Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

An Improved Lower Bound for Bayesian Network Structure Learning

Xiannian Fan and **Changhe Yuan**
Graduate Center and Queens College
City University of New York
365 Fifth Avenue, New York 10016
{xfan2@gc, change.yuan@qc}.cuny.edu

Abstract

Several heuristic search algorithms such as A* and breadth-first branch and bound have been developed for learning Bayesian network structures that optimize a scoring function. These algorithms rely on a lower bound function called static k-cycle conflict heuristic in guiding the search to explore the most promising search spaces. The heuristic takes as input a partition of the random variables of a data set; the importance of the partition opens up opportunities for further research. This work introduces a new partition method based on information extracted from the potentially optimal parent sets (POPS) of the variables. Empirical results show that the new partition can significantly improve the efficiency and scalability of heuristic search-based structure learning algorithms.

Introduction

This paper considers the problem of learning an optimal Bayesian network structure for given data and scoring function. Several heuristic search algorithms have been developed for solving this problem by formulating the problem as a shortest path problem (Yuan and Malone 2013), e.g., A* (Yuan, Malone, and Wu 2011), breadth-first branch and bound (BFBnB) (Malone et al. 2011), and anytime Window A* (Malone and Yuan 2013). Most of these algorithms need a lower bound function in estimating the quality of a search path so that they can prune paths that are guaranteed to lead to suboptimal solutions and focus on exploring the most promising search spaces.

A lower bound function called static k-cycle conflict heuristic (Yuan and Malone 2012) has demonstrated excellent performance when used by the search algorithms to solve many benchmark data sets. It takes as input a partition of the random variables of a data set, and computes the heuristic value of a path by enforcing the acyclicity constraint between the variables within each group and relaxing the acyclicity between different groups. The partition has a significant impact on the tightness of the heuristic. According to Yuan and Malone (2012), a good grouping method should reduce the number of directed cycles between different groups and enforce acyclicity as much as possible. Existing approaches (Fan, Yuan, and Malone 2014) achieve this objective only indirectly; they use independence tests to create an undirected graph, and then use graph partition algorithms to create a partition by minimizing the correlation between the variables in different groups.

This work introduces a more direct partition approach. Since the goal is to reduce the cycles between different groups of the partition, we should put variables that have more cycles between them in the same group. We will show that the cycles introduced in the static k-cycle conflict heuristic are originated from the potentially optimal parent sets (POPS) of the variables. We therefore identify groups of variables with more cycles based on information extracted from all or some of the POPS. Empirical results show that the new partition method can significantly improve the efficiency and scalability of heuristic search-based structure learning algorithms.

Background

This section provides an overview of Bayesian network structure learning, the shortest-path formulation, and the k-cycle conflict heuristic.

Bayesian Network Structure Learning

A Bayesian network (BN) consists of a directed acyclic graph (DAG) in which the vertices correspond to a set of random variables $\mathbf{V} = \{X_1, ..., X_n\}$ and a set of conditional probability distributions $P(X_i|PA_i)$, where all parents of X_i are referred to as PA_i. The joint probability over all variables factorizes as the product of the conditional probability distributions.

We consider the problem of learning a network structure from a discrete dataset $\mathbf{D} = \{D_1, ..., D_N\}$, where D_i is an instantiation of all the variables in $\mathbf{V}$. A scoring function s measures the goodness of fit of a network structure to $\mathbf{D}$ (Heckerman 1998). The goal is to find a structure which optimizes the score. We only require that the scoring function is *decomposable* (Heckerman 1998); that is, the score of a network $s(N) = \sum_i s_i(PA_i)$. The $s_i(PA_i)$ values are often called *local scores*. Many commonly used scoring functions, such as MDL (Lam and Bacchus 1994) and BDe (Buntine 1991; Heckerman, Geiger, and Chickering 1995), are decomposable.

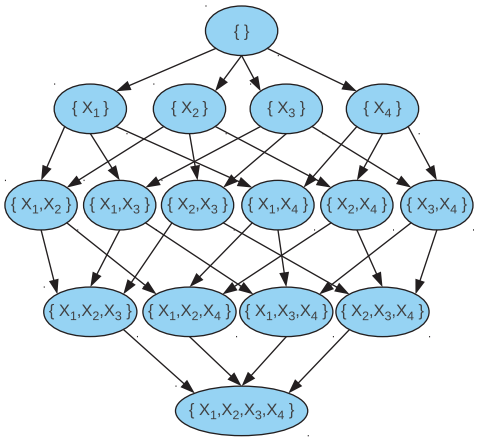

Figure 1: An order graph for four variables.

While the local scores are defined for all 2^{n-1} possible parent sets for each variable, this number is greatly reduced by pruning parent sets that are provably never optimal (de Campos and Ji 2011). The scores remaining after pruning are called *potentially optimal parent sets* (POPS). The POPS are given as input to the learning problem. We denote the set of POPS for X_i as $\mathcal{P}_i$.The *Bayesian network structure learning* problem (BNSL) can be defined as follows.

The BNSL Problem

INPUT: A set $\mathbf{V} = \{X_1, \ldots, X_n\}$ of variables and a set of POPS $\mathcal{P}_i$ for each X_i.

TASK: Find a DAG N^* such that

$$N^* \in \arg\min_N \sum_{i=1}^{n} s_i(PA_i),$$

where PA_i is the parent set of X_i in N and $PA_i \in \mathcal{P}_i$.

Shortest Path Formulation

The above structure learning problem was formulated as a shortest-path problem in (Yuan, Malone, and Wu 2011; Yuan and Malone 2013). Figure 1 shows the *implicit* search graph for four variables. The top-most node with the empty variable set is the *start* node, and the bottom-most node with the complete set is the *goal* node. An arc from $\mathbf{U}$ to $\mathbf{U} \cup \{X_i\}$ in the graph represents generating a successor node by adding a new variable X_i as a leaf to an existing subnetwork of variables $\mathbf{U}$; the cost of the arc is equal to the score of the optimal parent set for X_i out of $\mathbf{U}$, which is computed by considering all subsets of the variables in $PA \subseteq \mathbf{U}, PA \in \mathcal{P}_i$, i.e.,

$$cost(\mathbf{U} \to \mathbf{U} \cup \{X_i\}) = BestScore(X_i, \mathbf{U}) \quad (1)$$

$$= \min_{PA_i \subseteq \mathbf{U}, PA_i \in \mathcal{P}_i} s_i(PA_i). \quad (2)$$

In this search graph, each path from the start to the goal corresponds to an ordering of the variables in the order of their appearance, so the search graph is also called an *order graph*. Each variable selects optimal parents from the variables that precede it, so combining the optimal parent sets yields an optimal structure for that ordering. The shortest path gives the global optimal structure.

Static K-Cycle Conflict Heuristic

The following simple heuristic function was introduced in (Yuan, Malone, and Wu 2011) for computing lower bounds for A* search.

Definition 1. *Let* $\mathbf{U}$ *be a node in the order graph, its heuristic value is*

$$h(\mathbf{U}) = \sum_{X \in \mathbf{V} \setminus \mathbf{U}} BestScore(X, \mathbf{V} \setminus \{X\}). \quad (3)$$

The above heuristic function allows each remaining variable to choose optimal parents from all of the other variables. Therefore it completely relaxes the acyclicity constraint of Bayesian networks in the estimation. The heuristic was proven to be admissible, meaning it never overestimates the future distance (Yuan, Malone, and Wu 2011). Admissible heuristics guarantee the optimality. However, because of the complete relaxation of the acyclicity constraint, the simple heuristic may generate loose lower bounds.

In (Yuan and Malone 2012), an improved heuristic function called *static k-cycle conflict heuristic* was proposed by reducing the amount of relaxation. The idea is to partition the variables $\mathbf{V}$ into multiple groups $\mathbf{V_i}$ (typically two), i.e. $\mathbf{V} = \bigcup_i \mathbf{V_i}$, and enforce acyclicity within each group while still allowing cycles between the groups. For the partition, we need to compute a *pattern database* for each group $\mathbf{V_i}$. For this particular problem, a pattern database for group $\mathbf{V_i}$ is basically a full order graph containing all subsets of $\mathbf{V_i}$. We will use a backward breadth first search to create the graph layer by layer starting from the node $\mathbf{V_i}$. The cost for any reverse arc from $\mathbf{U} \cup \{X\}$ to $\mathbf{U}$ in this order graph will be $BestScore(X, (\bigcup_{j \neq i} \mathbf{V_j}) \cup \mathbf{U})$. We then enumerate all subsets of each group $\mathbf{V_i}$ as the *patterns*, which can be done by a reverse breadth-first search in an order graph containing only $\mathbf{V_i}$ (Yuan and Malone 2012). The patterns from different groups are guaranteed to be mutually exclusive, so we simply pick out the maximum-size pattern for each group that is a subset of $\mathbf{V} \setminus \mathbf{U}$ and add them together as the lower bound. Figure 2 shows two pattern databases for a 8-variable problem, as well as the procedure of calculating the heuristic value of node $\{X_2, X_3, X_5, X_7\}$.

The tightness of the static k-cycle conflict heuristic depends highly on the partition being used. The heuristic can avoid directed cycles for the patterns within the same group, but cannot avoid cycles between different groups. In the example shown in Figure 2, X_3 selects parents $\{X_1, X_5\}$ as parents (subset of $\{X_1, X_4\} \cup \mathbf{V_2}$) and X_5 selects $\{X_3, X_6\}$ as parents; there is a cycle between $\{X_3\}$ and $\{X_5\}$.

A New Partition Method for Improved Lower Bounds

Previously, Fan et al. proposed two methods called Parent Grouping (PG) and Family Grouping (FG) (Fan, Yuan, and

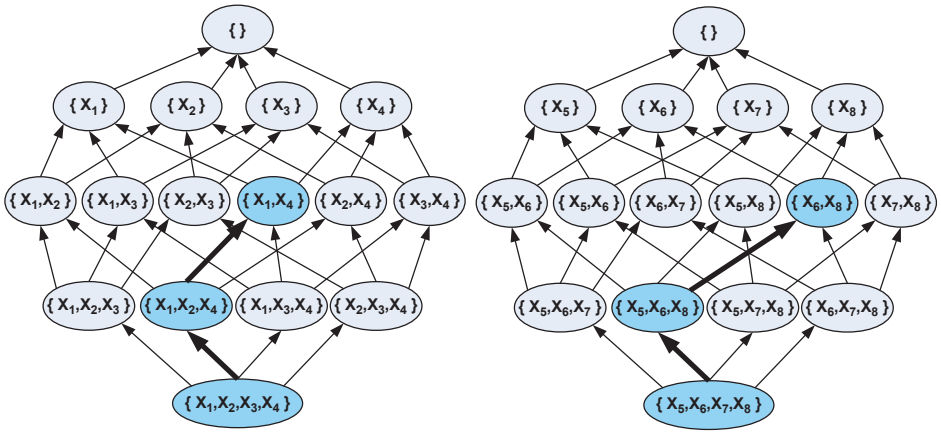

Figure 2: Two pattern databases for a 8 variables problem. 8 variables are partitioned into two groups, $\mathbf{V_1} = \{X_1, X_2, X_3, X_4\}$ and $\mathbf{V_2} = \{X_5, X_6, X_7, X_8\}$. **(Left)** The pattern database for group $\mathbf{V_1}$; bold arrows show the path corresponding to the score P_1 for pattern $\{X_2, X_3\}$, where $P_1 = BestScore(X_3, \{X_1, X_2, X_4\} \cup \mathbf{V_2}) + BestScore(X_2, \{X_1, X_4\} \cup \mathbf{V_2})$. **(Right)** The pattern database for group $\mathbf{V_2}$; bold arrows show the path corresponding to the score P_2 for pattern $\{X_5, X_7\}$, where $P_2 = BestScore(X_7, \{X_5, X_6, X_8\} \cup \mathbf{V_1}) + BestScore(X_5, \{X_6, X_8\} \cup \mathbf{V_1})$. The heuristic value for pattern $\{X_2, X_3, X_5, X_7\}$ is $P_1 + P_2$.

Malone 2014) to partition the variables based on correlation between the variables. There are three main steps in these two methods. First, they create an undirected skeleton graph. PG uses the best POPS of each variable to create the skeleton, while FG uses the Min-Max Parent Child (MMPC) algorithm (Tsamardinos, Brown, and Aliferis 2006) to get the skeleton. Second, they use the independence tests in MMPC to estimate the weights for the edges of the skeleton. Third, A graph partition algorithm called METIS (Karypis and Kumar 1998) is used to partition the skeleton into two balanced subgraphs by minimizing the total weights of the edges between the subgraphs.

Intuitively, since the edge weights measure the correlation between the variables, putting variables that are less correlated into different groups should reduce cycles between the groups. However, these approaches fail to consider the complex relation between a variable and another set of variables (parent set), and are heuristic in nature. This work introduces a new approach to obtaining a partition by directly reducing cycles between its groups.

Ancestral Constraints

We first use a concrete example to motivate the development of our method. It relies on the fact that all of the potential parent-child relations are contained in the potentially optimal parent sets (POPS) of the variables. Useful information can be extracted from the parent-child relations to find good partition strategies. As an illustrating example, Table 1 shows the POPS of eight variables in the example. Based on these POPS, we can extract the parent-child relations as follows. The set of all potential parents $\mathcal{P}_i$ for X_i can be collected by taking the union of all of its POPS. For example, X_1 can select parents from $\{X_2, X_5\}$, and X_2 can only select X_1 as its parent. We then use all of the $\mathcal{P}_i$s to create a directed graph, which was called *parent relation graph*

var.	**POPS**			
X_1	$\{X_2\}$	$\{X_5\}$		
X_2	$\{X_1\}$			
X_3	$\{X_1, X_5\}$	$\{X_1, X_2\}$	$\{X_2, X_4\}$	$\{X_1\}$
X_4	$\{X_3\}$	$\{X_6\}$	$\{X_7\}$	
X_5	$\{X_1, X_3\}$	$\{X_3\}$		
X_6	$\{X_2, X_7\}$	$\{X_7\}$		
X_7	$\{X_8\}$	$\{X_6, X_4\}$		
X_8	$\{X_6\}$	$\{X_7\}$		

Table 1: The POPS for eight variables. The ith row shows $\mathcal{P}_i$. The POPS for each variable are sorted according to their scores.

in (Fan, Yuan, and Malone 2014).

We can use the *ancestral relations* of the parent relation graph to partition the variables into sets. In particular if the variables in one set can be ancestors of the variables in another set, but not vice versa, there must be no cycles between the two sets. We can put the two sets of variables into different groups of the static k-cycle conflict heuristic in order to reduce cycles. We identify such ancestral relations by extracting *strongly connected components* (SCCs) from the parent relation graph. The SCCs of the parent relation graph form a DAG called *component graph* (Cormen et al. 2001); each component c_i corresponds to a SCC scc_i from the parent relation graph (which in turn corresponds to a set of variables in the Bayesian network). The component graph includes a directed edge from node c_i to c_j if the parent relation graph includes an edge from a variable $X_i \in scc_i$ to $X_j \in scc_j$. The component graph provides some obvious ancestral relation constraints: if c_j is a descendent of c_i in the component graph, variables in scc_j cannot be ancestors of variables in scc_i. We call such relations *ances-*

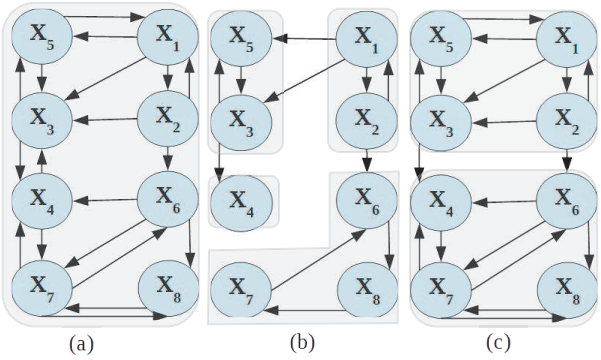

Figure 3: The parent relation graphs created from (a) all of the POPS, (b)top-1 POPS, and (c) top-2 POPS. Each shaded box represents a strongly connected component (SCC) and a node in a corresponding component graph.

tral constraints. We can obtain a tight static k-cycle conflict heuristic by putting the different SCCs into different groups of the heuristic as there would not be any cycles between the groups. As a matter of fact, the SCCs represent independent learning problems that can be solved separately (Fan, Malone, and Yuan 2014).

There are two possible pitfalls of the above approach. One is that it assumes there exist multiple SCCs. The component graph may only has a single SCC, especially when we use all of the POPS to create the parent relation graph. Figure 3(a) shows the parent relation graph created from the POPS in Table 1. The component graph includes one SCC containing all variables. Therefore, all variables can be ancestors of each other; we have no way of easily dividing the variables into groups. Second, even though there exist multiple SCCs, the largest one may be too large for the approach to be feasible. The reason is that we are performing an exhaustive search when building a pattern database for a group of variables; the search is only feasible for fewer than 30 variables within one group in our testing environment. We will address these issues in the following sections.

Ancestral Constraints from Top-K POPS

To avoid creating a dense parent relation graph that has either only a single SCC or a SCC that is too large, we propose to use only the top K POPS of each variable to create the graph. Fewer POPS reduce the total number of candidate parents for each variable, resulting in fewer arcs in the parent relation graph. Top K POPS allow us to focus on the most important parent-child relations so that the SCCs of the relation graph still allow the heuristic to remove the most critical cycles. Figure 3(b) shows the parent relation graph created from only the top 1 POPS of each variable. The component graph now has 4 SCCs with no cycles between them. Of course we are ignoring the ancestral constraints outside of the top-1 POPS. For example, after recovering the parent relation $X_5 \rightarrow X_1$, there will be cycles between SCCs $\{X_1, X_2\}$ and $\{X_3, X_5\}$ as well as between X_1 and X_5.

The parent relation graph created from the top-1 POPS is overly sparse; 4 SCCs are generated as a result. Fortunately, there is a happy medium to explore between using all POPS and using only top-1 POPS; we propose to use top-K POPS of each variable in creating the parent relation graph. Intuitively, including more POPS will introduce more cycles in the parent relation graph. More cycles in the parent relation graph would allow us to remove more cycles in the heuristic by enforcing acyclicity between the variables within each SCC, hence resulting in tighter heuristic values.

Figure 3(c) shows the graph created from the top-2 POPS of all variables. Now there are only two SCCs, $\{X_1, X_2, X_3, X_5\}$ and $\{X_4, X_6, X_7, X_8\}$. By using each SCC as a group in the static k-cycle conflict heuristic, we can remove the cycles within each SCC. For example, X_5 and X_1 are in the same SCC, the cycle between them can be removed by enforcing acyclicity between them in the heuristic. Moreover, there are no cycles between the SCCs, so putting them into different groups of the heuristic can reduce the possibility of cycles between the groups. Note that we cannot completely avoid cycles between the groups because we only consider top-2 POPS.

Finally, since a partition with two groups typically works best for the static k-cycle conflict heuristic, there is no need to increase K to consider more POPS in this case.

Components Grouping

It was shown in (Yuan and Malone 2012; Fan, Yuan, and Malone 2014) that it is best to divide the variables into only 2 groups in static pattern databases. The reason is that we enforce acyclicity within each group, and allow cycles between groups; using only two groups allows acyclicity to be enforced within the largest possible groups, hence lower probability for cycles between the groups. The parent relation graph created from top-K POPS may have more than 2 SCCs, regardless of what K is. We then have a grouping problem. Suppose there are l SCCs in the parent relation graph, denoted by $scc_1, scc_2, ..., scc_l$, the grouping problem is to divide these SCCs into two groups, $\mathbf{V_1}$ and $\mathbf{V_2}$.

We let γ be the number of variables in the largest pattern database we can build in our testing environment. If the largest SCC is already larger than γ, we cannot succeed in building the heuristic, not mentioning solving the learning problem. In this case we fall back on the parent grouping (PG) method (Fan, Yuan, and Malone 2014), that is, we ignore the direction of the edges in the parent relation graph from Top-1 POPS and use the graph as the skeleton, assign the graph weights, and partition the skeleton to give the two grouping.

On the other hand, if the largest SCC is smaller than γ, we should combine it with some other SCCs to get a group as large as possible as long as it is still smaller than γ. Moreover, we want to combine nearby SCCs into a group because variables in nearby SCCs are more likely to have cycles between them outside of the top-K POPS. We therefore propose the following Prim algorithm-like method for dividing the SCCs into two groups.

First, we create the parent relation graph and its component graph based on top-K POPS. Initialize $\mathbf{V_1}$ to contain the largest SCC in the graph. Second, we perform the following iteratively until $\mathbf{V_1}$ hits the threshold γ. For all of

the SCCs in $\mathbf{V_1}$, we find their neighboring SCCs that are not yet in $\mathbf{V_1}$. Then we select the largest one out of these SCCs to add to $\mathbf{V_1}$ subject to the constraint that $|\mathbf{V_1}| \leq \gamma$. Third, all the remaining SCCs form the second group $\mathbf{V_2}$. It is straightforward to generalize this method to more groups; we only consider two groups in this paper.

The K is given in the above procedure. Let P be the maximum number of POPS any variable can have. Since we do not know what the optimal K is, we try each K starting from 1 to P until we get only a single SCC or exhaust all POPS for each variable, or the largest SCC exceeds the γ threshold. We finally accept the grouping of the highest K that produces at least two SCCs subject to the γ threshold. We name the new method *Components Grouping* (CG).

Related Work

POPS for all variables are the input to the score-based exact BNSL algorithms. To the best of our knowledge, Fan et al. (2014) are the first to extract constraints from POPS to improve the performance of heuristic search on the exact BNSL problem. They decomposed the parent graph into a set of strongly connected components (SCCs). Each SCC corresponds to a smaller subproblem which can be solved independently of the others, thus the search space was reduced. They also used the top-K POPS to further reduce the search space to get a sub-optimal solution, but lost the optimality guarantee. We use the top-K POPS to get groupings for the lower bound. We do not lose global optimality no matter which K we use.

The Parent Grouping (PG) method in (Fan, Yuan, and Malone 2014) creates an undirected skeleton from the top-1 POPS, assigns the skeleton weights, and partition the graph into two groups. Our method works directly on the directed graph created from the top-K POPS.

Finally, this research focuses on heuristic search-based algorithms for learning optimal Bayesian networks (Yuan and Malone 2013; Yuan, Malone, and Wu 2011; Malone et al. 2011; Malone and Yuan 2013). Other exact algorithms have been developed for solving the same problem based on dynamic programming (Koivisto and Sood 2004; Ott, Imoto, and Miyano 2004; Silander and Myllymaki 2006; Singh and Moore 2005; Malone, Yuan, and Hansen 2011) and integer linear programming (Cussens 2011; Jaakkola et al. 2010). Please refer to (Yuan and Malone 2013) for a comprehensive study comparing the different approaches.

Empirical Results

We empirically evaluated our new CG lower bound using the A* and BFBnB algorithms on benchmark datasets from UCI machine learning repository and Bayesian Network Repository (*http://compbio.cs.huji.ac. il/Repository/*). The experiments were performed on an IBM System x3850 X5 with 16 core 2.67GHz Intel Xeon Processors and 512G RAM.

Parameter K

We first tested the effect of K on the performance of A* algorithm as it guarantees to expand the minimal number of nodes. The size of the largest SCC is monotonically increasing in terms of K. We did not limit the size of the largest SCC as long as at least two SCCs are obtained. In this experiment, Dataset *soybeans* (36 variables) was tested bacause it was found to be challenging from a previous study (Fan, Malone, and Yuan 2014). FG failed to solve this dataset within the time limit of 2 hours; PG solved it in more than 518 seconds with more than 9 million nodes expanded. For CG, we varied K and measured the running time and the number of expanded nodes. Figure 4 showed the results. Even for $K = 1$, the running time of CG was only 76 seconds, 7 times faster than PG, and the number of expanded nodes was around 1 million, 9 times fewer than that of PG; when $K = 2$, the running time was less than 1 second, and the number of expanded nodes was just around 1 thousand, which were negligible compared to PG; from $K = 3$ and on, the running time and number of expanded nodes were further reduced. The size of largest SCC was 20 when $K = 1$, 26 when $K = 2$, and 27 when $K = 3, 4, 5$. $K = 6$ and above only generated a single SCC.

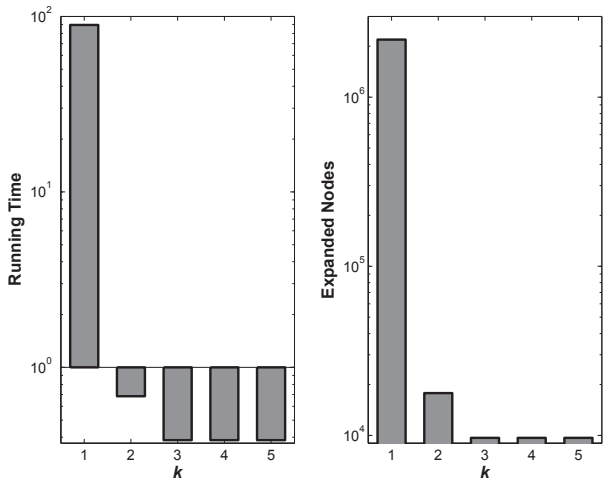

Figure 4: The running time and number of expanded nodes needed by A* to solve *Soybeans* with different K.

Generally, we notice that the larger the K, the better performance of the CG lower bound, which is understandable because we can avoid cycles within a wider range of POPS. That is exactly why we set K to be the largest value such that at least two SCCs are obtained and the size of the largest SCC is smaller than the threshold γ.

Results on Benchmark Datasets

We compared the components grouping (CG) against the two existing strategies, the family grouping (FG) and parents grouping (PG) (Fan, Yuan, and Malone 2014), using A* and BFBnB on a set of benchmark datasets. We set the threshold γ to be 22 for small datasets with fewer than 35 variables as the heuristic can be built within 5 seconds, and to be 25 for large datasets with 35 or more variables as the heuristic can be built within 60 seconds. The results are shown in Table 2.

A* is guaranteed to expand the least number of search

Dataset			Results						
Name	n	N		FG (A*)	PG (A*)	CG (A*)	FG (BFBnB)	PG (BFBnB)	CG (BFBnB)
Autos	26	159	Time (s)	16.91	26.78	7.20	9.12	13.77	4.79
			Nodes	0.94	1.63	0.46	0.94	1.63	0.46
Insurance*	27	1,000	Time (s)	7.51	62.41	22.27	4.57	28.74	10.88
			Nodes	0.53	3.77	1.50	0.53	3.77	1.64
Horse Colic	28	300	Time (s)	1.13	6.01	0.09	1.72	5.71	0.71
			Nodes	0.09	0.58	0.00	0.09	0.87	0.00
Flag	29	194	Time (s)	61.04	213.53	1.40	50.31	112.24	1.61
			Nodes	2.36	9.96	0.09	3.45	13.41	0.09
Mildew*	35	1,000	Time (s)	OT	20.92	0.58	OT	21.28	1.30
			Nodes	OT	1.97	0.03	OT	4.43	0.03
Soybean	36	307	Time (s)	OT	518.35	76.22	OT	1,203.78	59.13
			Nodes	OT	9.64	1.37	OT	129.77	2.46
Alarm*	37	1,000	Time (s)	4.08	4.09	4.00	3.18	3.15	3.23
			Nodes	0.25	0.24	0.24	0.25	0.25	0.25
Sponge	45	76	Time (s)	OT	OT	23.23	OT	OT	20.93
			Nodes	OT	OT	1.60	OT	OT	2.99
Barley*	48	1,000	Time (s)	OT	3.16	0.08	OT	1.53	1.14
			Nodes	OT	0.08	0.00	OT	0.08	0.00

Table 2: The number of expanded nodes (in millions) and running time (in seconds) of the A* and BFBnB algorithm on a set of benchmark datasets with different lower bounds. The lower bounds are family grouping (FG), parents grouping (PG) and components grouping (CG). "n" is the total number of variables, and "N" is the number of data points; "*"indicates the dataset was generated from a repository network using logic sampling; all other datasets are from UCI; OT means out of time (2 hours).

nodes (Yuan, Malone, and Wu 2011). The table shows that the benefit of the new grouping method CG on A* is rather obvious. The improvement brought by CG in running time and the number of expanded nodes ranges from three times to over two orders of magnitude on most of datasets.

FG is inferior than CG on all of the datasets except *insurance*. It failed to solve datasets *Mildew, Soybean, Sponge* and *Barley* within 2 hours. In comparison, three out of the four datasets turned out to be quite easy to solve by CG; the other one, *Soybean*, was solved efficiently by CG within 77 seconds (Note γ is set to be 25 in this experiment, so the final results are from $K = 1$). Upon further investigation, we found that the skeleton of FG found by the MMPC (Tsamardinos, Brown, and Aliferis 2006) method was very sparse and not well connected. MMPC used a parameter $p-$value to control the generation of the skeleton, and the default value $p = 0.05$ was shown to result in stable performance. We varied the value of p from 0.01 to 0.99 to test the effect for the network. The results showed that the issue cannot be solved with a larger p value.

PG is more robust on large datasets than FG. However, CG consistently outperformed PG; the speedup ranged from several times faster (e.g., *Autos*) to several orders of magnitude faster (e.g., *Flag*). Moreover, CG solved the Sponge efficiently (within 24 seconds) while PG and FG failed within the time limit (7,200 seconds). The only exception is the easy dataset *alarm* for which all of the grouping methods can solve it efficiently in around 4 seconds.

We also evaluated the different grouping methods with BFBnB (Malone et al. 2011) because it can scale to larger datasets by using external memory. Anytime window A* (AWA*) was used to provided an upper bound for pruning since a previous study (Malone and Yuan 2013) has shown that AWA* is effective at finding high quality solutions quickly, so we provided the upper bound by running AWA* for 5 seconds on small dataset (fewer than 35 variables) and 10 seconds for larger datasets. The results of BFBnB demonstrate rather similar patterns to A*. Therefore, the new grouping method seems to be generally applicable to other search methods.

Conclusion

This paper investigates a new method for improving the lower bound of heuristic search for Bayesian networks structure learning proposed in (Yuan and Malone 2012). The main idea is to create a parent relation graph from the top-K potentially optimal parent sets (POPS) of each variable, and extract better groupings for the static k-cycle conflict heuristic. Empirically, we showed that the new lower bound can significantly improve the efficiency and scalability of the heuristic search-based learning methods.

As we said earlier, the need for good groupings in the k-cycle conflict heuristic opens up opportunities for future research. Even though our current method has excellent performance, there may be better methods. For example, one possibility is to investigate how to integrate domain knowledge and the information contained in the potentially optimal parents sets.

Acknowledgments This work was supported by NSF grants IIS-0953723, IIS-1219114, and a PSC-CUNY enhancement award

References

Buntine, W. 1991. Theory refinement on Bayesian networks. In *Proceedings of the seventh conference (1991) on Uncertainty in artificial intelligence*, 52–60. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Cormen, T. H.; Stein, C.; Rivest, R. L.; and Leiserson, C. E. 2001. *Introduction to Algorithms*. McGraw-Hill Higher Education.

Cussens, J. 2011. Bayesian network learning with cutting planes. In *Proceedings of the Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-11)*, 153–160. Corvallis, Oregon: AUAI Press.

de Campos, C. P., and Ji, Q. 2011. Efficient learning of Bayesian networks using constraints. *Journal of Machine Learning Research* 12:663–689.

Fan, X.; Malone, B.; and Yuan, C. 2014. Finding optimal Bayesian network structures with constraints learned from data. In *Proceedings of the 30th Annual Conference on Uncertainty in Artificial Intelligence (UAI-14)*.

Fan, X.; Yuan, C.; and Malone, B. 2014. Tightening bounds for Bayesian network structure learning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI-14)*.

Heckerman, D.; Geiger, D.; and Chickering, D. M. 1995. Learning Bayesian networks: The combination of knowledge and statistical data. 20:197–243.

Heckerman, D. 1998. A tutorial on learning with Bayesian networks. In Holmes, D., and Jain, L., eds., *Innovations in Bayesian Networks*, volume 156 of *Studies in Computational Intelligence*. Springer Berlin / Heidelberg. 33–82.

Jaakkola, T.; Sontag, D.; Globerson, A.; and Meila, M. 2010. Learning Bayesian network structure using LP relaxations. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Karypis, G., and Kumar, V. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20(1):359–392.

Koivisto, M., and Sood, K. 2004. Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research* 549–573.

Lam, W., and Bacchus, F. 1994. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence* 10:269–293.

Malone, B., and Yuan, C. 2013. Evaluating anytime algorithms for learning optimal Bayesian networks. In *In Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI-13)*.

Malone, B.; Yuan, C.; Hansen, E.; and Bridges, S. 2011. Improving the scalability of optimal Bayesian network learning with external-memory frontier breadth-first branch and bound search. In *Proceedings of the Twenty-Seventh Annual Conference on Uncertainty in Artificial Intelligence (UAI-11)*, 479–488. Corvallis, Oregon: AUAI Press.

Malone, B.; Yuan, C.; and Hansen, E. 2011. Memory-efficient dynamic programming for learning optimal Bayesian networks. In *Proceedings of the 25th national conference on Artifical intelligence*.

Ott, S.; Imoto, S.; and Miyano, S. 2004. Finding optimal models for small gene networks. In *Pac. Symp. Biocomput*, 557–567.

Silander, T., and Myllymaki, P. 2006. A simple approach for finding the globally optimal Bayesian network structure. In *Proceedings of the 22nd Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)*. Arlington, Virginia: AUAI Press.

Singh, A., and Moore, A. 2005. Finding optimal Bayesian networks by dynamic programming. Technical report, Carnegie Mellon University.

Tsamardinos, I.; Brown, L.; and Aliferis, C. 2006. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning* 65:31–78. 10.1007/s10994-006-6889-7.

Yuan, C., and Malone, B. 2012. An improved admissible heuristic for finding optimal Bayesian networks. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence (UAI-12)*. AUAI Press.

Yuan, C., and Malone, B. 2013. Learning optimal Bayesian networks: A shortest path perspective. *Journal of Artificial Intelligence Research (JAIR)* 48:23–65.

Yuan, C.; Malone, B.; and Wu, X. 2011. Learning optimal Bayesian networks using A* search. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*.