**Anomaly Detection and Analysis Using Growing Neural Gas**

By Matt Paisner

**Abstract**

Metacognitive architectures provide one solution to the problem of brittleness for agents operating in complex, changing environments. The Metacognitive Loop (MCL), in which a system notes an anomaly, assesses the problem and guides a solution, is one form of such an architecture. This paper extends prior work on implementing the note phase of MCL in symbolic planning domains, using a growing neural gas algorithm to construct a network which represents various normal and anomalous states. Testing shows that this technique allows for improved detection of anomalies in the note phase as well as categorization of anomalies by severity. These results suggest that the use of GNG will provide a significant head start to more complex analysis of problems encountered by an agent by placing each new anomaly into a category based on previous experience.

**Introduction and Related Work**

Artificial intelligence agents guided by machine learning algorithms are often very successful at well-defined, consistent tasks, even when those tasks are very complex. However, for such agents to be effective in many real-world settings, they must be able to deal with situations that are more fluid. A pathfinding program might learn to navigate mazes better than a person ever could, but if it tries to apply that knowledge, unmodified, to the problem of climbing a wall, it will fail badly.

Of course, what the agent ought to do is to temporarily forget its maze navigation mastery and either start learning from scratch or apply some alternative algorithm that is more suited to its new task. To accomplish this, an agent must possess two distinct capabilities. First, it must realize that there is a problem. Ideally, it would see the (presumably unclimbable) maze walls receding, realize that the wall in front of it is too long to go around, and pause to assess its options. Failing this, it might walk along the wall for a ways in each direction before realizing it has a problem. Either way, a necessary first step in affecting a useful strategy change is to realize that some change is required.

Once the agent has decided to consider a change in approach, the second capability it must possess is some capacity for choosing among possible new strategies based on the changed situation. It may have a set of learning algorithms it can use, for example, and some heuristic for choosing between them. Or, it may simply choose algorithms at random or in some predetermined order until it finds one that works. However, the former solution leaves the agent very dependent on the foresight of its human programmers (who presumably designed the heuristic), while the latter may result in many abortive efforts for each success. An alternative, as articulated by Anderson and Perlis (2005), is for an agent to explicitly monitor and interpret both the world around it and its own cognitive architecture, culminating in a more informed decision about how to modify its strategy.

The structure of such a system is defined by adding a third step in between the observation of a problem and the determination of a response. Anderson and Perlis refer to this process as the metacognitive loop (MCL), which consists of three phases: *noting* an anomaly, *assessing* it to determine the nature of the problem, and *guiding* a solution. MCL is intended to work with an existing agent, using information about both the environment and the agent’s strategies and capabilities to advise it on the right course of action when its usual approach is ineffective.

Several different approaches to the concept of MCL have been taken since. Anderson et al. (2006) and Josyula et al. (2007) use the concepts of self-monitoring and assessment of failures to design domain-specific metacognitive systems in Q-learning and natural language dialog systems, respectively. Both demonstrate improved performance in settings where the environment is perturbed and the agent must adjust. A domain-independent version of MCL based on a Bayesian net was also developed (Schmill et al., 2011) and used to guide a simulated mars rover, as well as in other tasks.

Recently, work has begun on a version of MCL that works in symbolic domains, such as deterministic classical planning domains like blocks world (in which agents arrange blocks in stacks) and logistics (a package delivery scenario). The advantage of these domains is that they are readily accessible to logic- and explanation-based approaches to assessing anomalies, given the logical formulation of state representations. The challenge is to find a way to apply the many effective machine learning and statistical analysis tools that are designed to operate on vectors of real-valued time-series data.

One such approach (Cox et al., 2012) utilizes a novel state representation to apply A-distance, a statistical distance metric often used to detect anomalies in real-valued data streams (see Kifer et al., 2004), to the two symbolic planning domains mentioned above. A-distance has been used previously in domain adaptation work (Blitzer et al., 2006; Blitzer et al., 2007; Dredze et al., 2010) This approach was quite successful as an implementation of MCL’s note phase, but left some questions unanswered. First, A-distance requires the choice of a threshold value, epsilon, to determine how anomalous a state must be to be called an anomaly. One approach (used by Cox et al.) is to try several different values. However, in a real-time anomaly detection situation this may not always be practical, though it is reasonable to suggest that this choice might itself be amenable to metacognitive analysis. Additionally, while the adaptation of A-distance to these types of domains is creative, the analysis of results to this point has been fairly simplistic, leaving room both for improved performance in anomaly detection and the use of some of the information gained in the assessment phase of the MCL loop.

This paper aims to begin the work of filling that space. Previous work with A-distance has generated results which consist of several streams of values representing changes in predicate distributions as anomalous world states are encountered. The next sections will describe the use of these streams as inputs to a growing neural gas (GNG) net (Fritzke, 1995). This results in a net in which certain nodes correspond to anomalous data. This provides benefits both in terms of defining the anomalous without guesswork and in characterizing the type and severity of the anomalies that are detected, information which can be used to assess the problem and provide a solution.

**Domain**

Experiments were conducted in the logistics world, a symbolic planning domain. Logistics is a deterministic world in which states are defined by sets of statements in predicate logic. Actions are defined by a set of preconditions that must pertain for them to be performed and a set of results that become true upon completion. In the logistics world, there is no concept of time, and there is only a single actor. Therefore every action taken, if applicable to the current state, will be completed before another action is begun.

Objects in the logistics domain include:

* **Packages**, which can be moved from place to place
* **Cities**, each of which contains a **Post Office** and an **Airport**
* **Trucks**, each of which stays in one city and moves packages between its airport and post office
* **Planes**, which transport objects between airports in different cities

Predicates include:

* **At-Truck**, which gives a truck’s location
* **At-Plane**, which gives a plane’s location
* **At-Package**, which gives a package’s location when at an airport or post office
* **Inside-Truck** and **Inside-Plane**, which give a package’s location when it is being transported

Actions which operate on these predicate include:

* **Drive-Truck** and **Fly-Plane**
* **Load-Plane**, **Load-Truck**, **Unload-Plane** and **Unload-Truck**

An action is invoked on a specific set of objects, so to get a package P from city A’s airport to city B’s using plane PL, the action sequence:

**Load-Plane(PL, P, Airport\_City1)**

**Fly-Plane(PL, Airport\_City1, Airport\_City2)**

**Unload-Plane(PL, P, Airport\_City2)**

Would be called, with the result that both the plane and package would be moved from the airport at city1 to the airport at city2. These results would be added to the state:

State{

.

.

.

**At-Airplane(PL, Airport\_City2) &**

**At-Package(P, Airport\_City2) &**

**.**

**.**

**.**

}

**Planning**

A goal in the logistics domain is a statement that is desired to be true or false, given in the same predicate logic that defines states. A plan is defined as an ordered list of actions designed to achieve one or more goals. For example, given a state such that the following predicates are true:

**At-Airplane(PL, Airport\_City1)**

**At-Package(P, Airport\_City1)**

The same list of actions given as an example above:

**Load-Plane(PL, P, Airport\_City1)**

**Fly-Plane(PL, Airport\_City1, Airport\_City2)**

**Unload-Plane(PL, P, Airport\_City2)**

Would constitute a viable plan to achieve the goal

**At-Package(P, Airport\_City2)**

Planning for these experiments was conducted using A-star search. The search space was the set of possible states in a logistics world with the given start state, and an efficient heuristic function was constructed using domain knowledge to optimize the search. Additionally, domain knowledge was used to narrow down the list of potential actions at each node, reducing the branching factor substantially.

**Alternate State/Plan Representations**

Consider the following visualization of a state in the logistics domain:



The predicate representation of this state would be:

State{

**At-Truck(TruckC, Airport\_A)**

**At-Truck(TruckB, Airport\_B)**

**At-Plane(PlaneA, Airport\_A)**

**At-Plane(PlaneB, Airport\_B)**

**At-Package(ObjectC, PostOffice\_A)**

**Inside-Truck(ObjectA, TruckA)**

**Inside-Truck(ObjectB, TruckB)**

}

Now, we write the same state in another way:

**[2 2 1 2 0]**

This vector represents the number of times each predicate occurs in the state representation – At-Truck twice, At-Plane twice, At-Package once, Inside-Truck twice and Inside-Plane zero times. Using this representation loses information, since many different states could be represented by the same predicate count vector. However, it accomplishes two goals which will be essential in the following analysis:

1. Where we had a logical statement, we now have a numeric representation.
2. In part due to the abandonment of some information, we have greatly simplified the representation of this state.

If we have a plan which originates in this start state, we can represent it by using the sequence of states that result when it is followed. A certain plan, then, might be represented as:

**[2 2 1 2 0] #(unload-truck)**

**[2 2 2 1 0] #(load-plane)**

**[2 2 1 1 1] #(fly-plane)**

**[2 2 1 1 1] #(unload-plane)**

**[2 2 2 1 0] #(end)**

This sequence of vectors represents, among others, a plan to transport Object B to the airport in City A using Plane B, with the actions shown in simplified form on the right.

We will see in the next section how this type of plan representation is used.

**A-Distance**

The A-distance algorithm (Kifer et al., 2004) is a technique for anomaly detection in data streams. It uses a simple statistical metric to compare two windows of data. Both windows contain a constant number n of data values at all times. The first window is stationed at the beginning of the stream, which is assumed to be normal data. The second window moves with the stream, always containing the n newest available data values. When the distance between the two windows exceeds a given threshold, an anomaly has been detected.

As discussed previously, A-distance has been used frequently for anomaly detection in real-valued domains. [Cox et al., 2012] suggest a method for extending it to symbolic domains as well. Using the vector plan representation described above, sequences of states in the logistics domain can be analyzed and anomalies detected. Recall the plan representation:

**[2 2 1 2 0]**

**[2 2 2 1 0]**

**[2 2 1 1 1]**

**[2 2 1 1 1]**

**[2 2 2 1 0]**

Now, instead of viewing each state as a row vector, use the values for each predicate over the all time states as a column vector:

**[2] [2] [1] [2] [0]**

**[2] [2] [2] [1] [0]**

**[2] [2] [1] [1] [1]**

**[2] [2] [1] [1] [1]**

**[2] [2] [2] [1] [0]**

Each of these five column vectors can now be separately fed into A-distance as a data stream. So, at each time step, there will be five A-distance values returned. Each of these values represents, for a single predicate, the degree to which its recent counts differ from expected values for a standard distribution, which is characterized by the observed distribution in the stationary first window.

Based on experimentation by Cox et al., this method proves more effective if the raw predicate counts are replaced by the differences between counts at adjacent time steps. So, the five vectors above are transformed into:

**[0] [0] [1] [-1] [0]**

**[0] [0] [-1] [0] [1]**

**[0] [0] [0] [0] [0]**

**[0] [0] [1] [0] [-1]**

These, then, are the types of inputs to A-distance that will be discussed in the following sections.

**Growing Neural Gas**

The basic neural gas algorithm uses one-shot single-winner competitive learning, in which the winner is picked based on Euclidean distance from the input vector. The learning rule for the winner is:

*dw = n(a - w)*

Where a is the input vector, n is the learning rate and w is the node’s weight vector. Neighbors of the winning node also learn, but with a smaller value of n.

Growing neural gas (GNG) adds the dynamic of allowing nodes to be created and eliminated to better fit the structure of the input data. In Fritzke’s classic model, an error value is stored for each node, based on the total squared distance from that node to all inputs for which it was the winner. New nodes are then added near the nodes with the highest error rates, at fixed intervals. Each new node has an edge connecting to the high error node near to which it was placed.

This dynamic is intended to induce higher sensitivity in areas where there is a greater input density. However, it is less effective at accounting for small anomalous data clusters that are disconnected from the larger mass of normal data, especially when those clusters are present only during a short interval and then cease. One reason why this pattern is less easily captured is that the outlying data will occur much less frequently, so that it is difficult for a node to move all the way out to the location of the actual cluster without being pulled back towards the center by the more frequent input patterns. Also, even if a node does reach the correct location, if the outlying input patterns stop being expressed if will drift away due to the pull of its neighbors, and the network will “forget” about the anomaly rather quickly.

The solution to this issue employed in the present experiments is two-fold. First, the method of adding nodes has been changed. For the purposes of anomaly detection, it is actually counterproductive to add more nodes in a small area, as the old algorithm tends to do. Therefore, the version of GNG used herein adds nodes based on distance – when an input pattern is not near any currently existing node, a new node is created at the point specified by the input. This allows brief or unusually anomalous events to be detected. Also, the neighbor learning rate has been reduced to near zero, preventing the pull of outlying nodes back to the center. This prevents GNG from effectively performing its function as a topological map, but makes it more useful as a flexible clustering method, our present purpose.

**A-Distance -> GNG**

A-distance results are used with GNG as follows:

1. Plan vectors sets, as described above, are fed into several A-distance streams running in parallel. Each stream represents values for one predicate in a state representation, and each value in a given stream represents one time step in the plan.
2. At each step, the outputs of the A-distance streams are normalized to provide a distance metric with a value between 0 and 1. These values are organized into a vector, which represents the “anomalousness” of the world at that time step. This vector, which has dimensionality equal to the number of predicates/A-distance streams, is sent to GNG as an input.
3. Over time, the GNG network develops a set of nodes which correspond to normal and anomalous world states. The degree to which each state is anomalous is determined by its node’s distance from the origin, and the type of any new input state can be determined by association with an existing node.
4. To provide a yes/no anomaly detection result, this algorithm is run on a set of known normal data. The requirement for such a guaranteed set is not a significant issue, since A-distance already makes this assumption for the calibration of the initial window. The minimum distance from the origin for an anomalous node in testing is defined to be the maximum distance of any node generated by normal data, subject to a minimum number of updates that were performed on it. This last restriction is designed to avoid outliers at the end of the data set that generate new nodes which do not have time to be influenced by learning from more standard inputs.

The diagram below shows a single step of this process:

**Experimental Setup**

All experimental data were drawn from randomly generated world states and goals in the logistics domain described above. Inputs to A-distance were the vector representations of plans that had been created to achieve the given goals. To create anomalous data, planning was done with one operator removed, and only world state/goal combinations for which a plan could be successfully generated were used.

There were three data sets used for testing. The first, the “airplane-anomaly” set, consisted of 500 normal plans and 100 plans with the “unload-airplane” operator removed. The second, designated “truck-anomaly”, also had 500 normal plans and 100 with the “unload-truck” operator removed. The third, “two-anomaly”, had 500 normal plans and two separate anomalous sections of 100 plans, the first of which had the “unload-airplane” operator removed, and the second “unload-truck.”

The first step of testing was designed to determine the effectiveness of the combined GNG/A-distance process as an anomaly detection tool, or in MCL’s “note” phase. Trials were conducted using variable concentrations of anomalous plans in the anomalous sections, simulating anomalies of different intensities. For example, at intensity 0.2, only 20% of the 100 plans in each anomalous section were actually anomalous, signifying a very faint anomaly. For comparison, we ran trials on each data set using A-distance alone as well as the combined process. Since A-distance requires an input parameter epsilon to determine its sensitivity level, we calculated A-distance results using both a selected range of epsilon values and by using a standard statistical technique to calculate appropriate epsilons for each predicate stream. 50 trials were run for each data set and at each intensity level from 0 to 1 by units of 0.1, so testing was done on a total of 1650 individual sets of predicate streams (50 tests \* 3 data sets \* 11 intensity levels). Each stream consisted of an average of \_\_\_\_\_\_ time steps worth of data. Results from these tests were compared using several statistical techniques described below.

In the second step of testing, we analyzed the usefulness of the GNG network as a tool for anomaly analysis – the “assess” phase of MCL. One goal of applying GNG to A-distance data was to generate a data-driven method for clustering anomalies into types. To test this, we ran additional experiments on each data set, and examined which nodes were being assigned to the anomalous data. In other words, at each time step during an anomaly, the closest GNG node to the incoming data point (the “winner” node) was recorded. The results of these trials are also described below.

**Results**

For each method, reported anomalies at each step were recorded. Then reported values were compared to expected values, where the expected value at a time step was ‘anomalous’ iff at least 50% of the values within the sliding window at that step were in the anomalous section of the data and the intensity of the anomaly was non-zero. Values for precision, recall, and F1 were then calculated for each method. Detailed results are shown below. For clarity, some charts do not show all epsilon values tested. All values from 0.1 to 0.6 at intervals of 0.05 were tested.

Overall, F1 for the GNG method was 0.396, equivalent to the best possible choice of threshold (epsilon) for A-distance alone and far better than the average performance over all epsilon values. GNG performed slightly worse with extremely low anomaly intensities. However, the low threshold values that outperformed GNG on these intensities did so by permitting a very high false positive rate, which may not be a good tradeoff for the return of identifying a larger fraction of relatively insignificant anomalies. In general, what the graphs show is that the GNG method maximized the potential of A-distance, picking a middle ground of sensitivity that allowed superior overall performance without leaning too far towards either sensitivity (recall) or confidence (precision).

Besides improving anomaly detection, the second purpose of this method was to provide useful information as the MCL cycle moves into the assessment phase. GNG’s contribution to this goal was to provide groupings of anomalies by type and severity so that a newly noted anomaly could be quickly categorized, allowing a more robust assessment procedure to narrow down its list of options. More work still needs to be done on this idea, but early results are promising. The figure below graphs assignments of world states to nodes during the anomalous segment of a test run. The nodes are ordered by distance from the origin, which is a simple proxy for how anomalous they are. Node 7 is the most anomalous.

As the graph shows, node 7 is consistently mapped to when the window (width = 100) is completely filled with anomalous data. As the window first enters the anomalous section, it hits nodes that are progressively more anomalous, gets to 7, then hits the same nodes in reverse order as it exits the anomaly. In other words, the different nodes correspond to different degrees of anomaly, and can be used to distinguish between anomalies at least in that sense. More work is needed to determine how effectively GNG can distinguish truly different types as well as severities of anomalies, but even the limited ability demonstrated here would be useful. Imagine, for example, an agent trying to decide whether all the planes in its fleet were damaged or only a few of them. This is the difference between node 7 and node 4 above, and it is a significant one in terms of deciding on an appropriate course of action.

**Conclusion and Future Work**

The combination of GNG with previously established anomaly detection techniques using A-distance has produced some interesting successes. The problem of choosing an appropriate cutoff between the normal and the anomalous seems to be largely solved by the technique of generating a normal network to use as a baseline, which appears to naturally split the difference between precision and recall. Additionally, the nodes that correspond to anomalous data appear to effectively differentiate, at least in some cases, between anomalies of different severities. This information is clearly of value to an agent in assessing its next move.

A good deal of work still remains. An obvious next step would be to introduce several different types of anomalies to see if GNG could correctly classify them, as well as determining accuracy in this more complex case. In addition, testing based on a series of planning events in a persistent world rather than a sequence of randomly generated ones would be more realistic and interesting.

On a more macro level, the next step is to use the information GNG provides as a tool for assessment. For example, a case-based reasoning agent might be able to discover that an anomaly that corresponds to node x means that all airplanes are broken and no one can fly. Therefore he does not need to drive to the airport to pick up his girlfriend and can go play basketball. Node y, on the other hand, means that only some airplanes are grounded, and he should probably check her flight status online before he makes any decisions. (The observant reader will have noted that nodes x and y in this example correspond directly to nodes 7 and 4 from the end of the results section.)

Even more generally, the utility of this method for more than anomaly detection has yet to be determined. The examples used, while illustrative, are relatively simple and do not reflect the diversity of experiences that an actual metacognitive agent could be expected to deal with. The A-distance/GNG method was never intended to be able to handle that diversity on its own, but the degree to which it can contribute useful information to that process in a more challenging domain remains to be seen.

References

Michael L. Anderson, Tim Oates, Waiyian Chong and Don Perlis. *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. 2006.

Michael L. Anderson and Donald R. Perlis . *Logic, self-awareness and self-improvement: The metacognitive loop and the problem of brittleness*. Journal of Logic and Computation. . 15. 1. 2005.

John Blitzer, Ryan McDonald, and Fernando Pereira. *Domain adaptation with structural correspondence learning*. In EMNLP. 2006.

John Blitzer, Mark Dredze, and Fernando Pereira. *Biographies, Bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification*. In Association for Computational Linguistics (ACL). 2007.

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

Darsana Josyula, Scott Fults, Michael L. Anderson, Shomir Wilson, and Don Perlis. *Application of MCL in a dialog agent.* In Papers from the Third Language and Technology Conference, 2007.

Dredze, M., Oates, T., Piatko, C. *We're not in Kansas anymore: detecting domain changes in streams*. In EMNLP '10 Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, 585-595. 2010.

B Fritzke. *A Growing Neural Gas Network Learns Topologies*. In Advances in Neural Information Processing Systems 7,. MIT Press, Cambridge MA, 1995.

Kifer, D., Ben David, S., & Gehrke, J. *Detecting change in data streams*. In Proceedings of the Thirtieth Very Large Databases Conference, 80-191 (2004)

Michael L. Anderson and Donald R. Perlis. *Logic, self-awareness and self-improvement: The metacognitive loop and the problem of brittleness*. Journal of Logic and Computation. 15. 1. 2005.

M. D. Schmill, M. L. Anderson, S. Fults, D. Josyula, T. Oates, D. Perlis, H. Haidarian, S. Wilson, D. Wright. *The metacognitive loop and reasoning about anomalies*. In M. T. Cox & A. Raja (Eds.), Metareasoning: Thinking about thinking (pp. 183-198). Cambridge, MA: MIT Press. 2011.

**Appendix: A-distance**

Note: the following is taken from a tech report that is currently being written. It is a description of the way in which A-distance values are calculated between two windows of data.

The A-distance detects differences between two arbitrary probability distributions by dividing the range of a random variable into a set of (possibly overlapping) intervals, and then measuring changes in the probability that a value drawn for that variable falls into any one of the intervals. If such a change is large, a change in the underlying distribution is declared. Let be a set of real intervals and let be one such interval. For that interval, P(A) is the probability that a value drawn from some unknown distribution falls in the interval. The A-distance between P and P′, i.e., the difference between two distributions over the intervals, is defined in equation 1.

(*P,P′*) = |*P(A) – P’(A)*| (1)

Two distributions are said to be different when, for a user-specified threshold ε, (*P,P′*) > ε