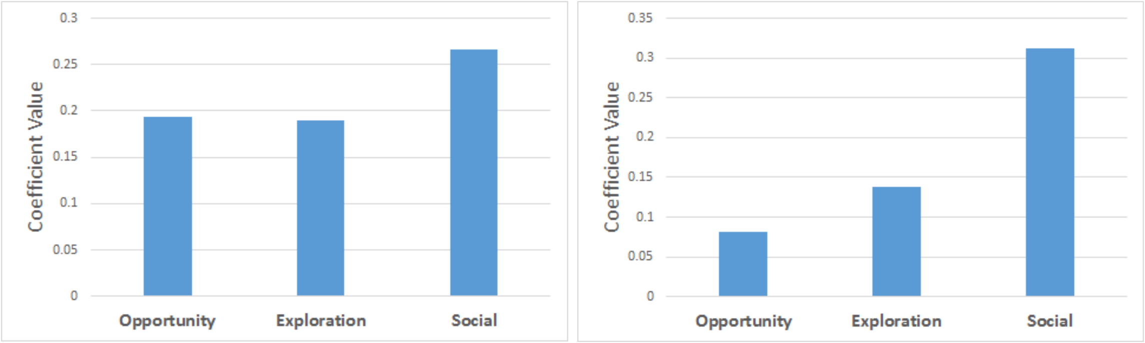
(a) Scenarios with high observability *(b)* Scenarios with low observability

Figure 4. Coefficients of motivator weight for agent performance

- **Observability:** We compared the original scenarios with modified versions that initially revealed none of the packages, gas stations, or connections, except those relevant to the trucks' starting locations.
- **Resource capacity:** We altered the scenarios to provide complete observability of the graph (to avoid any impact the Exploration Motivator might have on resource consumption by revealing shorter routes). We then compared these completely-observable scenarios using the original resource levels with modified versions that provide higher initial fuel levels and fuel capacity (95 and 150, compared to 25 and 40, respectively).

For each alteration to the scenarios, we evaluated the agent as in Section 4.1, using the same weight-mixture points and limiting the agent to 500 actions. We then compared the motivators' contributions to agent performance under the differing scenario properties by fitting a canonical linear mixture model, $y = \sum \beta_i x_i$, to the data for the original scenarios and the alterations. The results were as follows:

- **Observability:** Figure 4 depicts the motivator coefficients β_i in the linear fit model. These values indicate how strongly each motivator is correlated with agent performance in the 25 scenarios for the original high-observability scenarios and the matching low-observability scenarios. In the high-observability scenarios, the Social Motivator correlates most strongly with agent performance. By contrast, the Exploration Motivator correlates most strongly with agent performance in the low-observability scenarios, supporting our hypothesis that the importance of exploration increases (i.e., the agent achieves more goals when the Exploration Motivator is heavily weighted) when the environment exhibits low initial observability, as the agent cannot achieve user goals without discovering routes and gas stations, and the need to discover those features outweighs the need to conserve resources. In fact, in this extreme scenario, conserving resources contributed nothing to agent performance. The smaller numerical values of the coefficients in the low-observability scenarios are due to lower overall agent performance, as the environment is more challenging. (Note that, while the linear fit was significant for all three components in the high observability scenarios, it was not significant for the Opportunity Motivator in the low observability scenarios, supporting the conclusion that the Opportunity Motivator was not a significant contributor to agent performance in those experiments.)



(a) Scenarios with high resource capacity

(b) Scenarios with low resource capacity

Figure 5. Coefficients of motivator weight for agent performance

- **Resource capacity:** Figure 5 depicts the motivator coefficients β_i in the linear fit model. These values indicate motivator correlation with agent performance in the 25 high-resource scenarios and the matching low-resource scenarios. While the Social Motivator is the dominant contributing motivator to agent performance in all scenarios, the Opportunity Motivator contributes much more to agent performance in the high-capacity scenarios than in the low-capacity scenarios, supporting our hypothesis that the importance of resource acquisition increases as a function of the agent's ability to gather and retain more resources to assist in achieving goals. (The linear fit was significant for all components in these experiments.)

5. Conclusion and Future Work

The paradox of Buridan's ass exemplifies the need to balance between an agent's motivations. We reviewed a GR agent (M-ARTUE) that reasons with Social, Exploration, and Opportunity motivators to deliberate among goal choices. We introduced the Metric Transportation Domain to test the motivators' value in scenarios with varying properties, and showed that (1) varying the motivators' relative weights can impact agent performance and (2) the relative importance of each motivator is context-dependent.

In future work we will investigate the impact of motivator weight settings in other domains (e.g., an underwater vehicle domain and a Mars rover domain). We will also investigate the effects of the motivator weight settings in other extreme scenarios (e.g., when resource consumption is extremely low or resource availability is extremely high). We will use the results of these investigations to identify further predictable domain and problem characteristics that affect motivators' correlation with agent performance, and we will pursue the creation of a more formal model of domain and problem characteristics and their interaction with motivator weights. We will also investigate the effect of motivator weights using alternative metrics of agent performance (e.g., how quickly user goals are achieved and how many resources the agent expends while achieving them). We will investigate the use of other motivators (e.g., a directed information motivator). Finally, we will investigate how an agent can learn weight settings that will enable it to outperform its behavior using fixed equal weights. We will do so using fixed environmental conditions, as we used for these experiments (e.g., resource capacity is static throughout an experiment), and environments in which these conditions may change dynamically.

Acknowledgements

This work was supported in part by the National Science Foundation.

References

Coddington, A.M., Fox, M., Gough, J., Long., D., & Serina, I. (2005). MADbot: A motivated and goal directed robot. *Proceedings of the Twentieth National Conference on Artificial Intelligence* (pp. 1680-1681). Pittsburgh, PA: AAAI Press.

Cox, M.T. (2007). Perpetual self-aware cognitive agents. *AI Magazine*, *28*(1), 32-45.

Do, M.B., Benton, J., van den Briel, M., & Kambhampati, S. (2007). Planning with goal utility dependencies. *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 1872-1878).

Draper, D., Hanks, S., & Weld, D. S. (1994). Probabilistic planning with information gathering and contingent execution. *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems* (pp. 31-36). Chicago, IL: AAAI Press.

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

Jaidee, U., Muñoz-Avila, H., & Aha, D.W. (2011). Integrated learning for goal-driven autonomy. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*. Barcelona, Spain.

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.

Kuffner, J.J. (2004). Efficient optimal search of uniform-cost grids and lattices. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 1946-1951). Sendai, Japan: IEEE Press.

Langley, P., Laird, J. E., & Rogers, S. (2009). Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, *10*(2), 141-160.

Muñoz-Avila, H., Aha, D.W., Jaidee, U., Klenk, M., & Molineaux, M. (2010). Applying goal directed autonomy to a team shooter game. *Proceedings of the Twenty-Third Florida Artificial Intelligence Research Society Conference* (pp. 465-470). Daytona Beach, FL: AAAI Press.

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

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

Roberts, M., Vattam, S., Alford, R., Auslander, B., Karneeb, J., Molineaux, M., Apker, T., Wilson, M., McMahon, J., & Aha, D.W. (2014). Iterative goal refinement for robotics. In A. Finzi & A. Orlandini (Eds.) *Planning and Robotics: Papers from the ICAPS Workshop*. Portsmouth, NH: AAAI Press.

Smith, D.E., (2004). Choosing objectives in over-subscription planning. *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling* (pp. 393-401). Whistler, BC, Canada: AAAI Press.

Stentz, A. (1995). The focussed D* algorithm for real-time replanning. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1652-1659). Montreal, Quebec, Canada: Morgan Kaufmann.

Vattam, S., Klenk, M., Molineaux, M., & Aha, D.W. (2013). Breadth of approaches to goal reasoning: A research survey. In D.W. Aha, M.T. Cox, & H. Muñoz-Avila (Eds.) *Goal Reasoning: Papers from the ACS Workshop* (Technical Report CS-TR-5029). College Park, MD: University of Maryland, Department of Computer Science.

Veloso, M.M. (1994). *Planning and learning by analogical reasoning*. Springer: Berlin.

Williamson, M., & Hanks, S. (1994). Optimal planning with a goal-directed utility model. *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems* (pp. 176-181). Chicago, IL: AAAI Press.

Cleaning Efficiently:
A Case Study in Modeling Goal Reasoning and Learning

Mark Roberts MARK.ROBERTS.CTR@NRL.NAVY.MIL
NRC Postdoctoral Researcher, Naval Research Laboratory (Code 5514), Washington, DC USA

David W. Aha DAVID.AHA@NRL.NAVY.MIL
Adaptive Systems Section, Naval Research Laboratory (Code 5514), Washington, DC USA

Abstract

Goal Reasoning concerns actors that deliberate about their own goals, and learning is often applied to improve performance of such actors. A recent model of Goal Reasoning proposed by Roberts et al. lacks a simplified working example with an explanation of how the model can incorporate learning. We describe a thought experiment modeling the decision making of floor-cleaning robots, which are simple goal reasoning actors. We observe that such robots often have a less-than-optimal action policy that, though sufficient for cleaning in the limit, could be improved through learning to reduce the total cleaning time or reduce energy usage. We start with by modeling the straightforward policy for a simple robot named Vacuous. We then ponder a hypothetical robot, named KleeneStar, which optimally cleans in the fastest time possible. Finally, we describe a model for an improved robot, named Vakleene, which iteratively reduces its cleaning time through learning. While these examples are only a thought experiment, this work provides a model of a simplistic goal reasoning process that can learn.

1. Motivation

Small, floor-cleaning robots have become a popular home appliance to supplement regular floor cleaning performed by a human. These systems are best exemplified by the Roomba, introduced by iRobot in 2002. The Roomba contains a front bumper to detect a collision, infrared proximity sensors to detect nearby objects or sudden changes in height, and a radio sensor to detect beacons. It can clean a variety of floor surfaces for several hours between charges and newer versions use radio beacons to navigate multiple rooms, avoid hazards, or return to a self-charging base station.

Anyone who has interacted with these devices can observe that its task planning is rudimentary. The devices exhibit three simple behaviors as shown in Figure 1 taken from the Roomba’s User Manual (iRobot 2008). The Spiral behavior allows the device to clean a large area in a spiral fashion. The Wall Following behavior traces a wall or another obstacle to the right side while cleaning close to the edge. Finally, the Room Crossing behavior randomly turns and crosses the room while cleaning. Even with random selection of the actions, the robot should eventually traverse the entire floor in the limit. While sufficient, this action selection mechanism could be improved with learning that accounts for the room and obstacles within the room.

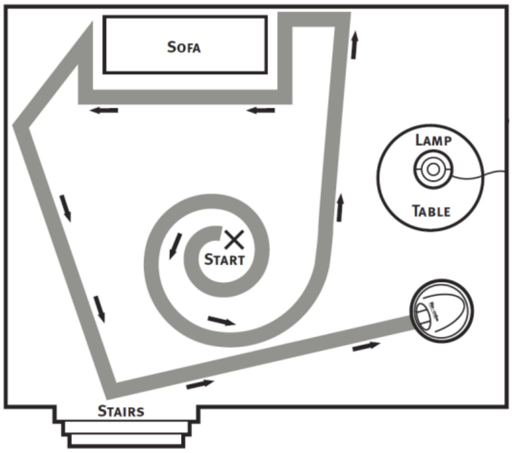


Figure 1. A typical Roomba cleaning pattern from the Roomba User Manual (iRobot 2008).

We will show in this paper how a floor-cleaning robot can be modeled as an instantiation of a Goal Reasoning process. We created a Goal Reasoning Model that captures the variety systems that perform Goal Reasoning (Roberts et al. 2014, 2015). Our model distinguishes systems by their design choices and, thus, facilitates their comparison. For example, instantiations of this model can represent iterative plan repair (e.g., Chien et al. 2000), replanning (e.g., Yoon et al. 2007), and Goal-Driven Autonomy (e.g., Klenk et al. 2013).

We have two aims in this paper. First, we model the robot vacuum as a Goal Reasoning process using the Goal Reasoning Model. Second, we describe how learning could augment this basic model to reduce cleaning time. We present a thought experiment to elucidate instantiations of Goal Reasoning, using the robot vacuum as a case study. We begin with a brief exposition of the model, and then use it to model three robot vacuum agents: (1) Vacuous is a simple robot with a straightforward, fixed action selection policy that is extremely suboptimal. (2) KleeneStar[1] is a hypothetical optimal robot vacuum that can guarantee cleaning in the minimal cleaning time but whose policy could never be realistically computed on such a limited robot platform. (3) Vaklene is a learning-enabled robot vacuum whose cleaning time asymptotically approaches KleeneStar.

2. Background: The Goal Reasoning Model

Deliberating about objectives – how to prioritize and attain (or maintain) them – is a ubiquitous activity of all intentional entities (i.e., actors). We apply the Goal Reasoning Model presented by Roberts et al. (2014, 2015) that models Goal Reasoning as a State Transition System consisting of goal nodes that track a goal's state and transitions defined by a Goal Lifecycle. Note that we adopt the notation provided by Roberts et al. (2015).

Let g_i be the actor's i^{th} goal of m goals ($0 \leq i \leq m$) that the actor wishes to attain (maintain). To avoid confusion with the use of the word *state* as it is typically applied in planning systems, we will use L to represent the *language* of the actor, $L = L_{external} \cup L_{internal}$. Often, $L_{external}$ will concern external state the actor is tracking (e.g., its location, its sensor values). $L_{internal}$

[1] With apologies to Stephen Kleene.

represents the predicates and state required internally to the actor (e.g., the predicates $attain(g)$ or $maintain(g)$, the state of all goals).

A goal g is tracked within the actor as part of a goal node N^g that transitions according to a Goal Lifecycle (Figure 2). Decisions consist of applying a *strategy* (arcs in Figure 2) to transition a goal node among *modes* (rounded boxes). Goal nodes in an *active* mode are those that have been formulated but not yet dropped. The **formulate** strategy determines when a new goal node is created. Vattam et al. (2013) describe goal formulation strategies. The **drop** strategy causes a goal node to be "forgotten" and can occur from any active mode; this strategy may store the node's history for future deliberation. To **select** N^g indicates intent and requires a formulated goal node. The **expand** strategy decomposes N^g into a goal network (e.g., a tree of subgoal nodes) or creates a (possibly empty) set of expansions X. Expansion is akin plan generation, but is renamed here to generalize it from specific planning approaches. The **commit** strategy chooses an expansion $x \in X$ for execution; a static strategy or domain-specific quality metrics may rank possible expansions for selection. The **dispatch** strategy slates x for execution; it may further refine x prior to execution (e.g., it may allocate resources or interleave x's execution with other expansions).

Goal nodes in executing modes (Figure 2, dashed lines) can be subject to transitions resulting from expected or unexpected state changes. The **monitor** strategy checks progress for N^g during execution. Execution updates arrive through the **evaluate** strategy. In a nominal execution, the information can be either resolved through a **continue** strategy or the **finish** strategy marks the goal node as *finished*.

During execution, the **evaluate** strategy determines how events or new information impacts goal node and the **resolve** strategies define the possible responses. If the evaluation does not impact N^g, the actor can simply **continue** the execution. However, if the event impacts the current execution then other strategies may apply. One obvious choice is to modify the world model using **adjust**, but this does not resolve the mode of N^g and further refinements are required. The **repair** strategy repairs expansion x so that it meets the new context; this is frequently called *plan repair*. If no repair is possible (or desired) then the **re-expand** strategy can reconsider a new plan in the revised situation for the same goal; this is frequently called *replanning*. The **defer** strategy postpones the goal, keeping the goal node *selected* but removing it from execution. Finally, **formulate** creates a revised goal g'; the actor may then drop the original goal g to pursue g' or it could consider pursuing both goals in parallel; this is similar to the concept of goal transformation provided by Cox and Veloso (1998).

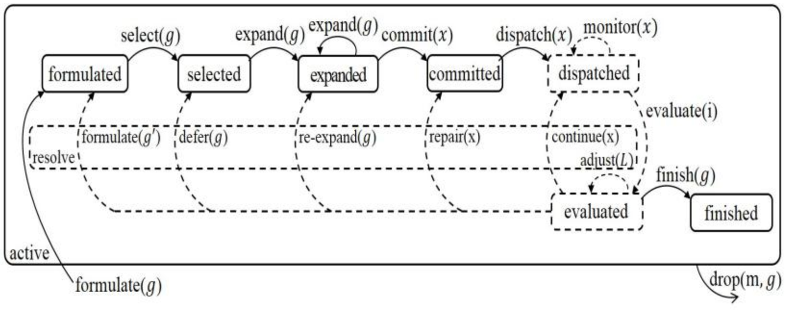


Figure 2: The goal lifecycle (Roberts et al. 2014). Strategies (arcs) denote possible decision points of an actor, while modes (rounded boxes) denote the status of a goal (set) in the goal memory.

Goal Reasoning can be modeled as a goal State Transition System $\mathrm{Z} = (M, R, \delta_{GR})$, where M is a goal memory of goal nodes, R is a set of refinement strategies that transition goals of the system, and $\delta_{GR} : M \times R \rightarrow M'$ is a transition function restricting the allowed transitions for M.

A goal memory M stores a goal node for each of the m goals. For goal g_i, $N^{g_i} = (g_i, parent, subgoals\ , C, o, X, x, q)$ is a goal node where:

- g_i is the goal that is to be achieved (or maintained);
- $parent$ is the goal whose subgoals include g_i;
- $subgoals$ is a list containing any subgoals for g_i;
- C is a set of constraints on g_i. Constraints could be temporal (finish by a certain time), ordering (do x before y), maintenance (remain safe), resource (use a specific resource), or computational (only use so much CPU or memory). A partition $C = C^{provided} \cup C^{added}$ separates constraints into those provided to the actor independent of whatever invoked it (e.g., a human operator, meta-reasoning process, or coach) and those added during refinement. Top-level constraints can be pre-encoded or based on drives (e.g., (Coddington et al. 2005; Young and Hawes 2012)). Hard constraints in C must be satisfied at all times, while soft constraints should be satisfied if possible.
- o is the current goal lifecycle mode (detailed below).
- X is a set of expansions that will achieve goal g_i. The types of expansions for a goal depend on its type. For goals tracking external state, expansions might include a set of plans Π. Other goals might expand into a goal network, a task network, a set of parameters for flight control, etc. The **expand** strategy creates X.
- $x \in X$ is the currently selected expansion, performed with the **commit** strategy.
- q is a vector of one or more quality metrics. For example, these could include the *priority* of a goal, the *inertia* of a goal indicating a bias against changing its current mode because of prior commitments, the net value (e.g., cost, value, risk, reward) associated with achieving g_i using selected expansion x.

The refinement strategies R are drawn from the Goal Lifecycle (Figure 2). For convenience, we sometimes refer to the goal node N^g as simply the goal g, though it should be clear that all strategies are functions that transition some N^g. We partition the refinement strategies $R = R^{provided} \cup R^{added} \cup R^{learned}$ to distinguish between strategies that the actor was provided prior to the start of its lifetime (e.g., through design decisions), representations that were added to its model as a result of execution in an environment (e.g., a new object is sensed), and those it learned for itself (e.g., the actor adjusts its expectations for an action after experience).

The transition function δ_{GR} specifies the allowed transitions between modes because not every strategy will apply to every goal or every situation. In a domain-independent fashion, δ_{GR} is defined by the arcs in the lifecycle. However, a system or domain may modify (through composition, ablation, or additional constraints) the transitions for M.

Once Goal Reasoning is modeled as Z, it is easy to see that it is a process through which M is iteratively refined through transitions of R as restricted by δ_{GR}. The Goal Reasoning Problem and one way of solving it, called Goal Refinement, is defined by Roberts et al. (2014, 2015b). For space reasons, we focus only on the definition of Z for our cleaning robots in this paper.

3. Modeling the robots

As mentioned, the available actions of a Roomba include Spiraling, Wall Following, and Room Crossing. Here we outline our central assumptions and more carefully define these actions so that we can compose them into policies for the three robots Vacuous, KleeneStar, and Vakleene.

We make several assumptions to perform our analysis: static environment, perfect sensing, perfect localization, and deterministic action outcome. These assumptions are unrealistic, given that the robotic platform's sensors and actuators are inexpensive and produce noisy observations. Our assumptions are easily violated by the presence of certain furniture (e.g., short legs that cause the robot to get stuck under an edge) or dynamic objects (e.g., a pet or person). Nevertheless, making these assumptions is necessary to gain traction on our analysis. Because we can model each of these assumptions in a simulator, we can accurately measure their impact on our analysis.

The actuator action space of the robot consists of *Vacuum*, *Turn(radians)*, and *Drive(timeInSeconds)*. Vacuum simply turns on the vacuum motor. Turn is a straightforward action that is always possible. Drive is conditioned on sensor observations. The *bumper* is true iff the robot has driven into an obstacle anywhere along its front side. The *right* obstacle sensor returns the distance to any object on the robot's right side up to 50 centimeters; similarly there are *left* and *front* obstacle sensors. These sensors are strategically angled toward the floor so that they should always return some value within 10 centimeters. If the sensor traverses an edge (e.g., the top step of a stairwell) it returns a very large value, indicating a drop that the robot should avoid. With these sensors observations, it is possible to define a simple, effective *drive* action as follows:

```
drive(timeInSeconds)
    startInSeconds ← getTimeInSeconds()
    elapsedInSeconds ← 0
    while (isSafe() && (elapsedInSeconds < timeInSeconds))
        move_forward(0.1 seconds)
        elapsedInSeconds ← getTimeInSeconds() – startInSeconds
```

where the function isSafe() returns true if (bumper == false) && (front < 10) && (left < 10) && (right < 10) && (front > 3) && (left > 3) && (right > 3).

Turn and Drive are composed into the following complex tasks:

- **Sprial(timeInSeconds)** is composed of turning on the vacuum and executing a series of alternating turn right and drive commands where the radians turned decreases and the time driven increases over time.
- **Follow(timeInSeconds)** is composed of a loop that ensures the robot is between 3 and 5 centimeters of an obstacle on the right and drives forward by half-second intervals, checking to perform corrective turns as needed.
- **Cross(turnInRadians, timeInSeconds)** alternates Turn(turnInRadians) followed by Drive(timeInSeconds). If an obstacle is detected during Drive, the robot continues to the next Turn action.
- **ReturnToBase()** reactively applies Turn and Drive actions to move the vehicle toward the Base, which is sending out a beacon signal.

3.1 Vacuous: The sufficient but sub-optimal robot vacuum

Let us now define a simple task selection policy to clean a floor for the Vacuous. This policy should eventually traverse the entire floor, though it will cover certain areas repeatedly. There may be some cases where obstacles or room shapes cause cycling (partially avoided by using a prime number in the turn radius), in which case it would be easy to add a random component to this policy to break such cycling.

```
VacuousPolicy(cleaningTimeInSeconds)
    startInSeconds ← getTimeInSeconds()
    elapsedInSeconds ← 0
    **while** (isSafe() && (elapsedInSeconds < cleaningTimeInSeconds))
        time ← randomInt(30)
        **if** (3< right < 5) Follow(time)
        **else**
            task ← randomInt(1) //randomly select either 0 or 1
            **if** (task == 0) then Spiral(time)
            **else**
                turn ← $-(19 \times \frac{\Pi}{180})$ // Turn by prime closest to 20 degrees
                Cross(turn, time)
        elapsedInSeconds ← getTimeInSeconds() – startInSeconds
    returnToStation()
```

To model this policy as a Goal Reasoning process, we define a system architecture and its goals. Figure 3 shows the system architecture of our model. The Goal Reasoner manages goals, refines them via strategies, and sends a single instantiated task to the Executive, which applies Turn or Drive commands to the robot. The Executive receives sensor readings from the robot and abstracts them into updates for the Goal Reasoner.

We define goals for maintaining safety and signaling that the cleaning time has expired plus a goal for each task. All goals in this model have identical formulate and select strategies. When the goal memory of Vacuous initializes, it formulates and selects all goals. The goals are then triggered by specific events. The TimeExpiredGoal signals the end of the cleaning time. The expand strategy is a NoOp and this goal is immediately dispatched. It monitors the system time and calls evaluate(TimeExpired) to all other goals. The goal is then marked as finished and dropped. The MaintainSafetyGoal ensures the robot remains safe. Similar to the TimeExpiredGoal, the expand strategy is a NoOp and this goal is immediately dispatched. It monitors the sensor status and, when the bumper is activated or when left/right/front detects a drop, it calls evaluate(Unsafe) to all other dispatched goals. Since this goal is central the robot's safety, it is never marked as finished or dropped (i.e., its finish and drop strategies are undefined thus prohibiting these transitions).

The remaining goals are named according to the task they manage: SprialGoal performs the Sprial(timeInSeconds) task, FollowGoal performs Follow(timeInSeconds), CrossTaskGoal performs Cross(turnInRadians, timeInSeconds), and ReturnToBaseGoal performs ReturnToBase(). Each of these goals is selected automatically (as are all goals in this domain) and will then cycle through the expanded, committed, dispatched, and evaluated modes during the execution of a cleaning cycle. Thus, the strategies of the task-oriented goals form the core functionality of this model for controlling the robot. For each goal managing a task, the expand strategy creates a single policy that is eventually dispatched to perform its respective task with the appropriate random time (or turn) variables. Since only one task can be sent to the Executive at

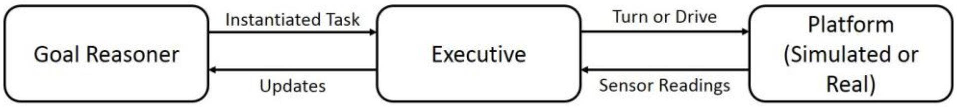


Figure 3: The system architecture of the floor-cleaning robot.

one time, the goals must coordinate via a global semaphore, ExecutiveLock. Therefore, the commit strategies of the goals check this lock before they commit to performing the task and release the lock when applying a resolution strategy that results in moving back to selected (via the defer strategy). This provides an opportunity for other goals to apply commit if they are able to do so. Otherwise, a task goal stays in dispatched until it receives information, via the integrate strategy, that it should pause. The TimeExpiredGoal or MaintainSafetyGoal can signal TimeExpired or Unsafe and a goal transition to committed if one of these is active. At this point, any dispatched task goal should apply an evaluate strategy followed by a resolve strategy.

Once formulating the goals, the Goal Reasoner simply iterates through the goals in the system and attempts to apply the next available strategy. If a goal can proceed, it does. Otherwise it remains in its current mode.

3.2 KleeneStar: The optimal robot vacuum

It is straightforward to show that an optimal policy exists for the robot vacuum given the simplifying assumptions we made above. We can reduce this problem to the Travelling Salesperson Problem (TSP). First, we discretize the floor into cells small enough such that a "visit" by the robot equates to having cleaned that cell. We then label each cell and create an adjacency map of the entire map. We solve the problem with any algorithm suitable for the TSP.

Unfortunately, TSP is an NP-Hard problem, and we face some obstacles implementing it on the limited computing platform of the robot's microcontroller. This leads us to our approximation using machine learning techniques, as outlined next.

3.3 Vakleene: The sub-optimal robot vacuum that learns

We claim that applying learning could improve the cleaning time of Vacuous. The parameters for learning are straightforward in this model, and consist of learning how often to apply each task and how long to perform each task. Let $cleanTime$ be the maximum time for the robot to clean. For task i, let p_i denote the probability of performing the task, let $t_i = (\mathrm{minTime_i}, \mathrm{maxTime_i})$ denote the minimum and maximum time to perform the task where $0 \leq \mathrm{minTime_i} \leq \mathrm{maxTime_i} < \mathrm{cleanTime})$, and let $r_i = (\mathrm{minRadian_i}, \mathrm{maxRadian_i})$ denote the minimum and maximum radians the robot can turn for the task where $-\pi \leq \mathrm{minRadian_i} \leq \mathrm{maxRadian_i} \leq \pi$. Then an instantiation of the system is determined by the parameters $F = (cleanTime, reward, p, t, r)$, where $reward$ is a count of the number of cells visited, p, t, r are the task probabilities, time and radian values and $p_{Spiral} + p_{Follow} + p_{Cross} = 1$.

It should be clear that the Vacuous is one instantiation of F. The parameters chosen for Vacuous may have given the best performance across a variety of rooms and obstacles. We hypothesize that a specific room (and room obstacles) lead to a structure that favors particular instantiations of F. If true, then Vacuous could be improved by learning F for the room that it is currently cleaning, leading to Vakleene, a cleaning robot that can learn.

Vakleene could apply learning (at least) two ways. First, it could perform online modification of F based on how much cumulative reward each task earns over time. For example, Vacuous could proportionally increase the time t_i for task i if, during the past k tasks run, it earned more reward; it might even adjust F as a function of the discounted reward over the past k tasks run (cf. Kaebling, Littman, & Moore 1996). Similarly, Vakleene could modify r_i for a task i that is getting insufficient reward. To help it avoid local minima in the parameter space, Vakleene could use a stochastic component (e.g., simulated annealing) to broaden its parameter search. Online learning like this can be modeled as part of the defer strategy, which checks the last k tasks run and modifies F appropriately.

There are a number of problems with this approach. Learning is performed during cleaning, when computational resources are probably better spent maintaining the robot's safety. This process is an iterated search over a large parameter space. Given the relative infrequency of interaction with its environment (i.e., daily cleaning), converging to a maximal (or even reasonable) policy may likely take many months and Vakleene may traverse many poor configurations for F. Finally, our solution is brittle from a software engineering perspective because changes to the learning procedure require modifying every task goal.

Another approach may be to introduce an offline learning goal, which adjusts the parameters while the vehicle is charging. Consider a new goal, **ApproximateKleeneStarGoal**, which could be dispatched during charging to consider the rewards earned during the past k runs, consider any global constraints for good configurations of F, consider how often obstacles or drops were encountered, etc. If the robot were provided more accurate localization in a later version, this goal could also consider this information. Software updates to the system then consist of patching one or more strategies of ApproximateKleeneStarGoal. As an advantage, this goal could be provided (or learn) a table of common room types to speed up its learning.

There are many other ways that learning could be modeled in the Goal Reasoning Model, but these two examples suffice to demonstrate our point.

4. Related Work on Machine Learning and Goal Reasoning

Several researchers have investigated methods for applying machine learning techniques to improve the performance Goal Reasoning agents. For example, Goal Reasoning has been used in the context of learning to compose web services (Burstein et al. 2008). The most common focus is applying learning for goal formulation or goal priorities (Weber et al. 2010, 2012; Powell et al. 2011; Young and Hawes 2012; Maynord et al. 2013; Silva et al. 2013). Other researchers have studied learning in the context of explanation generation (Molineaux and Aha 2014, 2015) or learning a combination of state expectation and goal formulation knowledge (Jaidee et al. 2013).

The Goal Lifecycle bears close resemblance to that of Harland et al. (2014) and earlier work (Thangarajah et al. 2010). They present a goal lifecycle for BDI agents, provide operational semantics for their lifecycle, and demonstrate the lifecycle for an agent that controls a simulated Mars rover. In future work we plan to characterize how this lifecycle relates to the one we presented in (Roberts et al. 2014). Winikoff et al. (2010) have linked Linear Temporal Logic to the expression of goals.

5. Closing Remarks

We have applied our Goal Reasoning Model to a floor-cleaning robot and described how learning could be incorporated into the model. Although this model is presented as a thought experiment, we plan to implement the model to analyze a comparison of the approaches we outlined for a variety of simulated room types. Future work will also include characterizing when and why particular instantiations of Vakleene, the learning robot, outperforms Vacuous, the simple robot. We also plan to formally characterize how close Vakleene's learned policies can approximate KleeneStar's optimal policies.

Acknowledgements

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

References

Burstein, Mark H., Robert Laddaga, David D. McDonald, Michael T. Cox, Brett Benyo, Paul Robertson, Talib S. Hussain, Marshall Brinn, and Drew V. McDermott. “POIROT-Integrated Learning of Web Service Procedures.” In *AAAI*, 1274–79, 2008.

Chien S., Knight R., Stechert A., Sherwood R., and Rabideau, G. (2000) Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling. Proc. of the Conf. on Auto. Plan. and Sched.(pp. 300-307). Menlo Park, CA: AAAI.

Cox, M. T., & Veloso, M. (1998). Goal Transformations in Continuous Planning. In M. desJardins (Ed.), Proc. of the Fall Symposium on Distributed Continual Planning (pp. 23-30).Menlo Park, CA: AAAI Press.

Harland, J., Morley, D., Thangarajah, J., & Yorke-Smith, N. (2014). An operational semantics for the goal life-cycle in BDI agents. Auton. Agents and Multi-Agent Systems, 28(4), 682–719.

iRobot. 2008. Roomba Manual.

Jaidee, U., Munoz-Avila, H., & Aha, D.W. (2011). Integrated learning for goal-driven autonomy. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*. Barcelona, Spain.

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.

Kaelbling, L.P., Littman, M.L., & Moore, A.P. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 237-285.

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

Maynord, Michael, Michael T. Cox, Matt Paisner, and Don Perlis. “Data-Driven Goal Generation for Integrated Cognitive Systems.” In *2013 AAAI Fall Symposium Series*, 2013.

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

Molineaux, M., & Aha, D.W. (2015). Continuous explanation generation in a multi-agent domain. To appear in *Proceedings of the Third Conference on Advances in Cognitive Systems*. Atlanta, GA: Cognitive Systems Foundation.

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.

Roberts, M., Apker, T., Johnson, B., Auslander, B., Wellman, B. & Aha, D.W. (2015a). Coordinating Robots for Disaster Relief. Proc. of the Conf. of the Florida AI Research Society (to appear) Hollywood, FL: AAAI.

Roberts, M., Vattam, S., Alford, R., Auslander, B., Karneeb, J., Molineaux, M., Apker, T., Wilson, M., McMahon, J., & Aha, D.W. (2014). Iterative goal refinement for robotics. In Working Notes of the Planning and Robotics Workshop at ICAPS. Portsmouth, NH: AAAI.

Roberts, M., Vattam, S., Alford, R., Auslander, B., Apker, T., Johnson, & Aha, D.W. (2015b). Goal Reasoning to Coordinate Robotic Teams for Disaster Relief. In Working Notes of the Planning and Robotics Workshop at ICAPS. Jerusalem, Israel. AAAI.

Silva, Michael, Silas McCroskey, Jonathan Rubin, Michael Youngblood, & Ashwin Ram. “Learning from Demonstration to Be a Good Team Member in a Role Playing Game.” In *FLAIRS Conference*, 2013.

Thangarajah, J., Harland, J., Morley, D., & Yorke-Smith, N. (2011). Operational behaviour for executing, suspending, and aborting goals in BDI agent systems. In Declarative Agent Lang. and Technologies VIII (pp. 1–21). Toronto, Canada: Springer.

Vattam, S., Klenk, M., Molineaux, M., & Aha, D. W. (2013). Breadth of approaches to goal reasoning: A research survey. In D.W. Aha, M.T. Cox, & H. Muñoz-Avila (Eds.) Goal Reasoning: Papers from the ACS Workshop (Tech. Report CS-TR-5029). College Park, MD: Univ. of Maryland, Dept. of Comp. Science.

Weber, Ben George, Michael Mateas, & Arnav Jhala. “Applying Goal-Driven Autonomy to StarCraft.” In *AIIDE*, 2010.

Weber, Ben George, Michael Mateas, and Arnav Jhala. “Learning from Demonstration for Goal-Driven Autonomy.” In Proc. *AAAI*, 2012.

Winikoff, M., Dastani, M., & van Riemsdijk, M. B. (2010). A unified interaction-aware goal framework. In Proc. of ECAI (pp. 1033–1034). Lisbon, Portugal: IOS Press.

Yoon, S.W., Fern, A., & Givan, R. (2007). FF-Replan: A baseline for probabilistic planning. Proc. of the 17th Int’l Conf. on Auto. Plan. and Sched. (pp. 352-359). Providence, RI: AAAI Press.

Young, Jay & Nick Hawes. “Evolutionary Learning of Goal Priorities in a Real-Time Strategy Game.” In *AIIDE*, 87–92, 2012.

Meta-Reasoning Over Goals: A Summary of the GAIA Project

Spencer Rugaber SPENCER@CC.GATECH.EDU

Ashok K. Goel GOEL@CC.GATECH.EDU

Design & Intelligence Laboratory, School of Interactive Computing, Georgia Institute of Technology, Atlanta, Georgia 30308, USA

Lee Martie LMARTIE@UCI.EDU

Department of Informatics, University of California at Irvine, California 92697, USA.

Abstract

We present a summary of GAIA, an interactive environment for designing game-playing agents. A game-playing agent in GAIA contains not only knowledge about the game world, but also a model of its own goals, methods and knowledge. If and when the agent fails to achieve a goal, it uses its self-model to reason about its goals, identifies repairs to its design, and retrospectively adapts itself. We illustrate GAIA through experiments in designing agents that play a version of the turn-based strategy game called Freeciv.

1. Introduction

The GAIA project explores the use of teleology in modeling agents that play turn-based strategy games. Its overall hypothesis is that the use of teleology supports the diagnosis and revision of such agents to better adapt them to their game environments. Specific research questions include how to represent agent goals, how to relate the goals to the means by which the goals are accomplished and how to reason over the models in support of diagnosis and adaptation.

GAIA itself is an interactive environment for designing game-playing agents. In GAIA, a designer interactively designs a game-playing agent in a high-level agent modeling language called TMKL2. The designed agent contains not only knowledge about the game world, but also models of its goals, methods and knowledge. Given the model of the agent, GAIA automatically compiles the agent program code and executes the agent in the game world. If and when the agent fails to achieve a goal specified in the model, a meta-reasoning component called REM uses the agent's model to reason about its goals and failures, identifies repairs to its design, and retrospectively adapts the agent. We evaluate GAIA, TMKL2 and REM through experiments in designing agents that play parts of the turn-based strategy game called Freeciv[1]. In this paper, we

[1] http://freeciv.wikia.com/

present a summary of the GAIA project in order to build a connection with the newly formed Cognitive Systems and Goal Reasoning communities.

2.TMKL2

TMKL2 is an agent modeling language that is used to express the teleology of an agent's design and the means by which the teleology is realized. TMKL2 includes vocabulary for specifying agent goals, the mechanisms to accomplish the goals, and the agent's knowledge of its internal design and its external environment. GAIA realizes TMKL2 models using an interpreter. When used to model a Freeciv agent, the interpreter is capable of executing a model in conjunction with Freeciv's server program to play the game. GAIA assumes that the division between the agent's model and external parts is clearly defined, and that this division takes the form of an (Application Programming Interface) API to the external code. GAIA also makes an explicit distinction between adaptation-time modeling of the agent and run-time execution of the adapted agent: Adaptation takes place on a model of the agent and then the model is interpreted to effect agent behavior in the game world.

1.1. Goals

TMKL2 comprises three sublanguages for modeling Goals, Mechanisms and Environment, corresponding to the Tasks, Methods and Knowledge portions of the earlier TMKL language [Murdock & Goel 2008], respectively. The first sublanguage describes the agent's goals. A Goal expresses a reason that the agent does what it does, in terms of its intended externally visible effects on the agent's world. Goals may be parameterized, enabling the agent to target specific elements of its Environment, such as, for example, a specific city. A Goal is expressed via a pair of logical expressions describing the precondition for Goal accomplishment (called its Given condition) and the expected effect of Goal accomplishment on the agent's Environment (its Makes condition). The final element of a Goal specification is a reference to the means by which the Goal is to be accomplished. In this version of TMKL2, each Goal is associated with the single Mechanisms by which it is to be achieved.

1.2. Mechanisms

The Mechanism portion of a TMKL2 model describes how the agent accomplishes its Goals. There are two kinds of Mechanisms, Organizers and Operations, that are each defined in terms of two logical expressions describing their precondition for execution (Requires conditions) and their effect (Provides condition). An Organizer mechanism is defined as a finite state machine comprising States and Transitions. Start, failure and success States are all explicitly indicated. States, in turn, refer to subGoals, enabling hierarchical refinement of an agent's specification. Transitions may be conditional (dependent on a DataCondition) with respect to the agents current perception of the world, as expressed in its Environment. The other kind of mechanism is an Operation. Operations are parameterized invocations of computational resources provided to the software agent via its API to external software, such as the Freeciv server. That is, each Operation models one of the agent's computational capabilities.

1.3. Environment

A TMKL2 program includes a description of the agent's understanding of itself and the world in which its exists. In particular, the agent's `Environment` comprises a set of typed `Instances` and `Triples` (3-tuples) relating the `Instances` to each other. In order to describe `Instances` and `Triples`, TMKL2 provides two modeling constructs, `Concepts` and `Relations`. A `Concept` is a description of a set of similar `Instances`. It is defined in terms of a set of typed `Properties`. Moreover, `Concepts` are organized in a specialization hierarchy promoting compositionality and reuse. There is a built-in `Concept` called `Concept`. When a TMKL2 model is constructed, `Instances` of `Concept` are automatically added to it for each defined `Concept,` enabling reflection by the agent over its own definition. A `Relation` describes a set of `Triples` allowing the modeling of associations among `Concepts`. In particular, an `Instance` of one `Concept` can be related to `Instance` of another via a `Triple`.

1.4. Semantics

A TMKL2 model of an agent connects the `Goals` of the agent to the `Mechanisms` by which the `Goals` are accomplished. The program is declarative in the sense that all behavior is defined in terms of logical expressions (`Given, Makes, Requires, Provides`). Consequentially, one semantic interpretation of a TMKL2 program is that it declaratively describes the behavior that a software agent must exhibit in order for it to accomplish a set of top-level `Goals`. TMKL2 programs are not just descriptive, however: They can be used to actually control the modeled agent. This requires an operational semantics of TMKL2: A TMKL2 model prescribes the detailed behavior of the agent in the world. Operationally, a TMKL2 Model can be interpreted as a hierarchy of finite state machines controlling communication with the external software with which the agent interacts. Superior state machines in the hierarchy effect the accomplishment of superior `Goals`. FSMs corresponding to `Goals` without any `subGoals` are called *leaf* FSMs. All state machines execute synchronously; that is, at any given time, each machine is in a specific `State`. At the next virtual clock tick, all pending `DataConditions` for active leaf machines are evaluated, and the outgoing `Transitions` evaluating to `true` are traversed, resulting in entry into new `States`. Upon entry into a `State`, the corresponding `subGoal` and its `Mechanism` are interpreted. `Mechanism` interpretation ultimately resolves into `Operation` invocations and updates to the `Environment`. After all invocations have been processed, the `Environment` is updated to reflect any changes to the agent’s run-time data structures made by the invocations. Interpretation terminates if the `Organizer` for the top-level `Goal` enters either a success or failure `State`.

1.5. An Example of a TMKL2 Model of a Freeciv Agent

Figure 1 presents part of the model of an agent, called Alice, capable of playing a simplified version of Freeciv. The figure illustrates a visual syntax for TMKL2. The partial model includes Alice's top level `Goals` and `Organizers`. In particular, the top (green) rectangle of the diagram denotes Alice's top-level `Goal` of collecting gold pieces. Contained within this rectangle is another (slate blue), depicting an `Organizer` comprising three States—an initial `State` (black circle), a `subGoal` reference (gray) and a final `State` (yellow). The `subGoal` itself is shown as the rightmost of the two (orange) rectangles on the second level. Its `Organizer`, in turn, refers to two `subGoals`—one that continually mints more gold until enough has been produced and the other determining when to end the game. The bottom two rectangles contain

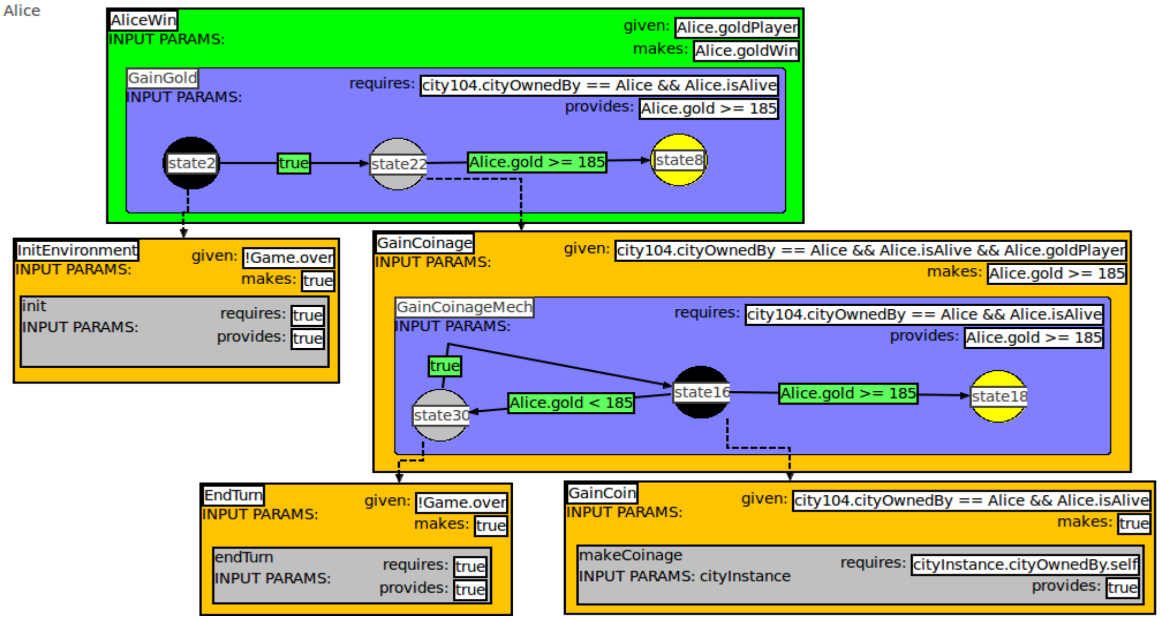


Figure 1: A portion of the TMKL2 model of Alice, an agent that plays a part of Freeciv.

Operations (gray), responsible for interacting with the Freeciv server. Complementing the Goals and Mechanisms shown in the figure is Alice's Environment (not shown). Example Concepts represented by Instances in the Environment include City, Tile, Player, and Unit.

2. GAIA

TMKL2 models can be constructed, modified and executed using the GAIA interactive development environment. GAIA is written in the Java programming language and is built using the Eclipse[2] software development environment.

2.1 The GAIA Architecture

The conceptual architecture for GAIA is presented in Figure 2. In the center left of the figure is SAGi, the GAIA user interface. REM is the reasoning module responsible for adapting models. Also part of GAIA is the model manager responsible for encapsulating access to agent models and persisting them to permanent storage. In-memory representation of TMKL2 models take the form of Java objects that are interpreted by the TMKL2 interpreter, which interacts with the world via the Runtime Communications Manager and associated queues.

SAGi invokes the TMKL2 interpreter to execute a model and thereby interact with the Freeciv server. The interpreter walks the TMKL2 tree of state machines iteratively until the agent either succeeds or fails to achieve its top-level Goals. When the interpreter attempts to accomplish a subGoal whose Mechanism is an Operation, it must place into the Operation Request Queue a request to the Freeciv server to execute a game action, encoding parameters as necessary.

[2] http://www.eclipse.org/

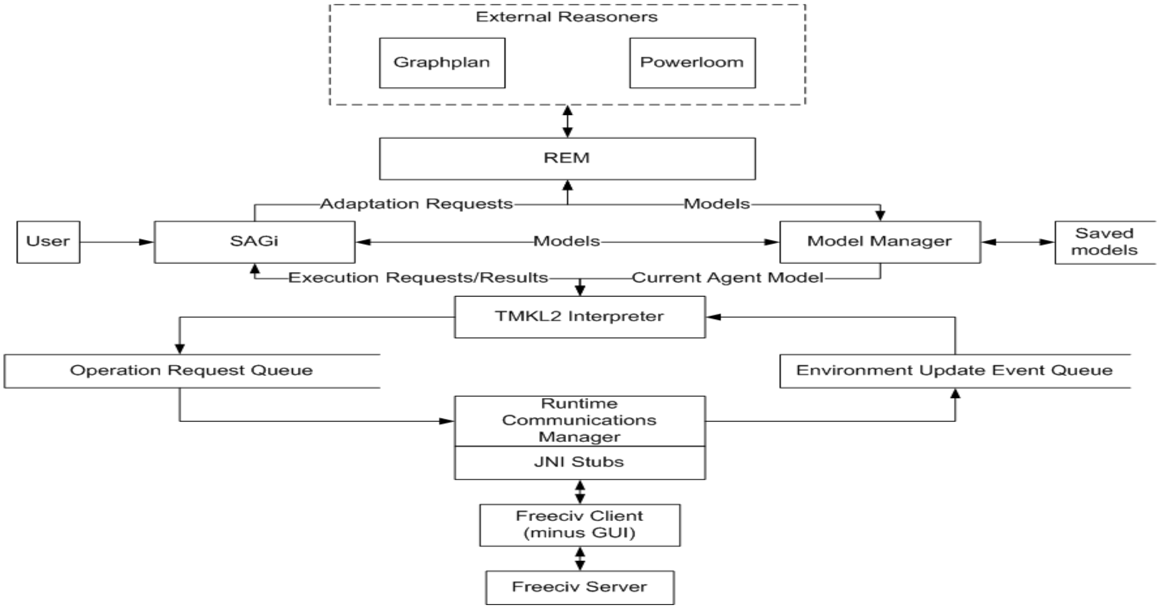


Figure 2: The Conceptual Architecture of GAIA.

REM is the other major component of GAIA. REM, when given an agent model and a situation—either a failed `Goal` or an altered `Environment`—produces an updated agent model engineered either to successfully accomplish the `Goal` or to take advantage of the new knowledge in the `Environment`. To achieve retrospective adaptation, REM performs three steps: localization (determining which of an agent's `subGoals` and associated `Mechanisms` were inadequate to accomplish the agent's overall `Goal`), transformation (devising an alternative `Goal`), and realization (providing/altering a `Mechanism` to accomplish the alternative `Goal`). Localization is accomplished in REM using a heuristic to find a low-level `State` in an `Organizer` such that the state's `Provides` condition suffices to accomplish the failing `Goal`. Further, the detected `State` must have a failing precondition (`Requires` condition). The presumption is that the `State` had not been reached, and, if it had been reached, then the agent would have succeeded.

Realization and transformation are accomplished by matching the failing situation against a library of adaptation plans, choosing a candidate transformation from the library and applying the result to the agent's model to produce a revised model. REM sits atop the Powerloom[3] knowledge representation and reasoning system. Powerloom supports automatic classification (truth maintenance) as well as natural deduction. TMKL2 logical expressions are easily mapped to/from Powerloom, and REM algorithms are easily expressed in Powerloom's variant of first-order logic.

3. Adaptation Scenarios and Results

To validate our approach to model-based adaptation, we have conducted several experiments, each involving variants of the Alice agent depicted in Figure 1. In the experiments, Alice plays a simplified variant of Freeciv against other agents. The simplified game consists of two agents.

[3] http://www.isi.edu/isd/LOOM/PowerLoom/

Each agent controls a civilization and is responsible for its government, economy, citizen morale, and military. Each civilization has one city, citizens in that city, and a number of warriors. Each game tile yields a quantity of food, production, and trade points each turn of the game. Food points feed a city's civilians; production points are used to support existing warriors or generate new warriors; and trade points are distributed among luxury, tax, and science resources. Initially both players start out in year 4000BC, with fifty gold pieces, zero warriors, and one worker that collects resources from a nearby tile. A city is either producing a warrior or collecting gold pieces on any given turn. Alice can win by collecting a specified number of pieces of gold, and the other agent can win by capturing Alice's city. An experiment consists of running Alice against its opponent and noting the results. Then, REM is tasked by the experimenter to adapt Alice, and the game is rerun. In order to best understand the results, FreeCiv was run deterministically throughout the experiments.

3.1. Experiment #1

The purpose of the first experiment we conducted was to test whether REM could make a trivial adaptation to improve Alice's performance versus Freeciv's built-in robot player. In general, a player of this reduced game has to make a decision about allocating resources between collecting gold and creating warriors to defend its city. At the beginning of the first experiment, Alice's strategy was to devote all of her resources to the former pursuit. An obvious adaptation is to adjust Alice to balance her resource allocation, and the first experiment tested whether REM could make this adaptation. In the experiment Alice played against Freeciv's robot player, which we call Frank, configured at its highest skill level. Although Alice had knowledge that Frank could win by capturing her city, she was unaware that Frank had more powerful weaponry and more production capacity than she had. When played against Frank, unadapted Alice directly succumbed to his attacking chariots, legions, and horsemen. Before losing, Alice was able to acquire 175 units of gold and lived for 3075 years. However, Alice failed to acquire sufficient gold to accomplish her `Goal`, thereby requiring retrospective adaptation. In this experiment, no transformation was needed. That is, the failure was that an `Organizer` rather than a `Goal` was flawed. Realizing a replacement `Organizer` took place by interjecting a new `State`, whose success would satisfy the preconditions of a problem `State`. The new `State` was created by first searching a small library of generic `Goal` patterns to see if any satisfy the preconditions of the problem `State`. After an instantiated `Goal` pattern was found it was assigned as the `Goal` of the new `State`. This new `State` was then inserted into the localized `Organizer` just prior to the problem `State`. This guarantees the problem `State's` precondition is satisfied upon its visitation. In Experiment #1, the new `State` was added with a `Goal` to build additional warriors. This `Goal` increases the defense of Alice's city if she is visibly outgunned on the game map. After performing this adaptation, the new agent, Alice', was tested against Frank. While still outgunned, Alice' fared better in longevity and defense. She lasted 3125 years and killed one of Franks powerful attacking units. Because some of her resources had been allocated to defense, she fared worse in gold acquisition, acquiring only 147 units. The lesson learned was that compensating for a well-understood limitation could be accomplished by making use of a simple heuristic alteration of a TMKL2 agent model and a small library of patterns.

3.2. Experiment #2

The previous experiment was an example of *retroactive* adaption in which a failure was mitigated. In Experiment #2, *proactive* adaption was attempted to take advantage of a slightly altered game rule. In particular, it now takes more gold units for Alice to win a game. Tests were run on Alice to see if Alice's model was still valid after the rule change. REM tested if each `Mechanism's Provides` condition satisfies its parent `Goal's Makes` condition; that is, if the `Mechanism` was capable of accomplishing the new `Goal`. If one of these tests failed, REM then located the responsible `Mechanism`. In this experiment, REM localized Alice's GainGold `Organizer`. Next, a replacement `Organizer` was created to achieve the new win condition. To do this, REM used an external planning tool, called Graphplan[4]. REM translated the initial game `Environment` into a Graphplan *facts* file. Then all `Organizers, Operations`, and game rules were translated into a Graphplan *operators* file. After pruning out operators with no effects, the resulting Graphplan file contained 10 operators. Next, REM ran Graphplan on the facts and operators files. Graphplan generated a three-stage plan capable of accomplishing Alice's top-level `Goal`. This plan was then translated back into an `Organizer` to replace GainGold. The lesson learned from this experiment was that for a simple numeric change, a no-longer valid TMKL2 `Organizer` can be located and adapted using an external planner.

3.3. Experiment #3

The first experiment described above was *off-line* in the sense that the adaptations were made after a game was completed. Experiment #3 is an *on-line* adaption in that Alice is changed while she is running. Moreover, her opponent, Barbra is also adapted during the game. In this experiment, both Alice and Barbra were reconfigured into two parts, one *allopoietic* and the other *autopoietic*. These term are borrowed from the literature of self-organizing systems and denote, respectively, the part of a system that changes and the part that does the changing. Alice's allopoietic part used a parameter, alpha, to determine how Alice should allocate her resources between obtaining gold or producing warriors. The autopoietic part of Alice adapted the allopoietic part by adjusting alpha to produce gold only if she had sufficient defensive capability to fend off Barbra's visible attackers. Similarly Barbra's allopoietic part used a parameter, beta, to determine the number of warriors with which to attack Alice's city. The autopoietic part of Barbra adapts the allopoietic part by adjusting the number of warriors Barbra attacks Alice. For both agents, the autopoietic part was itself a (meta-) agent. In particular, the meta-agent's `Environment` consisted of a description of the allopoietic part, including `Goals, Mechanisms` and (allopoietic) `Environment`. By monitoring game status, the meta-agent could make appropriate adjustments to the base agent's parameter by executing (meta) `Operations`. Running Alice versus Barbra resulted in the agents engaging in an arms race. Eventually Alice was able to defeat Barbra. In winning, Alice collected 186 gold units, Barbra had 6 dead warriors, Alice had 3 live warriors and never lost a battle. Barbra adapted herself 4 times, and Alice adapted herself 6 times. The lesson learned was that TMKL2 models allow for simple real-time adaptations by using meta `Operations` to control the agent strategy.

4. Discussion and Future Work

The GAIA development environment provides infrastructure enabling exploration of teleological modeling and reasoning over goals and the means to realize them. The experiments conducted so

[4] http://www.cs.cmu.edu/~avrim/graphplan.html

far have been a limited proof of concept. As such, many questions have been raised and future topics for research suggested.

- **Language extensions:** TMKL2 as described is limited to one `Mechanism` for each `Goal`. Alternative, possibly concurrent, `Mechanisms` and the reasoning necessary to choose among them might be provided. Also, as currently designed, TMKL2 is primarily aimed at expressing achievement `Goals`. The addition of invariants, along with augmenting REM to directly support truth maintenance, would extend GAIA modeling range. Much more ambitious is the ability to deal with non-functional concerns, such as performance.
- **Game design:** In addition to FreeCiv, we have used GAIA with several other games. For example, we have built an agent model to play TicTacToe and a TicTacToe game server. We then watch GAIA adapt the agent to play variants, such as *misère* (play to lose) and allowing players to use either X or O on any move. We have added some abilities to GAIA to support the designer in specifying game server APIs and generating stub `Operations`. Nevertheless, adding a new game still requires significant work on the part of the designer.
- **Adaptations:** The experiments described above illustrate but a few of the many kinds of adaptations imaginable. We have cataloged about a dozen such adaptation types, but the list is ad hoc. What is lacking is a unifying framework for them using which REM could specialize its adaptation capabilities. Also, the framework would enable a more systematic compilation of adaptation patterns.
- **Reasoning:** As it exists, GAIA makes use of PowerLoom, which is a truth maintenance reasoner. As described above, we have also hooked GraphPlan into GAIA, albeit on an ad hoc basis. The question then remains as to what reasoning capabilities are required by teleological and specifically adaptive questions. Among the possibilities are machine learning for better localizing failures from execution logs. Needed also are specialized capabilities for determining what existing `Mechanism` might best be applied (or adapted) to realize a modified `Goal`, and how to formulate a high-level `Mechanism` that combines existing low-level `Mechanisms`.
- **Meta-Reasoning:** Experiment #3, described above, was a very simple example of meta-reasoning, amounting to just parameter tweaking. Nevertheless, the generality of TMKL2's modeling capabilities are such that much more general schemes are possible. In particular, imagine a (meta-)agent formulated to deal with specific adaptation opportunities, including detection, localization, etc. That is, the process by which the experimenter designed Experiment #3's meta-agent could itself be coded in TMKL2 to deal with this particular class of adaptations.
- **Evaluation:** The experiments described above do not constitute a thorough evaluation of GAIA. Such a study would explore a variety of further questions such as: How robust is TMKL2 in dealing with different games? What is the relationship between class of adaption and required reasoning power? Important also are issues such as reasoning performance and usability of GAIA as a design environment. Ultimately, the key question will be the extent to which our approach to teleology, as manifest in the tight connection between `Goals` and `Mechanisms`, can address the general problem of adaptation.

5. Related Research

In this summary paper, we will only briefly cover closely related research; our technical papers cover related research in more detail. Our work perhaps is most directly related to research on meta-reasoning (Cox & Raja 2011). Much of the research on meta-reasoning for self-adaptation has used self-models of agents that help localize modifications to the agent design, e.g., (Anderson et al., 2006; Fox & Leake 2001; Jones & Goel 2012; Murdock & Goel 2008). We can trace several themes in model-based self-adaptation in intelligent agents. For example, self-adaptations can be retrospective, (Anderson et al. 2006; Fox & Leake 2001; Jones & Goel 2012), i.e., after the agent has executed an action in the world and received some feedback on the result, or proactive (Murdock & Goel 2008), i.e., when the agent is given a new goal similar and related to but different from its original goal. As another example, self-adaptations may pertain to domain knowledge (e.g., Fox & Leake 2001; Jones & Goel 2012) or reasoning processes (Anderson et al. 2006; Murdock & Goel 2008). The GAIA architecture supports both kinds of adaptations.

Our earlier research on model-based reflection and self-adaptation (Murdock 2008) suggested that (1) goal-based models of intelligent agents that captures the teleology of the agent design can help localize the changes to the agent design needed for classes of adaptations, and (2) hierarchical organization of the goal-based models of the agent designs helped make the above localization efficient. The work reported here extends earlier work on self-adaptation in two ways. Firstly, the agent specification language TMKL2 has a better defined syntax and semantics than its predecessor TMKL (Murdock & Goel 2008). This adds clarity, precision and rigor. While many agent specification languages specify the goals, mechanisms, structure and domain knowledge of agents, TMKL2 explicitly organizes the agent's mechanisms and domain knowledge around its goals. Together, the goals and the mechanisms that achieve them specify the teleology of the agent's design. Goel & Rugaber (2014) describe GAIA in more detail.

6. Conclusions

While much of earlier work on model-based reflection and self-adaptation pertained to agents that operated in small, fully-observable, deterministic, and largely static worlds, GAIA operates in large, complex, dynamic, partially observable and non-deterministic worlds such as Freeciv. The experiments in self-adaptation described here cover a small range of retrospective and proactive agent adaptations. They demonstrate that (i) it is possible in principle to design game-playing agents so that their teleology can be captured, specified and inspected, (ii) the specification of the teleology of the agent's design enables localization of modifications needed for the three experiments in self-adaptation, and (iii) this self-adaption in turn enables the agent to play an interactive game, monitor its behavior, adapt itself, play the game again, and so on.

Acknowledgements

We thank Andrew Crowe, Joshua Jones, and Chris Parnin for their contributions to this work. We also thank the US National Science Foundation for its support for this research through a Science of Design Grant (#0613744) entitled *Teleological Reasoning in Adaptive Software Design*.

References

Anderson, M., Oates, T., Chong, W., & Perlis, D. (2006) The metacognitive Loop I: Enhancing Reinforcement Learning with Metacognitive Monitoring and Control for Improved Perturbation Tolerance. *Journal of Experimental and Theoretical Artificial Intelligence*, 18(3), 387–411, 20.

Cox, M., & Raja, A. (2011) *Meta-Reasoning: Thinking About Thinking.* MIT Press, 2011.

Fox, S., Leake, D. (2001) Introspective Reasoning For Index Refinement In Case-Based Reasoning. *Journal of Experimental and Theoretical Artificial Intelligence,* 13:63-88, 2001.

Goel, A., & Rugaber, S. (2014) Interactive Meta-Reasoning: Towards a CAD-Like Environment for Designing Game-Playing Agents. In *Computational Creativity Research: Towards Creative Machines* T. Besold, K-U. Kuehnberger, M. Schorlemmer & A. Smaill (editors), Chapter 17, *347-370*, Atlantis ress.

Jones, J., & Goel, A. (2012) Perceptually Grounded Self-Diagnosis And Self-Repair Of Domain Knowledge. *Knowledge-Based Systems*, 27:281-301, 2012.

Murdock, J., & Goel, A. (2008) Meta-case-based reasoning: Self-improvement through self-understanding. *Journal of Experimental and Theoretical Artificial Intelligence,* 20(1), 1-36.

Character-Oriented Narrative Goal Reasoning in Autonomous Actors

Alexei V. Samsonovich ASAMSONO@GMU.EDU
Krasnow Institute, George Mason University, University Drive, MS 2A1, Fairfax, VA 22030 USA

David W. Aha DAVID.AHA@NRL.NAVY.MIL
Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory (Code 5514),
Washington, DC 20375 USA

Abstract

The cognitive autonomy capability understood in a certain sense is vital for actors working as teams in unpredictable environments with limited communications. Such actors need to be able to reason about their goals and decide which to pursue. Therefore, methods of autonomous goal reasoning (GR) would be useful in such cases. However, most previously studied models of GR are limited in their abilities to reason in situations involving multiple characters, viewing the latter from a third-person perspective. In this context, the connection between GR and narrative reasoning is beginning to be explored. Recently, a theoretical framework for narrative team planning (NTP) has been developed and used to generate character goals (not team goals). This approach, however, relies on a global team planning, which may not be suitable for a team operating with limited communications and without a central command. Accepting NTP as a baseline, here we argue that an appropriate introduction of characters and narratives into a GR framework can (in some environments) be beneficial, both conceptually and in terms of performance measures. We study this claim analytically using a Character Reasoner model that we formulate, and apply it to several example scenarios. This model separates characters from actors and employs a hierarchical network for narrative and goal selection. Expected benefits for teams of autonomous actors include more efficient and more robust goal reasoning abilities.

Keywords: Goal reasoning; autonomy; narrative intelligence; character arc; metacognition; distributed multi-agent reasoning

1. Introduction

Goal reasoning (GR) refers to the ability of a cognitive system to deliberate on, generate, and select its own goals in unforeseen situations (Vattam et al., 2013; Klenk et al., 2013; Roberts et al., 2014). Unlike most research on intelligent agents, GR is inspired by observations of (highly autonomous) human cognitive behavior. The high level of cognitive autonomy provided by GR can be vital for actors working in teams and/or in unpredictable environments. As an illustration, we consider a simple scenario (please see Sections 3 and 5.1 for further details of this example).

Suppose a team of two combat pilots are engaging an enemy in a 2-versus-2 beyond-visual-range air combat scenario. At the same moment, they receive information from their commanding officer about a developing situation at a distant location. They infer that the immediate help of at least one air vehicle is vital to that situation, is more important than their current mission, and requires a vehicle that is fully armed. The team needs to reassess, and possibly formulate, goal(s) to pursue. On the one hand, they cannot simply abandon the ongoing engagement: disengaging may result in the destruction of friendly assets. Therefore, one option is to pursue the previous goal until it is achieved, then switch to the new goal. However, in this case none of the team members will be guaranteed to have a full load of offensive assets, so their help at another location could be useless. An alternative choice is to arrange for one of the vehicles to escape before firing any of its missiles. For this to happen, the pilot of the remaining aircraft must keep both enemies engaged, allowing the partner to escape (while still fully loaded). Once the two goals are formulated, the associated risks and values can be estimated, and the selection among them can be made. This task requires more than traditional planning: for example, the team needs to select a top-level goal at the start of their engagement. Moreover, the need for team-level GR may repeatedly be required in a dynamically changing scenario, as we exemplify below. Existing GR agents aren’t designed for these kinds of open-ended team decision-making scenarios.

While GR has only recently been an active research topic, many branches of AI research relate to it. Examples include topics like intelligent agents and cognitive architectures (Gray, 2007), motivated agents, narrative planning, intent and plan recognition, self-regulated learning (Zimmerman, 2002), and many more. Because it is not our present goal to review all these areas, we contrast our approach to only one related topic, namely *narrative planning* (Riedl et al., 2008; O’Neil & Riedl, 2014; Riedl & Young, 2010, 2014; Young et al., 2013) applied in the context of a team mission. We refer to it as *narrative team planning* (NTP) and take it as a baseline.

The essence of NTP can be outlined as follows. First, the narrative that determines future team actions is generated by a planner, given one desired outcome as the goal (some systems allow for goal replacement in the case of a plan generation failure; see (Riedl et al., 2008)). Second, the actions of each character must be believable (i.e., consistent with individual character goals and motives, which are not necessarily consistent with each other or with the team’s goal). This constraint allows the actors to execute plans locally within the team. Here “locally” means focusing on their individual tasks and tasks of selected peers rather than on the entire team mission. Third, plans are generated from a “global” team perspective. The global narrative is created by eliminating unmotivated commitments (e.g., by assigning certain individual goals to characters, which transforms NTP into a GR process). Finally, the set of NTP characters is known a priori and is fixed, because the term “character” is synonymous to the term “actor” in NTP.

The approach that we introduce here as an alternative to NTP can be characterized as “character-oriented narrative goal reasoning”, or, for short, “character reasoning” (CR). A character in this case is an abstraction, and is not identical to an actor. We define a *Character Reasoner* as an autonomous, embodied intelligent agent (an actor) capable of (a) formulating believable characters applicable to the developing situation, that help with achieving team’s goals, and (b) performing roles of selected formulated characters, using narrative GR and planning in cooperation with the team. CR relies on the concepts of a narrative, a character, a character arc (i.e., a plot or a storyline describing the evolution of a character and its goals in a story), and related concepts, which are useful in many (e.g., military) domains (Finlayson & Corman, 2013) and are also parts of the NTP formalism (Riedl & Young, 2010; Ware & Young, 2014). So, what is essentially new in CR, besides separation of characters from actors?

To answer this question, we shall return to our example, assuming now that there are $N \geq 2$ vehicles in each team, M vehicles need to escape without firing any of their missiles, and $1 \leq M < N$. As explained above, for the air combat team to select a new goal using NTP, the future actions of all team members (actors) need to be considered and optimized as a whole. This could make the task computationally demanding and practically intractable in the case of a large team size. Moreover, the necessary information about all partners may not be available locally to team members engaged in NTP. Therefore, an important question is whether and how the global team-GR task can be effectively decomposed into individual, local GR tasks. This is exactly what the present work intends to address (Section 3 explains intuitively how, using the selected example).

The paper is organized as follows. In Section 3 we show how to perform the decompositions discussed above, but only after we provide a minimal background on the topic in Section 2. We also explain in Section 3 why CR can be more efficient than NTP, using the selected example at an intuitive level. We then formalize CR in Section 4, casting it as a form of narrative GR, and consider its likely benefits based on a more detailed analysis of other examples in Section 5. We finally discuss implications and summarize the main points in Sections 6 and 7.

2. Background and related works

In narratology, a *narrative* is defined to have two related components (Bal, 1998; Riedl & Young, 2010; Schmid, 2010): the *fabula*, which is a partially ordered set of causally, logically or temporally related events that form a consistent story, and the *sjuzet*, which is a (possibly incomplete) linear presentation of this story as viewed from a storyteller perspective[1]. Narrative techniques and their application have received recent attention in AI research, yet not sufficiently for team-level GR and planning and military applications (see, however, Young et al., 2013; Finlayson & Corman, 2013). At the same time, there is no general consensus on a formal definition of narratives, and how they should be generated (Kapadia et al., 2015). Within the popular formalism developed by Young's research group (Riedl et al., 2008; Riedl & Young, 2010, 2014; Young et al., 2013, Ware & Young, 2014), a fabula is defined as a sound plan with the additional requirement of character believability, enforced through the consistency of character goals, intentions and actions. Accordingly, within this approach, narratives are generated by planning algorithms. However, this use of planning imposes severe limitations on the outcome, many of which are identified by the authors themselves (Riedl & Young, 2010).

Outside this relatively narrow understanding of a narrative, no precise criteria are defined to distinguish narrative and non-narrative planning or GR. For example, John McCarthy defined a narrative more loosely than a plan: as a temporal partially-ordered collection of related situations and events, without requiring their consistency (McCarthy, 1994; McCarthy & Costello, 1998). In the situation calculus, narratives are first-order objects. From this point of view, today virtually any symbolic cognitive architecture can be regarded as a narrative-based modeling framework. McCarthy (1994; 1998) also discussed the concept of a *proper narrative,* which is a narrative without anomalies. Among other formal approaches in narratology, Abell (1987; 1993) represents a narrative as a graph, the nodes of which represent actions performed by actors that change states

[1] This dichotomy originates from the Russian mechanistic formalism of literary criticism developed early in the 20th century. Other terms are also used (e.g., "story", "discourse", or "plot").

of the world. Edges of the graph are directed and represent causal or dependency relations, indicating, for example, that one action is a prerequisite for another. Some modern authors go further and require that a narrative should be understood more narrowly than a plan constrained by the believability of characters. Additional requirements include "narrativity", character interaction, a conflict or suspense, struggle, characters changing their goals and values, and more (Huhn, 2013; Simon-Shoshan 2013; Haven, 2007). Finlayson and Corman (2013) refer to this kind of a narrative as a Level-II narrative.

In this paper, we distinguish between CR and non-CR approaches (we define CR formally in Section 3.2) using the notion of a character, which we contrast with the notion of an actor. For Haven (2007), characters are abstractions that are central in understanding a narrative, or story. Haven defines a story as a detailed, character-based narration of a character's struggle to overcome obstacles and reach an important goal. A character, according to Haven (2014), is an abstraction defined by five elements that can be interpreted (in our words) as follows:

1. drives, values and motives (the *core*);
2. personality type and features;
3. autobiographical memory and general knowledge;
4. behavioral activity and capabilities; and
5. subjective viewpoint, appearance, and self-image.

A character in a narrative is a perspective and a viewpoint that allows us "to see who is doing the action and to gauge relevancy…", "to interpret emotional state, beliefs, attitudes… to create meaning and relevance". A character's behavior is necessarily intentional. Character *intent* is composed of two components: the *goal* (the outcome that is being pursued by the character) and the *motive* (i.e., the reason why this goal is important for the character) (Haven, 2007).

Thus understood, characters are distinguished from actors (intelligent agents that are given entities), as well as from cognitive systems and from objects in the environment. A *character* is understood in this Section and below as an abstraction in the form of a virtual subject (an ego, a self, a persona), defined by its subjective perspective, together with a system of values, motives (given by top-level guidance and/or bottom-level drives), beliefs (autobiographical and general), and capabilities. A character is associated with a particular *arc* in a narrative, and can be performed by an actor.

Although both the character and the actor could be virtual agents, the notion of an actor is not redundant in this context. An actor is a fixed entity for a given specific scenario, while characters can be dynamically created, modified and deleted, as they exist in the "minds" of actors. An actor may have a suit of characters and make selections among them depending on the situation. In this context, "character" is a synonym of "role". For actors, characters replace goals and commitments, while being richer and more powerful constructs. Playing a certain character means more than pursuing a certain goal. Typically it involves an evolution of the character's goal(s). This notion of a character is further illustrated using a set of example scenarios in Section 5.

The primary distinction to be made between the present work and related works is that this work proposes to use characters as specifically defined narrative structures (distinct from goals and actors) to assist GR in complex scenarios (e.g., involving multiple actors and characters).

3. Local decomposition of team-GR using CR: An intuitive preamble

As the name suggests, character reasoning involves the concepts of a character and a character arc. As mentioned in Section 1, we distinguish characters from actors, which is not the case in NTP. A *character* in CR is an abstraction: it is a virtual rational agent with its own goals, motives, senses, affordances, knowledge, and recent history. In general, there is no 1-1 correspondence between actors and characters in CR. One actor can play multiple characters in a sequence, and in certain cases in parallel. The purpose of introducing characters in CR is to decompose the global team-level GR into local, single-character GR. This decomposition involves two steps: (1) separation of the global GR task into two stages (in the first of which characters and prototype character arcs are selected that will form the narrative), and (2) decomposition of the second stage (which maps characters to actors and further specifies character arc details) into individual-character GR tasks (Figure 1). Therefore, in this character-oriented version of GR, the notion of a character is central and includes the notion of a goal.

To illustrate how CR works, we extend our example air combat scenario. Two types of characters can be identified in our example: one, a "deserter", who escapes the fight without deploying any assets, and another, a "hero", who keeps the enemy engaged. These characters are believable (they act according to their goals and motives), yet their goals, motives, and strategies differ. Once defined as a part of the narrative, these two character types essentially determine the team's strategy (we assume that this decision is made independently by both pilots at the beginning). Yet, the character mapping to actors may not be decided immediately, is ambiguous, and may change over time. If all combat vehicles are identical and the pilots have similar expertise, then the choice should be determined by their relative positions. In the absence of a central command, team-level GR is performed independently by each actor, who must decide which character to play. The choice of each actor should be consistent with the team, and may need to be corrected based on observations of the partner's behavior or communications. For example, if you are an actor in the team, then upon noticing that your partner's behavior is inconsistent with your character choice, you can adjust your choice or communicate with your partner (i.e., to convince them to change their character choice). Once the mapping of characters to actors is decided by the team, each actor assumes the goal, motives, etc. of the selected character, and starts planning and acting based on this choice, possibly while keeping only marginal awareness of the rest of the team. In other words, the task becomes locally decomposed.

Both stages of CR in our example are executed locally. Stage-1 CR is performed by each actor independently in parallel, producing the same result (deciding that the two character types will determine the narrative: suppose that actors produce the same set of character types). Stage 2 involves local communications (Figure 1). This stage may recur during actor-level plan execution. Indeed, imagine that at a certain point swapping the characters becomes advantageous for the team (e.g., if both enemies follow the deserter). Character swapping can be performed using a global team-level GR. However, it can also be done using local communications, if we add the "swapping affordance" as a special privileged action to the deserter's repertoire (instantly switch positions with the partner). When feasible, this action should also transfer the character's current plan and recent memory from one actor to another. This will allow each character to preserve continuity, while reasoning locally (we consider the jump as a local action). In either case, the task remains decomposed at all times.

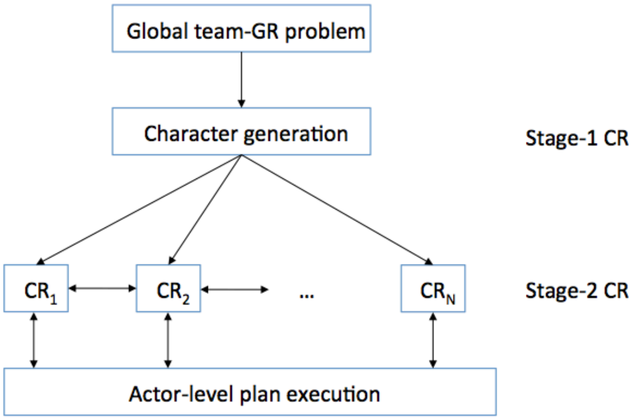


Figure 1. A general decomposition scheme of the team-level GR problem using character reasoning for the *N*-aircraft example. Each CR$_i$ is an individual-character GR process, including the mapping of the character to an actor. Horizontal arrows show local communications, possibly including the swapping of characters.

The practical difference between CR and NTP in this example may be small, given that the number N of aircraft is 2, but it could be noticeable when N is large. For example, suppose there must be M deserters and $N - M$ heroes, where $M \sim \frac{N}{2} \gg 1$. Then the number of choices

$$\binom{N}{M} = \frac{N!}{M!(N-M)!}$$

grows superexponentially with N, and could become too large to explore in real time using NTP. In contrast, if using CR all choices can be made locally in parallel. While there is some tradeoff (communications among team members are required), this does not necessarily assume all-to-all individual communications. For example, suppose that a nearly random mapping of character types to actors is accepted by the team initially, when each actor decides independently which character to play based on his local surrounding, and broadcasts his choice to the team. Then, further optimization of the mapping can be obtained using local character swapping. As a result, the team can be expected to produce nearly optimal behaviors for large N using CR. This illustrates a primary distinction of CR and NTP, and applies to other examples.

4. General conceptual basis and the CR formalism

In this section we define CR and its main building blocks formally. We start from a top structure that we call a *hierarchical narrative network* (HNN), related to the notion of a narrative network (Pentland & Feldman, 2007), which is defined as the tuple:

$$\text{HNN} = \langle S, E, C, \mathcal{A}, P \rangle, \qquad \text{(Eq. 1)}$$

where S is a set of nodes, E is a set of directed edges, C is a set of characters, $\mathcal{A}$ is a set of character arcs, and P is a set of performing actors. Now we will explain these elements intuitively. An HNN includes a graph with a set of nodes S and a set of directed edges E. Here nodes represent actual and possible states (see the definition of a state below in this section) and fragments (i.e., fragments of the graph), and edges represent causal and temporal relations among nodes, inducing a partial order. Possible states, relations and rules of dynamics are given by the *domain theory*: a knowledge base that is assumed given, yet is not explicitly included in (Eq. 1). The network is hierarchical, because some of its nodes represent fragments of the same network. Fragments can be collapsed into nodes, and nodes can be expanded into fragments, as necessary.

In addition to S and E, an HNN (Eq. 1) includes a set of characters $C=\{c_j\}$, a set of possible character arcs $\mathscr{A}=\{A_j\}$, and the set of performing actors $P = \{P_i\}$. Our intuitive notion of a character was introduced in Section 2. Technically, a character c is represented by a tuple

$$c = < p,\ m,\ A >, \qquad \text{(Eq. 2)}$$

in which the character is given by the perspective p, motives m, and the arc A. Now we shall explain these terms. The perspective p represents the current character's viewpoint, including senses of "now" and "here", "self" and "others" (own identity), as well as any specific features and capabilities, and other contextual variables determined by embodiment (i.e., the performing actor P_i). The set of character's motives m includes drives, values, and top-level guidance, that determine the selection of goals and intentions and usually do not change their nature within a character arc. A character arch was defined intuitively in Section 2. Formally, the character arc A is a set of character's attitudes $\{a_i\}$, such as beliefs, goals, intentions, memories, percepts and affordances, taken as functions of time. In general, a character c's attitudes are formed from states by attributing them to the character together with a certain modifier, e.g.:

```
c.does(s0), c.intends(s1), c.ignored(s2),
c.achieved(s3), c.saw(s4), c.committed(s5).
```

A character in an HNN is not committed to a particular arc or actor, and therefore may not have a unique perspective. However, each character c in a given narrative is committed to an arc A and to an actor p (the embodiment of c). In the spirit of Abell's formalism (2009; 2011), we define a *fabula* as any part of an HNN written as the tuple (Eq. 1) that is internally closed (i.e., all intentions and actions are *motivated* and placed into arcs), consistent, and includes exactly one arc per character. Here "motivated" means that character intentions can be explained by, or derived from character motives. Also, a mapping of performing actors to characters needs to be specified in the fabula. Then, we define the *sjuzet* to be the arc corresponding to the storyteller (usually the protagonist or the author; in our case of interest it is the character of the reasoning actor).

We define a state $s \in S$ to be a class of possible physical states of the world, such that a particular given fact applies to all those and only those physical states of the world. For example, a state can corresponds to a specified place and/or a moment or an interval in time, and/or can represent a particular object that is present there, or an event, a condition, a feature or property, etc., and any collection of them. States are therefore not necessarily mutually exclusive: for example, there may be more than one actual current state represented in the actor's working memory. Fractions and unions of states are themselves states. Therefore, states can be added and subtracted as sets. Controlled actions and processes are also states. Due to the hierarchical nature of HNNs, HNN fragments are also states, when represented by nodes. When a state is included in a narrative, it allows characters to form attitudes based on this state.

Finally, we define the term *Character Reasoner* as a system that implements the formalism described in Equations 1 and 2, and operates on an HNN, producing characters, character arcs, and a narrative. The architecture of a Character Reasoner is shown in Figure 2. In our case, the Character Reasoner is the author, the storyteller and the actor at the same time (e.g., the sjuzet is presented from the perspective of its character). The general CR procedure is outlined below (here we assume that the set of performing actors *P* and the domain theory *D* are given).

The CR procedure:

(1) Start with populating/updating HNN states and relations, using the available input and referring to the domain theory.
(2) Formulate relevant possible characters. Select characters that will determine a narrative.
(3) Generate possible expected character arcs for selected characters.
(4) Combine the arcs into a set of possible narratives (fabula).
(5) Evaluate and compare generated narratives, select the working narrative.
(6) Map characters in the working narrative to actors.
(7) Generate a sjuzet with the selected character(s) as the storyteller(s).
(8) Use the sjuzet to plan and execute the task at the actor level.

In a team, the outcomes of (2), (5), and (6) should be confirmed by partners (either explicitly, via communications, or implicitly, by observable behavior). Conflicts need to be resolved before plan execution.

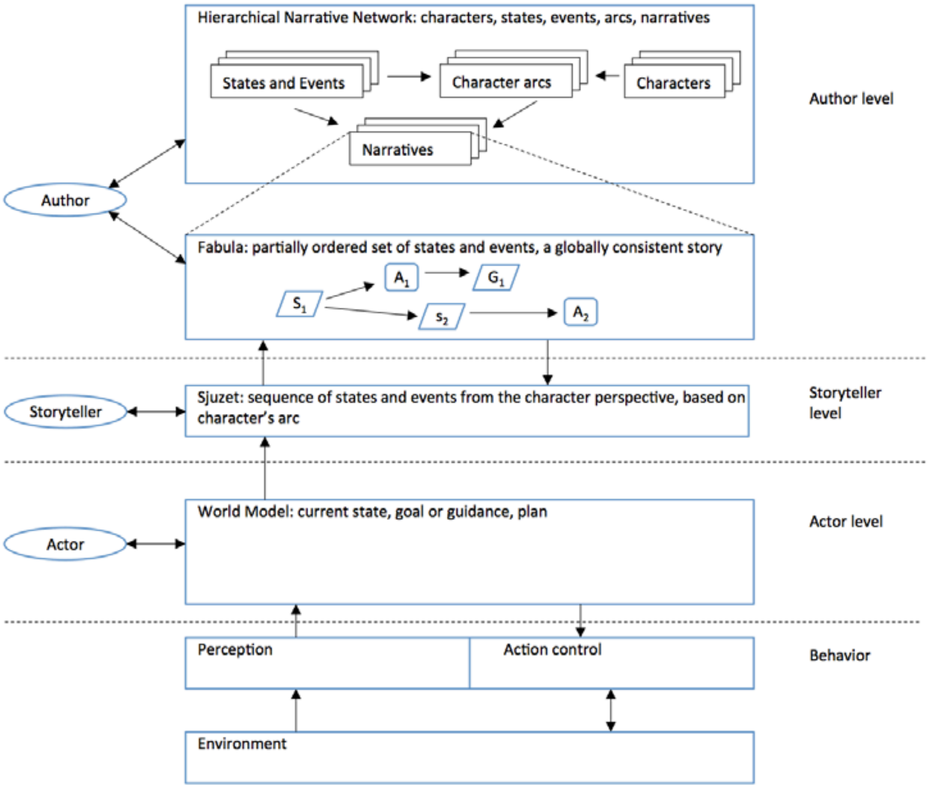

Hierarchical Narrative Network: characters, states, events, arcs, narratives
States and Events
Character arcs
Characters
Narratives
Author level
Author
Fabula: partially ordered set of states and events, a globally consistent story
A_1
G_1
S_1
S_2
A_2
Storyteller
Sjuzet: sequence of states and events from the character perspective, based on character's arc
Storyteller level
Actor
World Model: current state, goal or guidance, plan
Actor level
Perception
Action control
Behavior
Environment

Figure 2. General architecture of a Character Reasoner that is also an actor.

5. Analysis of examples

To assess the scope and benefits of our formalism, we consider the following examples at a more detailed level, each formulated in a virtual environment, allowing for their future computer simulations that will be presented elsewhere. We start with the example already used above.

5.1 Air combat (one possible scenario)

Initial situation: the Blue team consisting of two fighter jets, Snake1 and Snake2, are patrolling a political border. The Red team (bandits), consisting of two enemy aircrafts, has just crossed the border. Each aircraft carries two long-range missiles (we assume that no other weapons are used). The bandits fly close to each other, while Snake1 and Snake2 fly at a distance from each other. Commanding officer informed Snake1 and Snake2 that two bandits crossed the border and need to be eliminated. The initial distance between the two teams is beyond the radar / missile range.

Original working narrative:
T1. Bandits search for Blue aircraft using their radars.
T2. Snake1 and Snake2 search for bandits using their radars.
T3. Snake2 detects Red radars and communicates to Snake1.
T4. Snake1 detects Red radars and communicates to Snake2.
T5. Snake1 and Snake2 triangulate Red coordinates.
T6. Bandits detect Blue radars.
T7. Snake2 snoozes the radar.
T8. Snake1 activates a radar jammer.
T9. Bandits activate jammers.
T10. Bandits turn to pursue Snake1 but cannot achieve a missile shot.
T11. Snake2 fires two long-range missiles at bandits.
T12. One of the two bandits is destroyed.
T13. Snake2 illuminates the remaining bandit with a radar, as a distraction.
T14. The bandit pursues Snake2.
T15. Snake1 takes a shot at the remaining bandit.
T16. The bandit is destroyed.

Working narrative altered at T10, no character-actor separation:
T10.1. The blue team receives request for help at a distant location.
T11.1. Snake 2 is set to be the defector, and Snake1 is set to be the hero.
T12.1. Bandits turn to pursue Snake2.
T13.1. Snake1 fires two missiles at bandits.
T14.1. Bandits fire one missile each at Snake2.
T15.1. One of the bandits is destroyed.
T16.1. Snake2 is destroyed.
T17.1. Bandits fire one missile each at Snake1.
T18.1. Snake1 is destroyed.

Working narrative altered at T10 with character-actor separation (character swapping allowed):
T10.2. Snake 2 is set to be a defector, Snake1 is set to be a hero.
T11.2=T12.1. Bandits turn to pursue Snake2.
T12.2. Snake1 and Snake2 swap their characters. Snake1 proceeds to escape.
T13.2=T11. Snake2 fires two long-range missiles at bandits.
T14.2=T12. One of the two bandits is destroyed.
T15.2=T13. Snake2 illuminates the remaining bandit using the radar.
T16.2=T14. The bandit fires two missiles at Snake2.
T17.2. Snake 2 is destroyed, Snake1 is beyond the radar detection range.

5.2 Fruit collection

Two humanoid robots need to collect fruit (apples and oranges) scattered in equal numbers in a square room, placing them separately in two baskets (sorting fruits after collection is not allowed). Each robot can collect fruit with both hands, carry one basket and collect fruit with one hand, or carry two baskets. One of the robots is weak and unable to carry a heavy basket. A challenging situation occurs when this condition is discovered in the middle of task execution. The expected solution involves two characters: a basket carrier and a fruit collector.

Following the CR procedure, the actors will formulate several potentially useful characters: a basket carrier, a fruit collector (with a basket), and a fruit collector without a basket who will use both hands to collect fruit. The HNN will include the following states.

```
       State S1: The room contains two robots, two baskets, and
                 scattered apples and oranges.
       State G1: All apples are in one basket.
       State G2: All oranges are in one basket.
       Goal = G1 and G2.
```

Then, among the generated narratives will be the following:

```
Narrative 1: S1 → (Arc1, Arc2) → Goal
       Character1: Apple collector. Performer: Robot1.
       Arc1: Goal=G1. Pick a basket, collect all apples. Reach G1.
       Character2: Orange collector. Performer: Robot2.
       Arc2: Goal=G2. Pick a basket, collect all oranges. Reach G2.
Narrative 2: State1 → (Arc3, Arc4) → Goal
       Character3: Basket carrier. Goal=G1 & G2. Performer: Robot1.
       Arc3: Pick baskets, follow the partner, carry baskets and
             ensure that each type of fruit is separated. Reach Goal.
       Character4: Fruit collector without a basket. Goal=G1 and G2.
       Performer: Robot2.
       Arc4: Collect apples and oranges, placing them separately in
             baskets carried by the partner. Reach Goal.
```

Narrative 1 is potentially more efficient than Narrative 2, and the actors can infer this from simulations. Indeed, suppose that each type of fruit is uniformly distributed in one of the two equal halves of the square environment, and the distributions do not overlap. Then Narrative 2 may take approximately twice the time required by Narrative 1. However, Narrative 1 cannot be continued when the weak robot has a heavy basket in its hands. When this condition is not known a priori and is discovered in the middle of task execution, CR in the weak actor will be triggered, resulting in its switching to Narrative 2. When the partner observes the weak robot approaching empty handed, it will put its behavior in the context of the two narratives, and will find the observed behavior consistent with Narrative 2, not with Narrative 1. Therefore, the robot-partner will infer that its partner decided to switch to Narrative 2, and will accept the new choice, even if the reason is not understood (the information about weakness may not be available to the partner). This example also illustrates how the global team GR is decomposed into local CR.

5.3 Bucket brigade

A firefighting team is distributed in a building affected by fire (Figure 3) where they have limited mobility. Because of the fire and building damage, actors are confined in local domains (rooms) and cannot easily cross obstacles separating them, but can pass buckets of water to each other through holes in the obstacles. Their task is to deliver water from the source to the fire sites, returning empty buckets back to the source of water. However, the actors cannot use verbal communications due to noise. Only simple gestures can be used to send two possible messages: (a) “do not send water here”, (b) “send (more) water here”. Messages can be sent to nearest neighbors only. The first message (a) is sent when a full bucket should be returned. It may indicate that all fires in that direction are extinguished, or that access to them is blocked. The second message (b) is initiated whenever a new fire is detected, and is transmitted along the chain toward a water source.

The team uses a fixed, small number of buckets. All buckets look distinct from each other and can be identified by their unique color. Actors have perfect episodic memory. In particular, they remember: which buckets they have passed and received; when, to/from whom, and in what condition; and when, from whom and in what number did the water requests come. Actors do not know the entire floor plan and the locations of fires, but remember how they got to their locations from the entrance (which is also the location of the water source; see Figure 3). Therefore, each actor knows where to send requests for water.

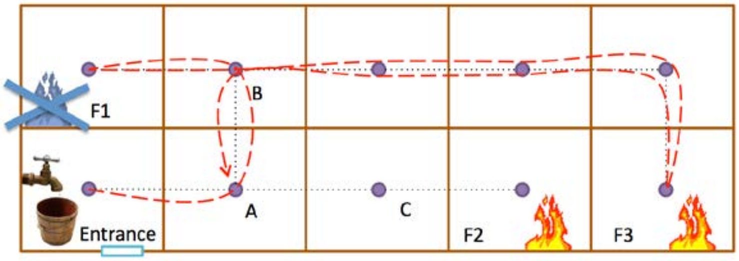


Figure 3. Floor plan of a building under fire with actors {A, B, C} and fires F1-F3. The water source is at the building’s entrance. Dotted lines show possible routes of buckets and messages. The red dashed line shows the long path of the bucket given by A to B, after F1 was extinguished: B passes the bucket toward F1, but the bucket returns full, then it follows through B to F3 (cannot reach F2), back empty to B and to A.

Imagine the following scenario (Figure 3). At the beginning, requests for water arrive from F1, F2, and F3. Given that A received two requests from B and one from C, he will send twice as many buckets to B than to C. When F1 is extinguished, the cancellation message (a) will not reach A, because B will not pass the full bucket back to A: instead, he will pass it toward the remaining active fire F3. However, if A tracks individual buckets, then he will notice that it took an unusually long time for a bucket to return empty. Then A will interpret this event by simulating the bucket as a character, who returned from an extinguished fire and went to the other fire (A knows that there were two fires served by B and one served by C). This will explain the time increase. Therefore, A will start sending less water to B. In contrast, in a simple-minded approach without bucket tracking and without CR, A will not have a reason to change behavior.

5.4 Discreet pursuit

This scenario relates to the times when widespread surveillance was not in use. Suppose that a team of secret agents (actors), dressed like ordinary pedestrians, are monitoring a suspect in a city. Their goal is to watch the suspect continuously from a distance not exceeding the range of visibility (e.g., one block), in order to identify a secret location in the city, toward which the suspect is presumably heading. At the same time, their trajectories should be "believable" (not causing suspicion) from the suspect's perspective. We also assume actors immediately see and recognize each other (or the suspect) within the range of visibility. They use wireless communication to broadcast messages to the team (e.g., a suspect's coordinates). The suspect does not know the secret agents. His goal is to reach a secret location in the city without being followed, and he suspects that he might be followed. Therefore, he will change his direction randomly several times, and will not proceed to his destination if he observes that somebody followed his random turns without an obvious reason. In the initial state, one actor follows the suspect, while others are located nearby but outside of range. What strategy should the team use?

A CR solution can be formulated using three character types: a pedestrian, a shadow, and a double. A pedestrian trajectory should be directed toward some location in the city (this location may remain ambiguous, but should not change inconsistently or improbably) and should not follow the suspect for too long. The shadow (only one) stays within the range from the suspect. A double follows the suspect outside of the range (on a parallel street) based on broadcasts. The actor who follows the suspect plays the shadow and a pedestrian at the same time; other actors play doubles. The current shadow continues following the suspect until the two characters come into conflict with each other (e.g., when the suspect makes a random turn). At this moment, the actor transfers the shadow character to a double, and continues playing a pedestrian. It would be difficult to formulate a solution so concisely without using CR.

5.5 Black box recovery

The following scenario illustrates that CR can be useful not only in a team, but also in isolation. The actor is an unmanned underwater vehicle (UUV) that autonomously performs a task of search of a dead UUV. Upon locating it, the actor will go to the surface and communicate the location of the detected UUV to the operator (this operation takes some time). This task requires exploration of a large area of the seabed. The actor has already searched exhaustively one half of it, area A. The other half, area B, remains unexplored. At this point, the actor receives new information from its human operator: a plane has crashed somewhere in the same area. The black box (BB) will continue to emit signals for a month, and locating it is very important. Recovery of the UUV is equally important, but there is no short deadline associated with it. The two search tasks conflict with each other. A search for BB involves listening to a certain frequency of sound that can be heard at a greater distance compared to the distance of the UUV's visibility. Therefore, an exhaustive BB-search can be performed faster, although it may not guarantee locating the UUV. The area A that is already searched for the UUV may contain BB. Assume that an exhaustive BB search of A and B will guarantee locating BB and will take up to 15+15 days. Similarly, assume that an exhaustive UUV search of B will guarantee locating the UUV and will also take 30 days. In responding to this situation, one approach could be to formulate possible goals and select one of them. In this case, the choice of the first goal would be to search A and B for BB, until it is found and its coordinates are communicated to the operator. However, the order in which A and B are searched remains ambiguous. Also, with this approach, if a UUV is spotted by chance during the BB search, it would be irrelevant to the current goal and will have no effect. An alternative approach for the actor is to play two characters in parallel: one is active (searching for a BB) while the other is dormant (searching for the UUV). The dormant character will bias decision making, attention, and behavior in cases when the active character has no preference. Thus, the actor will start its BB search from Area B, and will remember the location of a detected UUV to communicate it later (its immediate surfacing will disrupt the BB search).

6. Discussion

We presented a model of a Character Reasoner, which is combined with an actor serving as a team member. This means that one and the same cognitive system implemented in a robot needs to perform reasoning at multiple levels (Figure 2): (i) the author level, reasoning for the team and producing a whole-team fabula; (ii) the storyteller level, translating the fabula into a sjuzet: the arc of one character that will guide GR and planning from the local perspective; and (iii) the actor level, at which the system performs GR and planning guided by the sjuzet associated with one selected character. We illustrated this scheme in use and its benefits in five scenarios (none of them were implemented computationally: this is left for a future publication). Their comparison and specific points are summarized in Table 1, that also presents a comparison of CR with NTP. We see, in particular, that local decomposition of team-GR based on CR is possible in all cases.

Table 1. Comparison of the five scenarios. *Asterisks mark features that are unavailable in NTP.

	Sec.5.1	Sec. 5.2	Sec. 5.3	Sec. 5.4	Sec. 5.5
Feature or characteristic	Air combat	Fruit collection	Bucket brigade	Discreet pursuit	Black box recovery
Team-GR decomposed into local actor-GR		✔			
Team-GR decomposed into local CR*	✔	✔	✔	✔	✔
New characters *designed* in a new situation*	✔				✔
Characters associated with inanimate objects			✔		
Team or mission top goal selection	✔				✔
Narrative switching given a new situation	✔	✔	✔		✔
Character swapping during task execution*	✔	possible	✔	✔	
Character's top goal changes within an arc		✔	✔	✔	✔
Actor performs two characters in parallel*				✔	✔
Dormant character usage*					✔

We used the analyzed abstract examples to illustrate benefits of CR, point by point. Thus, in Section 5.2 (fruit collectors), the case of switching from Narrative 1 to Narrative 2 addresses the core question: is this model adaptive to changing environments or changing conditions? Our analysis of the Character Reasoner model gives an answer, explaining how and when the switching will occur, and why a global team-level GR may not be required. The analysis in Section 5.3 briefly conveyed that applying CR to objects (in this case, buckets) may benefit team performance. Section 5.4 further illustrates the capabilities of CR as opposed to NTP; we explain how one actor can play two characters simultaneously and benefit from this strategy. Section 5.5 extends this last point to a scenario involving only one actor in isolation, showing that the Character Reasoner model may be useful not only in teams or scenarios involving multi-agent interactions. Overall, CR is conceptually distinct from non-CR or non-narrative reasoning. It would be informative to compare them empirically using appropriate performance metrics; we expect that introducing characters in GR could be beneficial in some scenarios, as suggested by our example scenarios. In this case, the CR method will find applications in various domains. One possible example is the virtual football, the state of the art in which does not involve CR or NTP (Nadarajah & Sundaraj, 2013).

More generally, we would like our artificial actors to generate, understand and manage top-level goals in unpredicted situations. This level of cognitive autonomy is vital in scenarios involving multiple actors operating in dynamic unpredictable environments, especially when communications are limited. Scenarios usually involve goal-directed behavior that needs to be planned. In classical planning, an actor pursues a fixed goal. However, in real-world situations goals often need to be reprioritized, altered, or modified (Vattam et al., 2013). Autonomy in this case means that actors can perform GR both locally and consistently with their team, which is the focus of the CR framework we presented.

Young and his colleagues have described related work addressing GR in narrative generation (e.g., Riedl & Young 2010; Young et al., 2013). Their primary focus was to use automated planning to generate narratives and scripts for written or visual storytelling. Their developed NTP framework uses planning algorithms to produce desired narratives. As a result, the goals of individual characters are generated and managed during this process. Thus, planning can be used as a GR device to generate narratives.

7. Conclusions

- This paper introduced CR, a conceptual GR framework that uses narrative techniques to control the multi-actor behavior in multi-character scenarios. It extends the state-of-the-art on GR research, which previously did not focus on scenarios and solutions of this type (see Klenk, Molineaux, & Aha, 2013; Vattam et al., 2013, Samsonovich, 2014).
- Prior work related to this topic that has been studied in the literature on narrative intelligence has focused on planning (Riedl & Young, 2010; Young et al., 2013; Finlayson & Corman, 2013) and as such, has limitations. We have argued that the potential relative benefits of CR include overcoming limitations of the NTP approach identified earlier (Riedl & Young, 2010), and allowing a team of actors to use local GR instead of a global, team-level GR, thereby reducing the time and resources required for GR.
- We illustrated CR using example scenarios of cooperative problem solving. While we did not present empirical or formal analytical proofs in support of our claims, the presented informal analysis of selected examples strongly suggests that this topic is worthy of further study. Our

informal analyses of these scenarios suggests that reasoning in terms of characters differs from reasoning in terms of actors or narratives (when characters are not distinguished from actors), and that this may yield practical benefits in these and related scenarios. Future work suggestions include empirical computational studies using the formalism introduced here applied to examples similar to the analyzed scenarios.

Acknowledgements

Thanks to researchers collaborating with NRL's Adaptive Systems Section, especially Matt Molineaux, Ron Alford, Swaroop Vattam, and Mak Roberts, for their helpful discussions. The funding for this research was provided by OSD ASD (R&E). 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 OSD or NRL.

References

Abell, P. (1987) *The Syntax of Social Life: The Theory and Method of Comparative Narratives*. Oxford University Press, Oxford.

Abell, P. (1993) Some aspects of narrative method. *Journal of Mathematical Sociology*, 18, 2-3, 93-134.

Abell, P. (2009) A Case for cases: Comparative narratives in sociological explanation. *Sociological Methods and Research*, 38(1):38-70.

Abell, P. (2011). Singular mechanisms and Bayesian narratives. In: Demeulenaere, Pierre, (ed.) *Analytical Sociology and Social Mechanisms*, pp. 121-135. Cambridge University Press, Cambridge, UK. ISBN 9780521154352.

Bal, M. (1998). *Narratology: An Introduction to the Theory of Narrative*. Toronto UP.

Finlayson, M. A., & Corman, S. R. (2013). The Military Interest in Narrative. *Sprache und Datenverarbeitung*, 37 (1-2).

Gray, W. D. (Ed.) (2007). *Integrated Models of Cognitive Systems. Series on Cognitive Models and Architectures.* Oxford, UK: Oxford University Press.

Haven, K. (2014). *Story Smart: Using the Science of Story to Persuade, Influence, Inspire, and Teach*. Santa Barbara, CA: ABC-CLIO, LLC. ISBN: 9781610698115.

Haven, K. (2007). *Story Proof: The Science Behind the Startling Power of Story.* Westport, Connecticut: Libraries Unlimited. ISBN 978-1-59158-546-6.

Huhn, P., Meister, J.C., Pier, J., and Schmid, W. (Eds.) (2013). *The Living Handbook of Narratology* (online). Hamburg: Hamburg University Press. URL: http://wikis.sub.uni-hamburg.de/lhn/

Kapadia, M., Falk, J., Zund, F., Marti, M., Sumner, B., and Gross, M. (February 27, 2015). Computer-assisted authoring of interactive narratives. *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D 2015).*

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. doi: 10.1111/j.1467-8640.2012.00445.x

McCarthy, J. (1994). Situation calculus with concurrent events and narrative. Retrieved from http://jmc.stanford.edu/articles/narrative.html on 4.28.2015.

McCarthy, J. and Costello, T. (1998). Combining narratives. Retrieved from http://jmc.stanford.edu/articles/narrative2.html on 4.28.2015.

Nadarajah, S., & Sundaraj, K. (2013). A survey on team strategies in robot soccer: team strategies and role description. *Artificial Intelligence Review*, *40*(3), 271-304.

O'Neill, B. and Riedl, M.O. (2014). Dramatis: A Computational Model of Suspense. *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, Quebec City, Ontario.

Pentland, B. T., & Feldman, M. S. (2007). Narrative networks: Patterns of technology and organization. *Organization Science*, 18(5), 781-795. doi: 10.1287/orsc.1070.0283

Riedl, M.O., Stern, A., Dini, D., and Alderman, J. (2008). Dynamic experience management in virtual worlds for entertainment, education, and training. *International Transactions on Systems Science and Applications, Special Issue on Agent Based Systems for Human Learning*, 3(1).

Riedl, M.O. & Young, R.M. (2010). Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Research* 39: 217-268.

Riedl, M.O. & Young, R.M. 2014. The importance of narrative as an affective instructional strategy. In R. Sottilare, A. Graesser, X. Hu, and B. Goldberg (Eds.). *Design Recommendations for Adaptive Intelligent Tutoring Systems: Adaptive Instructional Strategies*, volume 2. Army Science Laboratory.

Roberts, M., Vattam, S., Aha, D.W., Wilson, M., Apker, T., & Auslander, B. (2014). Iterative goal refinement for robotics. In: A. Finzi & A. Orlandini (Eds.) Planning and Robotics: Papers from the ICAPS Workshop. Portsmouth, NH: AAAI Press (in press).

Samsonovich, A. V. (2014). Goal reasoning as a general form of metacognition in BICA. Biologically Inspired Cognitive Architectures, 9: 105-122. DOI: 10.1016/j.bica.2014.07.003.

Schmid, W. (2010). *Narratology: An Introduction*. Walter de Gruyter GmbH & Co. KG, Berlin/New York. ISBN 978-3-11-022631-7.

Simon-Shoshan, M. (2013). *Stories of the Law: Narrative Discourse and the Construction of Authority in the Mishnah.* Oxford: Oxford University Press.

Vattam, S., Klenk, M., Molineaux, M., & Aha, D.W. (2013). Breadth of approaches to goal reasoning: A research survey. In D.W. Aha, M.T. Cox, & H. Muñoz-Avila (Eds.) Goal Reasoning: Papers from the ACS Workshop (Technical Report CS-TR-5029). College Park, MD: University of Maryland, Department of Computer Science.

Ware, S.G., and Young, R.M. (2014). Glaive: A state-space narrative planner supporting intentionality and conflict, in *Proceedings of the 10th Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE2014)*, pp. 80-86. Raleigh, NC.

Young, R.M., Ware, S.G., Cassell, B.A., and Robertson, J. (2013). Plans and planning in narrative generation: A review of plan-based approaches to the generation of story, discourse, and interactivity in narratives. *SDV: Sprache und Datenverarbeitung: Intenational Journal of Language Processing*, 37 (1-2): 41-64.

Zimmerman, B.J. (2002). Becoming a self-regulated learner: An overview. *Theory into Practice*, *41*(2): 64-70.

Author Index