Structure Search and Stability Enhancement of Bayesian Networks

Hanchuan Peng and Chris Ding
Computational Research Division, Lawrence Berkeley National Laboratory,
University of California, Berkeley, CA, 94720, USA
Email: hpeng@lbl.gov, chqding@lbl.gov

Abstract

Learning Bayesian network structure from large-scale data sets, without any expert-specified ordering of variables, remains a difficult problem. We propose systematic improvements to automatically learn Bayesian network structure from data. (1) We propose a linear parent search method to generate candidate graph. (2) We propose a comprehensive approach to eliminate cycles using minimal likelihood loss, a short cycle first heuristic, and a cut-edge repairing. (3) We propose structure perturbation to assess the stability of the network and a stability-improvement method to refine the network structure. The algorithms are easy to implement and efficient for large networks. Experimental results on two data sets show that our new approach outperforms existing methods.

1. Introduction

The rapidly increasing quantity of data in many data mining fields allows a great opportunity to model and understand the relationships among a large number of variables. Bayesian Networks (BNs) [13][6][8][1] provide a consistent framework to model the probabilistic dependencies among variables, e.g. in medical image mining [15]. A BN [6][8] is a Directed Acyclic Graph (DAG) $G = (V, E)$ that models the probabilistic dependencies among a group of variables (nodes). The joint distribution can be factorized into the product of conditional probabilities of every variable given its parents: $P(\{g\}) = \Pi_{g\in V} P(g|\pi_g)$, where g stands for a variable, π_g is the parents of g; the directed edges among nodes encode the respective conditional distributions.

Directly identifying the BN structures from input data D remains a challenge. The problem is NP-hard [3][6]. Many heuristic search methods have been proposed (for reviews see [8][1]). If there is a predefined ordering of variables, the well-known K2 algorithm [6] can efficiently determine the structure. For many applications where there is no sufficient knowledge to provide such an ordering, the BN-learning methods, e.g. conditional independence test based method [2], often have at least $O(n^4)$ complexity. Other Monte Carlo methods have even larger complexity, e.g. the search method based on random-sampling and model averaging in the space of ordering [7]. Clearly, an efficient algorithm to identify BN structures, without requiring ordering of variables, is particularly important. There exist several methods/software. For example, the WinMine software of Chickering [4] has the strength to learn large BN structure. Cheng designed PowerConstructor [[1]] and won data mining contest KDD-Cup-2001.

We propose a new $O(n^2)$ algorithm to infer locally stable Bayesian networks without requiring predefined ordering of variables or predefined thresholds to terminate the model search. Our algorithm consists of three main steps. (1) We develop an efficient algorithm to search optimal parents, which form a candidate graph (See §2). (2) We propose a new graph-based method to eliminate possible cycles in the candidate graph that would violate the acyclic assumption of BNs (See §3). (3) We evaluate the network stability using structural perturbation. The structural perturbation can detect unstable local structures; an algorithm for improving the stability is proposed (See §4).

We assume a uniform prior of the structure of G. The posterior (log-likelihood) of G given the data D, $\ell(G) = \log P(G|D) \propto \log P(D|G)$, is used to judge the optimality of G. The posterior can be evaluated using different scores, including the Bayesian score [6] and its variant [5], MDL [1], BDe [8], etc. In this paper, we use the Bayesian score, but other scores can be equally well adopted in our structure identification algorithms.

2. Candidate Graph

The candidate graph G_c is a directed graph containing important associations of variables where the redundancy of associations should be minimized. Our approach is to identify the optimal parent set for each node based on the Bayesian score ℓ. Here our emphasis is on how to efficiently search for optimal parent set, $\pi = \{g_i^*, i=1,\ldots,m\}$. The locally optimal parent set is similar to dependency graph of Heckerman *et al* [9][11]; the difference is that they used regression to determine the dependency while we directly search based on the Bayesian score.

Our algorithm is an extension of K2 algorithm [6]. K2 uses a simple incremental search strategy: it first searches for the best singleton parent g_1^*, i.e., $g_1^* = \text{argmax}_i\ \ell(g_i \to g)$, and $\ell(g_i \to g) > \ell(g) + \ell(g_i)$. It then searches for further parent(s) to maximize the score increase in each step, until no better score can be found.

We extend K2 in two directions. (*a*) We constrain the search in the most probable space to reduce the computational complexity. Note that once a parent or parents are found, many of the rest nodes are rendered conditionally independent. Thus in searching for the second parent g_2^*, we do not need to search through all the rest variables, $\Omega_1 = \{V \backslash g_1^*\}$; instead we need only search

$$\Omega_1^+ = \{g_i \in V \mid g_i \neq g_1^*, \ell(g_i \to g) > \ell(g_i) + \ell(g)\}. \quad (1)$$

Note that Ω_1^+ is obtained automatically when searching for g_1^*. Similarly, when searching for g_3^*, we need only search Ω_2^+, instead of $\Omega_2 = \{V \backslash \{g_1^*, g_2^*\}\}$; etc. This restriction saves a large fraction of the searched space.

(*b*) We systematically search a larger space than K2. In K2, g_1^* corresponds to the largest $\ell(g_i \to g)$. Denote the respective parent set as $\pi^{(1)}$. We can search another set of parents beginning with the second largest $\ell(g_i \to g)$, denoted as $\pi^{(2)}$. If $\pi^{(2)}$ leads to better

Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)
0-7695-1978-4/03 $ 17.00 © 2003 IEEE

score than $\pi^{(1)}$, then we take $\pi^{(2)}$ as the final parent set. We call this 2-max search. This can be extended to k-max search. Clearly, the Bayesian score of $\pi^{(k)}$ increases monotonically with k, at the expense of linearly enlarged complexity.

We call this modified method the K2+ algorithm. It has the complexity $O(\alpha kmn)$, where α counts for the reduction using $\Omega_1^+, \Omega_2^+, \ldots, \Omega_m^+$ instead of $\Omega_1, \Omega_2, \ldots, \Omega_m$ (often Ω_i^+ contains a much smaller number of variables than Ω_i). Accordingly, the complexity to construct the whole candidate graph is $O(n^2)$.

3. Cycle Elimination

Since the candidate graph G_c is generated via local optimal search, it is possible that G_c contains many cycles that violate the basic acyclic assumption of BNs.

A simple approach is to enumerate all possible DAGs that could emerge from G_c and select the one with the largest score. However, this method is impractical due to its exponential complexity. Approximation methods based on random edge cut [12] have been studied. A heuristic decision-tree based approach has also been studied in [11]. In this paper, we resolve this problem via graph algorithmic approach. Our comprehensive approach consists of three methods that can be implemented efficiently.

Any cycle must lie in a Strongly Connected Component (SCC) of the graph. An efficient $O(n)$ algorithm based on depth-first search can locate SCCs in a directed graph. We first find all the SCCs in G_c, and eliminate cycles within each SCC.

3.1 Bayesian Likelihood Loss Function

If a SCC contains one cycle, we can break one cycle at a time. We break cycles based on loss function. For each edge $g_i \to g_j$, we define the loss as the reduction of Bayesian log-likelihood for g_j due to the loss of one of its parent

$$w(g_i \to g_j) = \ell(g_j \mid \pi) - \ell(g_j \mid \pi \setminus g_i). \qquad (2)$$

Note that $w(g_i \to g_j) \neq w(g_i \leftarrow g_j)$. Although mutual information might be another possible choice as the loss, it does not reflect the joint association between different parents and g_j.

If a SCC contains several cycles, sometimes they share one or more common edges, such as the cycles in Figure 1. For example, in Figure 1(a) the edge $g_2 \to g_3$ is shared by the cycles C_{1231} (i.e. $g_1 \to g_2 \to g_3 \to g_1$) and C_{2342}.

There are several criteria to break the cycles. (*a*) We can simply cut edges with the smallest loss. (*b*) We can identify the common edges and cut the one shared by most cycles. In Figure 1 (b), cutting the common edge $g_2 \to g_3$ will eliminate two cycles. (*c*) The loss function criterion indicates there could be better choices. Suppose $g_1 \to g_2$ and $g_3 \to g_4$ are the edges with the minimal loss in cycles C_{1231} and C_{2342}. If the condition

$$w(g_1 \to g_2) + w(g_3 \to g_4) < w(g_2 \to g_3) \qquad (3)$$

holds, then we break edges $g_1 \to g_2$ and $g_3 \to g_4$; otherwise, we break the edge $g_2 \to g_3$.

Figure 1 (b) illustrates a more complicated SCC with four 3-node cycles C_{2312}, C_{2342}, C_{2542}, C_{2642}. The edge $g_2 \to g_3$ is shared 2 times and the edge $g_4 \to g_2$ is shared 3 times. We start cycle elimination from the most-shared edge (i.e. $g_4 \to g_2$) and use a minimal-likelihood-loss strategy similar to Eq.(3). If the edge $g_4 \to g_2$ is cut, then only C_{2312} remains and we will further cut its minimal loss edge; otherwise we use Eq.(3) to decide which edge(s) in cycles C_{2312} and C_{2342} should be broken.

This minimal-likelihood-loss criterion is summarized as follows. If there is no nested cycle, for each cycle we break the edge with the minimal loss. When several cycles nest among themselves, we identify the edge e_{ij} shared by most cycles and compare its loss with the sum of the minimal loss edges in participating cycles; if breaking e_{ij} leads to less loss, we cut e_{ij}; otherwise we cut the minimal loss edges in every participating cycle.

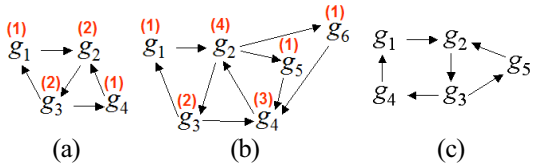

Figure 1. (a) A SCC with two 3-node cycles. (b) A SCC with four 3-node cycles. (c) A SCC with a 3-node cycle and a 4-node cycle. Multiplicities of nodes are shown in parentheses.

3.2 Short-Cycle-First Heuristic

Finding the set of cut edges with regarding to the minimal-likelihood-loss criterion could be complicated due to the existence of many cycles and the large number of common edges they share. We propose a short-cycle-first heuristic to minimize the complexity (for both computation and implementation):

(*a*) In BNs, information propagates multiplicatively because of the probability calculation. Along a fixed path of m edges, the influence of the starting node on the ending node is $P_1 P_2 \cdots P_m$ approximately. Therefore, in general, a long cycle violates the acyclic assumption less severely than a short cycle. If a SCC contains cycles of different lengths, our short-cycle first heuristic breaks the 2-node cycle first, and the 3-node cycle second, etc. In Figure 1 (c), we first break the 3-node cycle C_{2352}. Afterwards, we break the cycle C_{12341} if it still exists.

(*b*) When cycles of different lengths share edges, it is typically more efficient to break the shorter cycle first. For example, suppose two cycles C_a and C_b (with lengths a and b, respectively) share one common edge, and the cut-edge-loss-function is homogeneous. Approximately there is $1/a$ chance to break the common edge in C_a, and $1/b$ chance to break it in C_b. If $a<b$, then it is more likely (i.e. $1/a > 1/b$) that first breaking C_a (the shorter one) will simultaneously break C_b.

3.3 Cycle Identification by Matrix Multiplication

Short-cycle-first heuristic can be efficiently implemented through a matrix multiplication method. Let A be the adjacency matrix of a SCC. Diagonal elements of A are zeros. We compute A^m with the smallest m such that nonzero elements appear at matrix diagonal; with some elementary algebra, we can show that (1) nodes corresponding to nonzero diagonal elements in A^m must involve in m-node cycles; thus finding these cycles are restricted to the subgraph induced by these nodes; (2) the multiplicity of node i (i.e. value of $(A^m)_{ii}$) equals the number of times a cycle pass through node i (for example, in Figure 1 (a) and (b) the multiplicity of nodes are indicated by red numbers in parentheses). (3) Starting from the node with the highest multiplicity using breadth-first-search algorithm, restricting on the subgraph, we can

easily traverse all m-node cycles and identify the most-shared edges. For example, in Figure 1 (b), we can start from g_2 and quickly identify the most-shared edge e_{42}. We use the likelihood loss criterion to break cycles. Note that A is usually very sparse and the sparse matrix multiplications often involve much less computation than dense matrixes.

After one or more edges are cut, we re-run the SCC-detection algorithm to identify the new SCCs and the matrix multiplication method to identify shortest remaining cycles. This is repeated until all cycles are eliminated.

3.3 Repair of Local Structures

Once an edge $g_i \rightarrow g_j$ in the candidate graph G_c is cut, there is a loss of the likelihood of $\ell(g_j|\pi_j)$ because now g_j's parent set π_j is less optimal. Hence, we use K2+ parent-search to repair the parent set of each node whose incident edges have been cut. The repair is done locally, i.e., all other parents of g_j are retained during the repair of π_j. In addition, the repair is subject to the acyclic condition, i.e. the best replacement edge cannot cause cycles.

Suppose in cycle elimination, M edges are cut and the local structures of the involved nodes need repair. We notice the first-repaired local structures will give extra-constraints on the space of the later-repaired local structures due to the acyclic condition (i.e. potentially the search-space of the later-repaired local structures would be shrunk). By comparing the candidate graph G_c and the DAG G returned from cycle elimination, we first locate the nodes whose local structures need repair. We calculate the likelihood loss of a node due to the cutting of incident edges. We sort these loss values from large to small, and repair the nodes according to this ordering. This maximal-loss-first heuristic is consistent with the minimal-likelihood-loss criterion. Clearly, during the course of repair, the DAG after each local repair will always have a higher likelihood score than the DAG before this local repair. This repair algorithm has the complexity of $O(\beta n)$, where β is the number of nodes whose parent-sets are repaired.

4. Structure Perturbation and Stability Enhancement

To assess the quality of the obtained network G, we perform local structural perturbations to assess its stability. Here we consider the Edge Perturbation ("EP"), i.e., we attempt to eliminate an edge $e_{ij} = g_i \rightarrow g_j$ to see if the Bayesian likelihood is improved. A "brute force" perturbation is to simply cut e_{ij}. However, after e_{ij} is cut, g_j's parent-set is no longer optimal. For this reason, we use the K2+ algorithm to find the new optimal parents for g_2, excluding the cut edge (but keeping all other parents if any). We calculate $\Delta\ell_e^{EP}$ and the percentage of stable edges

$$r^{EP} = \frac{1}{|E|}\sum_{e\in E}\delta(\Delta\ell_e^{EP}),\ \Delta\ell = \ell(\hat{G}) - \ell(G) = \log\frac{P(D\mid\hat{G})}{P(D\mid G)}, \quad (5)$$

where $\hat{G}$ is the perturbed structure, $\delta(x) = 1$ if $x \le 0$ and 0 otherwise. The more negative $\Delta\ell_e^{EP}$, the more "stable" the edge e is. r^{EP} indicates the local stability of G. A stable G has $r^{EP} \sim 1$.

By using perturbation, we can identify those unstable edges whose replacements lead to better likelihood scores. We may improve the network structure by replacing these unstable edges with their replacements. The edge-stability-improvement algorithm first sorts the $\Delta\ell_e^{EP}$ of all unstable edges. Similar to the repair algorithm in §3.3, it then goes through all unstable edges following the sorted ordering (starting with the most unstable edge). For a given unstable edge e, the optimal replacement found in EP is first tried to see if there is cycle caused. If no, then the optimal replacement is used; otherwise, the K2+ search algorithm is invoked to search the best replacement (similar to §3.3, the search is subject to the acyclic condition, and all other parents of the current node are retained.). By applying the edge-stability-improvement algorithm, the Bayesian likelihood score of G is improved while the number of unstable edges is reduced.

Our goal is to detect and repair unstable edges to improve a single structure. This differs from other edge quality assessments, e.g. averaging over a large number of structures [14], where the edge importance is not associated with a particular structure.

5. Experiments

We use two data sets in this paper. The first is the well-known Alarm data set [6] (37 variables, 10000 samples). The intrinsic ordering of these variables is not used since our major concern is how to detect the network structure without the ordering information. We use the Alarm data accompanying the PowerConstructor software [2]. We compare our results with WinMine [4][11] (because it can generate DAGs without ordering of variables) and the best/mean results of random-sampling method in the space of variable ordering [7].

The second data set is a yeast genome [10] (481 real-valued gene variables, 300 data points). The variables are discretized to 3-states via thresholding at $\mu\pm0.4\sigma$ (σ - standard deviation, μ - mean). These states correspond to the over-expression, baseline, and under-expression of genes.

Beside the Bayesian score, we also compute the (normalized) data likelihood L based on the learned structure and conditional probabilities. We further compute the cross-validated likelihood (10-fold CV) L_{CV}, which is a better indicator for generalization.

5.1 Results on Alarm Data Set

The results on the Alarm data set are shown in Table 1. Results of our algorithms, WinMine, and ordering-space-search, are shown, together with those of the true Alarm structure and the null model (i.e. without edges). The k-max search in K2+ clearly improves the quality-measures ℓ, L, L_{CV} and r^{EP} (i.e. the 3-max search results are better than the 1-max search results). The edge stability algorithm of §3.3 clearly improves all the ℓ, L, L_{CV}, and r^{EP}. r^{EP} becomes 1 afterwards.

Compared to true model results, our best results (i.e. 3-max with improved stability) are very close. Remember that the ordering of variables is assumed unknown, thus it is highly unlikely that the true structure can be recovered from data. Hence, these results indicate our network can model the data almost equivalent to the true model, with a different network structure (57 edges in our model versus 46 edges in the true model).

We run WinMine using three different κ values, 0.01 (default value), 0.002, and 8e-12, to adjust the network to have the same number of edges as our results or as the true model. The quality metrics of these structures are not as good as our results.

When variables' ordering is unknown, one may generate many

random orderings, use K2 to learn structures, and select the best ones [7]. We perform this ordering-space-search for 100 random trials. Both the mean and best results are listed in Table 1. They are substantially worse than both our and WinMine's results, indicating it is hard to generate good models from random orderings of variables, even at great computation expense.

Table 1. Results on the Alarm data set. (ℓ, L, L_{CV} are all normalized by nN; κ is the WinMine parameter controlling the complexity of network structure. "ImpStab" means improving stability algorithm in §4.)

Method	Parent Search Method	Learning (all data)				CV (10-fold)
		ℓ	L	r^{EP}	$\|E\|$	L_{CV}
Our method (Before ImpStab)	1-max	-0.2587	-0.2543	0.9123	57	-0.2554
	3-max	-0.2581	-0.2539	0.9298	56	-0.2550
Our method (After ImpStab)	1-max	-0.2566	-0.2522	1.0000	56	-0.2533
	3-max	-0.2562	-0.2519	1.0000	57	-0.2530
True Model		-0.2555	-0.2517	0.9783	46	-0.2526
WinMine	κ = 0.01 (Default)	-0.2593	-0.2551	1.0000	57	-0.2561
	κ = 0.002	-0.2593	-0.2551	1.0000	57	-0.2561
	κ = 8e-12	-0.2655	-0.2622	0.9783	46	-0.2630
Search of Ordering Space (100 trials)	Best results	-0.2633	-0.2578	0.8730	63	-0.2592
	Mean results	-0.2701 ±0.0026	-0.2631 ±0.0023	0.8765 ±0.0521	74.0 ±5.4	-0.2650 ±0.0023
Null Model		-0.5822	-0.5813	---	0	-0.5815

5.2 Results on Yeast Gene Expression Data Set

Table 2 compares the results on the yeast gene data. In both our BNs and WinMine's results, there are more than 1600 edges for the 481 nodes. For learning, Table 2 shows that k-max search in K2+ improves ℓ, L, and r^{EP}. The edge-stability-improvement algorithm leads to steady improvements in all quality-measures.

We also run WinMine for a variety of parameter κ. The best results are obtained by setting κ to its maximal value, i.e. 1.0. Table 2 shows that in the best case, WinMine results are worse than that of 3-max, i.e. smaller ℓ, smaller r^{EP}, and less generalization strength L_{CV}. It is interesting to see that the training likelihood L of WinMine result is higher than that of 3-max, however L_{CV} of WinMine is lower than that of 3-max; this implies that the best network of WinMine might overfit data slightly.

Table 2 also lists the time (on PIII 1G CPU) of each method. A plus "+" in our results means the time spent for the current step for edge stability improvement. (Our algorithms were implemented in Matlab and C++, while WinMine was in C++). Our method uses less than 16 minutes to generate an initial BN, and 1 hour or so to refine the network structure. In contrast, WinMine takes about 4 hours to generate a network with the similar performance. These timing results show that our methods are much faster than WinMine, due to our algorithm's $O(n^2)$ complexity.

6. Discussions

A characteristic of the networks in our results is that they are rather sparse, which partially explains the high local stability of the obtained structures regarding to the perturbations. We use local structural perturbations to systematically assess the roles of individual edges in the network. Based on them, one could build larger subnet-level perturbations using clustering, seed growing, etc. This could help to detect sub-structures of BNs [16].

Acknowledgements: We thank Edward Herskovits for discussions on Bayesian learning, David Maxwell Chickering for discussion on using WinMine, and Dana Pe'er for providing the list of 481 genes. This work is supported by Department of Energy, Office of Science, under contract No. DE-AC03-76SF00098.

Table 2. Results on the Yeast gene expression data. b is the number of iterations in stability enhancement.

Search Method and Parameters			Learning (All Data)					CV (10-fold)
	k-max	b	ℓ	L	r^{EP}	$\|E\|$	T (min)	L_{CV}
Our Method	1-max	0	-0.9770	-0.8269	0.6222	1326	9	-0.9420
		1	-0.9691	-0.7831	0.8451	1517	+44	-0.9303
		2	-0.9662	-0.7687	0.9426	1586	+25	-0.9255
		3	-0.9654	-0.7640	0.9863	1611	+ 8.7	-0.9242
		4	-0.9651	-0.7627	0.9951	1620	+ 1.8	-0.9237
	3-max	0	-0.9710	-0.8019	0.7077	1433	16	-0.9338
		1	-0.9654	-0.7689	0.8901	1583	+40	-0.9250
		2	-0.9639	-0.7624	0.9579	1614	+27	-0.9226
		3	-0.9635	-0.7612	0.9821	1620	+14	-0.9220
		4	-0.9634	-0.7606	0.9889	1624	+ 6.2	-0.9218
		5	-0.9631	-0.7588	0.9957	1634	+ 0.09	-0.9212
WinMine	κ		ℓ	L	r^{EP}	$\|E\|$	T (min)	L_{CV}
	0.01 (Default)		-1.0218	-0.9918	0.2868	272	57	-1.0063
	0.50		-0.9916	-0.9373	0.4944	627	115	-0.9683
	0.99		-0.9644	-0.7744	0.9064	1528	229	-0.9239
	0.999		-0.9638	-0.7591	0.9468	1616	235	-0.9224
	1.00		-0.9636	-0.7554	0.9494	1641	268	-0.9220
Null Model			-1.1079	-0.1018	---	0	0	-1.0987

References

[1] Buntine, W., "A guide to the literature on learning probabilistic networks from data," *IEEE Trans KDE*, 8(2): 195-210, 1996.

[2] Cheng, J., Bell, DA, Liu, W., "Learning belief networks from data: an information theory based approach," *6th ACM Int Conf on Information and Knowledge Management*, 1997.

[3] Chickering, D., Geiger, D., and Heckerman, D., "Learning Bayesian Networks is NP-Hard," *MSR-TR-94-17*, Microsoft Research, 1994.

[4] Chickering, D.M., "The WinMine toolkit," *MSR-TR-2002-103*, Microsoft Research, 2002.

[5] Cooper, G.F., and Yoo, C., "Causal discovery from a mixture of experimental and observational data," *UAI-1999*: 116-125, 1999.

[6] Cooper, G.F., and Herskovits, E., "A Bayesian method for the induction of probabilistic networks from data," *Machine Learning, 9*: 309-347, 1992.

[7] Friedman, N., and Koller, D., "Being Bayesian about network structure: a Bayesian approach to structure discovery in Bayesian networks," *Machine Learning*, 2002.

[8] Heckerman, D., "A tutorial on learning with Bayesian networks," in M.I. Jordan (Ed.) *Learning in Graphical Models*: 301-354, MIT Press, 2000.

[9] Heckerman, D., Chickering, D.M., Meek, C., Rounthwaite, R., and Kadie, C. "Dependency networks for inference, collaborative filtering, and data visualization," *J. Machine Learning Research*, *1*: 49-75, 2000.

[10] Hughe, T.R., et al. "Functional discovery via a compendium of expression profiles," *Cell, 102*: 109-126, 2000.

[11] Hulten, G., Chickering, D.M., and Heckerman, D., "Learning Bayesian networks from dependency networks: a preliminary study," *AI & Statistics 2003*: 54-61, 2003.

[12] Larranaga, P., Poza, M., Yurramendi, Y., Murga, R.H., and Kuijpers, C.M., "Structural learning of Bayesian networks by genetic algorithms: a performance analysis of control parameters," *IEEE Trans. PAMI, 18*: 912-926, 1996.

[13] Pearl, J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, San Mateo, CA: Morgan Kaufmann, 1988.

[14] Pe'er, D., Regev, A., Elidan, G., and Friedman, N., "Inferring subnetworks from perturbed expression profiles," *Bioinformatics, 17*: 215S-224S, 2001.

[15] Peng, H.C., Herskovits E, and Davatzikos C. "Bayesian clustering methods for morphological analysis of MR images," IEEE Int'l Symp on Medical Imaging: Macro to Nano: 875-878, 2002.

[16] Peng, H.C., and Ding, C., "An efficient algorithm for detecting modular regulatory networks using Bayesian subnets of co-regulated genes," LBNL Technical Report 53734, Aug 2003.