Abduction Made Easy:
A Modified Bayesian Network Approach

Tim Menzies, Jason Gookins

December 13, 2008

1 Abstract

Abductive reasoning is a difficult problem. NP-hard, in fact. As opposed to deduction and induction, which produce a single, concrete conclusion based either on direct proof or direct observation, respectively, abduction produces many possible conclusions, each of fluctuating merit depending on their applicability to various situations. This difficulty could be alleviated by mapping abduction onto a descriptive model and then using that model to derive an approximation of abductive reasoning that minimizes possible conclusions, and thus uncertainty.

We have developed a Bayesian network evaluation tool called Belief that models a system upon which the main functions of abductive reasoning can be mapped. This paper first presents a logical description of the model and describes some of the technical details of Belief. The report then describes abductive reasoning via the mapping of its main components onto the features contained in the Belief program. It then lists any further features still required to implement a fully-functional approximation of abductive reasoning. Finally, the paper briefly outlines some possible future strategies, both technical and theoretical, for solving abduction by building upon the concepts contained within the Belief program.

2 Introduction & Background

Abductive reasoning has been a complex problem in the field of logic for millenia. First was Aristotles epagoge: A syllogism with two premises, a major and a minor, wherein the major premise is proven to be true, but the minor premise which is, in fact, required to tie the entire proof together is only supposed to be true. Charles Peirce adapted this concept to modern logic in the late 19th century and was the first to coin the phrase abduction.

Bayesian networks, on the other hand, are a relatively recent concoction. Althrough based on and named after the work of the mathematician Thomas Bayes in the late 18th century, the networks themselves were not conceived in their entirety until being outlined by Pearl in 1985. Conceived as graphical models for deriving explanations based on probabilistic-based assumptions, they rely heavily on the concepts in Bayes work.

Interestingly, the classical examples of both abductive reasoning and Bayesian networks are one and the same. Given a closed system comprised of a sprinkler, an area of grass, and a rain storm, one can use a Bayesian network to compute the answers to questions such as What is the probability that it has just rained, given that the grass is currently wet? The corresponding answer comes from abductive reasoning: If the grass is wet, the most probable explanation is that it has just rained. Thus it can be seen that Bayesian networks and abduction contain many apparent similarities in both form and function.

XYPIC FIGURE OF SPRINKLER/RAIN/GRASS EXAMPLE BAYES NET HERE

Previously we have proposed an anomaly approach to dealing with uncertainty in software engineering planning. In that approach, rather than making dubious assertions about uncertain values, we proposed

a logging scheme where "normal" was learned from experience and "abnormal" was detected when new data was significantly different to the old.

On reflection, we realized that this scheme was actually a special case of a general abductive reasoning approach. We show here that with minimal changes to our anomaly detection scheme we can build an abductive inference engine suitable for numerous software engineering tasks, such as:

- Prediction
- Planning
- Classification
- Monitoring
- Explanation
- Tutoring
- Diagnosis & Probing
- Validation
- Verification

as well as just anomaly detection. We show all this with a case study of managing IV&V projects.

3 Belief Network

A Belief network `G` can be modelled as follows:

```
G = <V+, E*>
V = name description operation priority
    status distribution(s) cost isLeaf
Operation = and | or | not | null
Status = locked | free
E = from to
From = V
To = V
Distribution = minimum maximum buckets+
Bucket = value height
Distributions = goal given actual
```

A network g consists of vertices V, also known as nodes, and edges E. These edges consist of ends from and to which are themselves nodes. With the added precept of forbidding cycles, this creates a directed acyclic graph. When the tool is run, paths are taken through this graph, evaluating nodes as they are hit, and accruing mathematical equations that are used to derive end results.

Nodes within the graph consist of a set of descriptors: a name; an English description; an operation to be applied to routes taken through the node; a priority, which is used to determine the order of processing during a run of the network; a current status, either locked or free, which describes whether or not the nodes distribution, if it has one, is mutable; a possible set of distributions, which are histographic in nature and represent the domain knowledge associated with the particular node; and a boolean value describing whether or not it is located on an outer edge of the graph.

There are two components of nodes that are particularly critical. The first is the Distribution component, which is a histographic distribution describing the expected characteristics of the sub-system represented by a particular node. Each distribution is composed of a range, which is bounded by a minimum and a maximum, and a series of buckets that each have a value and a height. A distribution comprised of input domain knowledge is of the Given type; a distribution comprised of desired output from the model is of the Goal type; and a distribution comprised of data output by a traversal of the graph is of the Actual type.

The second important item is the Operation component, which consists of possible states and, or, not, or null, which correspond to the fuzzy logic operators and an additional null state. These operators are actuated as paths taken through the network propagate through the nodes that possess them. A node with an and operator applies the corresponding fuzzy logic and to the distributions of its child nodes; a node with an or operator applies the corresponding fuzzy logic or to the distributions of its child nodes; a node with a not operator applies the corresponding fuzzy logic not to the distribution of its child node; and a node with the operator null allows the distributions of any

child nodes to pass through it untouched.

4 Abductive Reasoning

This section explains and maps the main tenets of abductive reasoning onto the Belief model, from prediction through to validation.

4.1 Prediction

Prediction is the process of determining the possible outcome that can be derived from some set of input data. It is the core function of a Bayesian network. Given a set of input data about a system to be modeled and a set of goals to be reached, we evaluate the network and predict whether or not the goals can be attained with the input data.

This is handled within the Belief framework by propagating from the leaf nodes in to the root nodes, taking goal distributions we see along the way and aggregate distributions accrued during our journey, normalizing them, and comparing them.

```
fun delta (dist1, dist2)
  for (I in dist1)
  {
    tmp = dist1[i]  dist2[I]
    sum += tmp
    out[I] = tmp
  }

  for (I in out)
  {
    out[I] = out[I] / sum
  }

  return out
```

4.2 Classification

Classification is merely a unique prediction case wherein the output is the array of possible classifications.

Belief implements classification by a simple tagging system for grouping sub-classes into super-classes. A user can then find all nodes of a relevant type for further specification or for generalization.

4.3 Explanation

Explanation is essentially the reverse of prediction, and is more or less the core of abductive reasoning. Given a set of goals, explanation derives the configuration of the antecedents that most closely approximates the goals.

In the Belief model, after each evaluation of a graph, the outcome is compared to the last, best outcome. If it is deemed more optimum, it becomes the new best. This is repeated until the current evaluation reaches its run limit and terminates.

It is within the explanation phase that the tool exhibits its actual AI elements. Leaf node distributions, if they are not found to be locked and immutable, are randomized and run through the model. If the newly randomized distribution is not deemed a new best, it is thrown out and replaced.

4.4 Diagnosis & Probing

Diagnosis in the case of the Belief model is essentially a refinement of explanation; namely, minimum explanation. Diagnosis sets about the task of both deriving the best explanation and ensuring that the explanation chosen results in the least amount of change in the model.

Belief accomplishes this with a feature that works as follows:

1. Evaluate the network fully
2. Order the leaf nodes by their amount of alteration
3. Lock the distribution of the node with the highest degree of change
4. Re-evaluate the network with this new constraint
5. Repeat until the amount of change minimization affected falls below a certain threshold, say 5%

This method quickly and concisely pulls out the problem children in the network; i.e. those leaf nodes that contribute the most to change within the model.

Probing can be simply defined as being analagous to minimal explanation. Repeated forays are made into the network per the minimal explanation feature in an attempt to minimize change and maximize closeness to goals.

4.5 Tutoring

Suppose that a user evaluates a large network that contains domain knowledge from many disparate individuals. The user may not reach an optimal outcome due to an information overload. In other words, the user may not be able to understand or personally verify all of the information contained in the network. A tutoring system would solve this problem by displaying only that information which a particular user understands.

Belief implements this through a user profile system. Each network can be given a set of user profiles, each of which contains a specific subset of information about the modeled system that is essentially a targetted knowledge base for each user and/or each relevant situation within the system. The given and goal distributions that are assigned to each node are stored exclusively within each user profile, so that ten different users can feasibly have ten different views of the way in which a particular aspect of the modelled system actually works.

4.6 Validation

Validation checks a model against the semantic criteria imposed upon the model. In the case of the Belief model, the question is asked Can I achieve my goals with the given inputs? The model answers this in the easiest way possible.

After a Belief network has been fully evaluated, every node possessing a goal distribution lights up with a color ranging from red to green. The closer to red the node is, the farther away from the goal we are, and the closer to green the node is, the closer to the goal we are.

SCREENSHOT OF RED/GREEN NODES HERE

4.7 Verification

Verification is the testing of the model against a set of syntactic criteria. Simply put, verification ensures that the model itself is not fundamentally broken, which would ruin the chance of any meaningful insights being gleaned from the data.

The most obvious verification method within the Belief tool is the computation of the transitive closure of the graph to maintain its acyclic nature. This is done with the following recursive algorithm:

```
fun createsCycle (childNode, parentNode)
  for (Node in parentNodes parents)
  {
    if (Node == childNode)
    {
      return true
    }
    else
    {
      createsCycle(childNode, Node)
    }
  }
```

Two nodes in a parent-child relationship are evaluated. A journey is made recursively up the graph from the parent node. If the algorithm reaches the child node, a cycle has been found, and the relationship is disallowed. This check is performed at edge creation and reversal, thus dynamically maintaining the acyclicality of the graph.

Another verification method involves ensuring that no node is allowed to possess an operator that would make no logical sense. Nodes are allowed to have and or or operators if and only if they have multiple children, and a node can have a not operator if and only if it has one child. These operator checks are also made dynamically during node editing and deletion to ensure the integrity of the model.

5 Unimplemented Features

This section details features of abductive reasoning that do not yet have an analagous feature within the Belief tool.

5.1 Planning

The planning component consists of the pursuit of the model configuration which best arrives at the goal state from the current state. Planning is basically a cost-benefit analysis for the explanation and diagnosis processes. After running the network to completion, a best case may be arrived at which almost perfectly matches the goals required, but will take $1 trillion, thousands of man hours, and possibly hundreds of lives to achieve. Planning will illuminate the fact that there is a deceivingly sub-optimum plan which does not perfectly match the goals, but would take only $100,000 and a month to complete.

The Belief model will approximate planning by implementing a cost attribute for contributory leaf nodes. Evaluations of the network will then steer for solutions that plan for both proximity to the goals and the eventual cost of that proximity.

5.2 Monitoring

As the planning stage arrives at plans based on their cost-benefit analysis, these plans may come to be obsolete or no longer feasible. Monitoring sets up a way to perform long-term comparison of plans. As available data changes, so do assumptions inserted into the network. A monitoring system checks old plans against new assumptions, and removes those that are no longer relevant.

The Belief model will implement a monitoring system with two features. The first is a simple component that saves evaluation results across multiple evaluations of a network. The user can then compare these various result sets to derive the most suitable outcome based on the current prevailing knowledge of the system.

The second feature is a way to determine the worst-case scenario in a given system. Just as the best operator drives the model to come as close as possible to the goals, the worst operator drives the model to keep as far as possible from complete abandonment of the goals. The worst operator derives the distributions which result in the greatest departure from goal attainment in order to highlight any elements that are particularly contributory so that they may be avoided.

6 Tool Support

7 Screenshots

8 Future Work

Belief currently uses a simple, homebrew algorithm for evaluating networks. It is possible that the tool's efficiency and accuracy could be greatly improved through the exploration and testing of other bayesian network evaluation algorithms.

From an aesthetic and usability standpoint, it has been found through limited user trials that the Belief tool may potentially present users with too much information at one time. We have attempted to alleviate this problem with our simplified, slide-out list structure, as well as the ability to automatically view highlighted nodes. A possible improvement could be a graphical feature that extends the user profile system. When viewing a user profile, only those nodes which had been provided distributions in that profile would be displayed. The tool would automatically travel through the network and compute the aggregate connections between those nodes, supplying special icons with which now hidden sections of the network could be re-expanded.

8.1 Conclusion

We have introduced the Belief system as a tool for approximating abductive reasoning using naive bayesian networks. We have described the model behind the tool and mapped the basic components of abduction onto it.