Finding the optimal bayesian network given a constraint graph

Jacob M Schreiber Corresp., [1] , **William S Noble** [2]

[1] Department of Computer Science, University of Washington, Seattle, Washington, United States
[2] Department of Genome Science, University of Washington, Seattle, Washington, United States

Corresponding Author: Jacob M Schreiber
Email address: jmschr@cs.washington.edu

Despite recent algorithmic improvements, learning the optimal structure of a Bayesian network from data is typically infeasible past a few dozen variables. Fortunately, domain knowledge can frequently be exploited to achieve dramatic computational savings, and in many cases domain knowledge can even make structure learning tractable. Several methods have previously been described for representing this type of structural prior knowledge, including global orderings, super-structures, and constraint rules. While super-structures and constraint rules are flexible in terms of what prior knowledge they can encode, they achieve savings in memory and computational time simply by avoiding considering invalid graphs. We introduce the concept of a "constraint graph" as an intuitive method for incorporating rich prior knowledge into the structure learning task. We describe how this graph can be used to reduce the memory cost and computational time required to find the optimal graph subject to the encoded constraints, beyond merely eliminating invalid graphs. In particular, we show that a constraint graph can break the structure learning task into independent subproblems even in the presence of cyclic prior knowledge. These subproblems are well suited to being solved in parallel on a single machine or distributed across many machines without excessive communication cost.

Finding the Optimal Bayesian Network Given a Constraint Graph

Jacob Schreiber
Department of Computer Science
University of Washington
Seattle, WA 98195
`jmschr@cs.washington.edu`

William Stafford Noble
Department of Genome Science
University of Washington
Seattle, WA, 98195
`william-noble@uw.edu`

Abstract

Despite recent algorithmic improvements, learning the optimal structure of a Bayesian network from data is typically infeasible past a few dozen variables. Fortunately, domain knowledge can frequently be exploited to achieve dramatic computational savings, and in many cases domain knowledge can even make structure learning tractable. Several methods have previously been described for representing this type of structural prior knowledge, including global orderings, super-structures, and constraint rules. While super-structures and constraint rules are flexible in terms of what prior knowledge they can encode, they achieve savings in memory and computational time simply by avoiding considering invalid graphs. We introduce the concept of a “constraint graph" as an intuitive method for incorporating rich prior knowledge into the structure learning task. We describe how this graph can be used to reduce the memory cost and computational time required to find the optimal graph subject to the encoded constraints, beyond merely eliminating invalid graphs. In particular, we show that a constraint graph can break the structure learning task into independent subproblems even in the presence of cyclic prior knowledge. These subproblems are well suited to being solved in parallel on a single machine or distributed across many machines without excessive communication cost.

1 Introduction

Bayesian networks are directed acyclic graphs (DAGs) in which nodes correspond to random variables and directed edges represent dependencies between these variables. Conditional independence between a pair of variables is represented as the lack of an edge between the two corresponding nodes. The parameters of a Bayesian network are typically simple to interpret, making such networks highly desirable in a wide variety of application domains that require model transparancy.

Frequently, one does not know the structure of the Bayesian network beforehand, making it necessary to learn the structure directly from data. The most intuitive approach to the task of Bayesian network structure learning (BNSL) is “search-and-score," in which one iterates over all possible DAGs and chooses the one that optimizes a given scoring function. Recent work has described methods that find the optimal Bayesian network structure without explicitly considering all possible DAGs (Malone et al., 2011; Yuan et al., 2011; Fan et al., 2014; Jaakkola et al., 2003), but these methods are still infeasible for more than a few dozen variables. In practice, a wide variety of heuristics are often employed for larger datasets. These algorithms, which include branch-and-bound (Suzuki, 1996), Chow-Liu trees (Chow & Liu, 2003), optimal reinsertion (Moore & Wong, 2003), and hill-climbing (Tsamardinos et al., 2006), typically attempt to efficiently identify a structure that captures the majority of important dependencies.

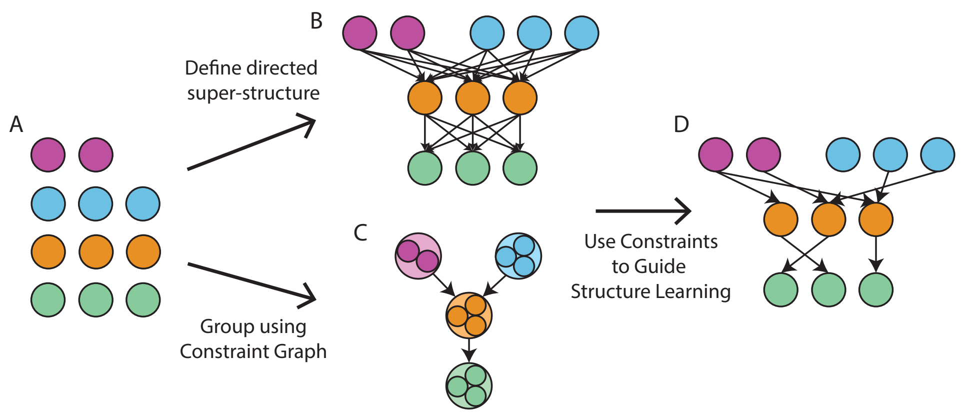
A
B
C
D
Define directed super-structure
Group using Constraint Graph
Use Constraints to Guide Structure Learning

Figure 1: **A constraint graph grouping variables.** (a) We wish to learn a Bayesian network over 11 variables. The variables are colored according to the group that they belong to, which is defined by the user. These variables can either (b) be organized into a directed super structure or (c) grouped into a constraint graph to encode equivalent prior knowledge. Both graphs define the superset of edges which can exist, but the constraint graph uses far fewer nodes and edges to encode this knowledge. (d) Either technique can then be used to guide the BNSL task to learn the optimal Bayesian network given the constraints.

In many applications, the search space of possible network structures can be reduced by taking into account domain-specific prior knowledge (Gamberoni1 et al., 2005; Zuo & Kita, 2012; Schneiderman, 2004; Zhou & Sakane, 2003). A simple method is to specify an ordering on the variables and require that parents of a variable must precede it in the ordering (Cooper & Herskovits, 1992). This representation leads to tractable structure learning because identifying the parent set for each variable can be carried out independently from the other variables. Unfortunately, prior knowledge is typically more ambiguous than knowing a full topological ordering and may only exist for some of the variables. A more general approach to handling prior knowledge is to employ a "super-structure," i.e., an undirected graph that defines the super-set of edges defining valid learned structures, forbidding all others (Perrier et al., 2008). This method has been fairly well studied and can also be used as a heuristic if defined through statistical tests instead of prior knowledge. A natural extension of the undirected super-structure is the directed super-structure (Ordyniak & Szeider, 2013), but to our knowledge the only work done on directed super-structures proved that an acyclic directed super-structure is solvable in polynomial time. An alternate, but similar, concept is to define which edges must or cannot exist as a set of rules (Campos & Ji, 2011). However, these rule-based techniques do not specify how one would exploit the constraints to reduce the computational time past simply skipping over invalid graphs.

We propose the idea of a "constraint graph" as a method for incorporating prior information into the BNSL task. A constraint graph is a directed graph where each node represents a set of variables in the BNSL problem and edges represent which variables are candidate parents for which other variables. The primary advantage of constraint graphs versus other methods is that the structure of the constraint graph can be used to achieve savings in both memory cost and computational time beyond simply eliminating invalid structures. This is done by breaking the problem into independent subproblems even in the presence of cyclic prior knowledge. An example of this cyclic prior knowledge is not knowing anything about how a group of variables relate to each other. In addition, constraint graphs are visually more intuitive than a set of written rules while also typically being simpler than a super-structure, because constraint graphs are defined over sets of variables instead of the original variables themselves. This intuition, combined with automatic methods for identifying parallelizable subproblems, makes constraint graphs easy for non-experts to define and use without requiring them to know the details of the structure learning task. This technique is similar to work done by Fan et al. (2014), where the authors describe the same computational gains through the identification of "potentially optimal parent sets." One difference is that Fan et al. define the constraints on individual variables instead of on sets on variables, as this work does. Given that two types of graphs will be discussed throughout this paper, the Bayesian network we are attempting to learn and the constraint graph, we will use the terminology “variable" exclusively in reference to the Bayesian network and “node" exclusively in reference to the constraint graph.

2 Constraint Graphs

A constraint graph is a directed graph in which nodes contain disjoint sets of variables from the BNSL task, and edges indicate which sets of variables can serve as parents to which other sets of variables. A self-loop in the constraint graph indicates that no prior knowledge is known about the relationship between variables in that node, whereas a lack of a self-loop indicates that no variables in that particular node can serve as parents for another variable in that node. Thus, the naive BNSL task can be represented as a constraint graph consisting of a single node with a self-loop. A constraint graph can be thought of as a way to group the variables (Fig. 1a), define relationships between these groups (Fig. 1c), and then guide the BNSL task to efficiently find the optimal structure given these constraints (Fig. 1d). In contast, a directed super-structure defines all possible edges that can exist in accordance with the prior knowledge (Fig. 1b). Typically, a directed super-structure is far more complicated than the equivalent constraint graph. Cyclic prior knowledge can be represented as a simple cycle in the constraint graph, such that the variables in node A draw their parents solely from node B, and B from A.

Any method for reducing computational time through prior knowledge exploits the “global parameter independence property” of BNSL. Briefly, this property states that the optimal parents for a variable are independent of the optimal parents for another variable given that the variables do not form a cycle in the resulting Bayesian network. This acyclicity requirement is typically computationally challenging to determine because a cycle can involve more variables than the ones being directly considered, such as a graph which is simply a directed loop over all variables. However, given an acyclic constraint graph or an acyclic directed super-structure, it is impossible to form a cycle in the resulting structure; hence, the optimal parent set for each variable can be identified independently from all other variables. A convenient property of constraint graphs, and one of their advantages relative to other methods, is that independent subproblems can be found through global parameter independence even in constraint graphs which contain cycles. We describe in Section 3.2 the exact algorithm for finding optimal parent sets for each case one can encounter in a constraint graph. Briefly, the constraint graph is first broken up into its strongly connected components (SCCs) that identify which variables can have their parent sets found independently from all other variables (“solving a component”) without the possibility of forming a cycle in the resulting graph. Typically these SCCs will be single nodes from the constraint graph, but may be comprised of multiple nodes if cyclic prior knowledge is being represented. In the case of an acyclic constraint graph, all SCCs will be single nodes, and in fact each variable can be optimized without needing to consider other variables, in line with theoretical results from Ordyniak & Szeider (2013). In addition to allowing these problems to be solved in parallel, this breakdown suggests a more efficient method of sharding the data in a distributed learning context. Specifically, one can assign an entire SCC of the constraint graph to a machine, including all columns of data corresponding to the variables in that SCC and all variables in nodes which are parents to nodes in the SCC. Given that all subproblems which involve this shard of the data are contained in this SCC of the constraint graph, there will never be duplicate shards and all tasks involving a shard are limited to the same machine. The concept of identifying SCCs as independent subproblems has also been described in Fan et al. (2014).

It is possible to convert any directed super-structure into a constraint graph and vice-versa. To convert from a directed super-structure to a constraint graph, one must tabulate the unique parent and children sets a variable can have. All variables with the same parent and children sets can be grouped together into a node in the constraint graph. Edges then connect these sets based on the shared parent sets specified for each node. To convert from a constraint graph to a directed super-structure one would simply draw, for each node, an edge from all variables in the current node to all variables in the node’s children. We suggest, however, that constraint graphs are the more intuitive method both due to their simpler representation and ease of extracting computational benefits from the task.

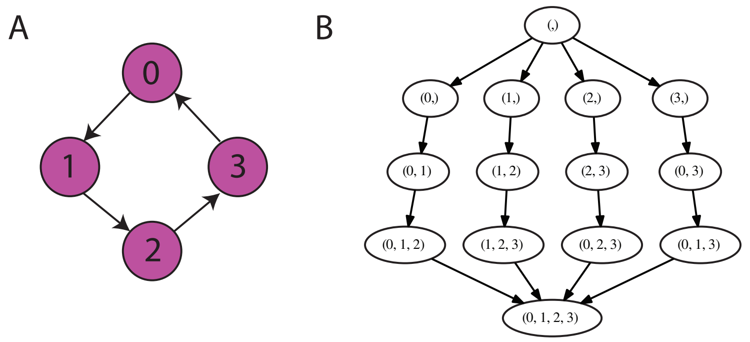

Figure 2: **An example of a constraint graph and resulting order graph** (a) A constraint graph is defined as a cycle over four nodes with each node containing a single variable. (b) The resulting order graph during the BNSL task. It is significantly sparser than the typical BNSL task because after choosing a variable to start the topological ordering the remaining variables must be added in the order defined by the cycle.

3 Methods

3.1 Bayesian Network Structure Learning

Although solving a component in a constraint graph can be accomplished by a variety of algorithms, we assume for this paper that one is using some variant of the dynamic programming algorithm proposed by Malone et al. (2011). We briefly review that algorithm here.

The goal of the algorithm is to identify the optimal Bayesian network defined over the set of variables without having to repeat any calculations and without having to use excessive memory. This is done by defining additional graphs, the parent graphs and the order graph. We will refer to each node in these graphs as "entries" to distinguish them from the constraint graph and the learned Bayesian network. A parent graph is defined for each variable and consists of all combinations of other variables along with the dynamically calculated optimal subset of parents given each possible set of parents and the associated score. The order graph consists of all combinations of all variables ordered in a lattice, where edges store the score associated with adding a given variable to the topological sort and its optimal parent set. Each path from the empty root node to the leaf node containing the full set of variables encodes the optimal network given a topological sort of the variables, and the shortest path encodes the optimal network. This data structure reduces the time required to find the optimal Bayesian network from super-exponential in the number of variables to simply exponential in the number of variables without the need to keep a large cache of values.

3.2 Solving a Component of the Constraint Graph

The strongly connected components of a constraint graph can be identified using Tarjan's algorithm (Tarjan, 1971). Each SCC corresponds to a subproblem of the constraint graph and can be solved independently. In many cases the SCC will be a single node of the constraint graph, because prior knowledge is typically not cyclic. In general, the SCCs of a constraint graph can be solved in any order due to the global parameter independence property.

The algorithm for solving an SCC of a constraint graph is a straightforward modification of the dynamic programming algorithm described above. Specifically, parent graphs are created for each variable in the SCC but defined only over the union of possible parents for that variable. Consider the case of a simple, four-node cycle with no self-loops such that $W \to X \to Y \to Z \to W$. A parent graph is defined for each variable in $W \cup X \cup Y \cup Z$ but only over valid parents. For example, the parent graph for X_1 would be over only variables in W. Then, an order graph is defined with entries that violate the edge structure of the constraint graph filtered out. The first layer of the order graph would be unchanged with only singletons, but the second layer would prohibit entries with two variables from the same layer because there are no valid orderings in which X_i is a parent of X_j, and would prohibit entries in which a variable W is joined with a variable of Y. One can identify valid

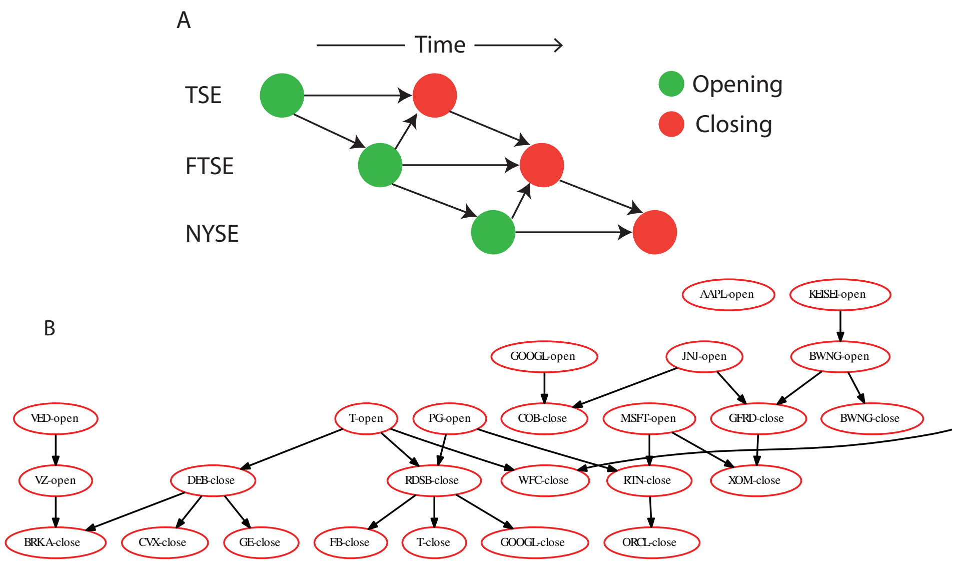

Figure 3: **A section of the learned Bayesian network of the global stock market.** (a) The constraint graph contains six nodes, the opening and closing prices for each of the three markets. These are connected such that the closing prices in a market depend on the opening prices but also the most recent international activity. (b) Since this is another example of an acyclic constraint graph, we can easily parallelize for near-linear speed gains. (c) The most connected subset of stocks from the learned network covering 25 variables.

entries by taking the entries of a previous layer and iterating over each variable present, adding all valid parents for that variable which are not already present in the set.

A simple example illustrating the algorithm is a constraint graph made up of a four node cycle where each node contains only a single variable (Fig 2a). The parent graphs defined for this would consist solely of two entries, the null entry and the entry corresponding to the only valid parent. The first layer of the order graph would be all variables as previously (2b). However, once a variable is chosen to start the topological ordering the order of the remaining variables is fixed because of the constraints, producing a far simpler lattice.

This algorithm has five natural cases, which we consider separately.

One node, no parents, no self loop: The variables in this node contain no parents, so nothing needs to be done to find the optimal parent sets given the constraints.

One node, no parents, self loop: This is equivalent to exact BNSL with no prior knowledge. In this case, the previously proposed dynamic programming algorithm is used to identify the optimal structure of the subnetwork containing only variables in this node.

One node, one or more parent nodes, no self loop: In this case it is impossible for a cycle to be formed in the resulting Bayesian network regardless of optimal parent sets, so we can justify solving every variable in this node independently by the global parameter independence property. There is also no need to store the full parent graphs because the optimal scoring entry in the parent graphs is the resulting optimal parent set.

One node, one or more parents, self loop: Initially, one may think that solving this SCC could involve taking the union of all variables from all involved nodes, running exact BNSL over the full set, and simply discarding the parent sets learned for the variables not in the currently considered node. However, in the same way that one should not handle prior knowledge by learning the optimal graph over all variables and discarding edges which offend the prior knowledge, one should not do the same in this case. Instead, a modification to the dynamic programming algorithm itself can be made to restrict the parent sets on a variable-by-variable basis. For simplicity, we define the variables in the current node of the constraint graph as X and the union of all variables in the parent nodes in the constraint graph as Y. We begin by setting up an order graph, as usual defined over X. We then

	Exact				Constraint Graph			
Variables	PGN	OGN	OGE	Time (s)	PGN	OGN	OGE	Time (s)
4	2304	512	2304	0.080	1024	16	32	0.033
8	53248	8192	53248	1.30	32768	256	1024	0.545
12	1114112	131072	1114112	27.03	786432	4096	24576	9.56
Parents								
4	2304	512	2304	0.087	1280	32	80	0.045
8	53248	8192	53258	1.401	20480	32	80	0.356
12	1114112	131072	1114112	27.22	327680	32	80	4.01

Table 1: **Algorithm comparison on a node with a self loop and other parents.** The exact algorithm and the constrained algorithm proposed here were on a SCC comprosied of a main node with a self loop and one parent node. Shown are the results of increasing the number of variables in the main node while keeping the variables in the parent node steady at 5, and the results of increasing the number of variables in the parent node while keeping the number of variables in the main node constant. For both algorithm we show the number of nodes across all parent graphs (PGN), the number of nodes in the order graph (OGN), the number of edges in the order graph (OGE) and the time to compute.

add Y to each node in the order graph such that the root node now is now comprised of Y instead of the empty set and the leaf node is comprised of $X \cup Y$ instead of just X. Because the primary purpose of the order graph is to identify the optimal parent sets that do not form cycles, this addition is intuitive because it is impossible to form a cycle by including any of the variables in Y as parents for any of the variables in X. In other words, if one attempted to find the optimal topological ordering over $X \cup Y$ it would always begin with the variables in Y but would be invariant to the ordering of Y. Parent graphs are then created for all variables in X but are defined over the set of all variables in $X \cup Y$, because that is the full set of parents that the variables could be drawn from. This restriction allows the optimal parents for each variable in X to be identified without wasting time considering what the parent set for variables in Y should be, or potentially throwing away the optimal graph because of improper edges leading from a variable in Y to a variable in X.

Multiple nodes: The algorithm as presented initially is used to solve an entire component at the same time.

4 Results

While it is intuitive how a constraint graph provides computational gains by splitting the structure learning task into subproblems, we have thus far only alluded to the idea that prior knowledge can provide efficiencies past that. In this section we examine the computational gains achieved in the three non-trivial cases of the algorithm presented in Section 3.2.

4.1 Acyclic Constraint Graph

First, we examine the computational benefits of an acyclic constraint graph modeling the global stock market. In particular, we want to identify for each stock which other stocks are predictive to its performance. We chose to do this by learning a Bayesian network over the opening and closing prices of 54 top performing stocks from the New York Stock Exchange (NYSE) in the United States, the Tokyo Stock Exchange (TSE) in Japan, and the Financial Times Stock Exchange (FTSE) in England. Learning a Bayesian network over all 108 variables is clearly infeasible, so we encode in our constraint graph some common-sense restrictions (Fig. 3a). Specifically, opening and closing prices for the same market are grouped into separate nodes, for a total of six nodes in the constraint graph. There are no self-loops because the opening price of one stock does not influence the opening price of another stock. Naturally, the closing prices of one group of stocks are influenced by the opening price of the stocks from the same market, but they are also influenced by the opening or closing prices of any markets which opened or closed in the meantime. For instance, the TSE closes after the FTSE opens, so the FTSE opening prices have the opportunity to influence the TSE closing prices. However, the TSE closes before the NYSE opens, so the NYSE cannot influence those stock prices.

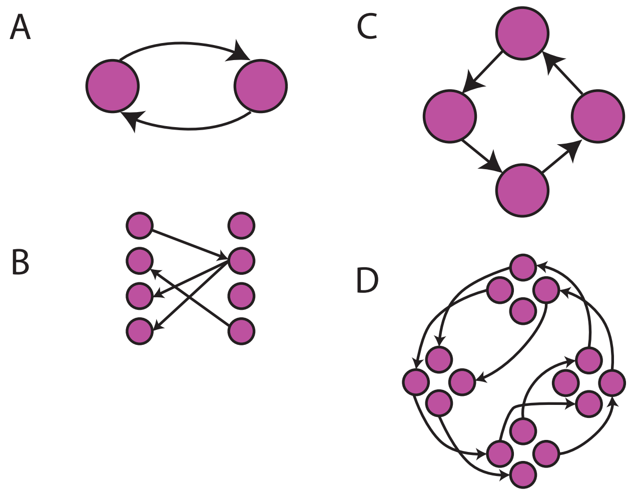

Figure 4: **Cyclic constraint graphs** (a) This constraint graph is comprised of a simple two node cycle with each node containing four variables. (b) The learned Bayesian network on random data where some variables were forced to identical values. Each circle here corresponds to a variable in the resulting Bayesian network instead of a node in the constraint graph. There were multiple possible cycles which could have been formed but the constraint graph prevented that from occuring. (c) This constraint graph now encodes a four node cycle each with four variables. (d) The learned Bayesian network on random data with two distinct loops of identical values forced. Again, no loops are formed.

The resulting Bayesian network has some interesting connections (Fig. 3b). For example, the opening price of Microsoft influences the closing price of Raytheon, and the closing price of Debenhams plc, a British multinational realtor, influences the closing price of GE. In addition, there were some surprising and unexplained connections, such as Google and Johnson & Johnson influencing the closing price of Cobham plc, a British defense firm. Given that this example is primarily to illustrate the types of constraints a constraint graph can easily model, we suggest caution in thinking too deeply about these connections.

It took only $\sim$35 seconds on a computer with modest hardware to run BNSL over 250 samples. If we set the maximum number of parents to three, which is the empirically determined maximum number of parents, then it only takes $\sim$2 seconds to run. In contrast it would be infeasible to run the exact BNSL algorithm on even half the number of variables considered here.

4.2 Self-Loops And Parents

We then turn to the case where the strongly connected component is a main node with a self loop and a parent node. Because an order graph is defined only over the variables in the main node its size is invariant to the number of variables in the parent node, allowing for speed improvements when it comes to calculating the shortest path. In addition, parent graphs are only defined for variables in the parent set, and so while they are not smaller than the ones in the exact algorithm, there are fewer. We compare the computational time and complexity of the underlying order and parent graphs between the exact algorithm over the full set of variables and the modified algorithm based on a constraint graph (Table. 1) We note that in all cases there are significant speed improvements and simpler graphs but that there are particularly encouraging speed improvements when the number of variables in the main node are increased. This suggests that it is always worth the time to identify which variables can be moved from a node with a self loop to a separate node.

4.3 Cyclic Constraint Graphs

Lastly, we consider constraint graphs that encode cyclic prior knowledge. We visually inspect the results from cyclic constraint graphs to ensure that they do not produce cyclic Bayesian networks

	Exact				Exact			
Variables	PGN	OGN	OGE	Time (s)	PGN	OGN	OGE	Time (s)
1	32	16	32	0.005	8	14	16	0.005
2	1024	256	1024	0.036	32	186	544	0.014
3	24576	4096	24576	0.611	96	3086	16032	0.320
4	524288	65536	525288	14.0	256	54482	407328	7.12
Nodes								
2	192	64	192	0.111	48	56	150	0.008
4	24576	4096	24576	0.634	96	3086	16032	0.217
6	2359296	262144	2359296	60.9	144	168068	1307358	26.12
Samples								
100	24576	4096	24576	0.357	96	3086	16032	0.311
1000	–	–	–	0.615	–	–	–	0.211
10000	–	–	–	2.670	–	–	–	0.357
100000	–	–	–	243.9	–	–	–	10.41

Table 2: **Algorithm comparison on a cyclic constraint graph.** The exact algorithm and the constrained algorithm proposed here were run for four node cycles with differing numbers of variables, cycles with different numbers of nodes but three variables per node, and differing numbers of samples for a four-node three-variable cycle. All experiments with differing numbers of variables or nodes were run on 1000 randomly generated samples. Shown for both algorithms are the number of nodes across all parent graphs (PGN), the number of nodes in the order graph (OGN), the number of edges in the order graph (OGE) and the time to compute. Since the number of nodes does not change as a function of samples those values are not repeated in the blank cells.

even when the potential exists. Two separate constraint graphs are inspected, a two node cycle and a four node cycle (Fig. 4a/c). In each case two cycles of variables are set to identical values in otherwise random data that may cause a careless algorithm to produce a cyclic Bayesian network. However, by jointly solving all nodes cycles are avoided while dependencies are still captured (Fig. 4b/d).

We then compare the exact algorithm without constraints to the use of an appropriate constraint graph in a similer manner as before (Table. 2). This is done first for four node cycles where we increase the number of variables in each node of the constraint graph and then for increasing sized cycles with three variables per node. The exact algorithm likely produces structures that are invalid according to the constraints and so this comparison is done solely to highlight that efficiencies are gained by considering the constraints. In each case using a constraint graph yields simpler parent and order graphs and the computational time is significantly reduced. The biggest difference is in the number of nodes in the parent graphs, as the constraints place significant limitations on which variables are allowed to be parents for which other variables. Since the construction of the parent graph is the only part of the algorithm which considers the dataset itself it is unsurprising that significant savings are achieved for larger datasets when much smaller parent graphs are used.

5 Discussion

Constraint graphs are a flexible way of encoding into the BNSL task prior knowledge concerning the relationships among variables. The graph structure can be exploited to identify potentially massive computational gains, and acyclic constraint graphs make problems tractable which would be infeasible to solve without constraints. This is particularly useful in cases where there are both a great number of variables and many constraints present from prior knowledge. We anticipate that the automatic manner in which parallelizable subtasks are identified in a constraint graph will be of particular interest given the recent increase in availability of distributed computing.

Although the networks learned in this paper are discrete, the same principles can be applied to all types of Bayesian networks. Because the constraint graph represents only a restriction in the parent set on a variable-by-variable basis, the same algorithms that are used to learn linear Gaussian or hybrid networks can be seamlessly combined with the idea of a constraint graph. In addition, most of the approximation algorithms which have been developed for BNSL can be modified to take into account constraints because these algorithms simply encode a limitation on the parent set for each variable.

One could extend constraint graphs in several interesting ways. The first is to assign weights to edges so that the weight represents the prior probability that the variables in the parent set are parents of the variables in the child set, perhaps as pseudocounts to take into account when coupled with a Bayesian scoring function. A second way is to incorporate “hidden nodes” that are variables which model underlying, onobserved phenomena and can be used to reduce the parameterization of the network. Several algorithms have been proposed for learning the structure of a Bayesian network given hidden variables (Elidan et al., 2001; Elidan & Friedman, 2005; Friedman, 1997). Modifying these algorithms to obey a constraint graph seems like a promising way to incorporate restrictions on this difficult task. A final way may be to encode ancestral relationships instead of direct parent relationships, indicating that a given variable must occur at some point before some other variable in the topological ordering.

Acknowledgments

We would like to acknowledge Maxwell Libbrecht, Scott Lundberg, and Brandon Malone for many useful discussions and comments on drafts of the paper.

References

Campos, C.P. and Ji, Q. Efficient structure learning of bayesian networks using constraints. *Journal of Machine Learning Research*, pp. 663–689, 2011.

Chow, C. and Liu, C. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, pp. 462 – 467, 2003.

Cooper, G. and Herskovits, E. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

Elidan, G. and Friedman, N. Learning hidden variable networks: The information bottleneck approach. *Journal of Machine learning Research*, 6:81–127, 2005.

Elidan, G., Lotner, N., Friedman, N., and Koller, D. Discovering hidden variables: A structure-based approach. *Advances in Neural Information Processing Systems (NIPS) 13*, pp. 479–485, 2001.

Fan, X., Malone, B.M., and Yuan, C. Finding optimal bayesian network structures with constraints learned from data. *UAI'14 Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, pp. 200–209, 2014.

Friedman, N. Learning belief networks in the presence of missing values and hidden variables. *ICML '97 Proceedings of the Fourteenth International Conference on Machine learning*, pp. 125–133, 1997.

Gamberoni1, G., Lamma1, E., Riguzzi1, F., Storari1, S., and Volinia, S. Bayesian networks learning for gene expression datasets. pp. 109–120, 2005.

Jaakkola, T., Sontag, D., Globerson, A., and Meila, M. Learning bayesian network structure using lp relaxations. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS-10)*, pp. 358–365, 2003.

Malone, B.M., Yuan, C., and Hansen, E.A. Memory-efficient dynamic programming for learning optimal bayesian networks. *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

Moore, A. and Wong, W. Optimal reinsertion: A new search operator for accelerated and more accurate bayesian network structure learning. pp. 552–559, 2003.

Ordyniak, S. and Szeider, S. Parameterized complexity results for exact bayesian network structure learning. *Journal of Artificial Intelligence Research*, 46:263–302, 2013.

Perrier, E., Imoto, S., and Miyano, S. Finding optimal bayesian network given a super-structure. *Journal of Machine Learning Research*, pp. 2251–2286, 2008.

Schneiderman, Henry. Learning a restricted bayesian network for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June 2004.

Suzuki, J. Learning bayesian belief networks based on the minimum description length principle: An efficient algorithm using the b b technique. *Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96)*, 1996.

Tarjan, R. Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 2:146–160, 1971.

Tsamardinos, I., Brown, L.E., and Aliferis, C.F. The max-min hill-climbing bayesian network structure learning algorithm. *Machine Learning*, pp. 462 – 467, 2006.

Yuan, C., Malone, B.M., and Wu, X. Learning optimal bayesian networks using a* search. *IJCAI proceedings-international joint conference on artificial intelligence*, 2011.

Zhou, H. and Sakane, S. Learning bayesian network structure from environment and sensor planning for mobile robot localization. *Proceedings of IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, MFI2003.*, 2003.

Zuo, Y. and Kita, E. Up/down analysis of stock index by using bayesian network. *Engineering Management Research*, 2012.