Goal-Driven Autonomy for Cognitive Systems

Matt Paisner (mpaisner@umd.edu)
Michael T. Cox (mcox@cs.umd.edu)
Michael Maynord (maynord@umd.edu)
Don Perlis (perlis@cs.umd.edu)
Computer Science Department, A.V. Williams Bldg.
College Park, MD 20742 USA

Abstract

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

Keywords: goal generation; anomaly handling; interpretation and explanation; TF-Tree; cognitive architecture; intelligent autonomy.

Introduction

Humans are astonishingly versatile, dealing with a wide range of unanticipated circumstances while still making headway on high-level goals. Humans can also recognize new problems and opportunities when they arise and react appropriately to them. Yet for the most part our machines cannot; they are like idiot-savants, very good at one narrow task and useless for anything else, even tasks very similar to the one they were designed for. This is the so-called brittleness problem, a major stumbling block for AI. What we appear to need is the opposite of expert systems: machines that might not excel at anything, but that can muddle through a wide range of circumstances and keep a strategic perspective. Yet more than 50 years of intense effort has failed to produce such machines. One approach to the problem of brittleness uses what we call goal-driven autonomy in a cognitive architecture. Here we describe some benefits of this approach in dynamic environments.

Goal-Driven Autonomy (GDA) is a unique conception that gives full independence to autonomous agents (Cox, 2007; Klenk, Molineaux, & Aha, 2013; Munoz-Avila, Jaidee, Aha, Carter, 2010). Rather than arbitrary anomaly-detection, the agent searches for problems in the context of its current goals and mission. Not all anomalies are problems, nor are all problems important enough to attend to. Rather than general assessment of an entire world state, the agent should abductively explain the causal factors giving rise to the problem. Given an explanation, a GDA agent may generate a (possibly new) goal that solves the problem (e.g., by removing its supporting conditions). In these terms, GDA is as much about problem recognition as it is problem-solving (Cox, 2013).

Consider a fire that breaks out at a construction site. This is a problem in many ways, not the least of which is that preconditions for actions (e.g., the integrity of building materials) will become unsatisfied. A standard planning algorithm might therefore generate a subgoal to extinguish the fire so that construction can continue. However a subsequent fire in quick succession might justify an investigation into a long-term threat to the construction site. One possible explanation would be the presence of an arsonist, leading to the goal of having the perpetrator in jail. Hence a direct reactive approach to fires would be to put them out; a GDA approach to this same situation would recognize the underlying problem in terms of its threat to the future of the enterprise.

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

Goal-Driven Autonomy in a Cognitive Architecture

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

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

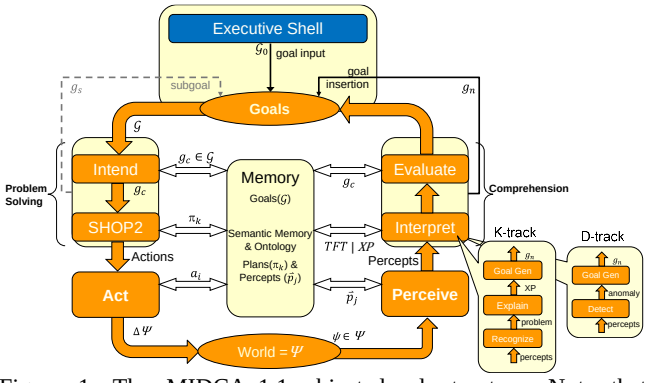

Figure 1: The MIDCA_1.1 object level structure. Note that execution-time subgoaling (dashes) is not currently implemented. TFT stands for TF-Tree, and XP stands for eXplanation Pattern.

Metacognitive Integrated Dual-Cycle Architecture Version 1.1

MIDCA_1.1 is an early version of the architecture whose components are shown in the schematic of Figure 1. It implements each phase of the cognitive loop, allowing the MIDCA agent to notice, analyze and respond to events in various simple domains.

Performance Domain To evaluate the performance of MIDCA in goal generation, we place the system in a modified blocksworld domain. This version of blocksworld includes both rectangular and triangular blocks, which compose the materials for simplified housing construction. The overarching goal in this domain is to build "houses" consisting of towers of blocks with a roof (triangle) on each. Specifically, the housing domain cycles through three goal states in building new houses. Figure 2 shows the three states and the goals that transition the system between them.

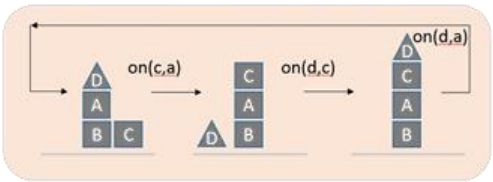

Figure 2: Three classifiers generate goals for subsequent states.

We use a simple world simulator in which actions, specified using predicate logic, are given prior to startup in a domain file. MIDCA's actions will be simulated, as well as actions performed by other agents and natural events. For the purpose of generating interesting anomalies for MIDCA to deal with, we have added a hidden arsonist, who can set blocks on fire. Furthermore there are two additional actions available to MIDCA to deal with fires. The three new actions are as follows:

- **light-on-fire (block)** *if block not on fire, lights it*
- **put-out-fire (block)** *if block on fire, extinguishes it*
- **apprehend (arsonist)** *imprisons arsonist*

MIDCA_1.1's task is to build houses while also dealing appropriately with fire. In the next subsection, we describe the techniques it uses in this task.

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

Perceive. The perceive phase is implemented very simply. The perceive component makes a copy of the current world state (defined in a predicate logic representation) and stores it in memory. As a result, MIDCA_1.1 always has a perfect, noise-free view of the current world state, though it has no direct knowledge of the arsonist's actions (it only sees that fires have started, not how). In our simple blocksworld example, MIDCA_1.1 copies to memory the same predicate representation of block relationships and attributes that the world simulator maintains as the current state.

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

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

Intend. The intend component determines which goals to pursue. If the evaluate phase reports that the previous goal

has been achieved, MIDCA_1.1 checks to see if a new goal was generated during the interpret phase. If so, it adopts that goal. Otherwise, it will do nothing until a new goal is generated. If the previous goal has not been achieved, it will also do nothing unless a goal with higher priority is generated, like a goal to put out a fire. In the latter case MIDCA_1.1 adopts the high-priority goal and puts the previous goal on hold. In MIDCA_1.1, goal priorities have been predetermined so that fire goals will be executed before construction goals. In blocksworld, the intend component converts the goal that has been generated into a task that can be taken as input by the planner. For example, the goal ¬onfire(A) would be transformed into put-out-fire(A).

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

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

Interpretation

The interpret phase of MIDCA has been the subject of much of our work, and is the focus of the experiments described below. It is implemented by two GDA processes that combine to generate new goals based on the features of the world the agent observes. We call these processes the D-track, which is a data driven, bottom-up approach, and the K-track, which is knowledge rich and top-down. A statistical anomaly detector constitutes the first step of the D-track, a neural network identifies low-level causal attributes of detected anomalies, and a goal classifier, trained using methods from machine learning, formulates goals. The K-track is implemented as a case-based explanation process.

The representations for expectations significantly differ between the two tracks. K-track expectations come from explicit knowledge structures such as action models used for planning and ontological conceptual categories used for interpretation. Predicted effects in the former and attribute constraints in the latter constitute expectations. By contrast, D-track expectations are implicit. Here the implied expectation is that the probabilistic distribution from which observations are sampled will remain the same. When the difference between expected and perceived distribution is statistically significant, an expectation violation is raised.

D-Track Goal Generation The D-track interpretation procedure uses a novel approach for noting anomalies. We apply the statistical distance metric called the A-distance to streams of predicate counts in the perceptual input (Cox, Oates, Paisner, & Perlis, 2012), yielding a measurement of how the distributions of predicates differ from a base state. This enables MIDCA to detect regions in which statistical distributions of predicates differ from previously observed input. MIDCA's implicit assumption is that where change occurs problems may exist.

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

Goal generation is achieved in MIDCA_1.1 using TF-Trees (Maynord, Cox, Paisner, & Perlis, 2013), machine-learning classification structures that combine two algorithms which work over the predicate representation of the blocksworld domain. The first of these algorithms is Tilde (Blockeel, & De Raedt, 1997), which is itself a generalization of the standard C4.5 decision tree algorithm. The second algorithm is FOIL (Quinlan, 1990), an algorithm which, given a set of examples in predicate representation reflecting some concept, induces a rule consisting of conjunctions of predicates that identify the concept. Given a world state, a TF-Tree first uses Tilde to classify the state into one of a set of scenarios. Each scenario is then associated with a rule generated by FOIL. Once that rule is obtained, groundings of the arguments of the predicates in that rule are permuted until either a grounding that satisfies the rule is found (in which case a goal is generated) or until all permutations have been eliminated as possibilities (in which case no goal is generated). The structure of a TF-Tree is a tree where in internal nodes are produced by Tilde and leaf nodes are rules produced by FOIL. Figure 3 depicts the structure of the TF-Tree MIDCA_1.1 uses in cycling through the 3 block arrangements.

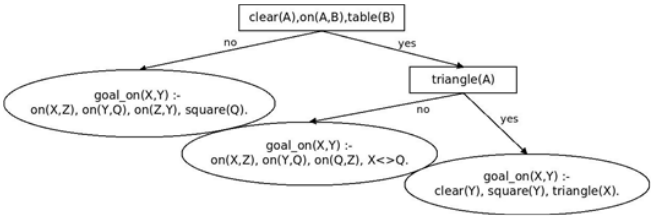

Figure 3: Depiction of the TF-Tree used in cycling through the 3 block configurations.

For example given the middle state of Figure 2, triangle D is clear, it is on the table, and the table is a table. Thus we take the right branch labeled "yes." Now triangle D is also a triangle, so again we take the "yes" branch to arrive at the right-most leaf of the tree. The leaf rule then binds the

variable Y to the clear square C, and the resulting goal is to have triangle D on square C.

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

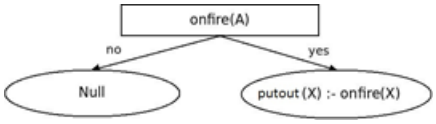

Figure 4: TF-Tree that generates goals to put out fires

K-Track Goal Generation The K-track GDA procedure uses the XPLAIN system (Cox & Burstein, 2008). XPLAIN is built on top of the Meta-AQUA introspective story understanding system (Cox and Ram 1999) and is used in MIDCA to detect and explain problems in the input perceptual representations. The system's interpretation task is to "understand" input by building causal explanatory graphs that link subgraph representations in a way that minimizes the number of connected components. XPLAIN uses a multistrategy approach to this problem. Thus, the top-level goal is to choose a comprehension method (e.g., script processing, case-based reasoning, or explanation generation) by which it can understand an input. When an anomalous or otherwise interesting input is detected, the system builds an explanation of the event, incorporating it into the preexisting model of the story. XPLAIN uses case-based knowledge representations implemented as frames tied together by explanation-patterns (Cox & Ram, 1999) that represent general causal structures.

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

Table 1: The arsonist explanation pattern

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

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

Evaluation: Autonomous goal formulation

The fires are problems because of their effect on housing construction and the supposed profits of the housing industry, and the threats they pose to life and property. Our approach to understanding fire problems is to ask why the fires were started and not just how. A scientific explanation of how the fire started would relate the presence of sufficient heat, fuel, and oxygen with the combustion of the blocks. For example, generating the negation of the presence of the oxygen would result in the goal ¬oxygen, which would put out the fire. But this does not address the reason the fire started in the first place. One might arrive at multiple answers to this question. Poor safety conditions might have led to fire, or an arsonist may have lit it. In the latter case, the arsonist causes the presence of the heat through a lighting action, which is hidden from the agent.

Given this explanation the agent can nevertheless anticipate the threat of more fires and generate a goal to remove the threat by finding the arsonist. Apprehending the arsonist then removes the potential of fires in the future rather than just reacting to fires that started in the past.

We tested the effectiveness of three methods for goal generation under these conditions. The first method was a simple baseline using predetermined, exogenous goals. The second method used the statistical, D-Track GDA method described in Section 2.2.1. The third method combined the D-Track approach with additional analysis using K-Track GDA as described in Section 2.2.2. Details appear in Table 2. For each test, MIDCA was run for 1000 time steps (equivalent to executing 1000 actions). At each step, the arsonist would have a probability p of starting a fire unless he had previously been apprehended. The value of p in the experiments described below was 0.4, allowing for enough fires to be significant without precluding progress in the tower construction project.

Table 1: Methods for goal generation

Exogenous Goals	Used a predetermined goal list that cycled between the 3 states constructing towers. Did not deviate from list in response to fires. Goals were [on(C,A), on(D,C), on(D,A), on(C,A), … on(D,A)]
D-Track GDA Goal Generation	Generated goals using TF-Trees. Trees were trained and implemented such that when no fire was present, they would generate the next goal in the 3-part cycle, but when a fire was present, they would instead generate a goal to put it out.
2-Track GDA Goal Generation	Generated goals using a combination of TF-Trees and a K-Track approach using XPLAIN. XPLAIN contained knowledge about possible arsonists and suggested a goal to search and apprehend an arsonist given fire. TF-Trees generated other goals as in 2 above

We tracked three scoring metrics: the number of towers completed; the overall prevalence of fires; and a combined score measuring completion of fire-free towers. Details on each scoring metric are shown in Table 2. At each time step in which a tower was completed – e.g. a triangular block was placed on a stack of rectangular blocks, – all fires were automatically put out, and the agent started on a new construction project.

Preliminary empirical results show that GDA approaches using only the D-Track as well as using both D-Track and K-Track perform significantly better than a baseline that does not use GDA. Also, the combined D- and K-Track implementation outperforms the purely statistical variant by a large margin. Figure 5 shows the detailed results of testing.

Table 2: Scoring metrics for testing

Towers Completed	Total number of 3- and 4-block towers completed in 1000 cycles
Fire Prevalence	The number of blocks on fire times the number of time steps they were on fire. If 3 blocks burn for 3 time steps and go out simultaneously the score is 9
Overall Score	Awards 1 point per block that is not on fire in a completed tower. A 4-block tower with 2 blocks on fire scores 2 points

The agent that used only exogenous goals completed the most towers, but, because it did not deal with fires in any way, most of the towers were burning as they were completed and received very low scores. Certainly, this baseline behavior does not seem to be sufficient for a fully autonomous house construction agent. The second agent used behavior dictated by TF-Trees to fight fires directly. It did not complete as many towers because it divided its attention between construction and extinguishing fires, but the towers it did construct were consequently much less likely to be on fire. Its total score was 367, 54.2% better than the baseline agent. Finally, the dual-track GDA agent analyzed the problem logically using XPLAIN, and thereby suggested an explanation of the fires as potentially caused by arson. As such, it generated a goal early in the process to apprehend the arsonist. This took some time, but afterwards it was able to devote its full attention to house construction without devoting time to firefighting. It completed nearly as many towers as the baseline agent, and did so with almost no incidence of fire, since no fires started after the arsonist was apprehended. The dual-track agent achieved a score of 584, 245.4% better than the baseline agent.

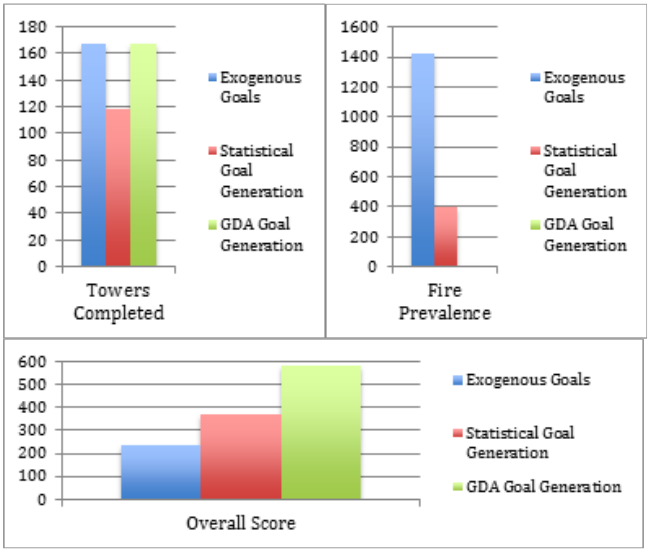

Figure 5: Results of testing using 3 methods. Note that the value of GDA Goal Generation in the Fire Prevalence panel is 2, which is too small to show clearly in the graph.

It should not be surprising that an agent that is capable of reacting to the unanticipated problem posed by fire performs better than one that heedlessly continues on a predetermined

course of action. Perhaps more telling is the large advantage gained by the dual-track agent, which has the knowledge to identify and address the true source of the problem, rather than simply treating its symptoms. Though this example is too simple to easily generalize, these results at least suggest the importance of combining a knowledge-rich approach with low-level data analysis to achieve the best possible results.

Related Work

Work has been done to expand the capacities of agents by making use of goal manipulation. (Hanheide et al., 2010) created a framework for managing goals to be used by a robot exploring an unknown space which autonomously classifies rooms into categories. They ran the robot with and without the framework, and concluded that a framework for goal management increases the performance of the robot.

Schermerhorn, Benton, Scheutz, Talamadupula, & Kambhampati (2009) sought to use modification of a robot's goal structure to confront the challenges of a partially observable, non-deterministic domain in which prior knowledge about the domain is limited, knowledge acquisition is non-monotonic, planning is subject to real time constraints, and goals and utilities can dynamically change during execution. Counterfactuals determine actions that lead to goal opportunities, and when opportunities are detected, the goal structure can be modified. Other work has taken advantage of the GDA model which we use in our work. For example, Munoz-Avila, Jaidee, Aha, and Carter (2010) merged the GDA framework with case based reasoning (CBR) and ran a comparison between a GDA system using CBR, a rule based variant of GDA, and a non-GDA based agent. The CBR based GDA system outperformed the others, and functioned by making use of a case base that mapped goals to expectations and a case base that mapped mismatches to new goals.

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

Conclusion

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

Acknowledgments

This is supported by ONR Grants N00014-12-1-0430 and N00014-12-1-0172 and by ARO Grant W911NF-12-1-0471.

References

Blockeel, H., & De Raedt, L. (1997). Lookahead and discretisation in ILP. *Proc. of the 7th intl. workshop on inductive logic programming* (pp. 77–84) Berlin: Springer

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

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., & Burstein, M. H. (2008). Case-based explanations and the integrated learning of demonstrations. *Künstliche Intelligenz 22*(2), 35-38.

Cox, M. T., Maynord, M., Paisner, M., Perlis, D., & Oates, T. (2013). The integration of cognitive and metacognitive processes with data-driven and knowledge-rich structures. *Proc. of Annual Meeting of the Intl. Association for Computing and Philosophy*.

Cox, M. T., Oates, T., Paisner, M., & Perlis, D. (2012). Noting anomalies in streams of symbolic predicates using A-distance. *Advances in Cognitive Systems 2*, 167-184.

Cox, M. T., & Ram, A. (1999). Introspective multistrategy learning. *Artificial Intelligence, 112*, 1-55.

Hanheide, M., Hawes, N., Wyatt, J., Göbelbecker, M., Brenner, M., Sjöö, K., Aydemir, A., Jensfelt, P., Zender, H. & Kruijff, G. J. (2010). A framework for goal generation and management. *AAAI Workshop on Goal-Directed Autonomy*.

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

Maynord, M., Cox, M. T., Paisner, M., & Perlis, D. (2013). Data-driven goal generation for integrated cognitive systems. C. Lebiere & P. S. Rosenbloom (Eds.), *Integrated Cognition: Papers from the 2013 Fall Symposium* (pp. 47-54). Menlo Park, CA: AAAI Press.

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

Munoz-Avila, H., Jaidee, U., Aha, D. W., Carter, E. (2010). Goal-driven autonomy with case-based reasoning. I. Bichindaritz & S. Montani (Eds.), *Case-Based Reasoning. Research and Development, 18th International Conference on Case-Based Reasoning, ICCBR 2010* (pp. 228-241). Berlin: Springer.

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

Paisner, M., Perlis, D., & Cox, M. T. (2013). Symbolic anomaly detection and assessment using growing neural gas. *Proceedings of the 25th IEEE Intl. Conf. on Tools with Artificial Intelligence* (pp. 175-181). Piscataway, NJ: IEEE Press.

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

Schermerhorn, P., Benton, J., Scheutz, M., Talamadupula, K., Kambhampati, S. (2009). Finding and exploiting goal opportunities in real-time during plan execution. *Proc. 2009 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems* (pp. 3912-3917). Piscataway, NJ: IEEE Press.