RC25589 (WAT1601-089) January 26, 2016
Mathematics

IBM Research Report

Recursion Graphs: Consistent Inference for Cyclic Noisy-Logical Graphs

David W. Buchanan
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598 USA

Research Division
Almaden – Austin – Beijing – Brazil – Cambridge – Dublin – Haifa – India – Kenya – Melbourne – T.J. Watson – Tokyo – Zurich

Recursion graphs: consistent inference for cyclic noisy-logical graphs

David W. Buchanan
IBM Research
dbuchan@us.ibm.com

Abstract

Directed, cyclic graphs occur in many applications of graphical models where feedback loops are not desired. For instance, consider a relational resource that potentially contains meaningful directed cycles, such as a knowledge base or social network. We may wish to convert such a resource into probabilistic graphical model. Many existing methods for performing such a conversion either lose potentially important information, or are not guaranteed to have a consistent joint distribution. Other solutions are not tractable for some applications. This paper introduces a new semantics, recursion graphs, that do not lose information, and have a consistent joint distribution, while they are tractable in more situations. In these graphs, nodes keep track of their "reasons," so that no node can be an indirect reason for itself. In addition to the main consistency result, some preliminary empirical results are shown. In these experiments, recursion graphs significantly outperform alternatives.

1 Introduction

Graphical models are widely used in artificial intelligence, in areas from causal inference to social network analysis. Consider a case in which we must infer the marginal probabilities of unknown variables given known values of some set of other variables ("inference"). The resources at hand include a set of relations between these variables (a "relational resource") that can be interpreted as a directed graphical model. Such resources might include a social network, a set of structured links between entities in a recommendation system, or a graph constructed from knowledge. A potential problem is that relational resource might not respect the formal requirements of the graphical model we wish to use. For instance, the resource might contain directed cycles, and the edges in the cycles might carry important information, but we might wish to use a Bayesian network or some other formalism that requires an acyclic graph. The simplest such case occurs when for some pair of nodes, a relation exists in each direction with a different strength.

An example graph that has this property is shown in Figure 1a: If Alice tells us that she saw someone wearing a raincoat (abbreviated as "A"), it is a reason to think, with strength 0.8, that someone is wearing a raincoat ("R"). Similarly, Bob's report ("B") can be a reason to think someone is carrying an umbrella ("U"). If someone is wearing a raincoat, this is a reason to think that people are carrying umbrellas, and vice versa. Consider for the sake of argument that the relation from raincoats to umbrellas is stronger (0.9) than the relation from umbrellas to raincoats (0.4). That is, if we know someone is wearing a raincoat, this is a strong reason to think that someone is carrying an umbrella. But if we know that someone is carrying an umbrella, this is a relatively weak reason to think that someone is wearing a raincoat. (This might occur if only stronger rain made some people put on raincoats). Finally, the fact that someone is wearing a raincoat is a reason to think that there will be coats on the hooks ("H"), with strength 0.8, and if someone is carrying an umbrella, this is a reason to think that there will be umbrellas in the umbrella stand ("S") with strength 0.8.

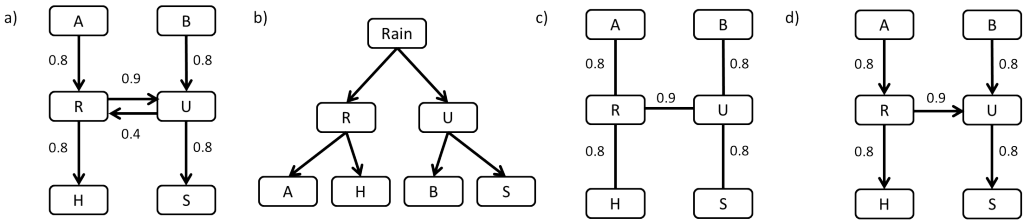

Figure 1: Example graphs illustrating the feedback problem and some existing solutions. A cyclic solution is shown in (a), and an approach that adds knowledge is shown in (b). Solutions that subtract knowledge include using an undirected model (c) and "combing" the graph (d).

It is intuitive to read certain properties from this graph: A provides strong support for H, and B provides strong support for S. Additionally, A provides good support for S, while B provides relatively weak but nonzero support for H. However, these intuitions do not work out mathematically unless we can find a coherent formalization of the graph as it is drawn. The directed cycle in the graph between R and U creates a formal problem. This is because, to paraphrase Friedman et al., [1], it is incoherent for a node to influence its own value. With the cycle present, U can provide support for R, and this very support can be used to support U in turn. Intuitively, it is obvious that such a feedback loop is not desired in this case: belief in raincoats should not reinforce itself. But it is not obvious how to formalize the graph so that such a feedback loop does not occur. We will call this the "feedback problem": Given a relational resource that potentially contains directed cycles, describe a procedure for building from the relational resource a graphical model over which we can perform probabilistic inference that captures the intuitions described above. We will call such a procedure a "semantics."

1.1 Existing solutions

One solution to the feedback problem is to use a semantics that constructs a fundamentally different graph. For instance, common sense tells us that rain is a common cause of both umbrellas and raincoats; we could construct an acyclic graph such that each was diagnostic of rain. (See Figure 1b) There exists some set of conditional probabilities for Figure 1b that could capture the conditions expressed above. Though such solutions exist in principle, it is not clear how to construct them in general, especially if knowledge is limited. This paper will take a different approach, tackling the practical problem directly: Given a potentially cyclic, noisy graph, and imperfect knowledge, we wish to directly make use of the graph without losing or adding important information. In other words, while restructuring the graph is allowed, we will assume we cannot add new knowledge to the graph.

Given such a constraint, another approach is to make a formal model that directly corresponds to the relational resource, then minimally modify the graph such that it is formally coherent. In other words, we can subtract information from the graph. For instance, we could construct a symmetric Markov random field in which the edge represents the compatibility between the values of the two nodes, but direction is not meaningful. Inference over such a graph tries to satisfy the constraint that nodes with strong compatibility take the same value. For every directed edge in Figure 1a, we can construct a undirected edge whose strength is a function of the edges between the two nodes. If there are multiple such edges, we combine them using some function (e.g., the maximum). See Figure 1c for an example. Note that no matter what combination function we use, we lose important properties: The graph becomes completely symmetric with respect to Alice and Bob. We lose the property that Alice tells us more about umbrellas in stands than Bob tells us about raincoats on hooks.

Another approach is to treat the network as directed, but orient edges such that cycles no longer exist. An example is shown in Figure 1d. In these graphs we must lose important properties as well: either Alice has no influence on umbrellas, or Bob has no influence on raincoats. This type of solution can be called a "combing" solution, because the cycles can be seen as "tangles" in the graph that are resolved by modifying the graph.

Yet another approach is to abandon the idea of a formally coherent representation, and use the graph as it is. That is, treat the graph as a Bayesian network and run inference despite the cycles. This is a version of an approach outlined by Heckerman et al. [2]. Some inference techniques, such as Markov chain Monte Carlo (MCMC) sampling, allow us to run inference despite not meeting the acyclicity condition of Bayesian networks. As Heckerman et al. [2] show, such inference can perform well in practice, for at least some applications. In the case of the raincoats example, we would accept that the values of R and U would be higher, in practice, than our intuition tells us. We would also accept that the graph was not guaranteed to specify a consistent joint distribution.

Finally, we could abandon using graphical models, and instead use a more expressive semantics such as a probabilistic logic. For instance, ProbLog [3] allows for directed cycles in sets of probabilistic implication rules. The rules specify a distribution over proofs; feedback is prevented by ensuring that each grounded proof is itself acyclic. While this can be a good solution in some contexts, the richer language of probabilistic logic comes with a higher computational cost that can be prohibitive in other contexts. In particular, grounding out each proof can cause an explosion in the number of variables to be considered. In the application described in section 3 we attempted to use probabilistic logics but ran out of memory on the majority of tasks. The solution we present in this paper represents a novel location on the spectrum of complexity/expressiveness tradeoffs: Rather than beginning with a fully expressive probabilistic logic, we begin with a simpler graphical model and add just the complexity needed to solve the feedback problem.

A note about feedback: This paper addresses problems that arise from applications (like the raincoats example) where feedback is not intuitively desired. In some applications, feedback is desired: For instance, obesity causes sedentary lifestyle, which causes obesity. Working from graphs where feedback is desired is another problem addressed by different research [4, 5].

Existing solutions to the feedback problem are unsatisfying on at least some applications: they are either impractical (e.g., changing the graph with new knowledge), formally incoherent (e.g., run inference anyway), too computationally expensive for some applications (e.g., probabilistic logic), or lose potentially important information (e.g., Markov random fields, combing). This paper proposes a novel solution to the feedback problem that has none of these problems. It allows us to directly construct from the relational resource a graphical model that has directed cycles, but is also probabilistically valid. We call it a "recursion graph." The core idea is that nodes keep track of the *reason* they are in a particular state, where reasons can be traced recursively through the graph. Keeping track of reasons allows us to prevent nodes from being reasons for themselves, without needing to eliminate cycles. The machinery for doing so will be presented in detail in the next section, which will also show that recursion graphs have a consistent joint distribution. The last section shows that recursion graphs outperform the other solutions just outlined, on a real world application.

2 Formal specification

2.1 Acyclic noisy-logical graphs

The formal specification of recursion graphs will be clearest if we begin by specifying acyclic noisy-logical graphs in the same way. The present formulation is a simplified version of the graphs presented by Yuille and Lu [6]. In an acyclic noisy-logical graph, nodes can be in one of two states, on and off, sometimes referred to as "active" and "inactive" respectively. Intuitively, a node is active if it has an active generative parent whose edge fires, and zero active inhibitory parents whose edges fire, and off otherwise. Formally, a graph consists of a set of nodes, for instance x, and a set of directed edges, for instance xy (an edge from x to y). We assume that there is at most one directed edge between each ordered pair of nodes. That is, there can be an edge from xy and a distinct edge yx, but not two distinct edges xy. The same effects as achieved by multiple edges between an ordered pair, can be achieved by combining the strengths of the edges. We use X to mean the random variable that x is active, (0 if it is active, 1 if it is not) and $P(X)$ to represent the probability it is active. Each directed edge xy has a strength s_{xy} between 0 and 1, and a valence, g_{xy} either generative ($g_{xy} = 1$) or inhibitory ($g_{xy} = 0$). An edge "fires" with probability equal to its strength. A node's parents R_x head the incoming edges. We denote all generative parents of x with R_x^g, and all inhibitory parents with R_x^h. Each node x has some "background" probability of being in the active state spontaneously. This is equivalent to having a generative edge from a special parent that

is always on ("the background") with strength equal to the background probability. Equations below will assume background is captured this way.

We will look at probabilistic inference as inferring marginal probabilities over "worlds." A world is an assignment of values to variables. The probability of a variable being in a particular state is the sum of the probabilities of all the worlds in which it is in that state. A semantics assigns a probability to each world. Looking at inference in this way allows us to apply techniques such as MCMC sampling. In a semantics with a consistent distribution, the probabilities of all worlds sum to one. It is well known that Bayesian networks, including acyclic noisy-logical graphs, are consistent. One way to see the consistency of noisy-logical graphs using a perspective we will call "edge worlds." An edge world $\mathbf{O}$ is an assignment of values to each edge variable O_{xy}. Each edge can be either *open*, $O_{xy} = 1$, or *closed*, $O_{xy} = 0$. The probability of an edge-world is given by:

$$P(\mathbf{O}) = \prod_{O_{xy} \in \mathbf{O}} \begin{cases} s_{xy}, & \text{if } O_{xy} = 1 \\ 1 - s_{xy}, & \text{if } O_{xy} = 0 \end{cases} \tag{1}$$

It is easy to see that the distribution over edge worlds is consistent. It is equivalent to the distribution over flipping a coin with bias s_{xy} for each edge xy, which is trivially consistent. Edge worlds can be used to determine activation worlds, which are used to determine the marginal probability of a node being in the active state. (The term "worlds" used unqualified refers to activation worlds.) A node is active in a given world if it has at least one active generative parent whose edge fires, and zero active inhibitory parents whose edges fire. Formally:

$$X = \left[\bigoplus_{y \in R_x^g} O_{yx} Y \right] \left[1 - \bigoplus_{z \in R_x^h} O_{zx} Z \right] \tag{2}$$

Where we introduce notation for the noisy-OR function: $\bigoplus x_1, ...x_j = 1 - \prod_{i=1:j} (1 - x_i)$. Given any edge world, the activation state of any given node can be determined using Equation 2. If the activation states of parents are needed, apply Equation 2 recursively. Note that multiple edge worlds can correspond to the same activation world. For instance, edges whose head nodes are not active are irrelevant with respect to activation worlds. Because edge worlds are consistent, and every edge world uniquely determines an activation world, acyclic noisy-logical graphs are consistent.

2.2 Cyclic noisy-logical graphs

Noisy-logical graphs where directed cycles are allowed, we will call cyclic noisy-logical graphs. The presence of directed cycles undermines the consistency arguments given above. In particular, these arguments rely on the guarantee that no node is its own ancestor. If such a guarantee were not present, then applying Equation 2 could lead to an infinite recursion loop. For instance, consider Figure 1a. Imagine a world in which only the edges ru and ur are open. The activation state of R is not clearly defined by Equation 2, because R's state depends on the state of U, which depends in turn on R. The procedure goes on forever. Spirtes [4] has described such graphs as "non-recursive."

2.3 Recursion graphs

To specify recursion graphs, we add to cyclic noisy-logical graphs in such a way that cycles are allowed, but consistency is restored. In a recursion graph, nodes are not just on or off: They are on or off for zero or more *reasons*, where a reason is a node in the graph. One node, the "background" is special in that it is always on for no reason. Other nodes are on if they have at least one generative reason and zero inhibitory reasons, and off otherwise. If the edge xy is generative ($g_{xy} = 1$), then x can be a generative reason for y, and if the edge is inhibitory ($g_{xy} = 0$), then x can be an inhibitory reason for y.

We trace reasons in order to enforce the key concept of *eligibility*: A node x can only be a reason for y if the edge xy exists, and x is eligible to be a reason for y. A node x is eligible to be a reason for another node y if and only if either x is the background, or x has at least one generative reason that is eligible to be a reason for y, and zero inhibitory reasons that are eligible to be a reason for y.

Crucially, nodes are not eligible to be a reason for themselves. This prevents feedback and leads to consistency, as we will see below.

Some properties of recursion graphs are worth noting at this stage. First, while node states can be categorized into "on" and "off," the nodes in the graph are not strictly speaking binary. Rather, if a node has k incoming edges, it has at least 2^k possible states. This is one state for every possible combination of reasons. Nodes with deeper ancestors have even more possible states. In practice, the extra states are not a significant problem because most of these states can be collapsed, especially with respect to another node. We define the variable $X[\mathbf{A}]$, to mean that X is eligible to be a reason for the set of nodes $\mathbf{A}$ (0 if eligibile, 1 if not). The boldface indicates a set. Adding to a set is represented with a comma: $X[\mathbf{A}, B]$ means that X is eligible to be a reason for the new set that includes everything in $\mathbf{A}$, and also B.

Recursion graphs, like other noisy-logical graphs, can be viewed in terms of edge worlds. (See equation 1). Recall that edge worlds are consistent; if we can define a procedure for generating an activation world for each edge world, then recursion graphs are consistent as well. In cyclic noisy-logical graphs, such a procedure can encounter a recursion loop. In a recursion graph, eligibility allows us to define a such a procedure that will not encounter a recursion loop, even in a graph with directed cycles. For each node, we can recursively trace reasons back to the background, abandoning paths that go through x. The recursion stops when we either reach the background (this means x is eligible) or have abandoned all paths (which means x is not eligible). Formally:

$$X[\mathbf{A}] = \left[\bigoplus_{y \in R_x^g} O_{yx} Y[\mathbf{A}, X]\right] \left[1 - \bigoplus_{z \in R_x^h} O_{zx} Z[\mathbf{A}, X]\right] \tag{3}$$

With the special cases that $X[\mathbf{A}] = 1$ for all $\mathbf{A}$ if X is the background, and if X is not the background, then $X[\mathbf{Y}] = 0$ if $X \in \mathbf{A}$. To determine whether a node is active, ask whether it is eligible to be a reason for the empty set, i.e. $X = X[]$. Then recursively apply Equation 3 as the activation states of parents are needed. We can prove that this process terminates without encountering an infinite recursion loop: At each step of recursion, the set of nodes for which we are checking eligibility strictly increases. Infinite recursion implies that this set can be arbitrarily large. But, if the set of reasons already contains X, then we return 0 and do not recurse further. Thus the size of the list of reasons is limited by the number of nodes in the graph. If the graph is finite, then the size of the list is bounded, but an infinite recursion loop would require it to be unbounded. Therefore the process is guaranteed to terminate for a finite graph. We have defined a procedure that uniquely determines an activation world for each edge world, and edge worlds are consistent. Therefore, recursion graphs are consistent.

2.4 Acyclic expansions of a recursion graph

Another approach to showing the consistency of recursion graphs would have been to expand the recursion graph into an acyclic noisy-logical Bayesian network. We sketch this process here primarily to illustrate the often prohibitive complexity of the expansion. Expansion is not part of any algroithm necessary for using recursion graphs. The overall apporach is to take each recursion graph node, which is set-valued, and create a binary-valued node for each set of descendants for which it might be eligible. One way to do this is to apply equation 3 to each node, and ensure the existence of a binary valued node for each list of reasons that is generated. The process continues recursively until every possible chain of reasons has been exhausted. While it may be true that such a graph exists for every recursion graph, it is easy to see that the complexity will increase steeply with the size and connectivity of the recursion graph. For a complete graph, the number of nodes in the expansion increases exponentially with the number of nodes in the recursion graph. We found the construction of such graphs to be intractable in practice on our application: Our hardware ran out of memory. We suspect that this is related to the reason why ProbLog ran out of memory on our application: It, like most probabilistic logics, begins by "grounding out" the logical rules into a graph that can be many times larger than the number of rules. When working with recursion graphs, there is no need to ground out or expand the graph: Using MCMC, we can directly sample the set-valued state of each node. By directly representing the chains of reasons, we avoid the need for an expansion or grounding out of the graph.

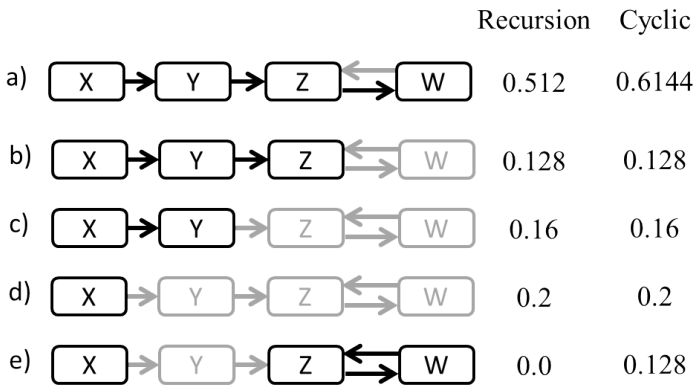

Figure 2: All non-zero likelihood worlds for a simplified example. Every edge has a strength of 0.8. Dark-bordered nodes and edges are active, while gray-bordered edges and nodes are inactive.

2.5 Working through an example

We will use a simplified example to work through how recursion graphs solve the feedback problem. See Figure 2. In this graph, X has background probability 1.0, and each edge has a strength of 0.8. All nodes other than X have zero background probability. If we are focused on Z, the path to and from W is irrelevant; it could be pruned away and should not affect inference. The path says that Z is a reason to think W, and W is a reason to think Z. Intuitively, the marginal probability of Z should simply be $0.8 \times 0.8 = 0.64$. We will call this property of graphs "the pruning property": Adding a cyclic path that leads only to and from a node but does not connect it to the background, should not affect its marginal probability. We will see that the pruning property holds in a recursion graphs semantics, but not in a cyclic semantics.

In practice, for efficient sampling and likelihood calculations, we will operate at the level of reason worlds, which are between edge and activation worlds. In a reason world, $\mathbf{N}$, an edge is active if its head is eligible and it is open, i.e. $N_{xy} = O_{xy}X[Y]$ We can generate reason worlds for a recursion and cyclic semantics as follows: Initialize all edges to random states. Determine activation of each node using equation 3. At each step of sampling, choose a node at random. For each incoming edge of this node, if the head is active, then re-sample whether the edge is active, changing the activation state of the node as appropriate. The likelihood of each reason world is the sum of the edge worlds that ground it. This can be found with the following product:

$$P(\mathbf{N}) = \prod_{N_{xy} \in \mathbf{N}} \begin{cases} 0, & \text{if } X[Y] = 0 \text{ and } N_{xy} = 1 \\ 1, & \text{if } X[Y] = 0 \text{ and } N_{xy} = 0 \\ s_{xy}, & \text{if } X[Y] = 1 \text{ and } N_{xy} = 1 \\ 1 - s_{xy}, & \text{if } X[Y] = 1 \text{ and } N_{xy} = 0 \end{cases} \tag{4}$$

That is, for every edge, if it is active, first check if its head is eligible to be a reason for its tail. If it is not eligible, return zero likelihood. If it is eligible, multiply by the edge's strength if it is active, and by one minus the edge's strength if it is not active. Using the above procedure and likelihood equation, it is straightforward to construct a sampling procedure (for instance, Gibbs or Metropolis-Hastings sampling). We can define a cyclic semantics using a similar procedure, but omitting reason-checking and asserting every node as eligible to be a reason for every other node (i.e., X[Y]=X for all X and Y). In a cyclic semantics, this procedure will not be sampling over a consistent joint distribution, but will nonetheless output a result.

The likelihoods of each non-zero likelihood activation world for each semantics are shown in Figure 2. This figure shows both activation and reason worlds, because for this example, each valid activation world has exactly one valid reason world that grounds it in the recursion semantics. Recall that for each node, its marginal probability is the sum of the likelihood of worlds in which it is active,

divided by the total likelihood of all worlds.[1] For the node Z, this number is 0.64 in recursion graphs, but 0.7074 in the cyclic semantics (violating the pruning property). This is largely because of world (e), which has zero likelihood in the recursive semantics but nonzero likelihood in the cyclic semantics. The recursion graph semantics assigns zero likelihood to world (e), because W and Z are not eligible to be reasons for each other.

3 Experimental test

Previous sections have shown that recursion graphs present a solution to the feedback problem, and have useful formal properties, such as consistency. These are the main points of the paper. It remains to be seen whether they yield practical results on a real-world data set. One such result is presented here. It is understood that more results are needed before a strong argument could be made about the superiority of recursion graphs in general. This paper makes a weaker argument: that recursion graphs show potential promise because of positive results on at least one application, in addition to their other useful properties.

3.1 Inputs

The algorithm, and all baseline algorithms, were applied to the same graphs from a real-world application. See [7] for details on the system that generated the graphs. Beginning with 1484 multiple-choice medical test-preparation questions, the system automatically identified a subset of 411 questions as involving medical diagnosis. The system takes a text question as input. It creates a node in a graphical model for each candidate answer, and an node for each sentence and key term in the question. Then a series of relation generators are run on the graph, potentially generating new nodes, and also potentially new relations between nodes. Most edges come from a relation generator that runs a probabilistic question answering system. The result is a directed graphical model, with a weight for each directed edge. Most graphs have hundreds of nodes and thousands of edges. Because many relation generators create edges in both directions, most nodes participate in some directed cycle. The problem that inference must address is: given the graph, infer a confidence for each candidate answer. For each graph, exactly one answer is correct.

3.2 Baselines

This section begins by describing exactly how recursion graphs were deployed, and then defines each baseline by its differences. Inference takes a graph as input, and begins by collapsing all redundant edges: All directed edges in the same direction and valence are collapsed by taking the maximum of all the weights. If multiple edges of different valences exist, the edge with the lower weight is deleted. Inference uses Metropolis-Hastings sampling as described in the previous section. We use equation 4 for the likelihoods. This procedure does not differ for any baseline, except for the differences described explicitly below.

The *cyclic* baseline was intended to represent the strategy under which we accept that the graphs have cycles, and run MCMC inference, despite the fact that it leads to feedback and to a possibly inconsistent joint distribution. This idea can be credited to Heckerman et al. [2]. This was run in an identical way to recursion graphs, except that we turned reason checking off. (Reasons were always eligible). Reason checking took up between 5 to 10 per cent of the processing time on most graphs. This serves as a rough estimate of the computational cost of using recursion graphs versus a cyclic semantics.

The *feedforward* baseline was intended to represent the "combing" strategy, in which we strategically orient edges to eliminate cycles. Under this baseline, we create the graph as above. Then for every pair ab that contains an edge ab, and an edge ba, we merge them in to one edge ab whose strength is the higher of the two strengths. After this, we "comb" the graph: Orient each edge in the graph such that there are no cycles, and each edge is oriented, if possible, so that belief flows from the background to a candidate node. Inference operates just as in the recursion graph baseline.

[1] Note that the marginal probabilities sum to one for the recursion graph semantics, but not for the cyclic semantics. This is not a calculation error; it is a side effect of the fact that the cyclic semantics does not specify a consistent joint distribution.

	Accuracy	Earnings Fraction	Mean Gap
Recursion	**0.623****	**0.246****	**0.112**
Cyclic	0.599	0.197	0.086
Feedforward	0.591	0.182	0.104
Undirected	0.387	-0.226	0.004

Table 1: Results from experimental test. Bold indicates highest number on that metric. Two asterisks indicates a statistically significant difference ($p < 0.05$) from the next-highest baseline on that metric. For mean gap, asterisks are not shown because recursion was not significantly higher than feedforward. Recursion was, however, significantly higher than cyclic.

Because the combed graph has no cycles, reasons are irrelevant: nodes are always eligible to be reasons for their children.

The *undirected* baseline used a symmetric Markov random field. This is similar to the feedforward baseline in that all edges between a and b, in any direction, are merged to a single weight w. During inference, the weight is treated as a compatibility function: the likelihood of the nodes taking the same value is (1.0) while the likelihood of them taking different values is $(1 - w)$. Each node is treated as having an equal prior probability of being on or off.

As mentioned previously, we also attempted to run ProbLog on our data set, but ProbLog ran out of memory on a large majority of questions, so results cannot be shown. This may be due to the problem of attempting to ground out the recursion graph into an acyclic representation.

3.3 Results

Results are shown in Table 1. The accuracy of an algorithm is the proportion of questions on which the highest confidence answer is the correct answer: $right/(right + wrong)$. There was a tie on one question; all baselines returned the same answer for this question. The remaining metrics address the efficacy of confidence estimation. Earnings fraction addresses the idea that high confidence in incorrect answers should be penalized. It considers only those questions on which the highest confidence was above 50 per cent. Earnings fraction among these questions is $(right - wrong)/(right + wrong)$. Gap reflects the average difference between the top right answer and the top wrong answer (negative if the top wrong answer had higher confidence). Significance was calculated using Fisher randomization. As can be seen in Table 1, recursion graphs significantly outperform the next-best baseline (cyclic graphs) on all metrics.

4 Discussion and further work

Recursion graphs present an efficient solution to the feedback problem: given graphs with directed cycles, recursion graphs allow us to preserve the information contained in the cycles without unintended feedback loops. We have also seen that recursion graphs have some desirable properties: Formally, they can be shown to have a consistent joint distribution. Practically, we can operate directly on the cyclic representation without having to ground the graph out into an acyclic form, making some applications newly tractable. Empirically, they perform better on at least one application. While they added a small computational cost, we believe this may be an attractive tradeoff given other benefits. Thus recursion graphs open up the possibility of directly using a cyclic graph in an application that encounters the feedback problem, with the confidence that we are standing on a solid formal foundation, and the possibility of better performance.

An obvious area for further work is to deploy recursion graphs in other applications that encounter the feedback problem. In preparing this paper, it was difficult to find established data sets that directly represented the feedback problem. After all, it is widely known that graphs with directed cycles are not formally valid for inference in graphical models, which may explain why few have made such graphs available. Thus a "chicken and egg" problem exists with respect to the literature. It is hoped that the formal and empirical results presented here will encourage others to try recursion graphs on applications that show the feedback problem.

References

[1] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *IJCAI*, volume 99, pages 1300–1309, 1999.

[2] David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, and Carl Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *The Journal of Machine Learning Research*, 1:49–75, 2001.

[3] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7, pages 2462–2467, 2007.

[4] Peter Spirtes. Directed cyclic graphical representations of feedback models. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 491–498. Morgan Kaufmann Publishers Inc., 1995.

[5] Alexander L Tulupyev and Sergey I Nikolenko. Directed cycles in bayesian belief networks: probabilistic semantics and consistency checking complexity. In *MICAI 2005: Advances in Artificial Intelligence*, pages 214–223. Springer, 2005.

[6] Alan L Yuille and Hongjing Lu. The noisy-logical distribution and its application to causal inference. In *NIPS*, 2007.

[7] Adam Lally, Sugato Bachi, Michael A Barborak, David W Buchanan, Jennifer Chu-Carroll, David A Ferrucci, Michael R Glass, Aditya Kalyanpur, Erik T Mueller, J William Murdock, et al. Watsonpaths: Scenario-based question answering and inference over unstructured information. Technical report, Technical Report Research Report RC25489, IBM Research, 2014.