Data-Driven Goal Generation for Integrated Cognitive Systems

Michael Maynord, Michael T. Cox, Matt Paisner, and Don Perlis
University of Maryland, College Park, MD 20742
maynord@umd.edu, mcox@cs.umd.edu, mpaisner@umd.edu, perlis@cs.umd.edu

Abstract

We describe our Meta-cognitive, Integrated, Dual-Cycle Architecture (MIDCA), whose purpose is to provide agents with a greater capacity for acting in an open world and dealing with unexpected events. We present MIDCA_1.0, a partial implementation which explores a novel machine-learning approach to goal generation using the Tilde and FOIL algorithms. We describe results from this goal generation algorithm and preview MIDCA_1.1, a partially implemented version that will guide the goal insertion process using statistical anomaly detection and categorization techniques. Finally, we outline the next steps towards a complete, functional implementation of the MIDCA architecture.

Introduction

A number of interesting research projects exist within the artificial intelligence and cognitive science communities that integrate multiple high-level cognitive functions and perform complex tasks in dynamic environments. Well known examples include ACT-R (Anderson, 1993), CogAff (Sloman, 2011), Companion Cognitive Systems (Forbus, Klenk, & Hinrichs, 2009), DIARC (Krause, Schermerhorn, & Scheutz, 2012), EPILOG (Morbini & Schubert, 2011), Icarus (Langley & Choi, 2006), MCL (Anderson, Oates, Chong, & Perlis, 2006; Schmill et al., 2011), SNePS(Shapiro, 2000), and Soar (Laird, 2012). However, most if not all intelligent agent architectures define autonomous behavior as goal-following behavior. That is, an agent is considered autonomous and intelligent if, when given a goal by a human, the agent can determine on its own the actions that achieve the goal. We claim that this is insufficient.

The automated planning community represents input problems as an initial state and a goal state along with a domain theory containing an action model (see Ghallab, Nau, & Traverso, 2004). The subsequent task for a planning agent is to generate a sequence of actions that when executed from the initial state achieve the goal state. Once the plan is successfully executed, the agent either halts or waits to be given another goal. This "wait to be told what to do" model of autonomy presents agents as autonomous because they do not have to be told how to achieve the goal that solves the problem. The alternative model of autonomy we advocate here consists of self-motivated processes of generating and managing an agent's own goals in addition to goal pursuit. This model - called Goal-Driven Autonomy (Cox, 2007; Klenk et al., 2013) - casts agents as independent actors that can recognize problems on their own and act accordingly. Goals are seen as dynamic and malleable and can arise in three cases: goals can be subject to transformation or abandonment (Cox & Veloso, 1998; Talamadupula, et al., 2010), they can arise from sub-goaling on unsatisfied preconditions during planning, and they can be generated from scratch during interpretation.

What is missing in the planning and agent communities is a recognition that autonomy is not just planning, acting and learning. It also must incorporate a first-class reasoning mechanism that interprets and comprehends the world as plans are executed. It is this comprehension process that not only perceives actions and events in the world, but can recognize threats to current plans, goals, and intentions. We claim that a balanced integration between planning and comprehension leads to agents that are more sensitive to surprise in the environment and more flexible in their responses.

In our approach, flexibility is realized through a process we call goal insertion, where an agent inserts a goal (that may already have been learned somehow) into its planning apparatus. Goals are produced through the process of goal generation, the general term that includes creation and deployment of autonomous goals where the agent takes initiative to come up with new concerns and to pursue new opportunitites. Goals arise as an agent detects discrepancies between its sensory inputs and its expectations. The agent explains what causes the discrepancy, then solves the problem by generating a new goal to remove or mitigate the problem's cause (Cox, 2007).

In this paper we present a cognitive architecture called MIDCA that integrates planning, comprehension, and metacognition. In the following section, we briefly overview the MIDCA architecture. We will describe a statistical data-driven approach to goal generation that can be contrasted with a knowledge-rich explanation-based method. Subsequently we describe a machine learning technique for data-driven goal generation and report preliminary results in a very simple blocksworld domain. The next section then discusses future research and we conclude with a brief summary.

The Metacognitive Integrated Dual-Cycle Architecture

Computational metacognition distinguishes agent reasoning about reasoning from reasoning about the world (Cox, 2005). The Metacognitive, Integrated, Dual-Cycle Architecture (MIDCA) (Cox, Maynord, Paisner, Perlis, & Oates, in press; Cox, Oates, & Perlis, 2011) consists of action-perception cycles at both the cognitive (i.e., object) level and the metacognitive (i.e., meta-) level. 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 world match the goal state. The agent perceives changes to the environment resulting from the actions, interprets the percepts with respect to the plan, and evaluates the interpretation with respect to the goal. At the object level, the cycle achieves goals that change the environment. At the meta-level, the cycle achieves goals that change the object level. That is, the metacognitive perception components introspectively monitor the processes and mental state changes at the cognitive level. The action component consists of a meta-level controller that mediates reasoning over an abstract representation of the object-level cognition.

To illustrate these distinctions, consider the object level as shown in Figure 1. Here the meta-level executive function manages the goal set G. In this capacity, the meta-level can add initial goals (g_0), subgoals (g_s) or new goals (g_n) to the set, can change goal priorities, or can change a particular goal (Δg). In problem solving, the Intend component commits to a current goal (g_c) from those available by creating an intention to perform some task that can achieve the goal (Cohen & Levesque, 1990). The Plan component then generates a sequence of actions (π_k, e.g., a hierarchical-task-net plan, see Nau, et al., 2001) that instantiates that task given the current model of the world ($W*$) and its background knowledge (e.g., semantic memory and ontologies). The plan is executed to change the actual world (W) through the effects of the planned actions (a_i). The goal and plan are stored in memory and constitute the agent's expectations about how the world will change in the future. Then given these expectations, the comprehension task is to understand the execution of the plan and its interaction with the world with respect to the goal.

Comprehension starts with perception of the world in the attentional field. Interpretation takes as input the resulting percepts (p_j) and the expectations in memory (π_k and g_c) to determine whether the agent is making sufficient progress. A Note-Assess-Guide (NAG) procedure (Anderson & Perlis, 2005; Perlis, 2011) implements the comprehension process. The procedure is to note whether an anomaly has occurred; assess potential causes of the anomaly by generating hypotheses; and guide the system through a response. Responses can take various forms, such as (1) test a hypothesis; (2) ignore and try again; (3) ask for help; or (4) insert another goal (g_n). In the absence of an anomaly, the agent incorporates the changes inferred from the percepts into the world model ($\Delta W*$) and the cycle continues. This cycle of problem-solving and action followed by perception and comprehension functions over discrete state and event representations of the environment.

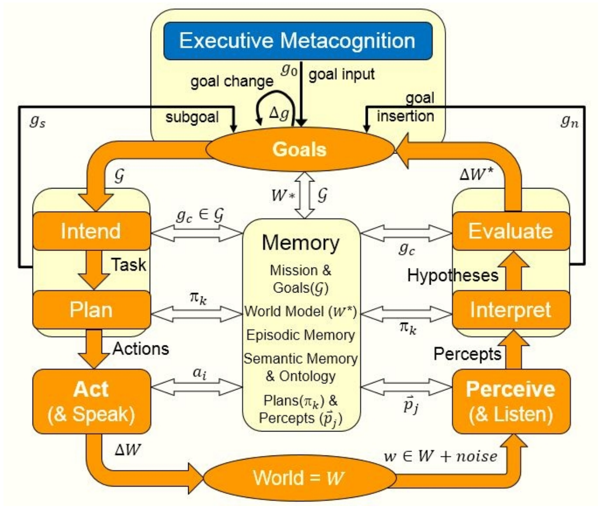


Figure 1: Object-level detail with meta-level goal management shown (taken from Cox, Maynord, Paisner, Perlis, & Oates, in press)

The NAG procedure at both meta- and object-levels has two variations that represent a bottom-up, data-driven track and a top-down, knowledge rich, goal-driven track. The data-driven track we call the D-track; whereas the knowledge rich track we call the K-track. The representations for expectations significantly differ between the two tracks. K-track expectations come from explicit knowledge structures such as action models used for planning and ontological conceptual categories used for interpretation. Predicted effects form the expectations in the former and attribute constraints constitute expectation in the latter. By contrast, D-track expectations are implicit. Here the implied expectation is that the distribution of observations will remain the same. Statistically significant change beyond a threshold constitutes an expectation violation. In past work on the INTRO (Cox, 2007) system, the K-track has been implemented as a case-based explanation process (Cox & Burstein, 2008). In the current implementation of MIDCA, comprehension is performed solely with D-track processes. For the purposes of this paper, we will describe the D-track implementation that generates new goals (i.e., g_n) during interpretation.

MIDCA_1.0

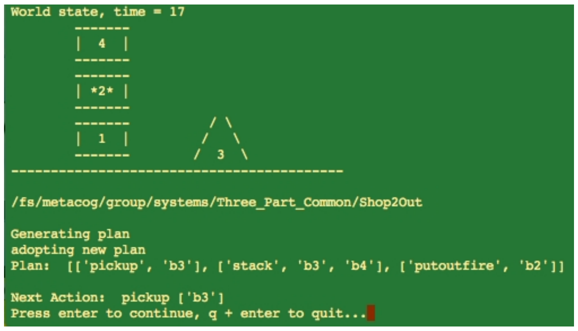
To determine the degree to which the introduction of reasoning improves the performance of agents, we must construct a baseline system which lacks reasoning. The performance of systems which make us of reasoning can then be compared to the performance of this baseline system. What follows is the construction and evaluation of such a baseline within the framework of MIDCA, in an early implementation called MIDCA_1.0.

For the implementation of MIDCA_1.0 we used a modified blocksworld for the domain. This version of blocksworld includes both rectangular and triangular blocks, as well as the possibility for blocks to catch on fire and for fires to be put out. Using blocksworld avoids many of the difficulties that come with more complex domains, such as intractability resulting from scale, inaccuracies stemming from imperfect perception, or complications introduced by stochastic processes. The fire element was added to create a fairly obvious anomaly to which the system could respond.

MIDCA_1.0 is a simplified version of the complete MIDCA architecture whose components are shown in the schematic of Figure 1. It is composed of three parts: a world simulator that generates successor states based on valid actions taken in the blocksworld domain; a goal generator; and a planner. For the planner, we used SHOP2 (Nau et al., 2003), a domain-independent task decomposition planner. This planner constitutes the Plan and Intend components of Figure 1 - the Intend component being implicit in the planner. Whereas the full MIDCA architecture has a metacognitive component, which manages goals, MIDCA_1.0 has no goal management, and simply passes goals from the goal generator to the planner. In MIDCA_1.0, the goal generator constitutes the Interpret component of Figure 1, and the Evaluate component of Figure 1, whose role it is to evaluate generated goals to help in goal selection and management, is absent. The Act component of Figure 1 is incorporated into the blocksworld simulator, and the Perception component of Figure 1 is implicit in the transfer of world state representation to the goal generator.

Goal generation is implemented using a conjunction of two algorithms, both of which work over predicate representations of the world. Tilde (Blockeel, & De Raedt, 1997) is an extension of C4.5 (Quinlan, 1993), the standard decision tree algorithm, and FOIL (Quinlan, 1990) is a rule generation algorithm producing conjunctions of predicates to match a concept reflected in a training set. Given a world state interpretation, the state is first classified using Tilde into one of multiple scenarios, where each scenario has an associated goal generation rule generated by FOIL. Given an interpretation and a scenario, different groundings of the variables of the FOIL rule are permuted through until either one is found which satisfies that rule, in which case a goal can be generated, or until all permutations of groundings have been attempted, in which case no goal can be generated. In effect, this approach results in a decision tree generated by Tilde in which each leaf corresponds to a rule generated by FOIL. We call this data structre a *TF-Tree*. This approach to goal generation is naïve in the sense that it constitutes a mapping between world states and goals which is static with respect to any context; there is no explicit reasoning in this goal generation scheme.

TF-Trees are constructed through a supervised learning process which requires the creation of domain-specific training sets. Both training and test sets consist of world states and their associated "correct" goal mappings. Using a test set reflective of states encountered in the world, we can determine the baseline level of accuracy with which this reasoning-free approach selects the right goal.



```
World state, time = 17
        -------
        |  4  |
        -------
        -------
        | *2* |
        -------
        -------          / \
        |  1  |         /   \
        -------        /  3  \
------------------------------------------

/fs/metacog/group/systems/Three_Part_Common/Shop2Out

Generating plan
adopting new plan
Plan:  [['pickup', 'b3'], ['stack', 'b3', 'b4'], ['putoutfire', 'b2']]

Next Action:  pickup ['b3']
Press enter to continue, q + enter to quit...
```

Figure 2: Block Visualization

When MIDCA_1.0 is launched, it is passed an initial world state. From that state, a TF-Tree trained on data constructed to match the domain is used to generate a goal appropriate to that state. That goal is passed to SHOP2, which generates a plan to achieve it. The plan is then executed to completion in the world simulator. Upon plan completion, the resulting world state is passed back to the goal generator, and the TF-Tree is used to select a new goal, restarting the process. Note that goals are not generated for the intermediate states(states between the starting and end state) in a plan.

An example of the output from this process is shown in Figure 2. Here we added fire as a feature to the blocksworld domain. This domain functions precisely in the same manner as standard blocksworld, with the addition that blocks that are on fire can be extinguished using a $putout$ operation. Goals can now consist of two components, a stacking component, and a putout component. In Figure 1, these components correspond to g_n, and together they constitute $\mathcal{G}$. In the state shown, a triangle block is on the table and three square blocks are stacked to form a tower. The middle block in the tower is on fire, which is represented by asterisks inside the block. This state has been passed to the goal generator, which has generated goals to stack the triangle block on top of the tower and to put out the fire ($on(b3, b4)$, $putout(b2)$). SHOP2 has then created a plan to accomplish these goals, which is shown, along with the next action to be taken, beneath the state representation. At the next time step, the world state will have changed in response to the action $pickup[b3]$, where $b3$ is the triangular block, being taken.

Goal Generation Results

We ran the TF-Tree algorithm over scenario sets to evaluate performance. The first scenario set consists of four scenarios of the following description:

```
1. A short tower, a square, a triangle.
2. A tall tower, a square, a triangle.
3. A tall tower topped by a triangle, a short tower,
   a square, a triangle.
4. A tall tower, a short tower, a square, a triangle.
```

Goals are of the form $on(X, Y)$, where X and Y are blocks. The associated goals for each scenario are:

```
    1. Place the square on the tower.
    2. Place the triangle on the tower.
    3. Transfer the triangle at the top of the tall
       tower to the short tower.
    4. Transfer the top of the tall tower onto the square.
```

For illustration purposes, this is what the decision tree generated by Tilde looks like:

```
triangle(A),on(A,B),square(B) ?
+--yes: [scen_003]
+--no:  on(C,D),on(D,E),on(E,F),square(F) ?
    +--yes: on(F,G),on(G,H),square(H) ?
    |   +--yes: [scen_002]
    |   +--no:  clear(I),on(I,J),on(J,K),on(K,L),table(L) ?
    |       +--yes: [scen_004]
    |       +--no:  clear(M),on(M,N),on(N,O),table(O) ?
    |           +--yes: [scen_004]
    |           +--no:  [scen_002]
    +--no:  [scen_001]
```

Here is what a block arrangement from each of the four scenarios could look like:

```
Scenario 1:
   ------
   |  s |
   ------
   ------          ------          /\
   |  s |          |  s |         /  \
   ------          ------        /  t \
-------------------------------------------------------

Scenario 2:
   ------
   |  s |
   ------
   ------
   |  s |
   ------
   ------
   |  s |
   ------
   ------          ------          /\
   |  s |          |  s |         /  \
   ------          ------        /  t \
-------------------------------------------------------

Scenario 3:
                                 /\
                                /  \
                               /  t \
                               ------
                               |  s |
                               ------
                               ------
                               |  s |
                               ------
                               ------        ------
                               |  s |        |  s |
                               ------        ------
   ------          /\          ------        ------
   |  s |         /  \         |  s |        |  s |
   ------        /  t \        ------        ------
-------------------------------------------------------


Scenario 4:
                               ------
                               |  s |
                               ------
                               ------
                               |  s |
                               ------
                               ------        ------
                               |  s |        |  s |
                               ------        ------
   ------          /\          ------        ------
   |  s |         /  \         |  s |        |  s |
   ------        /  t \        ------        ------
-------------------------------------------------------
```

Here are the rules generated by FOIL:

```
Scenario 1:
goal(A,B) :- square(A), clear(B), clear(A), A<>B, on(B,C),
    square(C).

Scenario 2:
goal(A,B) :- clear(B), on(A,C), on(B,D), square(D),
    not(square(A)).

Scenario 3:
The disjunction of:
goal(A,B) :- square(B), clear(B), clear(A), on(A,C), on(B,D),
    on(C,E), on(E,F), on(D,G), on(G,H), on(F,I), not(on(I,H)).
goal(A,B) :- clear(B), clear(A), on(A,C), on(B,D), on(C,E),
    on(E,F), on(D,G), table(G), on(F,H), not(on(H,G)).
goal(A,B) :- clear(B), clear(A), on(A,C), on(B,D), on(C,E),
    on(E,F), on(D,G), on(G,H), on(F,I), on(I,H).
goal(A,B) :- clear(B), clear(A), on(A,C), on(B,D), on(C,E),
    on(E,F), on(D,G), on(F,H), on(H,G).

Scenario 4:
goal(A,B) :- clear(A), square(B), clear(B), on(A,C), on(B,D),
    not(on(C,D)), table(D), on(C,E), not(on(E,D)).
```

Given this TF-Tree structure (the decision tree learned by Tilde with rules learned by FOIL at the leaves), generating a goal from a world state in predicate representation works as follows: The decision tree will classify the world state into one of multiple scenarios, each scenario being distinguished from others by the kind of actions considered appropriate. Each scenario has an associated rule generated by FOIL. To generate the goal, the variables in the rule must be grounded to blocks in the world state, and so groundings are permuted through until either one which satisfies the rule is found, in which case a goal can successfully be generated, or until all permutations have been determined invalid, in which case no goal can be generated.

For example, in generating a goal for the block arrangement belonging to scenario 1 shown above, the first node of the decision tree(also shown above) checks to see if it is the case that $triangle(A) \wedge on(A, B) \wedge square(B)$ for some blocks A and B. This is not the case, so we follow the "no" link to the next node, which checks to see if

$on(C, D) \wedge on(D, E) \wedge on(E, F) \wedge square(F)$ is true for some set of blocks C,D,E, and F. This is also not the case, so we follow the "no" link to a leaf of the tree, which classifies the block arrangement as belonging to scenario 1.

The variables A and B in the associated rule as generated by FOIL then will give us the arguments for $goal(A, B)$, we just need to find groundings of A,B, and C such that $square(A) \wedge clear(B) \wedge clear(A) \wedge A <> B \wedge on(B, C) \wedge square(C)$. The only groundings of variables in this case which satisfy the rule are if A is the single square on the ground, B is the second block of the tower, and C is the first block of the tower. So, we have just generated the goal of placing the single square onto the tower, the goal which we trained TF-Tree to learn.

We were interested in how the performance of the TF-Trees' degraded as the size of their training corpus decreased. Table 1 shows the accuracy across different training corpus sizes.

The units on training corpus size correspond to sets of nearly exhaustively listed valid and invalid goals for a given block arrangement. Each set will contain up to n^2 goals, where n is the number of blocks.

Table 1: Goal generation accuracy across training corpus sizes.

Training Corpus Size	Accuracy	Iterations	Std. Dev.
5	0.59	1	-
10	0.68	7	10
25	0.88	1	-
100	1.0	3	0.125
1000	1.0	1	-

Constructing a classifier from the output of Tilde and FOIL is time consuming, so the numbers in table 1 are from single iterations. To determine the variability of these accuracies, we ran 7 iterations over corpus sizes of 10, getting a mean accuracy of 0.68 with a standard deviation of 0.125. On 3 iterations over corpus sizes of 100, mean accuracy was 1.0, with standard deviation 0.

The second scenario set on which we tested TF-Tree is for a more rigorous test of the method and consists of 8 scenarios, with the following descriptions:

1. Three towers, two of height 2 or 3, one of height 4 or 5, and one triangle and one square on the ground.
2. Three towers, two of height 4 or 5, one of height 2 or 3, and one triangle and one square on the ground.
3. Three towers, the first of height 2 or 3, the second of height one greater than the first, the third of height one greater than the second, and one triangle and one square on the ground.
4. Two towers, each of height 2, 3, 4, 5, or 6, one topped by a triangle, and one triangle and one square on the ground.
5. Two towers, both of height 2, 3, 4, 5 or 6, both topped by a triangle, and on triangle and one square on the ground.
6. One tower of height 2 or 3, and one triangle and one square on the ground.
7. One tower of height 4, and one triangle and one square on the ground.
8. One tower of height 5, 6, 7, or 8, and one triangle and one square on the ground.

The goals associated with those 8 scenarios are as follows:

1. Place the ground triangle on top of the tall tower.
2. Place the ground square on the top of the short tower.
3. Move the top square of the tallest tower to the top of the shortest tower.
4. Place the ground triangle on the tower lacking a triangle top.
5. Place the ground triangle on the ground square.
6. Place the ground square on the tower.
7. Place the ground triangle on the tower.
8. Place the top square of the tower onto the ground square.

Goal generation accuracy over these 8 scenarios was 0.78, as was classification accuracy. The implication of these numbers being the same is that the error was introduced solely by the classification tree as generated by Tilde and not the rules learned by FOIL. Table 2 is a confusion matrix showing scenario classifications.

Table 2: Classification confusion matrix for the 8 scenario set.

Actual\Predicted	1	2	3	4	5	6	7	8
1	763	735	1502	0	0	0	0	0
2	716	780	1504	0	0	0	0	0
3	0	0	3000	0	0	0	0	0
4	0	0	0	3000	0	0	0	0
5	0	0	0	0	3000	0	0	0
6	0	0	0	0	0	3000	0	0
7	0	0	0	0	0	0	3000	0
8	0	0	732	0	0	0	0	2268

The difficulty that Tilde had in classifying scenarios 1 and 2 can be explained by the similarities between scenarios 1, 2, and 3. These three scenarios each contain three towers(more than the other five scenarios) and so in this way are more complex. In addition, the differentiating characteristics of these three scenarios are not as obvious as the other five scenarios. Here the only differentiating factor is the height of the three towers. Tilde proved incapable of learning to distinguish between these relatively complex and similar scenarios, but given the simplicity of the approach this isn't surprising.

Discussion

A more sophisticated approach to goal generation is needed to deal with similar, but importantly different, situations. In MIDA_1.0 world states in isolation are the only thing determining goals. In an approach which takes the context of how the present state relates to past states, and has some conception of the reasons behind the form of the present state, there is more information available for differentiation and interpretation. Generation of goals for arrangements such as scenarios 1, 2, and 3 are where our baseline performs poorly, and where future implementations of MIDCA will perform better.

Because MIDCA_1.0 employs goal generation based on statistics, rather than reasoning, it constitutes a useful performance baseline for evaluating future reasoning-based approaches to this problem. In addition, MIDCA_1.0 provides us with an early implementation of the MIDCA architecture that we intend to expand to operate on more complex domains and to utilize more advanced methods for anomaly detection, failure assessment and goal management.

The approach we present in this paper differs from the prevailing approach to agent behavior in that agents are not extrinsically provided with goal-states to achieve. Rather, starting from an initial state, the agent must determine on its own the appropriate goals to work toward. Once goals are produced, a plan to achieve a goal is generated and enacted within the environment in the standard fashion.

In MIDCA_1.0, the goal generator has no capacity for reasoning, rather it generates goals by means of a TF-Tree trained on a training corpus mapping states to appropriate goals. The blocksworld domain is amenable to this method of goal generation, as well as to methods employing reasoning.

In the blocksworld domain, an agent making use of reasoning may have a conception of purposes behind actions and the ultimate outcome the agent is trying to achieve. For example, the agent may have the notion that it is presently trying to build a building out of blocks. As such, when presented with some input the agent can reason over possible actions, evaluating them with respect to the degree to which each action increases the 'building-like' nature of the block set. Or, for example, an agent may desire to achieve the goal $on(triangle, square)$ because the triangle can be seen as a roof, the square can be seen as a house, and a roof on a house can protect against the elements. This approach is distinct from the approaches consisting of direct input/output mappings in that the rationale behind action selection is explicit in the reasoning, rather than implicit in the mapping. By including these rationales explicitly, the adaptability of the agent can be improved.

Future and Related Research

Related Research

Work on approaches conforming to the Goal-Driven Autonomy model includes (Munoz-Avila, Jaidee, Aha, and Carter, 2010). Munoz-Avila et. al. use a case based reasoning approach towards managing goals. Given a state, they construct a mapping from goals to expectations, along with a mapping from expectation violations to new goals. These mappings are then used for the dynamic modification and maintenance of a collection of goals.

Further work includes (Jaidee, Munoz-Avila, Aha, 2011). Here, a GDA approach is used in conjunction with case-based reasoning and the added element of reinforcement learning for acquiring domain knowledge, encoded as cases, which would otherwise have to be hand crafted for the domain. Using case-based reasoning, the approach of Jaidee et. al. creates distributions of (state, action) pairs over expected states, and (goal, discrepancy) pairs over discrepancy-resolution goals. Reinforcement learning is used to learn goals' expected values, in terms of utility.

Future Research

In MIDCA_1.0, each plan is executed to completion before a new goal is generated. This is not ideal, as situations may arise in the middle of the execution of a plan for which a new goal should be generated. There are difficulties, however, in simply generating a goal for every new world state. To address this issue, we have begun implementing MIDCA_1.1, our first attempt to improve on baseline performance using reasoning. This system adds D-track approaches to anomaly detection and categorization to the existing framework. We apply a statistical metric called the A-distance to streams of predicate counts in the perceptual input. This enables MIDCA to detect regions whose distributions of predicates differ from previously observed input (Cox, Oates, Paisner, & Perlis, 2012). These areas are those where change occurs and potential problems exist. The presence of an anomaly detection schema will allow MIDCA_1.1 to interrupt a plan in progress if it detects that a significant change in the world state has occurred, possibly preventing wasteful or damaging pursuit of goals that are no longer relevant.

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 (Fritzke, 1995) as the D-track processes perceptual input. This process leverages the results of analysis with A-distance to generate anomaly archetypes, each of which represents the typical member of a set of similar anomalies the system has encountered. When a new state is tagged as anomalous by A-distance, it is associated with one of these groups. In MIDCA_1.1, this categorization can provide evidence that could corroborate or refute the assignment of the current world state to a scenario by Tilde. More generally, knowing the type of an anomaly allows a system to prioritize explanations and goals that have worked with the same anomaly type in the past.

The K-track NAG procedure is still under development, but we plan to implement a process similar to that used by the Meta-AQUA system (Cox & Ram, 1999) and other case-based interpretation systems. In Meta-AQUA frame-based concepts in the semantic ontology provide constraints on expected attributes of observed input and on expected results of planned actions. When the system encounters states or actions that diverge from these expectations, an anomaly occurs. Meta-AQUA then retrieves an explanation-pattern that links the observed anomaly to the reasons and causal relationships associated with the anomaly. A response is then generated from salient antecedents of the instantiated explanation pattern (see Cox, 2007 for details).

One obvious approach to the interaction between the D-track and K-track would be simply to call K-track algorithms only on regions detected by D-track anomaly detection. This would be more efficient because the overhead for the K-track method is greater than that of the A-distance and growing neural gas methods. But more sophisticated approaches exist. For instance the weight of one procedure over the other may be a function of features including resources available and factors such as urgency. Many other issues remain to be examined in detail. These include the decision between plan change and goal change and the allocation of responsibility for this decision between meta-level goal management and the Intend component.

A more complete approach would introduce goal management to this process. Using this approach, the goal generator would produce new goals whenever D-track or K-track anomaly detectors report a potential problem, then leave it up to goal management to determine if each goal should be kept. A goal management framework would include a list of goals, partially ordered according to precedence and precondition relations. This list of goals would be maintained such that new goals are inserted into the correct locations, or discarded if they are deemed unhelpful, and goals which are no longer applicable would be discarded as well. A goal may cease to be applicable if it was a sub-goal of a goal which has already been achieved, if it was a sub-goal of a goal which itself has been discarded, or if it had a time window which has expired or preconditions which can no longer be met.

Concluding Remarks

In this paper we have presented an architecture, MIDCA, whose purpose is to increase an agents capacity to deal with novel and open ended situations by structuring that agent's behavior around dynamically generated goals. We covered a basic implementation conforming to the MIDCA architecture, MIDCA_1.0. Within MIDCA_1.0 goal generation is achieved using a comparatively simple approach which generates goals by considering world states in isolation. In future implementations of the MIDCA architecture, performance will be improved by introducing to goal generation the capacity to reason over causal relations in the environment, by introducing a goal management scheme, and by employing anomaly detection methods such as A-Distance to determine the appropriate times to make use of goals.

Acknowledgements

This material is based upon work supported by ONR Grants # N00014-12-1-0430 and # N00014-12-1-0172 and by ARO Grant # W911NF-12-1-0471.

References

Anderson, M. L., and Perlis, D. 2005. Logic, Self-Awareness and Self-Improvement: The Metacognitive Loop and the Problem of Brittleness. *Journal of Logic and Computation* 15(1).

Anderson, M. L.; Oates, T.; Chong, W.; and 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.

Anderson, J. R. 1993. *Rules of the mind*. Hillsdale, NJ: LEA.

Cohen, P. R., and Levesque, H. J. 1990. Intention is Choice with Commitment. *Artificial Intelligence* 42(2-3):213–261.

Cox, M. T., and Burstein, M. H. 2008. Case-Based Explanations and the Integrated Learning of Demonstrations. *Künstliche Intelligenz (Artificial Intelligence)* 22(2):35–38.

Cox, M. T., and Ram, A. 1999. Introspective Multistrategy Learning: On the Construction of Learning Strategies. *Artificial Intelligence* 12:1–55.

Cox, M. T., and Veloso, M. 1998. Goal Transformations in Continuous Planning. In DesJardins, M., ed., *Proceedings of the 1998 AAAI Fall Symposium on Distributed Continual Planning*, 23–30. Menlo Park, CA: AAAI Press / MIT Press.

Cox, M. T.; Oates, T.; Paisner, M.; and Perlis, D. 2012. Noting Anomalies in Streams of Symbolic Predicates Using A-distance. *Advances in Cognitive Systems* 2:167–184.

Cox, M. T.; Maynord, M.; Paisner, M.; Perlis, D.; and Oates, T. in press. The Integration of Cognitive and Metacognitive Processes with Data-driven and Knowledge-rich Structures. *Proceedings of the Annual Meeting of the International Association for Computing and Philosophy. IACAP-2013.* To appear.

Cox, M. T.; Oates, T.; and Perlis, D. 2011. Toward an Integrated Metacognitive Architecture. Technical Report FS-11-01, Menlo Park, CA. 74–81.

Cox, M. T. 2005. Metacognition in Computation: A Selected Research Review. *Artificial Intelligence* 169(2):104–141.

Cox, M. T. 2007. Perpetual Self-Aware Cognitive Agents. *AI Magazine* 28(1):32–45.

D., N.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, J.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *JAIR* 20:379404.

Forbus, K.; Klenk, M.; and Hinrichs, T. 2009. Companion Cognitive Systems: Design Goals and Lessons Learned so Far. *IEEE Intelligent Systems* 24:3646.

Fritzke, B. 1995. A Growing Neural Gas Network Learns Topologies. In Tesauro, G.; Touretzky, D.; and Leen, T., eds., *Advances in Neural Information Processing Systems 7*. Cambridge, MA: MIT Press.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. San Francisco, CA: Morgan Kaufmann.

H. Blockeel and L. De Raedt. 1997. Experiments with Top-Down Induction of Logical Decision Trees. Technical Report CW 247, Dept. of Computer Science, K.U.Leuven.

Jaidee, U.; Munoz-Avila, H.; and Aha, D. W. 2011. Integrated Learning for Goal-Driven Autonomy. *Proceedings of the Twenty-Second International Conference on Artificial Intelligence (IJCAI-11).*

Klenk, M.; Molineaux, M.; and Aha, D. 2013. Goal-Driven Autonomy for Responding to Unexpected Events in Strategy Simulations. *Computational Intelligence.* 29(2):187–206.

Krause, E.; Schermerhorn, P.; and Scheutz, M. 2012. Crossing Boundaries: Multi-Level Introspection in a Complex Robotic Architecture for Automatic Performance Improvements. In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence*. Palo Alto, CA: AAAI Press.

Laird, J. 2012. *The Soar Cognitive Architecture*. Cambridge, MA: MIT Press.

Langley, P., and Choi, D. 2006. A Unified Cognitive Architecture for Physical Agents. *AAAI-2006*.

Morbini, F., and Schubert, L. 2011. Metareasoning as an Integral Part of Commonsense and Autocognitive Reasoning. In Cox, M. T., and Raja, A., eds., *Metareasoning: Thinking about thinking*. Cambridge, MA: MIT Press. 267–282.

Munoz-Avila, H.; Jaidee, U.; Aha, D. W.; and Carter, E. 2010. Goal-Driven Autonomy with Case-Based Reasoning. *Proceedings of the 18th International Conference on Case Based Reasoning (ICCBR 2010)*.

Nau, D.; Muoz-Avila, H.; Cao, Y.; Lotem, A.; and Mitchell, S. 2001. Total-Order Planning with Partially Ordered Subtasks. In *IJCAI-2001*.

Perlis, D. 2011. There's No 'Me' in Meta - Or Is There? In Cox, M. T., and Raja, A., eds., *Metareasoning: Thinking about Thinking*. Cambridge, MA: MIT Press. 15–26.

Quinlan, J. R. 1990. Learning Logical Definitions from Relations. *Machine Learning* 5:239–266.

Quinlan, J. 1993. *C4.5: Programs for Machine Learning*. San Francisco, CA: Morgan Kaufmann.

Schmill, M.; Anderson, M.; Fults, S.; Josyula, D.; Oates, T.; Perlis, D.; Shahri, H.; Wilson, S.; and Wright, D. 2011. The Metacognitive Loop and Reasoning about Anomalies. In Cox, M. T., and Raja, A., eds., *Metareasoning: Thinking about Thinking*. Cambridge, MA: MIT Press. 183–198.

Shapiro, S. 2000. SNePS: A Logic for Natural Language Understanding and Commonsense Reasoning. In Iwanska, L., and Shapiro, S., eds., *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language*. Menlo Park, CA: AAAI Press/The MIT Press. 175–195.

Sloman, A. 2011. Varieties of Meta-Cognition in Natural and Artificial Systems. In Cox, M. T., and Raja, A., eds., *Metareasoning: Thinking about Thinking*. Cambridge, MA: MIT Press. 307–323.

Talamadupula, K.; Benton, J.; Schermerhorn, P.; Kambhampati, S.; and Scheutz, M. 2010. Integrating a Closed World Planner with an Open World Robot: A Case Study. In *Proceedings of AAAI*, volume 291, 599–600. Palo Alto, CA: AAAI Press.