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

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

1 Introduction

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

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

36 In many applications, the search space of possible network structures can be reduced by taking into
37 account domain-specific prior knowledge (Gamberoni1 et al., 2005; Zuo & Kita, 2012; Schneiderman,
38 2004; Zhou & Sakane, 2003). A simple method is to specify an ordering on the variables and
39 require that parents of a variable must precede it in the ordering (Cooper & Herskovits, 1992).
40 This representation leads to tractable structure learning because identifying the parent set for each
41 variable can be carried out independently from the other variables. Unfortunately, prior knowledge is
42 typically more ambiguous than knowing a full topological ordering and may only exist for some of the
43 variables. A more general approach to handling prior knowledge is to employ a "super-structure," i.e.,
44 an undirected graph that defines the super-set of edges defining valid learned structures, forbidding
45 all others (Perrier et al., 2008). This method has been fairly well studied and can also be used as a
46 heuristic if defined through statistical tests instead of prior knowledge. A natural extension of the
47 undirected super-structure is the directed super-structure (Ordyniak & Szeider, 2013), but to our
48 knowledge the only work done on directed super-structures proved that an acyclic directed super-
49 structure is solvable in polynomial time. An alternate, but similar, concept is to define which edges
50 must or cannot exist as a set of rules (Campos & Ji, 2011). However, these rule-based techniques
51 do not specify how one would exploit the constraints to reduce the computational time past simply
52 skipping over invalid graphs.

53 We propose the idea of a "constraint graph" as a method for incorporating prior information into the
54 BNSL task. A constraint graph is a directed graph where each node represents a set of variables in the
55 BNSL problem and edges represent which variables are candidate parents for which other variables.
56 The primary advantage of constraint graphs versus other methods is that the structure of the constraint
57 graph can be used to achieve savings in both memory cost and computational time beyond simply
58 eliminating invalid structures. This is done by breaking the problem into independent subproblems
59 even in the presence of cyclic prior knowledge. An example of this cyclic prior knowledge is
60 identifying two groups of variables that can draw parents only from each other, similar to a biparte
61 graph. It can be difficult to identify the best parents for each variable that does not result in a cycle in
62 the learned structure. In addition, constraint graphs are visually more intuitive than a set of written
63 rules while also typically being simpler than a super-structure, because constraint graphs are defined
64 over sets of variables instead of the original variables themselves. This intuition, combined with
65 automatic methods for identifying parallelizable subproblems, makes constraint graphs easy for
66 non-experts to define and use without requiring them to know the details of the structure learning
67 task. This technique is similar to work done by Fan et al. (2014), where the authors describe the same
68 computational gains through the identification of "potentially optimal parent sets." One difference is
69 that Fan et al. define the constraints on individual variables instead of on sets on variables, as this
70 work does. By defining the constraints on sets of variables instead of individual ones, one can identify
71 further computational gains when presented with cyclic prior knowledge. Given that two types of
72 graphs will be discussed throughout this paper, the Bayesian network we are attempting to learn and
73 the constraint graph, we will use the terminology "variable" exclusively in reference to the Bayesian
74 network and "node" exclusively in reference to the constraint graph.

2 Constraint Graphs

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

89 Any method for reducing computational time through prior knowledge exploits the "global parameter
90 independence property" of BNSL. Briefly, this property states that the optimal parents for a variable
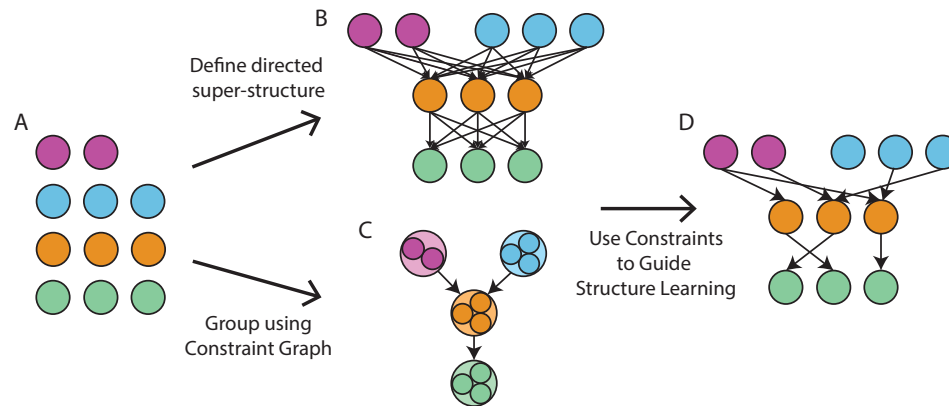
Figure 1: **A constraint graph grouping variables.** (a) We wish to learn a Bayesian network over 11 variables. The variables are colored according to the group that they belong to, which is defined by the user. These variables can either (b) be organized into a directed super structure or (c) grouped 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.

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 though it is far simpler to go from a constraint graph to a directed super-structure. To convert from a directed super-structure to a constraint graph, one must first identify all strongly connected components that are more than a single variable. All variables in a strongly connected component can be put into the same node in a constraint graph that contains a self loop. Then, one would tabulate the unique parent and children sets a variable can have. All variables outside of the previously identified strongly connected components 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. In the situation where a node in the constraint graph can draw parents from only a subset of the variables in a node created by the identification of the strongly connected components, the node must

125 be broken into two nodes that both have self loops and loops connecting to each other to allow for
126 only a subset of those variables to serve as a parent for another node. In contrast, to convert from a
127 constraint graph to a directed super-structure one would simply draw, for each node, an edge from
128 all variables in the current node to all variables in the node's children. We suggest that constraint
129 graphs are the more intuitive method both due to their simpler representation and ease of extracting
130 computational benefits from the task.

3 Methods

3.1 Bayesian Network Structure Learning

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

137 The goal of the algorithm is to identify the optimal Bayesian network defined over the set of variables
138 without having to repeat any calculations and without having to use excessive memory. This is done
139 by defining additional graphs, the parent graphs and the order graph. We will refer to each node in
140 these graphs as "entries" to distinguish them from the constraint graph and the learned Bayesian
141 network. A parent graph is defined for each variable and can be defined as a lattice, where the entries
142 to some layer i correspond to combinations of all other variables of size i. Each entry is connected to
143 the entries in the previous layers that are subsets of that entry such that (X_1, X_2) would be connected
144 to both X_1 and X_2. For each entry, the score of the variable is calculated using the parents in the
145 entry and compared to the scores held in the parent entries, recording only the best scoring value
146 and parent set amongst them. These entries then hold the dynamically calculated best parent set
147 and associated score, allowing for constant time lookups later on of the best parent set given a set
148 of possible parents. The order graph is structured in the same manner as the parent graphs except
149 over all variables. In contrast with the parent graphs, it is the edges that store useful information
150 in the form of the score associated with adding a given variable to the set of seen variables stored
151 in the entry and the parent set that yields this score Each path from the empty root node to the leaf
152 node containing the full set of variables encodes the optimal network given a topological sort of the
153 variables, and the shortest path encodes the optimal network. This data structure reduces the time
154 required to find the optimal Bayesian network from $O(n2^{n(n-1)})$ time in the number of variables to
155 $O(n2^n)$ time in the number of variables without the need to keep a large cache of values.

156 Structure learning is flexible with respect to the score function used to identify the optimal graph.
157 There are many score functions that typically aim to penalize the log likelihood of the data by
158 the complexity of the graph to encourage sparser structures. These usually come in the form of
159 Bayesian score functions, such as Bayesian-Dirichlet (Heckerman et al., 1995), or those derived from
160 information theory, such as minimum description length (MDL) (Suzuki, 1996). Most score functions
161 decompose across variables of a Bayesian network according to the global parameter independence
162 property, such that the score for a dataset given a model is equal to the product of the score of each
163 variable given its parents. While constraint graphs remain agnostic to the specific score function used,
164 we assume that MDL is used as it has several desirable computational benefits. For review, MDL
165 defines the score as the following:

$$MDL(D|M) = P(D|M) - \frac{1}{2}\log(N)|B| \qquad (1)$$

166 where $|B|$ defines the number of parameters in the network. The term "minimum description length"
167 arises from needing $\frac{1}{2}\log(N)$ bits to represent each parameter in the model, making the second
168 term the total number of bits needed to represent the full model. The MDL score function has the
169 convenient property that a variable cannot have more than $\log\left(\frac{n}{log(n)}\right)$ parents given n samples,
170 greatly reducing computational time.

3.2 Solving a Component of the Constraint Graph

172 The strongly connected components of a constraint graph can be identified using Tarjan's algorithm
173 (Tarjan, 1971). Each SCC corresponds to a subproblem of the constraint graph and can be solved
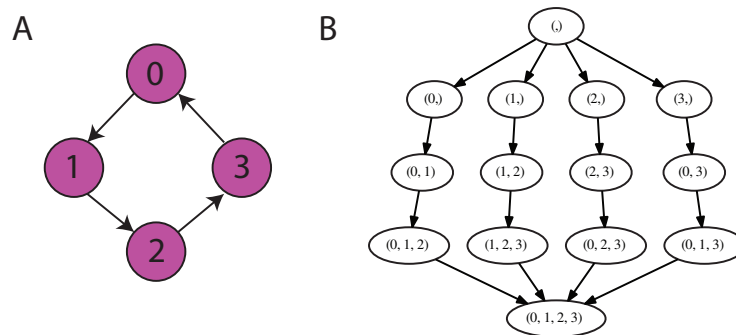
Figure 2: **An example of a constraint graph and resulting order graph** (a) A constraint graph is defined as a cycle over four nodes with each node containing a single variable. (b) The resulting order graph during the BNSL task. It is significantly sparser than the typical BNSL task because after choosing a variable to start the topological ordering the remaining variables must be added in the order defined by the cycle.

174   independently. In many cases the SCC will be a single node of the constraint graph, because prior
175   knowledge is typically not cyclic. In general, the SCCs of a constraint graph can be solved in any
176   order due to the global parameter independence property.

177   The algorithm for solving an SCC of a constraint graph is a straightforward modification of the
178   dynamic programming algorithm described above. Specifically, parent graphs are created for each
179   variable in the SCC but defined only over the union of possible parents for that variable. Consider
180   the case of a simple, four-node cycle with no self-loops such that $W \to X \to Y \to Z \to W$. A
181   parent graph is defined for each variable in $W \cup X \cup Y \cup Z$ but only over valid parents. For example,
182   the parent graph for X_1 would be over only variables in W. Then, an order graph is defined with
183   entries that violate the edge structure of the constraint graph filtered out. The first layer of the order
184   graph would be unchanged with only singletons, but the second layer would prohibit entries with two
185   variables from the same layer because there are no valid orderings in which X_i is a parent of X_j, and
186   would prohibit entries in which a variable W is joined with a variable of Y. One can identify valid
187   entries by taking the entries of a previous layer and iterating over each variable present, adding all
188   valid parents for that variable which are not already present in the set.

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

195   Because constraint graphs can encode a wide variety of different constraints, the complexity of the
196   task depends on the structure of the constraint graph. Broadly, the results from Ordyniak & Szeider
197   (2013) still hold, namely, that acyclic constraint graphs can be solved in quadratic time. As was
198   found in Fan et al. (2014), because each SCC can be solved independently, the time complexity for
199   constraint graphs containing a cycle corresponds to the time complexity of the worst case component.
200   Fortunately, although the complexity of a node engaging in a cycle is still exponential, it is only
201   exponential with respect to the number of variables that node interacts with. Adding additional,
202   equally sized nodes to the constraint graph only causes the algorithm to grow linearly in time and has
203   no additional memory cost if the components are solved sequentially.

204   The algorithm described above has five natural cases and are described below.

205   **One node, no parents, no self loop:** The variables in this node contain no parents, so nothing needs
206   to be done to find the optimal parent sets given the constraints. This naturally takes O(1) time to
207   solve.

208   **One node, no parents, self loop:** This is equivalent to exact BNSL with no prior knowledge. In
209   this case, the previously proposed dynamic programming algorithm is used to identify the optimal

210 structure of the subnetwork containing only variables in this node. This takes $O(n2^n)$ time where n
211 is the number of variables in the node.

212 **One node, one or more parent nodes, no self loop:** In this case it is impossible for a cycle to be
213 formed in the resulting Bayesian network regardless of optimal parent sets, so we can justify solving
214 every variable in this node independently by the global parameter independence property. Doing so
215 results in a significant improvement over applying the algorithm naively, because neither the parent
216 graphs nor the order graph need to be explicitly calculated or stored. The optimal parent set can be
217 calculated without the need for dynamic programming because the optimal topological ordering does
218 not need to be discovered. Because no dynamic programming needs to be done, there is no need to
219 store either the parent or order graphs in memory. This takes $O(nm^k)$ time, where n is the number of
220 variables in the node, m is the number of possible parents, and k is the maximum number of parents
221 that a node can have, in this case set by the MDL algorithm. If k is set to any constant value, then this
222 step requires quadratic time with respect to the number of possible parents and linear with respect to
223 the number of variables in the node.

224 **One node, one or more parents, self loop:** Initially, one may think that solving this SCC could
225 involve taking the union of all variables from all involved nodes, running exact BNSL over the full
226 set, and simply discarding the parent sets learned for the variables not in the currently considered
227 node. However, in the same way that one should not handle prior knowledge by learning the optimal
228 graph over all variables and discarding edges which offend the prior knowledge, one should not do
229 the same in this case. Instead, a modification to the dynamic programming algorithm itself can be
230 made to restrict the parent sets on a variable-by-variable basis. For simplicity, we define the variables
231 in the current node of the constraint graph as X and the union of all variables in the parent nodes in
232 the constraint graph as Y. We begin by setting up an order graph, as usual defined over X. We then
233 add Y to each node in the order graph such that the root node now is now comprised of Y instead
234 of the empty set and the leaf node is comprised of $X \cup Y$ instead of just X. Because the primary
235 purpose of the order graph is to identify the optimal parent sets that do not form cycles, this addition
236 is intuitive because it is impossible to form a cycle by including any of the variables in Y as parents
237 for any of the variables in X. In other words, if one attempted to find the optimal topological ordering
238 over $X \cup Y$ it would always begin with the variables in Y but would be invariant to the ordering of
239 Y. Parent graphs are then created for all variables in X but are defined over the set of all variables in
240 $X \cup Y$, because that is the full set of parents that the variables could be drawn from. This restriction
241 allows the optimal parents for each variable in X to be identified without wasting time considering
242 what the parent set for variables in Y should be, or potentially throwing away the optimal graph
243 because of improper edges leading from a variable in Y to a variable in X. This step takes $O(n2^{n+m})$
244 time, where n is the number of variables in the node and m is the number of variables in the parent
245 nodes. This is because we only need to define a parent graph for the variables in the node we are
246 currently considering, but these parent graphs must be defined over all variables in the node plus all
247 the variables in the parent nodes.

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

4 Results

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

4.1 Acyclic Constraint Graphs Can Model the Global Stock Market

256 First, we examine the computational benefits of an acyclic constraint graph modeling the global
257 stock market. In particular, we want to identify for each stock which other stocks are predictive to
258 its performance. We chose to do this by learning a Bayesian network over the opening and closing
259 prices of 54 top performing stocks from the New York Stock Exchange (NYSE) in the United States,
260 the Tokyo Stock Exchange (TSE) in Japan, and the Financial Times Stock Exchange (FTSE) in
261 England. Learning a Bayesian network over all 108 variables is clearly infeasible, so we encode in
262 our constraint graph some common-sense restrictions (Fig. 3a). Specifically, opening and closing
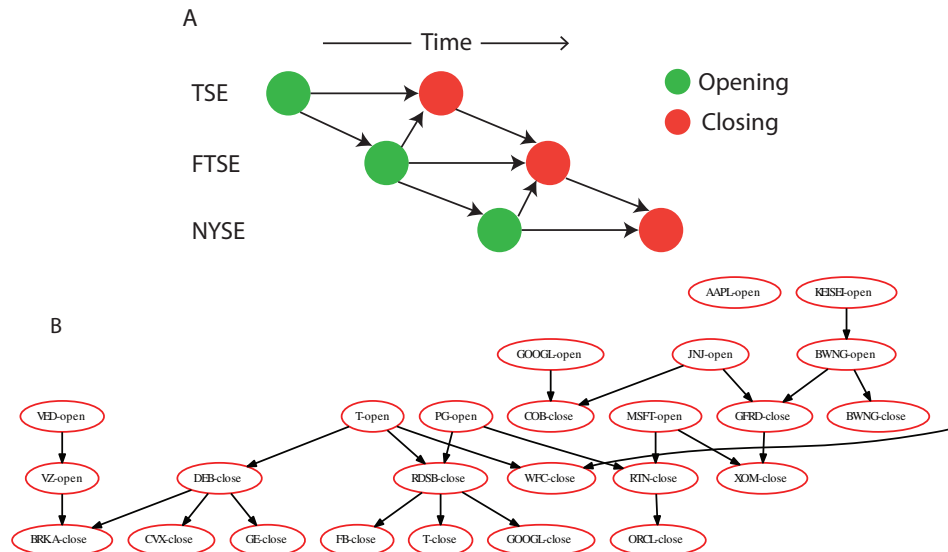
Figure 3: **A section of the learned Bayesian network of the global stock market.** (a) The constraint graph contains six nodes, the opening and closing prices for each of the three markets. These are connected such that the closing prices in a market depend on the opening prices but also the most recent international activity. (b) The most connected subset of stocks from the learned network covering 25 variables.

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. The dataset consists of opening and closing prices from these stocks between December 2nd 2015 and November 29th 2016, binarized to indicate whether the value was an increase compared to the prior price seen.

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 ∼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 ∼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 Constraint Graphs Allow Learning of Bayesian Network Classifiers

Bayesian network classifiers are an extension of Bayesian networks to supervised learning tasks by defining a Bayesian network over both the feature variables and the target variables together. Normal inference methods are used to predict the target variables given the observed feature variables. In the case where feature variables are always observed, only the Markov blanket of the target variables must be defined, i.e. their parents and children. The other variables are independent of the target variables and can be discarded, serving as a form of feature selection.

Model	Train Time (s)	Test Set Accuracy
Naive Bayes	0.05	0.79
BNC	0.5	0.81
Random Forest	1.4	0.89

Table 1: **Model comparison between naive Bayes, Bayesian network classifiers (BNC), and random forest.** Three algorithms were evaluated on the UCI handwritten digits dataset, fed in the binarized value corresponding to whether the intensity of a pixel was above average. The fitting time and test set accuracy are reported for each algorithm.

	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 2: **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 composied 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.

A popular Bayesian network classifier is the naive Bayes classifier that defines a single class variable as the parent to all feature variables. A natural extension to this method is to learn which features are useful, instead of assuming they all are, thereby combining feature selection with parameter learning in a manner that has some similarities to decision trees. This approach can be modeled by using a constraint graph that has all feature variables X in one node and all target variables y in its parent node, such that $y \rightarrow X$.

We empirically evaluated the performance of learning a simple Bayesian network classifier on the UCI Digits Dataset. The digits dataset is a collection of 8x8 images of handwritten digits, where the features are discretized values between 0 and 16 representing the intensity of that pixel and the labels are between 0 and 9 representing the digit stored there. We learn a Bayesian network, where the 64 pixels are in one node in the constraint graph and the class label is by itself it another node in the constraint graph that serves as a parent. We then train a Bayesian network classifier, a naive Bayes classifier, and a random forest classifier comprised of 100 trees, on a test set of 1500 images and test their performance on a held out 297 images. As expected, the learned Bayesian network classifier falls between naive Bayes and the random forest in terms of both training time and test set performance (Table. 1).

Futhermore, more complicated Bayesian network classifiers can be learned with different constraint graphs. One interesting extension is that instead of constraining all features to be children of the target variable, to allow features to be either parents or children of the target variable. This can be specified by a cyclic constraint graph where $y \rightarrow X \rightarrow y$, preventing the model from spending time identifying dependencies between the features. Finally, in cases where some features may be missing, it may be beneficial to model all dependencies between the features in order to allow inference to flow from observed variables not directly connected to the target variables to the target variables. This can be modeled by adding a self loop on the features variables X, allowing all edges to be learned except those between pairs of target variables. Learning a Bayesian network classifier in this manner will suffer from the same computational challenges as an unconstrained version, given the looseness of the constraints.
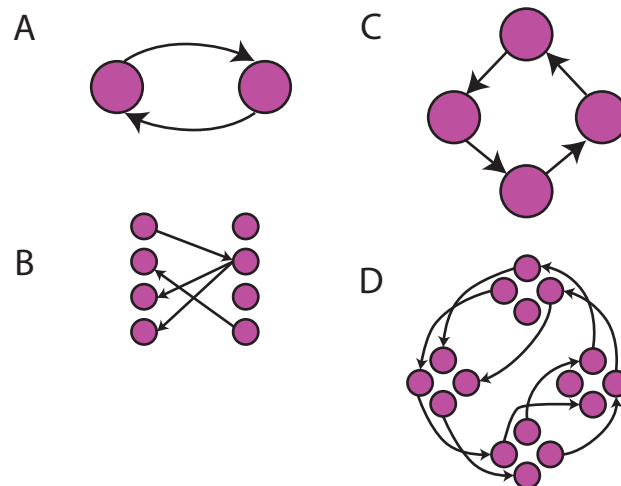
Figure 4: **Cyclic constraint graphs** (a) This constraint graph is comprised of a simple two node cycle with each node containing four variables. (b) The learned Bayesian network on random data where some variables were forced to identical values. Each circle here corresponds to a variable in the resulting 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.

4.3 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. 2). The data consisted of randomly generated binary values, because the running time does not depend on the presence of underlying structure in the data. 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.

4.4 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 even when the potential exists. Two separate constraint graphs are inspected, a two node cycle and a four node cycle (Fig. 4a/c). The dataset is comprised of random binary values, where the value of one variable in the cycle is copied to the other variables in the cycle to add synthetic structure. 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. 3). 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

	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 3: **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.

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

374 relationships, indicating that a given variable must occur at some point before some other variable in
375 the topological ordering.

Acknowledgments

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

References

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

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

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

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

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

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

393 Friedman, N. Learning belief networks in the presence of missing values and hidden variables. *ICML*
394 *'97 Proceedings of the Fourteenth International Conference on Machine learning*, pp. 125–133,
395 1997.

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

398 Heckerman, D., Geiger, D., and Chickering, D.M. Learning bayesian networks: The combination of
399 knowledge and statistical data. *Machine Learning*, pp. 197–243, 1995.

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

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

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

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

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

412 Schneiderman, Henry. Learning a restricted bayesian network for object detection. In *IEEE Confer-*
413 *ence on Computer Vision and Pattern Recognition*. IEEE, June 2004.

414 Suzuki, J. Learning bayesian belief networks based on the minimum description length principle:
415 An efficient algorithm using the b b technique. *Machine Learning, Proceedings of the Thirteenth*
416 *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.