Generating Random Bayesian Networks with Constraints on Induced Width

Jaime S. Ide and **Fabio G. Cozman** and **Fabio T. Ramos**[1]

Abstract. We present algorithms for the generation of uniformly distributed Bayesian networks with constraints on induced width. The algorithms use ergodic Markov chains to generate samples. The introduction of constraints on induced width leads to realistic networks but requires new techniques. A tool that generates random networks is presented and applications are discussed.

1 INTRODUCTION

It is often the case that theoretical questions involving artificial intelligence techniques are hard to answer exactly. Many such questions appear in the theory of Bayesian networks; for example, How does quasi-random sampling algorithms compare to pseudo-random sampling? Significant insight into such questions could be obtained by analyzing large samples of Bayesian networks. However it may be difficult to collect hundreds of "real" Bayesian networks for an experiment, or it may be the case that an experiment must be conducted for a specific type of Bayesian network for which few "real" examples are available. One must then randomly generate Bayesian networks that are somehow close to "real" networks. In fact, many researchers have used random processes to generate networks in the past, but without guarantees of that every allowed graph is produced with the same uniform probability (for example, [14, 15]).

We would like to have a method that generates Bayesian networks uniformly; that is, we would like to guarantee that averages taken with generated networks produce unbiased estimates. We would also like to have generation methods that are flexible in the sense that constraints on generated networks can be added with relative ease. For example, it should be possible to add a constraint on the maximum number of parents for nodes, the average number of children, or the maximum number of loops. Ad hoc methods are usually concocted for a particular set of constraints, and it is hard to imagine ways to add constraints to them.

Finally, we would like to generate "realistic" networks, however hard it may be to define what is a "real" Bayesian network. A reasonable strategy is to look for properties that are commonly used to characterize Bayesian networks, and to allow some control over them. This is the strategy followed by Ide and Cozman [5]: they allow control over the degree of a node, thus allowing some control over the "density" of the connections in the generated Bayesian networks. We have found that such a strategy is reasonable but not perfect. Restrictions solely on node degree and number of edges lead to "overly random" edges — real networks often have their variables distributed in groups, with few edges between groups.[2] Another strategy, suggested by T. Kocka (personal communication), would be to produce Bayesian networks with a large number of equivalent graphs, as this is a property observed in real networks. However we would like to use properties with clear intuitive meaning, so that users of our algorithms would quickly grasp the properties of generated networks.

A quantity that characterizes the algorithmic complexity of Bayesian networks, and is easy to explain and to understand, is the induced width. Indirectly, the induced width captures how dense a network is. Besides, it makes sense to control induced width, as we are usually interested in comparing algorithms or parameterizing results with respect to the complexity of the underlying network.[3] Unfortunately, the generation of random graphs with constraints on induced width is significantly more involved than the generation of graphs with constraints on node degree and number of edges. In this paper we report on new algorithms that accomplish generation of graphs with simultaneous constraints on all these quantities: induced width, node degree, and number of edges.

Following the work of Ide and Cozman [5], we divide the generation of random Bayesian networks into two steps. First we generate a random directed acyclic graph that satisfies constraints on induced width, node degree, and number of edges; then we generate probability distributions for the graph. To generate the random graph, we construct ergodic Markov chains with appropriate stationary distributions, so that successive sampling from the chains leads to the generation of properly distributed networks. The necessary theory and algorithms are presented in Sections 2 and 3.

The methods presented in this paper focus on Bayesian networks, but they convey a general method for generation of testing examples in artificial intelligence. The idea is to generate uniformly distributed examples using Markov chains. This strategy allows one to easily add and modify constraints on the generated examples, provided that a few steps are taken. The theory in Section 3 can serve as a guide for exactly what steps must be taken to guarantee appropriate results.

A freely distributed program for Bayesian network generation is presented in Section 4. In Section 4 we also discuss applications of random networks.

2 BASIC CONCEPTS

This section summarizes material from [5] and [3].

A *directed* graph is composed of a set of nodes and a set of edges. An edge (u, v) goes from a node u (the *parent*) to a node v (the *child*). A *path* is a sequence of nodes such that each pair of consecutive nodes is adjacent. A path is a *cycle* if it contains more than two nodes and the first and last nodes are the same. A cycle is *directed* if we can reach the same nodes while following arcs that are in the

[1] Escola Politécnica, Univ. de São Paulo, São Paulo, Brazil. Email: jaime.ide@poli.usp.br

[2] Tomas Kocka brought this fact to our attention.

[3] Carlos Brito suggested this strategy.

same direction. A directed graph is *acyclic* (a *DAG*) if it contains no directed cycles. A graph is *connected* if there exists a path between every pair of nodes. A graph is *singly-connected*, also called a *polytree*, if there exists exactly one path between every pair of nodes; otherwise, the graph is *multiply-connected* (or *multi-connected* for short). An *extreme sub-graph* of a polytree is a sub-graph that is connected to the remainder of the polytree by a single path. In an *undirected* graph, the direction of the edges is ignored. An *ordered graph* is a pair containing an undirected graph and an ordering of nodes. The *width* of a node in an ordered graph is the number of its neighbors that precede it in the ordering. The *width* of an ordering is the maximum width over all nodes. The *induced width* of an ordered graph is the width of the ordered graph obtained as follows: nodes are processed from last to first; when node X is processed, all its preceding nodes are connected (call these connections *induced connections* and the resulting graph *induced graph*). An example is presented at Figure 1. The *induced width* of a graph is the minimal induced width over any ordering; the computation of induced width is an NP-hard problem [3], and computations are usually based on heuristics [6].

A Bayesian network represents a joint probability density over a set of variables $\mathbf{X}$ [10]. The density is specified through a directed acyclic graph; every node in the graph is associated with a variable X_i in $\mathbf{X}$, and with a conditional probability density $p(X_i|\text{pa}(X_i))$, where $\text{pa}(X_i)$ denotes the parents of X_i in the graph. A Bayesian network represents a unique joint probability density [10]: $p(\mathbf{X}) = \prod_i p(X_i|\text{pa}(X_i))$ (consequence of a *Markov condition*). The *moral graph* of a Bayesian network is obtained by connecting parents of any variable and ignoring direction of edges. The *induced width* of a Bayesian network is the induced width of its moral graph. An *inference* is a computation of a posterior probability density for a *query* variable given *observed* variables; the complexity of inferences is directly related to the induced width of the underlying Bayesian network [3].

We use Markov chains to generate random graphs, following [8]. Consider a Markov chain $\{X_t, t \geq 0\}$ over finite domains S and $P = (p_{ij})_{ij=1}^{M}$ to be a M x M matrix representing transition probabilities, where M is the number of states and $p_{ij} = Pr(X_{t+1} = j|X_t = i)$, for all t [11, 13]. The s-step transition probabilities is given by $P^s = p_{ij}^{(s)} = Pr(X_{t+s} = j|X_t = i)$, independent of t. A Markov chain is *irreducible* if for all i,j there exists s that satisfies $p_{ij}^{(s)} > 0$. A Markov chain is irreducible if and only if all pair of states intercommunicate. A Markov chain is *positive recurrent* if every state $i \in S$ can be returned to in a finite number of steps; it follows a that finite irreducible chain is positive recurrent. A Markov chain is *aperiodic* if the greatest common divisor of all those s for which $p_{ii}^{(s)} > 0$, is equal to one (that is, $G.C.D.(s|p_{ii}^{(s)} > 0) = 1$). Aperiodicity is ensured if $p_{ii} > 0$ (p_{ii} is a *self-loop probability*). A Markov chain is *ergodic* if there exists a vector π (the *stationary* distribution) satisfying $\lim_{s \longrightarrow \infty} p_{ij}^{(s)} = \pi_j$, for all i and j; a finite aperiodic, irreducible and positive recurrent chain is ergodic. A transition matrix is called *doubly stochastic* if the rows and columns sum to one (that is, if $\sum_{j=1}^{N} p_{ij} = 1$ and $\sum_{i=1}^{N} p_{ij} = 1$). A Markov chain with such a transition matrix has a uniform stationary distribution [11].

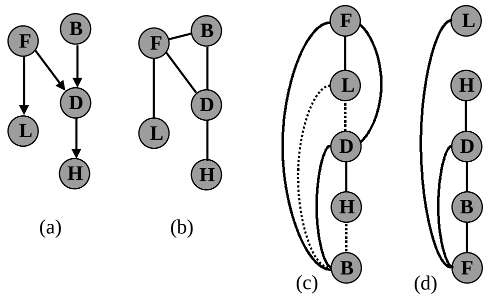

Figure 1. a) Network, b) moral graph, c) induced graph for ordering F, L, D, H, B, and d) induced graph for ordering L, H, D, B, F. Dashed lines represent induced connections.

3 GENERATING RANDOM DAGS

In this section we show how to generate random DAGs with constraints on induced width, node degree and number of edges. After such a random DAG is generated, it is easy to construct a complete Bayesian network by randomly generating associated probability distributions — if all variables in the Bayesian network are categorical, probability distributions are produced by sampling Dirichlet distributions. More general methods can be contemplated (for example, it may be interesting to generate logical nodes together with probabilistic nodes) and are left for future work.

To generate random DAGs with specific constraints, we construct an ergodic Markov chain with uniform limiting distribution, such that every state of the chain is a DAG satisfying the constraints. By running the chain for many iterations, eventually we obtain a satisfactory DAG.

Algorithm PMMixed produces an ergodic Markov chain with the required properties (Figure 2). The algorithm is significantly more complex than the algorithms presented by Ide and Cozman [5]. The added complexity comes from the constraints in induced width. Such a price is worth paying as the induced width is a property that characterizes a Bayesian network much more accurately than node degree.

The algorithm works as follows. We create a set of n nodes (from 0 to $n-1$) and a simple network to start. The loop between lines 03 and 08 constructs the next state (next DAG) from the current state. Lines 05 and 08 verify whether the induced width of the current DAG satisfies the maximum value allowed; constraints on maximum node degree and maximum number of edges must also be checked there. If the current DAG is a polytree, the next DAG is constructed in lines 04 and 05; if the current DAG is multi-connected, the next DAG is constructed in lines 07 and 08. Depending on the current graph, different operations are performed (the procedures AorR and AR correspond to the valid operations). Note that the particular procedure to be performed and the acceptance (or not) of the resulting DAG is probabilistic, parameterized by p.

Algorithm PMMixed is essentially a mixture of procedures AorR and AR. These procedures are used by Ide and Cozman [5] to produce respectively multi-connected graphs and polytrees with constraints on node degree. We need both to guarantee irreducibility of Markov chains when constraints on induced width are present; the procedure AR creates a needed "path" in the space of polytrees that is used in Theorem 3. The mixture of procedures has two other benefits: first, it creates more complex transitions, hopefully increasing the convergence of the chain; second, it eliminates a restriction on node degree that was needed by Ide and Cozman [5].

The PMMixed algorithm can be understood as a sequence of probabilistic transitions that follow the scheme in Figure 3.

We now establish ergodicity of Algorithm PMMixed.

Theorem 1 *The Markov chain generated by Algorithm PMMixed is aperiodic.*

```
Algorithm PMMixed: Generating DAGs with induced width control
Input: Number of nodes ($n$), number of iterations ($N$), maximum induced width, and possibly constraints on node degree and number of nodes.
Output: A connected DAG with $n$ nodes.
01. Create a network with $n$ nodes, where all nodes have just one parent, except the first node that does not have any parent;
02. Repeat $N$ times:
03.     If current graph is a polytree:
04.         With probability $p$, call Procedure AorR; with
                probability $(1-p)$, call Procedure AR.
05.         If the resulting graph satisfies imposed
                constraints, accept the graph;
                otherwise, keep previous graph;
06.     else (graph is multi-connected):
07.         Call Procedure AorR.
08.         If the resulting graph is a polytree and satisfies
                imposed constraints, accept with probabil-
                ity $p$; else accept if it satisfies imposed
                constraints; otherwise keep previous graph.
09. Return current graph after $N$ iterations.

Procedure AR: Add and Remove
01. Generate uniformly a pair of distinct nodes $i, j$;
02. If the arc $(i, j)$ exists in the current graph, keep the same state; else
03. Invert the arc with probability 1/2 to $(j, i)$, and then
04. Find the predecessor node $k$ in the path between $i$ and $j$, remove the arc between $k$ and $j$, and add an arc $(i, j)$ or arc $(j, i)$ depending on the result of line 03.

Procedure AorR: Add or Remove
01. Generate uniformly a pair of distinct nodes $i, j$;
02. If the arc $(i, j)$ exists in the current graph, delete the arc, provided that the underlying graph remains connected; else
03. Add the arc if the underlying graph remains acyclic, otherwise keep same state.
```

Figure 2. Algorithm for generating DAGs, mixing operations AR and AorR.

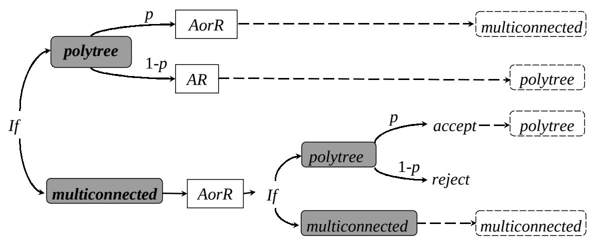

Figure 3. Structure of PMMixed.

Proof. It is always possible to stay in the same state for procedures AR and AorR; therefore, all states have a self-loop probability greater than zero. QED

Theorem 2 *The transition matrix defined by the Algorithm PMMixed is doubly stochastic.*

Proof. If we have symmetric transition probabilities between two neighbor states, its rows and columns sum one, because the self-loop probabilities are complementary to all other probabilities. Procedure AorR is clearly symmetric; procedure AR is also symmetric [5]. We just have to check that transitions between polytrees and multi-connected graphs are symmetric; this is true because transitions from polytree to multi-connected are accepted with probability p, and multi-connected to polytree transitions are also accepted with the same probability. QED

We need the following lemma to prove Theorem 3.

Lemma 1 *After removal of an arc from a multi-connected DAG, its induced width does not increase.*

Proof. When we remove an arc, the moral graph stays the same or contains less arcs; by just keeping the same ordering, the induced width cannot increase. QED

Theorem 3 *The Markov chain generated by Algorithm PMMixed is irreducible.*

Proof. Suppose that we have a multi-connected DAG with n nodes; if we prove that from this graph we can reach a simple sorted tree (Figure 4 (c)), the opposite transformation is also true, because of the symmetry of our transition matrix — and therefore we could reach any state from any other (during these transitions, graphs must remain acyclic, connected and must satisfy imposed constraints). So, we start by finding a loop cutset and removing enough arcs to obtain a polytree from the multi-connected DAG [10]. The induced width does not increase during removal operations by Lemma 1. From a polytree we can move to a simple polytree (Figure 4 (b)) in a recursive way. For all extreme sub-graphs of our polytree, for each pair of extreme sub-graphs (call them *branches*), it is possible to "cut" a branch and add it in the other branch, by the procedure AR, without ever increasing the induced width. Doing this we get a unique branch. If we have more than two branches connected to a node, we repeat this process by pairs; we do this recursively until get a simple polytree. Now that we have a simple polytree, we get a simple tree (Figure 4 (a)) just inverting arcs to the same direction, without ever getting an induced width greater than two. The last step is to get a simple sorted tree (Figure 4 (c)) from the simple tree. The idea here is illustrated in Figure 5. We want to sort labelled nodes from 1 to n. Start removing arc (n, k) and adding arc (l, i) (step 1 to 2). Remove arc (j, n) and add arc $(n-1, n)$ (step 2 and 3). Note that in this configuration, the induced width is one. Now, remove arc $(n-1, o)$ and add arc (j, k) (step 3 and 4). Repeat steps 2 and 4 for all nodes. So, from any multi-connected DAG it is possible to reach a simple sorted tree. The opposite path is clearly analogous, so we can go from any DAG to any other DAG, and the chain is irreducible. Note that constraints on node degree and maximum number of edges can be dealt with within the same processes. QED

By the previous theorems we obtain:

Theorem 4 *The Markov chain generated by Algorithm PMMixed is ergodic and its unique stationary converges to a uniform distribution.*

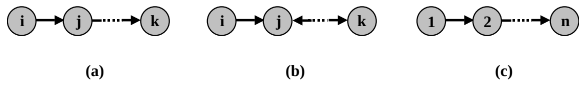

Figure 4. Simple trees used in our proofs: (a) Simple tree, (b) Simple polytree, (c) Simple sorted tree.

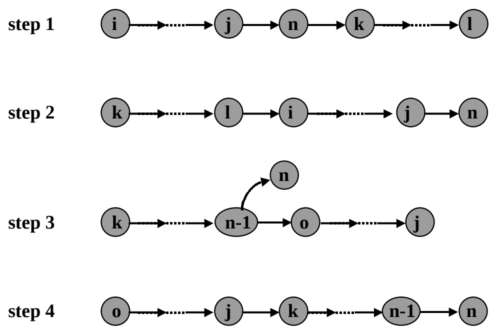

Figure 5. Basic moves to obtain a simple sorted tree.

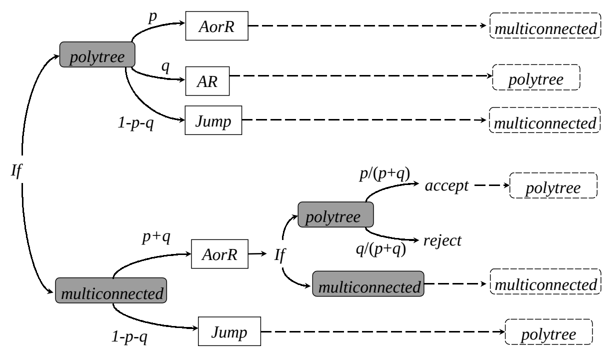

Figure 7. Structure of PMMixed with procedure J.

The algorithm PMMixed can be implemented quite efficiently, except for the computation of induced width — finding this value is a NP-hard problem with no easy solution. There are heuristics for computing induced width; some of which have been found to be of high quality [6]. Consequently, we must change our goal: instead of adopting constraints on exact induced width, we assume that the user specifies a maximum width *given a particular heuristic*. We call this width the *heuristic width*. Our goal then is to produce random DAGs on the space of DAGs that have constraints on heuristic width.

Apparently we could still use the PMMixed algorithm here, with the obvious change that lines 05 and 08 must check heuristic width instead of induced width. However such a simple modification is not sufficient: because heuristic width is usually computed with local operations, we cannot predict the effect of adding and removing edges on it. That is, we cannot adapt Lemma 1 to heuristic width in general, and then we cannot predict whether a "path" between DAGs can in fact be followed by the chain without violating heuristic width constraints. We must create a mechanism that would allow the chain to transit between arbitrary DAGs regardless of the adopted heuristic. Our solution is to add a new type of operation, specified by procedure J (Figure 6) — this procedure allows "jumps" from arbitrary multi-connected DAGs to polytrees. We also assume that any adopted heuristic is such that, *if the DAG is a polytree, then the heuristic width is equal to the induced width.* Even if a given heuristic does not satisfy this property, the heuristic can be easily modified to do so: test whether the DAG is a polytree and, if so, return the induced width of the polytree (the maximum number of parents amongst all nodes).

Procedure J must be called with probability $(1 - p - q)$ both after line 04 and after line 07 in the algorithm PMMixed. The complete algorithm can be understood as a sequence of probabilistic transitions that follow the scheme in Figure 7. All previous theorems can be easily extended to this new situation; the only one that must be substantially modified is Theorem 3. Transitions from polytree to multi-connected DAGs are performed with probability $(1 - q)$; transitions from multi-connected DAGs to polytrees are performed with probability $1 - (p + q) \times \frac{q}{p+q} = 1 - q$. The value of p and q control the mixing rate of the chain; we have observed remarkable insensitivity to these values.

```
Procedure J: Sequence of AorR
01. If the current graph is polytree:
02.     Generate uniformly a pair of distinct nodes $i, j$;
03.     If arc $(i, j)$ does not exist in current graph,
            add the arc; otherwise, keep the same state.
04. If the current graph is multi-connected:
05.     Generate uniformly a pair of distinct nodes $i, j$.
06.     If arc $(i, j)$ exists in current graph, remove the arc;
        otherwise, keep the same state.
07. If the new graph satisfies imposed constraints, accept the
    graph; otherwise, keep previous graph.
```

Figure 6. Procedure J.

4 THE BNGenerator AND APPLICATIONS

The algorithm PMMixed (with the modifications indicated in Figure 7) can be efficiently implemented with existing ordering heuristics, and the resulting DAGs are quite similar to existing Bayesian networks. We have implemented the algorithm using a $\mathcal{O}(n \log n)$ implementation of the minimum weight heuristic. The result is the *BNGenerator* package, freely distributed under the GNU license (at http://www.pmr.poli.usp.br/ltd/Software/BNGenerator). The software uses the facilities in the JavaBayes system, including the efficient implementation of ordering heuristics (http://www.cs.cmu.edu/˜javabayes). The BNGenerator accepts specification of number of nodes, maximum node degree, maximum number of edges, and maximum heuristic width (for minimum weight heuristic, but other heuristics can be added). The software also performs uniformity tests using a χ^2 test. Such tests can be performed only for small number of nodes (as the number of possible DAGs grows extremely quickly [12]), but they allowed us to test the

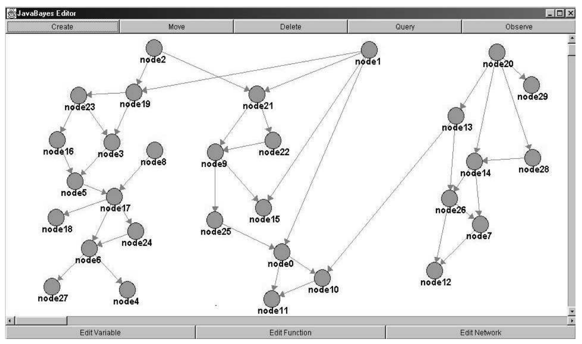

Figure 8. Bayesian network generated with BNGenerator: 30 nodes, maximum degree 20, maximum induced width 2.

algorithm and its procedures. We have observed the relatively fast mixing of the chain with the transitions we have designed.

To show how to use our previous results, we discuss the evaluation of a particular inference algorithm that has received attention in the literature but have no conclusive analysis yet. Due to the lack of space, we present a brief summary of rather extensive tests; more details can be found in a longer technical report [4]. Two other applications can be found in that technical report: a study on the relation between heuristic width and d-connectivity, and a study of convergence for loopy propagation in networks with non-zero probabilities.

Consider the behavior of Monte Carlo methods associated with quasi-random numbers. That is, numbers that form low discrepancy sequences — numbers that progressively cover the space in the "most uniform" manner [7, 9]. There have been quite successful applications of quasi-Monte Carlo methods for integration in low-dimensional problems; in high-dimensional problems, there has been conflicting evidence regarding the performance of quasi-Monte Carlo methods. As a positive example, Cheng and Druzdzel obtained good results in Bayesian network inference with importance sampling using quasi-random numbers [1]. We have investigated the following question: How does quasi-random numbers affect standard importance sampling and Gibbs sampling algorithms in Bayesian networks? We have used the importance sampling scheme derived by Dagum and Luby [2], and have investigated the behavior of Halton sequences in random networks. The summary of our investigation is as follows. First, pseudo-random numbers are clearly better than quasi-random numbers in medium-sized networks for Gibbs sampling. Second, pseudo-random number have a small edge over quasi-random numbers for importance sampling; however the differences are so small that both can be used. In fact it is not hard to find networks that behave better under quasi-random importance sampling than under pseudo-random importance sampling.[4]

The methodology indicated in this example can be applied to other inference algorithms and theoretical questions related to directed acyclic graphs and Bayesian networks.

5 CONCLUSION

In this paper we have presented a solution for the generation of uniformly distributed random Bayesian networks with control over key quantities. The main idea is to generate DAGs with control on induced width, and then generate distributions associated with the generated DAG. Given the NP-hardness of induced width, we have resorted to "heuristic width" — the width produced by one of the many high-quality heuristics available. We generate DAGs using Markov chains, and the need to guarantee heuristic width constraints leads to a reasonably complex transition scheme encoded by algorithm PMMixed and procedure J. The algorithm can be modified to accommodate a number of other constraints (say constraints on the maximum number of parents). The methodology used to derive these algorithms and proving their convergence can be employed to generate testing examples in other fields of artificial intelligence. The reliance on Markov chains demands convergence proofs and mixing times, but it allows the manipulation of constraints and guarantees of uniformity that do not seem to be handled by other methods.

We have observed that this strategy does produce "realistic-looking" Bayesian networks. Using such networks, we have confirmed comments in the literature that suggest that standard Gibbs sampling cannot profit from quasi-random samples, while straightforward importance sampling presents essentially the same behavior under pseudo- and quasi-random sampling for medium-sized networks. We have also investigated the relationship between heuristic width and d-connectivity and the performance of loopy propagation, and reported on those issues elsewhere.

[4] As a notable (not randomly generated) example of this phenomenon, the Alarm network does behave slightly better with quasi-random than with pseudo-random importance sampling (corroborating results by Cheng and Druzdzel [1]).

ACKNOWLEDGEMENTS

We thank Carlos Brito for suggesting the use of induced width, Robert Castelo for pointing us to Melançon et al's work, Guy Melançon for confirming some initial thoughts, Nir Friedman for indicating how to generate distributions, and Haipeng Guo for testing the BNGenerator. We also thank Jaap Suermondt, Tomas Kocka, Alessandra Potrich and Márcia D'Elia Branco for providing important ideas, and Y. Xiang, P. Smets, D. Dash, M. Horsh, E. Santos, and B. D'Ambrosio for suggesting valuable procedures. The first author was supported by FAPESP grant 00/11067-9. This work was (partially) developed in collaboration with HP Brazil R&D; the third author was supported by HP Brazil R&D and was responsible for investigating loopy propagation. The second author was partially supported by CNPq through grant 300183/98-4.

REFERENCES

[1] J. Cheng and M. Druzdzel, 'Computational investigation of low-discrepancy sequences in simulation algorithms for Bayesian networks', in *Conf. on Uncertainty in Artificial Intelligence*, pp. 72–81, SF, CA. Morgan Kaufmann.

[2] P. Dagum and M. Luby, 'An optimal approximation algorithm for Bayesian inference', *Artificial Intelligence*, **93**(1–2), 1–27, (1997).

[3] R. Dechter, 'Bucket elimination: An unifying framework for probabilistic inference', in *Conf. on Uncertainty in Artificial Intelligence*, pp. 211–219, SF, CA. Morgan Kaufmann.

[4] J. S. Ide and F. G. Cozman and F. T. Ramos, *Generation of Random Bayesian Networks with Constraints on Induced Width, with Applications to the Average Analysis of d-Connectivity, Quasi-random Sampling, and Loopy Propagation*, Tech. Report BT/PMR, University of São Paulo, Brazil, 2004.

[5] J. S. Ide and F. G. Cozman, 'Random generation of Bayesian networks', in *Brazilian Symp. on Artificial Intelligence*. Springer-Verlag, (2002).

[6] U. Kjaerulff, 'Triangulation of graphs — algorithms giving small total state space', Technical Report R-90-09, Department of Mathematics and Computer Science, Aalborg University, Denmark, (March 1990).

[7] J. G. Liao, 'Variance reduction in Gibbs sampler using quasi random numbers', *Journal of Computational and Graphical Statistics*, **7**(3), 253–266, (September 1998).

[8] G. Melançon and M. Bousque-Melou, 'Random generation of dags for graph drawing', Technical Report technical report INS-R0005, Dutch Research Center for Mathematical and Computer Science-CWI, (2000).

[9] H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*, volume 63 of *CBMS-NSF regional conference series in Appl. Math.*, SIAM, Philadelphia, 1992.

[10] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, Morgan-Kaufman, 1988.

[11] S. I. Resnick, *Adventures in Stochastic Processes*, Birkhäuser, Cambridge, MA, USA; Berlin, Germany; Basel, Switzerland, 1992.

[12] R. W. Robinson, 'Counting labeled acyclic digraphs', in *New Directions in the Theory of Graphs*, ed., F. Harary, pp. 28–43, Michigan, (1973). Academic Press.

[13] S. M. Ross, *Stochastic Processes*, John Wiley & Sons; New York, 1983.

[14] P. Spirtes, C. Glymour, and R. Scheines, *Causation, Prediction, and Search (second edition)*, MIT Press, 2000.

[15] Y. Xiang and T. Miller, 'A well-behaved algorithm for simulating dependence structure of Bayesian networks', in *International Journal of Applied Mathematics*, volume 1, pp. 923–932, (1999).