A Bounded Error, Anytime, Parallel Algorithm for Exact Bayesian Network Structure Learning

Brandon Malone[1,3] and Changhe Yuan[2,3]
[1]Department of Computer Science, University of Helsinki, Helsinki Institute for Information Technology
[2]Department of Computer Science, Queens College/City University of New York
[3]Department of Computer Science and Engineering, Mississippi State University
brandon.malone@cs.helsinki.fi, changhe.yuan@qc.cuny.edu

Abstract

Bayesian network structure learning is NP-hard. Several anytime structure learning algorithms have been proposed which guarantee to learn optimal networks if given enough resources. In this paper, we describe a general purpose, anytime search algorithm with bounded error that also guarantees optimality. We give an efficient, sparse representation of a key data structure for structure learning. Empirical results show our algorithm often finds better networks more quickly than state of the art methods. They also highlight accepting a small, bounded amount of suboptimality can reduce the memory and runtime requirements of structure learning by several orders of magnitude.

1 Introduction

Bayesian networks are a type of graphical model that are frequently used to capture relationships among variables in a domain. When a structure is unknown, we must learn it from data. In score-based structure learning, a scoring function measures the goodness of fit of a network to the data (Cooper and Herskovits, 1992; Heckerman et al., 1995). The goal is to find the structure with the optimal score. This problem is NP-hard (Chickering, 1996), so early worked focused on approximation algorithms (Chickering, 2002; Moore and Wong, 2003; Teyssier and Koller, 2005). These algorithms cannot guarantee the quality of the learned structure, but they do have good anytime behavior. An anytime algorithm quickly finds a solution and improves its quality throughout the search.

Recently, dynamic programming (DP) algorithms (Koivisto and Sood, 2004; Ott et al., 2004; Singh and Moore, 2005; Silander and Myllymaki, 2006) with optimality guarantees have been developed. These algorithms learn optimal small subnetworks and grow them one leaf at a time until having the optimal network over all of the variables. Naively, though, DP algorithms store an exponential number of subnetworks. Several DP variants (Parviainen and Koivisto, 2009; Malone et al., 2011a) have reduced the memory requirement. None of these algorithms have anytime behavior.

Tamada *et al.* (2011) developed a parallel DP algorithm. The intuition is to group subnetworks with many overlapping computations on the same processor. They propose an indexing function which partitions variables in such a manner that provably maximizes that overlap. Consequently, it also minimizes the communication overhead. As with other DP algorithms, this algorithm does not have anytime behavior. The authors also note that, for large networks, despite minimizing communication, their MPI communication time still accounted for over 80% of the runtime.

De Campos and Ji (2011) proposed a branch and bound (BB) algorithm which searches in the space of cyclic structures to find optimal networks. They begin with a (cyclic) structure in which all variables have their optimal parents, and use a best-first search to break cycles until finding the optimal structure. To add anytime behavior, they use an approximation algorithm to initialize the search with a suboptimal network as an upper bound and sometimes vary their search strategy. As the search explores structures, it provably increases a lower bound. When the upper and lower bounds agree, the current best structure is optimal.

Several authors (Jaakkola et al., 2010; Cussens, 2011) have also developed mathematical programming (MP) algorithms to learn optimal networks. They use a series of MPs to search through a polytope with exponentially many facets to find the optimal network. These algorithms also have anytime behavior; the solution to the primal problem gives a lower bound on the score of the optimal network, and the solution to the dual can be used to decode a valid network which gives an upper bound.

Yuan *et al.* (2011) gave a shortest-path formulation of the problem. A* and breadth-first branch and bound (Malone et al., 2011b) algorithms were developed to leverage that formulation. These algorithms also lack anytime behavior.

In this paper, we take advantage of the shortest-path formulation and dovetailing (Valenzano et al., 2010) to develop a parallel algorithm that has anytime behavior and bounds the error of learned solutions. An efficient, sparse representation for calculating search information is a key ingredient of the algorithm. Experimentally, we show that our algorithm often exhibits better anytime behavior than BB and sequential anytime search techniques. It also reveals that accepting a small, bounded amount of suboptimality in the network can reduce the memory and runtime requirements of structure learning by several orders of magnitude.

The remainder of this paper is structured as follows. Section 2 gives an overview of Bayesian network structure learning and the shortest-path formulation. Section 3 details our new algorithm. Section 4 gives experimental results in which we compare it to other state of the art algorithms.

2 Background

This section reviews score-based structure learning and the shortest-path formulation.

2.1 Learning Bayesian Network Structures

A Bayesian network is a directed acyclic graph whose vertices correspond to a set of random variables $\mathbf{V} = \{X_1, ..., X_n\}$, and the arcs describe dependence relationships among the variables. The parents of X_i are called PA_i. The parameters of the network give a conditional probability distribution, $P(X_i|PA_i)$, for each X_i.

Given a dataset $\mathbf{D} = \{D_1, ..., D_N\}$, where each D_i is a complete instantiation of $\mathbf{V}$, and a scoring function, the structure learning problem is to find a network structure which optimizes the scoring function for that dataset (Heckerman, 1996). We assume the use of a *decomposable* (Heckerman, 1996) scoring function, such as MDL (Lam and Bacchus, 1994) or BDe (Heckerman et al., 1995). We assume that local scores, $score(X_i|PA_i)$, are calculated at the beginning of the algorithm.

2.2 A* Heuristic Search

A* (Hart et al., 1968) is a state space search algorithm which guarantees to find a shortest path from a *start* node to a *goal* node. The algorithm uses an evaluation function f to measure the quality of search nodes. For a search node $\mathbf{U}$, the value of f is the sum of the cost from the start node to $\mathbf{U}$, denoted as $g(\mathbf{U})$, and an estimate of the cost from $\mathbf{U}$ to a goal node, denoted as $h(\mathbf{U})$. That is, $f(\mathbf{U}) = g(\mathbf{U}) + h(\mathbf{U})$, and f gives an estimate on the cost of the shortest path from the start to the goal which passes through $\mathbf{U}$.

The search algorithm uses a min priority queue called $open$ list to organize the search frontier. The nodes on $open$ are sorted according to their f values. Initially, $open$ contains only the start node. At each search step, the top node $\mathbf{U}$ is removed from the priority queue and *expanded* by generating its successors; the expanded node is placed in a hash table called $closed$. The g cost of each successor $\mathbf{S}$ is equal to $g(\mathbf{U})$ plus the cost from $\mathbf{U}$ to $\mathbf{S}$.

This process continues until a goal node, $\mathbf{G}$, is expanded. The cost of the shortest path from *start* to $\mathbf{G}$ is $g(\mathbf{G})$.

An *admissible* heuristic ensures that $h(\mathbf{U})$ is always optimistic. That is, it always underestimates the distance from $\mathbf{U}$ to *goal*. When h is also *consistent*, the shortest path to a node is found the first time it is expanded. Therefore, if a successor is in $closed$, it is discarded. Otherwise, $\mathbf{S}$ is added to $open$; duplicates on $open$ are merged and the best cost is f is retained.

2.3 Shortest-path Structure Learning Formulation

Yuan *et al.* (2011) formulated the learning problem as a shortest-path finding problem. Figure 1 shows an *implicit* state space search graph for four variables. The *start* node is the top node with the empty variable set. The *goal* node is the bottom node with the full variables set. Successors are shown by arcs in the figure. An arc from $\mathbf{U}$ to $\mathbf{U} \cup \{X\}$ represents adding X as a leaf to a subnetwork $\mathbf{U}$; the cost of the arc is equal to the score of the optimal parent set for X out of $\mathbf{U}$, i.e.,

$$BestScore(X, \mathbf{U}) = \min_{PA_X \subseteq \mathbf{U}} score(X|PA_X).$$

The g cost of a node $\mathbf{U}$ is equal to the sum of the arc costs from start to $\mathbf{U}$, and $g(\mathbf{U})$ corresponds to the score of the optimal subnetwork over $\mathbf{U}$.

In this formulation, a path from $start$ to $goal$ induces an order on the variables, so this graph is called the *order graph*. Because each variable selects optimal parents from the existing subset when it is added, the best structure over that ordering is found by combining all of the parent sets. The shortest path corresponds to the global optimal network. In contrast to most other structure learning algorithms (Silander and Myllymaki, 2006; Jaakkola et al., 2010; de Campos and Ji, 2011; Cussens, 2011), etc., this formulation searches for the structure with the minimum score. Multiplying local scores by -1 can transform between maximization and minimization.

The $BestScore(\cdot)$ values are calculated with *parent graphs*. The parent graph for X contains all subsets of $\mathbf{V} \setminus \{X\}$. A parent graph containing local scores for X_1 is shown in Figure 2(a). Figure 2(b) shows that, by propagating $BestScore(\cdot)$ from subsets to supersets, the same score can be used for many nodes. Node $\mathbf{U}$ in the parent graph of X gives the cost from $\mathbf{U}$ to $\mathbf{U} \cup \{X\}$ in the order graph.

A consistent heuristic was also given which estimates a lower bound on the cost from a node $\mathbf{U}$ to $goal$. The distance from $start$ to $\mathbf{U}$ is $g(\mathbf{U})$, and the estimated cost to $goal$ is $h(\mathbf{U})$. The f value, which is $f(\mathbf{U}) = g(\mathbf{U}) + h(\mathbf{U})$, always gives an optimistic estimate on the cost of a path through $\mathbf{U}$.

An A* algorithm (Yuan et al., 2011) was shown to be an order of magnitude faster than DP; however, it requires both the parent and order graphs in RAM. Malone *et al.* (2011b) developed a breadth-first branch and bound (BFBnB) algorithm to search

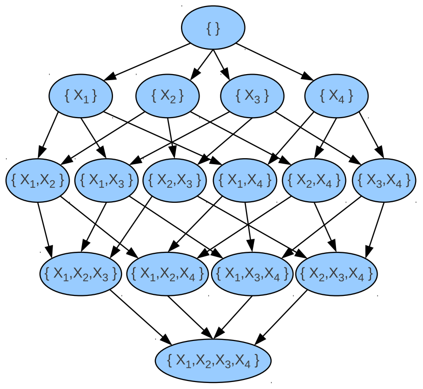

Figure 1: The order graph.

the graph layer by layer. This strategy allowed most data structures to be stored on disk and efficiently read into RAM when needed. BFBnB ran as fast as A* and scaled to datasets with more variables. An improved heuristic function was recently proposed (Yuan and Malone, 2012) which further improved the A* and BFBnB search algorithms.

3 A Bounded Error, Anytime Parallel Algorithm

The structure learning algorithms described in Section 2.3 do not have anytime behavior. They also do not take advantage of modern, multi-core architectures. In this section, we present a *bounded error, anytime parallel* search algorithm (BEAP) that has anytime behavior and bounded error for all solutions. Given enough resources, it guarantees to find the optimal network structure. It greatly improves the scalability compared to A* and BFBnB.

We give details of the algorithm in the following subsections. Section 3.1 describes a new sparse representation for the parent graphs and integration of those into A*. Section 3.2 presents our new parallel algorithm based on weighted A* and dovetailing to give bounded error and anytime behavior.

3.1 Sparse Parent Graph Representation

Existing formulations of the parent graphs for each variable X explicitly store $BestScore(X, \mathbf{U})$ for all subsets $\mathbf{V} \setminus \{X\}$. More efficient representations (Malone et al., 2011a; Malone et al., 2011b) store $O(C(n-1, \frac{n}{2}))$ scores for each variable at each layer of the search. In many cases, though, the following theorem guarantees that far fewer parent sets could be optimal (Teyssier and Koller, 2005).

Theorem 1. *Let* $\mathbf{U} \subset \mathbf{T}$. *If* $score(X|\mathbf{U}) < score(X|\mathbf{T})$, $\mathbf{T}$ *is not the optimal parent set for* X.

We adopt a different approach to leverage the pruning offered by Theorem 1. After pruning, we *sort* the remaining parent scores for each variable X in a list called $scores_X$ in increaseing (worsening) order of scores; we store the associated parent sets in an analogous list called $parents_X$. Because of its sorted order, the first item in $scores_X$ gives $BestScore(X, \mathbf{V} \setminus \{X\})$. To find $BestScore(X, \mathbf{U})$, we scan the list from the beginning; the first parent set which is a subset of $\mathbf{U}$ corresponds to $BestScore(X, \mathbf{U})$. Yuan and Malone (2012) describe an efficient scanning technique. In effect, this data structure allows us to efficiently operate on the pruned parent graph shown in Figure 2(c).

3.2 Bounded Error, Anytime, Parallel Search

In this section, we develop a *bounded error, anytime, parallel* search algorithm (BEAP). This algorithm is an example of parallel dovetailing (Valenzano et al., 2010) using Weighted A* (Pohl, 1970; Pearl, 1984).

The bounded error aspect of our algorithm results from using Weighted A* (WA*) (Pohl, 1970; Pearl, 1984), which is a variant of A* which weights the heuristic function by a factor ϵ. That is, $f(\mathbf{U}) = g(\mathbf{U}) + \epsilon \times h(\mathbf{U})$. By weighting the heuristic, it is no longer admissible, so the f value for $\mathbf{U}$ may over-estimate the cost of a path to the goal through $\mathbf{U}$. However, upon expanding a goal node, its cost is guaranteed to be no more than a factor of ϵ greater than the globally optimal solution (Pearl, 1984). For example, if $\epsilon = 1.05$, and we expand a goal node with cost f, then the globally optimal solution is guaranteed to be no more than 5% better than f. WA* never re-expands any nodes.

We add the anytime and parallel behavior to our algorithm by adapting parallel dovetailing (Valenzano et al., 2010). Valenzano *et al.* (2010) observed that many search algorithms require some sort of parameter configuration. For example, WA* is parameterized by the weight ϵ. Parallel dovetailing begins by selecting a variety of parameter configurations and running each configuration in parallel processes. All configurations run until any process finds a solution, regardless of its optimality. At that point, all processes receive a message that a solution has been found, and the search stops. Because of this behavior, the version of parallel dovetailing presented is a suboptimal search strategy.

BEAP blends the advantages of WA* and parallel dovetailing. In this algorithm we select a range of ϵ values and run one WA* process for each value in parallel. Unlike previous versions of dovetailing, we do not stop after finding a solution; instead, each process continues until completion. We adapt the A* search algorithm (Yuan et al., 2011) into WA* by passing ϵ_i as an input to the algorithm in process i. The only algorithmic change is that, when calculating the heuristic value h, we multiply by ϵ_i, so the f value for a node is $f(\mathbf{U}) = g(\mathbf{U}) + \epsilon_i \times h(\mathbf{U})$. The processes do not communicate, so they expand some of the same nodes.

The anytime behavior of the parallel algorithm results because, as the WA* instances complete, their solutions give an upper bound on the optimal score. Typically, processes with large ϵ_i values finish very quickly, but the scores of the learned network are high (always bounded by ϵ_i, though). Processes with lower ϵ_i values finish more slowly, but have better scores. Therefore, as the search progresses and WA* instances complete, the upper bound improves. A process in which $\epsilon = 1$, denoted as ϵ_1, corresponds to the unweighted, exact A* algorithm. Therefore, the completion of that process guarantees to give the globally optimal network.

As each WA* process completes, the quality of the solution is bounded by ϵ_i of that process. As more processes complete, the provable bound between the optimal network and the best learned network decreases. Running ϵ_1 offers another way to bound the error. Because ϵ_1 does not weight the heuristic, no optimal network could possibly have a score better than the f value of the most recently expanded node of ϵ_1, so that serves as a lower bound on the optimal network score. That lower bound is guaranteed to increase (or stay the same) with each node expanded in ϵ_1 because of the best-first expansion. Therefore, the ratio between the score of the best learned network and the lower bound of the

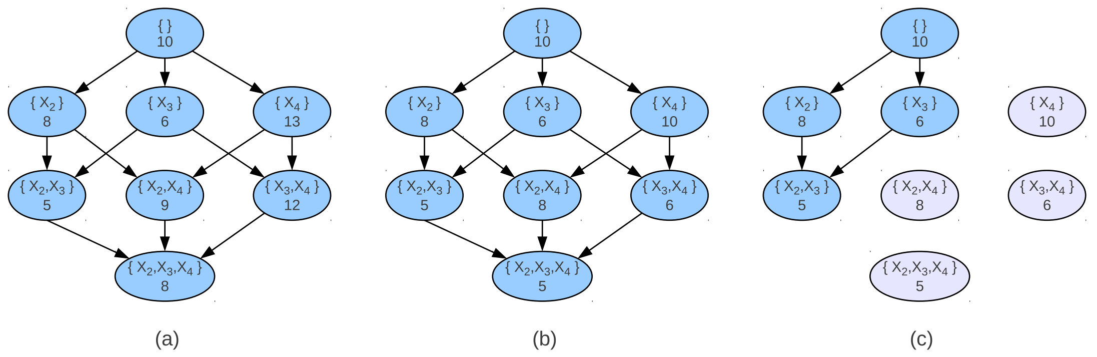

Figure 2: A sample parent graph for variable X_1. (a) The local scores, $score(X_1, \cdot)$ for all the parent sets. (b) The optimal scores, $BestScore(X_1, \cdot)$, for each candidate parent set. (c) The unique optimal parent sets and their scores. Pruned parent sets are in gray. A parent set is pruned if any predecessors has a better score.

most recently expanded node of ϵ_1 also bounds the solution quality. As shown in Section 4, the ratio bound is often tighter than the bound guaranteed by ϵ_i of the other WA* processes.

4 Experimental Results

We compared BEAP to BB and another serial anytime search algorithm, Anytime WA*.

4.1 Experimental Design

Anytime WA* (AWA*) (Hansen and Zhou, 2007) begins as the normal WA* algorithm; however, rather than stopping the search as soon as a solution is found, AWA* continues to expand nodes. As better paths to a goal are found, the best solution is updated, which gives the algorithm its anytime behavior. Eventually, unless interrupted, the search expands or prunes all nodes in the search space and terminates with the optimal solution. Because of the weighted heuristic, AWA* may find a better path to a closed node. To guarantee optimality of the final solution, AWA* must re-expand those nodes.

We evaluated BEAP on a set of benchmark datasets from the UCI repository (Frank and Asuncion, 2010). For all datasets, we removed records with missing values and discretized all variables into two states. The experiments were performed on a PC with 3.07 GHz Intel i7 processor and 16 GB of RAM. We compared BEAP to BB and a custom implementation of AWA*. The AWA* implementation is a straight-forward adaptation of the existing A* algorithm (Yuan et al., 2011). Even though they are anytime algorithms, we did not compare to any local search algorithms because they do not give an error bound. For BEAP, we used four different values of ϵ: 1.2, 1.08, 1.04 and 1. We empirically determined that $\epsilon > 1.2$ did not improve learning. We allowed all algorithms a total execution time of 30 minutes, not including local score calculations. BB and AWA* are sequential, so we gave them 30 minutes of wall clock time. Since BEAP used four processes (one for each value of ϵ), we gave it 7.5 minutes of wall clock time, so its total time was also 30 minutes. Each BEAP process had 4 GB of RAM.

4.2 Node Expansion

We first evaluated the number of nodes expanded by BEAP for each value of ϵ. The results in Figure 3 show that the algorithm typically found high quality solutions quickly. The figure also sheds insight into several characteristics of the search algorithm.

First, the searches with high ϵ usually expand a very small number of nodes. For example, on five of the datasets, the process with $\epsilon = 1.2$ expands the minimum number of nodes possible to find a solution ($n + 1$). This takes only a fraction of a second; that processor is idle for the rest of the search. A more sophisticated scheme could be used to more fully utilize the available resources.

Second, the figure suggests that, like other combinatorial optimization problems, Bayesian network structure learning has a *critical point* (Zhang and Pemberton, 1994). A critical point for a problem is a point at which the problem difficulty undergoes a major change. Based on Figure 3, the critical point for structure learning appears to be between

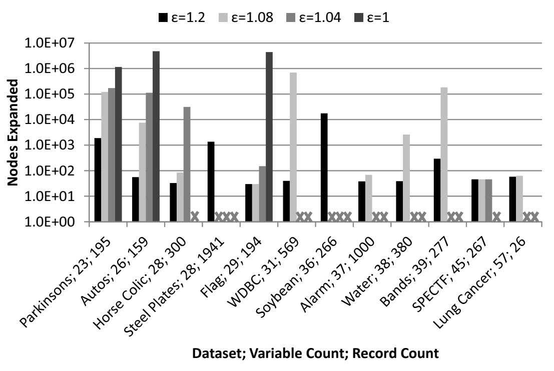

Figure 3: The number of order graph nodes expanded for each dataset and value of ϵ by BEAP. An "X" indicates that the search did not complete within the resource restrictions.

8% and 4% of optimal. Nearly all of the instances for $\epsilon = 1.08$ complete quickly; however, over half fail for $\epsilon = 1.04$. These results indicate that finding a network that is 8% of optimal is much easier than finding one that is 4% of optimal.

4.3 Comparison of Anytime Behavior

We next compared the convergence and anytime behavior of BEAP to BB. As the convergence curves in Figure 4 show, BEAP finds provably high quality solutions very quickly on all of the datasets. For both $Flag$ and $SPECTF$, within 2 seconds of wall clock time (8 seconds of CPU time), BEAP found networks with scores provably within 2.5% of optimal. The curves demonstrate that BEAP and BB improve error bounds differently. BB never improves its initial solution, but spends the entire 30 minutes improving its lower bound. As BEAP processes complete and ϵ_1 expands nodes, both upper and lower bounds improve.

4.4 Comparison of Solution Quality

Finally, we compared the solution quality of BEAP to AWA* and BB by comparing their upper and lower bounds. As Figure 5 shows, BEAP almost always finds a solution with a tighter error bound than the other algorithms. BEAP is the only algorithm which finds and proves the optimal structure on any of the datasets. It found tighter solutions than AWA* because BEAP never re-expands nodes within the same process; AWA* must re-

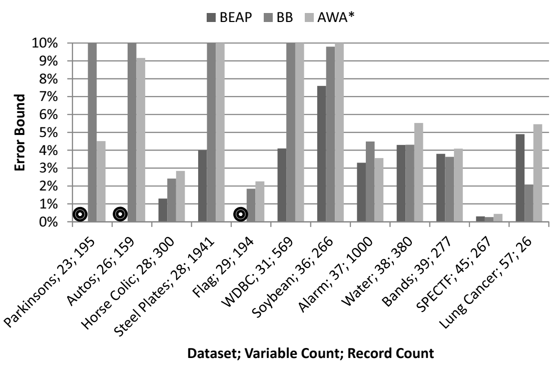

Figure 5: The error bound calculated by BEAP, AWA* and BB. An "O" indicates that BEAP found and proved the optimal network.

expand a node each time it finds a better path to it. BB searches in the space of cyclic graphs, so these results suggest that the heuristic search formulation more effectively guides the algorithm to higher quality solutions than breaking cycles.

The bounds for BEAP are always better than the best ϵ_i that was solved (shown in Figure 3). This shows that the bound given by the ratio between ϵ_1 and the best solution is always tighter.

For all algorithms, these results compare very favorably to those for parallel DP (Tamada et al., 2011). That algorithm took 483,874 seconds to find the optimal network for a 32 variable dataset. Of that time, 392,186 seconds were spent in MPI communication. Their algorithm also required 836.1 GB of RAM. In contrast, our algorithm used at most 16 GB, and typically less than 8 GB, which is an improvement of nearly two orders of magnitude.

5 Discussion

BEAP has several advantages compared to other parallel Bayesian network structure learning algorithms. First, it has very little communication overhead because each WA* process uses a different ϵ; the processes do not communicate. The limited communication ensures that runtime is not wasted passing messages or waiting for synchronization, which plagued the parallel DP algorithm (Tamada et al., 2011). Second, a proper range of ϵ_is gives the parallel algorithm very good anytime behavior. The parallel DP algorithm (Tamada et al., 2011) does not have anytime behavior at all.

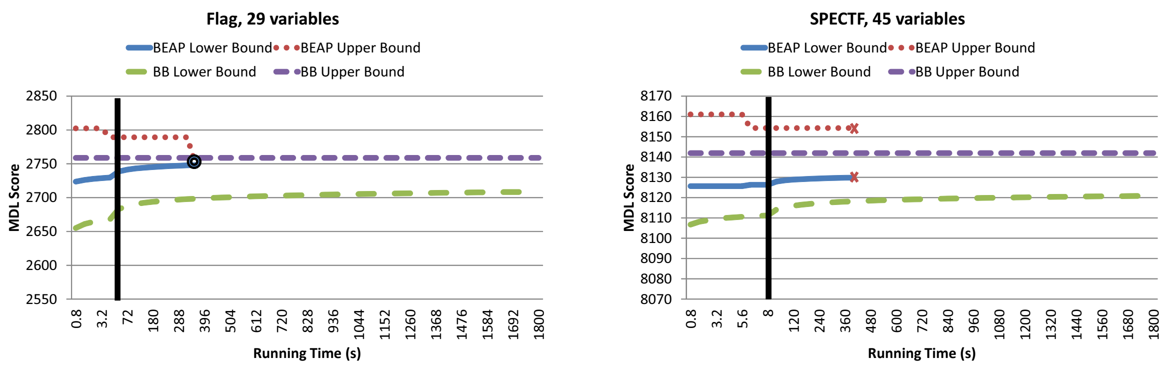

Figure 4: Convergence behavior of BEAP and BB. A black bar indicates a change in time scale. The running time is CPU, not wall clock, time. An "O" indicates that BEAP found and proved the optimal network. An "X" indicates that BEAP exceeded the RAM constraints. BB ran for the entire 30 mintues for both datasets.

BEAP also has some similarites to, and advantages over, several serial anytime search algorithms, including AWA* (Hansen and Zhou, 2007) and Anytime Repairing A*(ARA*) (Likhachev et al., 2003). All three algorithms use a weighted heuristic to provably bound the error of solutions. BEAP offers advantages over these serial anytime search algorithms, though. First, BEAP re-expands nodes in parallel rather than serially. Second, in order to calculate a tighter bound than that given by ϵ, AWA* and ARA* must search through the open list and calculate the true f value of each node. In constrast, BEAP simply uses the f value of the most recently expanded node of ϵ_1. Third, unlike ARA*, BEAP does not require any data structures other than those normally required by A*. Like AWA* and ARA*, though, BEAP is a general purpose search algorithm that could be applied to any heuristic search problem, not just structure learning.

6 Conclusions

In this paper, we have presented a general purpose bounded error, anytime, parallel search algorithm based on weighted A* and applied it to the Bayesian network structure learning problem. We described a sparse, efficient representation to calculate $BestScore(\cdot)$. Experimentally, we showed that our algorithm scales to many more variables than existing dynamic programming and shortest-path structure learning algorithms. We also showed that our algorithm often finds better solutions more quickly than existing error bounded, anytime algorithms. In the future, we would like to investigate other pruning techniques to examine the critical point behavior of the structure learning problem.

Acknowledgments

This work was supported by NSF CAREER grant IIS-0953723 and EPSCoR grant EPS-0903787.

References

David Maxwell Chickering. 1996. Learning Bayesian networks is NP-complete. In *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer-Verlag.

David Maxwell Chickering. 2002. Learning equivalence classes of Bayesian-network structures. *J. Mach. Learn. Res.*, 2:445–498.

Gregory F. Cooper and Edward Herskovits. 1992. A bayesian method for the induction of probabilistic networks from data. *Mach. Learn.*, 9:309–347, October.

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

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

A. Frank and A. Asuncion. 2010. UCI machine learning repository.

Eric A. Hansen and Rong Zhou. 2007. Anytime heuristic search. *Journal of Artificial Intelligence Research*, 28:267–297.

P E Hart, N J Nilsson, and B Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions On Systems Science And Cybernetics*, 4(2):100–107.

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

David Heckerman. 1996. A tutorial on learning with Bayesian networks. Technical report, Learning in Graphical Models.

Tommi Jaakkola, David Sontag, Amir Globerson, and Marina Meila. 2010. Learning Bayesian network structure using LP relaxations. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*.

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

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

M. Likhachev, G. Gordon, and S. Thrun. 2003. ARA*: Anytime A* search with provable bounds on sub-optimality. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Proceedings of Conference on Neural Information Processing Systems (NIPS)*. MIT Press.

Brandon Malone, Changhe Yuan, and Eric Hansen. 2011a. Memory-efficient dynamic programming for learning optimal Bayesian networks. In *Proceedings of the 25th national conference on Artifical intelligence*.

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

Andrew Moore and Weng-Keen Wong. 2003. Optimal reinsertion: A new search operator for accelerated and more accurate Bayesian network structure learning. In *Intl. Conf. on Machine Learning*, pages 552–559.

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

Pekka Parviainen and Mikko Koivisto. 2009. Exact structure discovery in Bayesian networks with less space. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, Montreal, Quebec, Canada. AUAI Press.

Judea Pearl. 1984. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Ira Pohl. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3-4):193 – 204.

Tomi Silander and Petri Myllymaki. 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.

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

Yoshinori Tamada, Seiya Imoto, and Satoru Miyano. 2011. Parallel algorithm for learening optimal bayesian network structure. *Journal of Machine Learning Research*, 12:2437–2459.

Marc Teyssier and Daphne Koller. 2005. Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *Proceedings of the Twenty-First Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 584–590, Arlington, Virginia. AUAI Press.

Richard Valenzano, Nathan Sturtevant, Jonathan Schaeffer, Karen Buro, and Akihiro Kishimoto. 2010. Simultaneously searching with multiple settings: An alternative to parameter tuning for suboptimal single-agient search algorithms. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*.

Changhe Yuan and Brandon Malone. 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.

Changhe Yuan, Brandon Malone, and Xiojian Wu. 2011. Learning optimal Bayesian networks using A* search. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*.

Weixiong Zhang and Joseph C. Pemberton. 1994. Epsilon-transformation: exploiting phase transitions to solve combinatorial optimization problemsinitial results. In *Proceedings of the twelfth national conference on Artificial intelligence (vol. 2)*, AAAI'94, pages 895–900, Menlo Park, CA, USA. American Association for Artificial Intelligence.