Finding Optimal Bayesian Networks by Dynamic Programming

Ajit P. Singh and Andrew W. Moore

June, 2005
CMU-CALD-05-106

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Finding the Bayesian network that maximizes a score function is known as *structure learning* or *structure discovery*. Most approaches use local search in the space of acyclic digraphs, which is prone to local maxima. Exhaustive enumeration requires super-exponential time. In this paper we describe a "merely" exponential space/time algorithm for finding a Bayesian network that corresponds to a global maxima of a decomposable scoring function, such as BDeu or BIC.

NSF IIS-0325581, NSERC PGS-B

Keywords: Bayesian networks, structure learning, dynamic programming, P-cache

1 Introduction

Learning the structure of a Bayesian network without restrictive assumptions is a hard problem. The number of possible structures is $O\left(n!2^{\binom{n}{2}}\right)$ (Robinson, 1973), making exhaustive enumeration impractical. Under a wide variety of assumptions, including those of this paper, structure learning is NP-hard (Chickering, 1996; Chickering et al., 2003).

We propose an algorithm, OptOrd, to find a Bayesian network that maximizes a decomposable scoring function, like BDeu. This algorithm is feasible for $n < 26$, and is useful for learning moderately sized networks. One of the uses of OptOrd is that we are able to quantify how close a heuristic structure learning algorithms gets to a global optimum. Another use is that it is now possible to study the properties of decomposable scoring functions without the potential for results being distorted by the behaviour of local search.

2 Problem Description

We are given a fully observed data set D on n categorical variables. A Bayesian network represents relationships among variables as a directed acyclic graph $G = (V, E)$ where each node corresponds to a variable and each edge to a dependence relationship. We are also given a scoring function DagScore(D) that is *decomposable*, that is

$$\textsc{DagScore}(G) = \sum_{x \in V} \textsc{NodeScore}(x|parents(x)) \tag{1}$$

where NodeScore depends on the parameters of the conditional density $x|parents(x)$. The most common scoring metrics are decomposible: BIC (Schwartz, 1979), BD (Cooper & Herskovits, 1992), BDeu (Buntine, 1991), BDe (Heckerman et al., 1995). Structure discovery is the task of finding a model that maximizes DagScore.

In this paper we use the notion of an variable ordering (order). An order π is a permutation of the variables where if $\pi_i = x$ then $parents(x) \subseteq \{\pi_1 \dots \pi_{i-1}\}$. Any digraph respecting the parent constraint must be acyclic. An order encodes a set of $O(2^n)$ possible structures, and a structure can be consistent with many orders.

3 Exact Structure Discovery

What if one were given a oracle that, for $A \subseteq V$ and $x \in A$, returns the best subset of parents from $A - \{x\}$ as well as the corresponding node score ? Our dynamic programming algorithm assumes the existence of such an oracle [1]. It suffices to find an ordering of variables that contains the highest scoring network[2]. To construct the best network apply the oracle to each node, with its predecessors in the ordering as potential parents.

We start with a simpler question than full structure discovery. Every acyclic digraph has at least one leaf. Can we identify a leaf of the best network, $x_\ell = \textsc{Leaf}(V)$? Since x_ℓ cannot be the parent of any other variable, the score of the best network on V consists of two parts: (i) the score of the best network on $V - \{x_\ell\}$ and (ii) the score of $\textsc{Leaf}(V)$

[1] The oracle is a P-cache (section 4).

[2] There can be many networks that achieve the highest score.

given the best set of parents from $V - \{x_\ell\}$, *i.e.*, the set of parents that maximizes the node score of x_ℓ. This argument requires a decomposable scoring metric. Recursively, we can ask which node is a leaf in the best network on $V - \{x_\ell\}$. This process induces a (reverse) order on the variables. It remains to be shown how to choose $\textsc{Leaf}(V)$.

Suppose you had the score for the best network on $V - \{x\}$ for each $x \in V$. Since a leaf cannot influence the node score of any other variable, $\textsc{Leaf}(V)$ is the node whose removal maximizes the score of the best network on the remaining variables. Formally

$$\textsc{Score}(V) = \max_{x \in V} \textsc{Score}(V - \{x\}) + \textsc{BestScore}(V, x) \tag{2}$$

$$\textsc{Leaf}(V) = \operatorname*{argmax}_{x \in V} \textsc{Score}(V - \{x\}) + \textsc{BestScore}(V, x) \tag{3}$$

$$\textsc{BestScore}(V, x) = \max_{\text{PS} \subseteq V - \{x\}} \textsc{NodeScore}(x | PS)$$

where $\textsc{Score}(\emptyset) \equiv 0$, $\textsc{Leaf}(\emptyset) \equiv nil$, and PS denotes a parent set.

To visualize the algorithm consider the subset lattice in figure 1. There is one state for each subset of V. The children (successors) of state S in the lattice correspond to potential choices for $\textsc{Leaf}(S)$.

A naïve depth-first search of the lattice corresponds to direct evaluation of recurrences 2 and 3. The algorithm begins at the bottom of the lattice in figure 1 and searches all paths to the top. Each transition from state S to a successor corresponds to choosing a leaf in a network on the nodes in S. Thus a single path from $\{1, 2, \ldots, n\}$ to $\{\}$ corresponds to a sequence of leaf node removals, a (reverse) variable ordering.

A simple depth-first search of the lattice would require $O(n!)$ time and $O(n)$ space. The alternative is more memory intensive. Once the value of $\textsc{Score}(S)$ is computed on one search path, it can be memoized. If $\textsc{Score}(S)$ is cached, future search paths that arrive at S can be pruned. This is dynamic programming on the subset lattice (algorithm 1).

When algorithm 1 terminates the value of $\textsc{Leaf}(S)$ is known for every $S \subseteq V$. The last element of the order is $\textsc{Leaf}(V)$. The second last element of the order is $\textsc{Leaf}(V - \textsc{Leaf}(V))$, etc. For memory efficiency, one can forgo storage of $\textsc{Leaf}(V)$, as it can be recovered in $O(n^2)$ time once all the $\textsc{Score}(\cdot)$ terms have been computed.

3.1 Complexity

OptOrd must store one floating point number for each state in the subset lattice, and this requires $O(2^n)$ memory. In the worst case, the oracle can require $O(n2^n)$ memory, but will almost certainly never reach the worst case in practice. Since $\textsc{Score}(\cdot)$ is computed for each state there are 2^n calls to the oracle. As will be seen in section 4 the worst-case cost of invoking the oracle on each $x \in S$ is $|S|2^{|S|-1}$. In practice the cost is much less; the oracle uses a branch-and-bound algorithm. Summing over all states in the subset lattice the complexity is $O(n3^n)$. If the indegree of a node is bounded the overall cost is reduced to $O(n2^n)$. These are loose upper bounds.

The cost of storing the lattice restricts OptOrd to moderate values of n. A simple alternative, Rand Orders, generates random orderings and uses the oracle to find the best network in each order. This is similar to the use of random orderings as inputs to K2, except that we find the provably best network in each order.

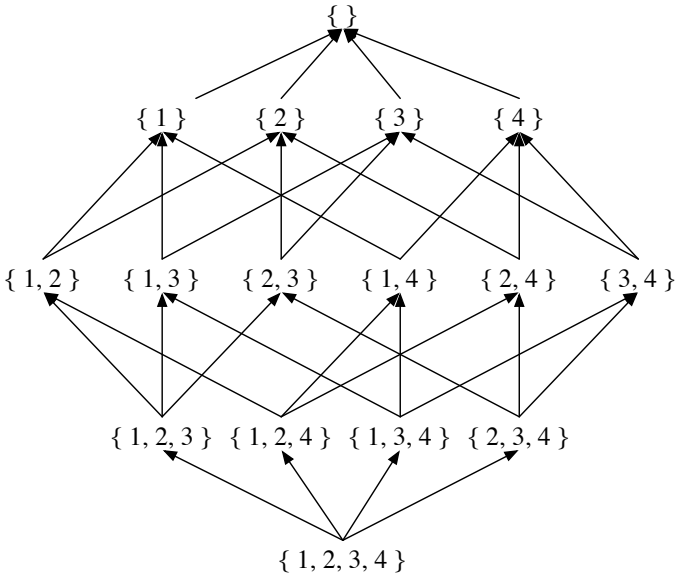


Figure 1: Subset lattice on a network with four nodes

4 P-Caches

Section 3 assumed the existence of an oracle that can choose the best subset of parents from a candidate set. In this section we describe an implementation of the oracle, using a set of data structures, known as P-caches.

Given n variables, a parent cache (P-cache) for variable x compactly records all pertinent information about the 2^{n-1} possible parent sets of variable x. Throughout this section we assume wlog that the $n-1$ potential parents of a variable are designated by integers $1, 2, \ldots, n-1$. Henceforth we assume the BDeu score is used, but these ideas work with any decomposable score.

A P-cache supports two operations: PSCORE(PS) which returns the node score of x in which PS $\subseteq \{1, 2, \ldots, n-1\}$ are the parents of x and BESTPS(x, PS$_{\text{avail}}$) which finds the subset of PS$_{\text{avail}}$, the set of potential parents, which maximizes the score.

4.1 Terminology

We introduce the following notation. An *extension* of a parent set PS is the set of all parent sets that include PS as a subset and, other than the members of PS, contain only variables that have a larger integer designation than the elements of PS. If a parent set is treated as a sorted list of integers PS $= \{i_1, i_2, \ldots, i_k\}$ then we can also define prefixes.

$$\begin{aligned}
\text{MAXMEMBER(PS)} &= \text{largest integer in PS} \quad (\text{MAXMEMBER}(\emptyset) = 0) \\
\text{EXTENSIONS(PS)} &= \{\text{PS'} | \text{PS} \subseteq \text{PS'} \wedge \forall i \in (\text{PS'} - \text{PS}), i > \text{MAXMEMBER}(PS)\} \\
\text{PREFIXES(PS)} &= \{\emptyset, \{i_1\}, \{i_1.i_2\}, \ldots, \{i_1, \ldots, i_{k-1}\}\}
\end{aligned}$$

A parent set is *useless* iff no extension of PS has a higher score than the highest scoring prefix of PS. If a parent set is useless then it is easy to prove that no extension of it can

Algorithm 1: OPTORD(S) — Store score of the best network on S in CACHE(S)

```
bestscore ← −∞
if S = ∅ then return 0.0
foreach x ∈ S do
    if CACHED(S-x) then
        s ← CACHE(S-x)        // if cached retrieve the score of best network on S − x
    else
        s ← OPTORD(S-x)       // if not cached compute score of best network on S − x
    s ← s + BESTSCORE(S,x)    // add in the score of the best subset of parents for x
    if bestscore < s then
        LEAF(S) ← x
        bestscore ← s
CACHE(S) ← bestscore
return bestscore
```

be the parent set for x in the optimal Bayesian network.

A P-cache for variable x is a data structure that stores a set of tuples

$$(\text{PS}_i, \text{PSCORE}(\text{PS}_i), \text{MAXEXT}(\text{PS}_i))$$

where

$$\begin{aligned} \text{PSCORE}(\text{PS}) &= \text{score for using PS as parents of } x \\ \text{MAXEXT}(\text{PS}) &= \max_{\text{PS'} \in \text{EXTENSIONS}(\text{PS})} \text{PSCORE}(\text{PS'}) \end{aligned}$$

where a parent set does not appear in the P-cache if it is useless, and appears once otherwise. Tuples are stored in a data structure that supports efficient mapping from any parent set to its cached value. Once the P-cache is built computing PSCORE($\cdot$) is simply a lookup. While we cannot retrieve values for useless parent sets, those values are not required to find the optimal structure. Naïvely computing BESTPS(x, PS) would require searching all possible subsets of PS

4.2 Constructing the P-cache

Instead of explicitly iterating over all 2^{n-1} parent sets, we perform a depth-first search in which the state is a parent set. In algorithm 2 the recursive function takes a set of potential parents for variable x, PS, as well as the node score of the best prefix, BPS, and returns the node score of x given the best set of parents from PREFIXES(PS) $\cup$ EXTENSIONS(PS).

If we can prove that the score of any extension must be less than the score of the best prefix of PS, we can prune the search over parent sets (note 1). The for-loop generates every possible extension of PS, and keeps track of the best score for extensions and prefixes of extensions of PS (note 2). If we find an extension of PS that is better than the best prefix of PS, we record the fact in the P-cache (note 3). Thus we never store a useless parent set. To compute all the P-caches on a network with variables V run CONSTRUCT($x, \{\}, 0$) $\forall x \in V$.

An important practical detail when evaluating scores, or upper bounds on scores, is that they require counting records that match a particular condition. We use a variant of AD-tree search (Moore & Lee, 1998) to amortize the cost of counting operations on larger data sets.

For the BDeu score[3] we propose the following bound for BOUNDEXTENSIONSCORE(PS). We have a conditional probability table representing the density $x_i|PS$. Imagine there exists a "virtual" variable x_i^* that, when added to PS, yields a perfect predictor for x_i. That is, all the data that matches an assignment to $PS \cup \{x_i^*\}$ shares the same value for x_i. Each row of $x_i|PS$ is shattered into a set of pure virtual rows. Formally, let the j^{th} row in $x_i|PS$ be denoted by $(N_{ij1}, N_{ij2}, \ldots, N_{ijr_i})$. This row is transformed into a set of virtual rows in $x_i|(PS \cup x_i^*)$, denoted $\{v_1, \ldots, v_m\}$. The virtual rows are pure iff $\forall k\ v_k = (0, \ldots, 0, N_{ijk}, 0, \ldots, 0)$. An illustration of the transformation from a row to pure virtual rows is shown in figure 2(a) in appendix B. Let $\mathcal{B}_{ij}^k(S, D)$ denote the row score of virtual row v_k derived from row ij in the original network. Note that the introduction of virtual rows respects marginal constraints, if x_i^* is marginalized out of the virtual rows $\{v_1, \ldots, v_m\}$ we get back the original row $(N_{ij1}, \ldots, N_{ijr_i})$.

Theorem 1. *Consider a row $(N_{ij1}, \ldots, N_{ijr_i})$ and the virtual rows $\{v_1, \ldots, v_m\}$ formed by adding variable x^* to the parent set. If the virtual rows are pure then the BDeu score is maximized.*

Proof. See appendix B for the proof. The virtual rows transformation of this theorem is how BOUNDEXTENSIONSCORE is implemented. □

CONSTRUCT is a branch-and-bound algorithm, and in the worst case the P-cache can have $O(2^n)$ tuples. However, good Bayes net scoring functions penalize complex models, and so many parent sets are useless and not stored.

4.3 Searching a P-cache

When constructing the P-cache for variable x all that was stored were useful parent sets, the node score of x given the parent set, and the best node score of x given an extension of the parent set. Algorithm 3 is a recursive function that finds the best subset of parents from a set of potential parents. Two invariants are maintained

$$\forall i \in PS_{avail}, i > \text{MAXMEMBER(PS)}$$

$$bestscore \geq \max_{\text{PS' } \in \text{ PREFIXES}(D)} \text{PSCORE(PS')}$$

The function returns the score of the best subset of parents. Typically the call is

$$\text{BESTSCORE}(x, \{\}, PS_{avail}, 0, PC_x).$$

If PS is non-empty those variables are forced to be parents of x.

BESTSCORE is a branch-and-bound algorithm where we consider the node score of x when we either throw away a potential parent (note 1) or make it an actual parent of x (note 2). This continues as long as there is an extension of PS with a better score. In the worst case the run-time is exponential in $|PS_{avail}|$, but is typically much faster.

[3]The Bayesian Dirichlet class of scoring metrics is explained in appendix A.

Algorithm 2: CONSTRUCT(x, PS, BPS) — Construct P-cache for x

x	child variable, the variable we are building the P-cache for.
PS	set of potential parents for x
BPS	node score of x given the best prefix of PS
score	node score of x given PS
mes	node score of x given the best extension of PS (the return value)
bps_i	node score of x given the best prefix of $PS \cup \{i\}$
$bpoe_i$	node score of the best prefix or extension of $PS \cup \{i\}$

if BOUNDEXTENSIONSCORE$(PS) \leq BPS$ **then return** BPS
$mes = score =$ NODESCORE$(x \mid PS)$
q = MAXMEMBER(PS)
for $i \in \{q+1, q+2, \ldots, n-1\}$ **do**
  $bps_i =$ MAX$(BPS,\ score)$
  $bpoe_i =$ CONSTRUCT$(x,\ PS \cup \{i\},\ bps_i)$
  $mes =$ MAX$(mes,\ bpoe_i)$
if $mes > BPS$ **then** add $(PS,\ score,\ mes)$ to P-cache for x
return *mes*

5 Experiments

The purpose of our experiments is to compare the performance of OPTORD to heuristic algorithms. Since our concern is with the algorithmic problem of structure discovery, and not the quality of scoring functions, we leave a comparison of the networks produced for future work. All experiments were done using BDeu scoring on an Opteron 1.8 GHz machine.

5.1 Data Sets

The data sets used are presented in table 1. Seven of the data sets (nursery, zoo, adult, letters, covtype, mushroom, and autos) are from the UCI ML repository (Blake & Mertz, 1998). Real-valued attributes were discretized to binary attributes by thresholding on the median value. The biosurv data set contains records on hospital admissions in Pennsylvania (Moore & Wong, 2003). For some of the data sets, we do not include all available records for computational reasons (*i.e.*, adult, covtype, biosurv, mushroom, alarm). Parity consists of uniform noise for the first 9 attributes, with the last attribute being the parity of the first 9.

5.2 Results

Before OPTORD can be used to find the optimal Bayesian network, the P-cache must be constructed (see table 1). While the virtual rows bound (§4.2) is a true upper bound, it is loose and yields only limited forward pruning in BOUNDEXTENSIONSCORE. However, once the node score of a parent set is computed it is possible to prove it is useless (post-pruning). Thus relatively small P-caches can take a long time to build.

To illustrate the behaviour of OPTORD we compare it to three other discovery algo-

Algorithm 3: BESTSCORE($x, PS, PS_{avail}, bestscore, PC_x$) — Find best parents for x

PC_x	P-cache for variable x
PS	Current set of best parents for x
PS_{avail}	Set of potential parents for x
bestscore	node score of x given PS
res	Intermediate variable, floating point
MES(PS)	Returns the score of the best extension PS from PC_x
SCORE(PS)	Returns the score of PS from PC_x

if $PS \notin$ *P-cache* **then return** *bestscore*
res = MAX(*bestscore,* SCORE(PS))
if MES(PS) $>$ *res* $\wedge$ $PS_{avail} \neq \{\}$ **then**
  m = MIN(PS_{avail})
1 *res* = BESTSCORE(*x, PS,* $PS_{avail} - \{m\}$*, res,* PC_x)
2 *res* = BESTSCORE(*x,* $PS \cup \{m\}$*,* $PS_{avail} - \{m\}$*, res,* PC_x)
return *res*

rithms: RAND ORDERS (§3); OPT REINSERT, the optimal reinsertion algorithm of (Moore & Wong, 2003), which originally used P-caches without forward pruning; HILLCLIMB, an optimized version of hillclimbing that uses multiple restarts and caching to ensure node scores are never recomputed. For zoo and autos, all algorithms use the indegree constraint.

The results in table 2 confirm that OPTORD is producing a higher scoring solution than the alternatives. If OPTORD is too slow then RAND ORDERS provides a reasonable anytime alternative. However, because the search over orders is uninformed, it will generally be beaten by optimal reinsertion. The run-time of OPTORD can be dramatically affected by the cost of P-cache lookups. For example, on biosurv when $k = 6$ the optimal structure is found in 5802s (vs. 12649s when $k = 22$). In both cases the score of the best network was the same. To put the results in perspective, even if we could score 100 structures per second it would take over 10^{77} years to test all possible structures for a 22 node network.

5.3 Discussion

Two forms of path pruning were attempted: (i) pruning if the path score was lower than the best network found thus far. This happens rarely, as it is unusual for a network on a small subset of variables to have a lower score than any network on all the variables. (ii) By ignoring acyclicity we note that $\sum_{x \in A}$ BESTSCORE($x, A - \{x\}$) is an upper-bound on the score of a network on A. If the path score, the score of leaves chosen so far, plus this bound is worse than the best known network, prune the search. Any combination of the pruning techniques *slowed* the search down. At any state A in the subset lattice, there are $(n - |A|)!$ paths to it – pruning one path leaves many others paths to be tested. Computing SCORE(A) requires $O(2^{|A|})$ time, but prevents all other paths from being expanded.

Given the performance of RAND ORDERS it would be worthwhile to consider a more informed way of sampling the space of orderings. The MCMC chain over variable orderings of (Friedman & Koller, 2000) may work. Other directions for future work include finding the k-best networks and adapting this work to find Markov blankets.

Table 1: P-cache construction. n and k indicate the number of variables and indegree bound, respectively. Italicized entries are the P-caches used in table 2.

NAME	n	k	RECORDS	TIME (S)	SIZE (MB)
nursery	9	9	12960	10	0.2
parity	10	10	1×10^6	650	1.4
adult	15	15	10000	4049	0.6
letters	17	17	20000	13093	22.1
covtype	21	21	1000	23577	489.6
biosurv	22	22	5000	117724	1082.4
zoo	17	6	101	30	6.0
biosurv	22	5	5000	516	16.1
biosurv	22	6	5000	1731	45.4
mushroom	23	8	4000	34583	313.4
autos	26	4	205	99	8.5

Table 2: Comparison of structure learning algorithms. Run times do not include P-cache construction. The method(s) that reached the optimal score are in bold. Under hillclimb score A refers to the score after 120s, score B after 600s.

DATASET	EXACT		RAND ORDERS		OPT REINSERT		HILLCLIMB	
	SCORE	TIME	SCORE	TIME	SCORE	TIME	SCORE_A	SCORE_B
nursery	**−125717**	< 1	**−125717**	120 s	**−125717**	< 1 s	**−125717**	same
parity	**−6238741**	< 1 s	−6238743	120 s	**−6238741**	2 s	−6931538	same
adult	**−112563**	1 s	−112570	120 s	−112603	1 s	−112796	−112785
letters	**−196359**	53 s	−196834	120 s	−196475	3 s	−198638	−198209
zoo	**−587**	18 s	−593	120 s	−592	2 s	−649	−647
covtype	**−7603**	98 m	−7654	120 s	−7628	27 s	−7684	same
biosurv	**−30297**	210 m	−30321	120 s	−30303	115 s	−30435	−30397
mushroom	**−38108**	621 m	−38142	120 s	−38122	91 s	−41010	−40711
autos	**−3054**	807 m	−3159	120 s	−3107	2 s	−3159	same

6 Related Work

Most work on Bayes net structure learning is based on stochastic local search, usually hillclimbing (Buntine, 1991). There are several approaches to dealing with local minima: multi-link lookahead (Xiang et al., 1997), more complex search operators (Moore & Wong, 2003), guiding the search towards promising candidates (Friedman et al., 1999; Brown et al., 2004), constraining the (causal) search (Spirtes et al., 1993; Cheng et al., 1997; Margaritis & Thrun, 2000), and searching in a smaller space such as equivalence classes (Chickering, 2002; Kocka & Castelo, 2001) or orderings (Friedman & Koller, 2000).

The idea of using dynamic programming for exact Bayes net structure discovery was first proposed by Koivisto & Sood (KS) (Koivisto & Sood, 2004). Their paper extends the work of (Friedman & Koller, 2000; Friedman & Koller, 2003), which computes the Bayesian posterior probability of structural features, such as edges, as well as a single MAP model, by

MCMC search over orderings. KS showed that the equations of (Friedman & Koller, 2000; Friedman & Koller, 2003) can be solved as a recursive function over intermediate terms. This recursive function can be computed by dynamic programming, but requires precomputation of the intermediate terms, which is done through the Fast Möbius Transform.

OptOrd has a different class of functions to optimize, the decomposable scoring metrics of equation 1. We have shown that incrementally removing leaves yields a (different) recursive equation, which can be solved by dynamic programming without the introduction of intermediate terms. Our basic operator is finding the best set of parents from a set of potential parents, which we compute efficiently using P-caches. An advantage of our implementation, which could also be applied to KS, is the use of AD-trees to greatly reduce the cost of computing sufficient statistics from the data set.

The cost of dynamic programming for both algorithms is $O(n2^n)$ time, but with indegree constraints, KS is faster because many of the intermediate terms are zero – *i.e.*, stored, but not computed. Using P-caches can be very expensive if there are no indegree constraints and the BDeu bound is loose; but if there are no indegree constraints computing the intermediate terms in KS is just as hard. Both algorithms have a worst case $O(n2^n)$ memory requirement, but the advantage is ours since the worst case is only achieved when there is no bounding in the construction of the P-cache. This roughly corresponds to the implausible scenario where all parents sets, regardless of size, are all equally good for a given node. In our algorithm the memory requirement is $O(2^n)$ to store the subset lattice plus the size of the P-cache; in KS the cost is $\Omega(n2^n)$ to store the intermediate terms. KS supports some forms of edge constraints (layering), which can greatly reduce the search space. Layering makes direct comparison of their results to ours difficult.

7 Conclusions

The most common framework for structure discovery in Bayesian networks is to maximize a decomposable scoring function, like BDeu, over the space of acyclic digraphs. Existing work has been almost entirely devoted to heuristic methods, even for networks with less than 25 variables. This paper (i) describes the first *exact* solution to this framework with less than superexponential complexity in the number of variables, (ii) describes the branch-and-bound construction and search of P-caches, which allow one to select the optimal set of parents for any variable under decomposable scoring metrics, and (iii) evaluates the performance of this algorithm on benchmark data sets.

Acknowledgements

The authors would like to thank Greg Cooper and Weng-Keen Wong for their careful reading of earlier drafts and many stimulating discussions.

A Bayesian Dirichlet (BD) Scoring

The Bayesian Dirichlet (BD) scoring function (Cooper & Herskovits, 1992) scores a structure S against data D

$$\begin{aligned} P(S, D) &= P(S)P(D|S) \\ &= P(S)\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})}\prod_{k=1}^{r_i}\frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})} \end{aligned}$$

Where

- i – indexes the variables in the network, *i.e.*, $x_i \in V$.
- j – indexes the q_i distinct assignments to the parents of x_i.
- k – indexes the r_i values of x_i.
- α_{ij} – the prior weight on the j^{th} assignment to the parents of x_i
- α_{ijk} – the prior weight of x_i taking on its k^{th} value given the j^{th} assignment to its parents.
- N_{ij} – the number of records that match the j^{th} assignment of the parents of x_i
- N_{ijk} – the number of records that match the k^{th} value of x_i and the j^{th} assignment to the parents of x_i.
- $P(S)$ – prior distribution over structures.

For the BDeu (likelihood equivalent uniform Bayesian Dirichlet) score (Buntine, 1991) we assume that $P(S)$ is uniform, $\alpha_{ij} = 1/q_i$, and $\alpha_{ijk} = \alpha_{ij}/r_i$. Thus the BDeu score is

$$\mathcal{B}(S, D) = \prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})}\prod_{k=1}^{r_i}\frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})}$$

and the score for the j^{th} row in the conditional probability table for x_i is

$$\mathcal{B}_{ij}(S, D) = \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})}\prod_{k=1}^{r_i}\frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})}$$

B Proofs

The following are proofs related to the BDeu scoring metric. The notation follows that established in section 4.2 and appendix A.

Fact 1. *The Pochhammer symbol, a.k.a. the rising factorial*

$$(x)_n = \frac{\Gamma(x+n)}{\Gamma(x)}$$

is monotonic nondecreasing on $[0, \infty)$ *for any fixed* $n \geq 0$.

Fact 2. *(Dragomir et al., 2000, Theorem 4). For a given $\alpha > 0$ the mapping $(\alpha)_x$ on $[0, \infty)$ is supermultiplicative:*

$$(\alpha)_{x+y} \geq (\alpha)_x (\alpha)_y$$

Fact 3. *(Dragomir et al., 2000, Theorem 7) The Digamma function $\Psi(x)$ is monotonic nondecreasing on $(0, \infty)$.*

Lemma 1. *Given a pure virtual row $v_k = (0, \ldots, 0, N_{ijk}, 0, \ldots, 0)$ of length r_i,*

$$\mathcal{B}_{ij}^k(S, D) = \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + N_{ijk})} \frac{\Gamma(\alpha_{ijk} + N_{ijk})}{\Gamma(\alpha_{ijk})}$$

is monotonically nonincreasing in α_{ij}.

Proof. Trivially true if $N_{ijk} = 0$. If $N_{ijk} > 0$ observe that

$$\frac{\partial}{\partial N_{ijk}} \mathcal{B}_{ij}^k(S, D) = \Psi(\alpha_{ijk} + N_{ijk}) - \Psi(\alpha_{ij} + N_{ijk}) < 0$$

by noting that $\alpha_{ij} > \alpha_{ijk}$ and applying fact 3. □

If any $N_{ijk} = 0$ then the corresponding pure virtual row v_k has no direct contribution to the score,*i.e.*, $\mathcal{B}_{ij}^k(S, D) = 1$. However, these extra rows have an indirect effect since they reduce the value of α_{ij}. Lemma 1 shows that including these extra rows cannot decrease the score of the other non-zero pure virtual rows. This lemma allows us to not worry about whether $N_{ijk} = 0$ in the proof of theorem 1.

Lemma 2. *Given a row $(N_{ij1}, N_{ij2}, \ldots, N_{ijr_i})$ where $N_{ij} = \sum_k N_{ijk}$ the row score is maximized when for some k, $N_{ijk} = N_{ij}$.*

Proof. Reexpress $\mathcal{B}_{ij}(S, D)$ as follows, and then apply fact 2

$$\begin{aligned}
\mathcal{B}_{ij}(S, D) &= \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + N_{ij})} \prod_{k=1}^{r_i} (\alpha_{ijk})_{N_{ijk}} \\
&\leq \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + N_{ij})} (\alpha_{ijk})_{N_{ij1} + \ldots + N_{ijr_i}}
\end{aligned}$$

□

Lemma 3. *Given a row $v = (0, \ldots, 0, m_k, 0, \ldots, 0, N_{ijk}, 0, \ldots)$ and a row $v' = (0, \ldots, 0, m_k + t_k, 0, \ldots, 0, N_{ijk}, 0, \ldots)$ for $m_k, t_k \geq 0$ the row score of v is no smaller than the row score of v'.*

Proof. Let the score of rows v, v' be denoted $\mathcal{B}_v(S, D)$ and $\mathcal{B}_{v'}(S, D)$ respectively.

$$\begin{aligned}
\frac{\mathcal{B}_v(S, D)}{\mathcal{B}_{v'}(S, D)} &= \frac{\Gamma(\alpha_{ij} + N_{ijk} + m_k + t_k)}{\Gamma(\alpha_{ij} + N_{ijk} + m_k)} \frac{\Gamma(\alpha_{ijk} + m_k)}{\Gamma(\alpha_{ijk} + m_k + t_k)} \\
&= \frac{(\alpha_{ij} + N_{ijk} + m_k)_{t_k}}{(\alpha_{ijk} + m_k)_{t_k}} \geq 1
\end{aligned}$$

This proof does not require that the counts m_k and $m_k + t_k$ be in the same position within their rows. □

Theorem 1. *Consider a row $(N_{ij1}, \ldots, N_{ijr_i})$ and the virtual rows $\{v_1, \ldots, v_m\}$ formed by adding variable x^* to the parent set. If the virtual rows are pure then the BDeu score is maximized.*

Proof. There must be at least r virtual rows for them to be pure while respecting the marginal constraints. If there are more rows, then purity is achieved at the cost of extra parameters, so $m = r$. The only way to improve the score is to remove a row (*i.e.*, reduce the number of parameters). Rearranging the mass within each row cannot help since each row is already pure (lemma 2). Wolg let $N_{ij1} \leq N_{ij2} \leq \ldots \leq N_{ijr_i}$, so we will remove virtual row v_1. To respect the marginal constraints the mass in v_1 is redistributed among $\{v_2, \ldots, v_r\}$. This process is illustrated in figure 2(b). The mass moved from N_{ij1} into v_k is denoted m_k. Note that all m_k are integral and $N_{ij1} = \sum_{k=2}^{r_i} m_k$. If instead we removed row $v_i \neq v_1$ the redistribution would force some row(s) to absorb more mass than before. By lemma 3 this leads to a lower score for those rows, and thus a lower overall score.

Let the prior weights associated with the virtual rows be denoted α'_{ij} and α'_{ijk}. The score for the pure virtual rows is

$$\text{pure} = \prod_{k=1}^{r_i} \frac{\Gamma(\alpha'_{ij})}{\Gamma(\alpha'_{ij} + N_{ijk})} \frac{\Gamma(\alpha'_{ijk} + N_{ijk})}{\Gamma(\alpha'_{ijk})}$$

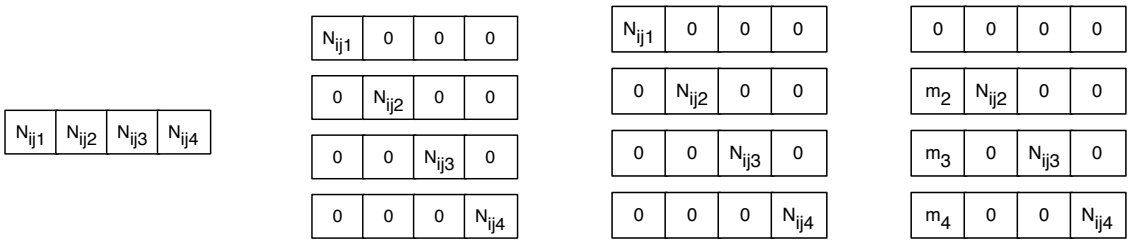
and the score for the impure virtual rows formed by removing v_1 is

$$\text{impure} = \prod_{k=1}^{r_i} \frac{\Gamma(\alpha'_{ij})}{\Gamma(\alpha'_{ij} + N_{ijk} + m_k)} \frac{\Gamma(\alpha'_{ijk} + N_{ijk})}{\Gamma(\alpha'_{ijk})} \frac{\Gamma(\alpha'_{ijk} + m_k)}{\Gamma(\alpha'_{ijk})}$$

and so the ratio of the scores is

$$\begin{aligned}
\frac{\text{pure}}{\text{impure}} &= \frac{\Gamma(\alpha'_{ij})}{\Gamma(\alpha'_{ij} + N_{ij1})} \frac{\Gamma(\alpha'_{ij}/r_i + N_{ij1})}{\Gamma(\alpha'_{ij}/r_i)} \prod_{k=2}^{r_i} \frac{\Gamma(\alpha'_{ij} + N_{ijk} + m_k)}{\Gamma(\alpha'_{ij} + N_{ijk})} \frac{\Gamma(\alpha'_{ij}/r_i)}{\Gamma(\alpha'_{ij}/r_i + m_k)} \\
&= \underbrace{\frac{(\alpha'_{ij}/r_i)_{N_{ij1}}}{(\alpha'_{ij})_{N_{ij1}}}}_{\leq 1} \prod_{k=2}^{r_i} \frac{(\alpha'_{ij} + N_{ijk})_{m_k}}{(\alpha'_{ij}/r_i)_{m_k}} \\
&\geq \left[\frac{(\alpha'_{ij}/r_i)_{N_{ij1}}}{(\alpha'_{ij})_{N_{ij1}}} \right]^{r_i - 1} \prod_{k=2}^{r_i} \frac{(\alpha'_{ij} + N_{ijk})_{m_k}}{(\alpha'_{ij}/r_i)_{m_k}} \\
&= \prod_{k=2}^{r_i} \underbrace{\frac{(\alpha'_{ij}/r_i)_{N_{ij1}}}{(\alpha'_{ij}/r_i)_{m_k}}}_{C_k} \underbrace{\frac{(\alpha'_{ij} + N_{ijk})_{m_k}}{(\alpha'_{ij})_{N_{ij1}}}}_{T_k}
\end{aligned}$$

Now $C_k \geq 1$ since $N_{ij1} \geq m_k, \forall k$. Next we observe that, since m_k and N_{ij1} are integral, we



(a) An example of a row of arity 4 (left) turned into pure virtual rows (right)

(b) Removing a row by redistributing its mass. $m_2 + m_3 + m_4 = N_{ij1}$

Figure 2: Operations involving virtual rows

can reexpress T_k as

$$T_k = \frac{\begin{array}{r} (\alpha'_{ij} + N_{ijr_i} + m_{r_i} - 1) \cdots (\alpha'_{ij} + N_{ijr_i}) \Big\} m_{r_i} \text{ terms} \\ \cdots \\ \times \quad (\alpha'_{ij} + N_{ij3} + m_3 - 1) \cdots (\alpha'_{ij} + N_{ij3}) \Big\} m_3 \text{ terms} \\ \times \quad (\alpha'_{ij} + N_{ij2} + m_2 - 1) \cdots (\alpha'_{ij} + N_{ij2}) \Big\} m_2 \text{ terms} \end{array}}{(\alpha'_{ij} + N_{ij1} - 1) \cdots (\alpha'_{ij}) \Big\} N_{ij1} \text{ terms}}$$

There are a total of N_{ij1} terms in the numerator and N_{ij1} terms in the denominator. Since $\forall k > 1, N_{ij1} \leq N_{ijk}$ each term in the numerator is larger than any term in the denominator. Thus $T_k \geq 1$. Since $\forall k, C_k \geq 1$ and $T_k \geq 1$ the overall ratio is greater than 1. It immediately follows that altering the pure virtual rows reduces the score. □

References

Blake, C., & Mertz, C. (1998). UCI repository of machine learning databases.

Brown, L. E., Tsamardinos, I., & Aliferis, C. (2004). A novel algorithm for scalable and accurate Bayesian network learning. *Proc. 11th World Congress on Medical Informatics*.

Buntine, W. (1991). Theory refinment on Bayesian networks. *UAI-7* (pp. 52–60).

Cheng, J., Bell, D., & Liu, W. (1997). An algorithm for Bayesian network construction from data. *Proc. 6th Intl. Workshop on AI & Stats, 1997*.

Chickering, D. M. (1996). Learning Bayesian networks is NP-complete. In *Learning from data*. Springer.

Chickering, D. M. (2002). Optimal structure identification with greedy search. *JMLR*, *3*, 507–554.

Chickering, D. M., Meek, C., & Heckerman, D. (2003). Large-sample learning of Bayesian networks is NP-hard. *UAI-9* (pp. 124–133). Morgan Kaufmann.

Cooper, G., & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, *9*.

Dragomir, S., Agarwal, R., & Barnett, N. (2000). Inequalities for Beta and Gamma functions via some classical and new integral inequalities. *J. Inequalities and Applications*, *5*.

Friedman, N., & Koller, D. (2000). Being Bayesian about network structure. *UAI-16*.

Friedman, N., & Koller, D. (2003). Being Bayesian about network structure: A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, *50*, 95–125.

Friedman, N., Nachman, I., & Peer, D. (1999). Learning Bayesian network structure from massive datasets: The "sparse candidate" algorithm. *UAI-13* (pp. 206–215).

Heckerman, D., Geiger, D., & Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, *20*, 197–243.

Kocka, T., & Castelo, R. (2001). Improved learning of Bayesian networks. *UAI-17* (pp. 269–276).

Koivisto, M., & Sood, K. (2004). Exact Bayesian structure discovery in Bayesian networks. *JMLR*, *5*, 549–573.

Margaritis, D., & Thrun, S. (2000). Bayesian network induction via local neighborhoods. *NIPS 12* (pp. 505–511).

Moore, A., & Wong, W.-K. (2003). Optimal Reinsertion: A new search operator for accelerated and more accurate Bayesian network structure learning. *ICML-20*.

Moore, A. W., & Lee, M. S. (1998). Cached sufficient statistics for efficient machine learning with large datasets. *JAIR*, *8*, 67–91.

Robinson, R. (1973). Counting labelled acyclic digraphs. In *New directions in the theory of graphs*, 239–273. Academic Press.

Schwartz, G. (1979). Estimating the dimensions of a model. *Ann. Stat.*, *6*, 461–464.

Spirtes, P., Glymour, C., & Scheines, R. (1993). *Causation, Prediction, and Search*. New York, N.Y., Springer-Verlag.

Witten, I. H., & Frank, E. (2000). *Data mining: Practical machine learning tools with java implementations*. Morgan Kaufmann.

Xiang, Y., Wong, S., & Cercone, N. (1997). A microscopic study of minimum entropy search in learning decomposable Markov networks. *Machine Learning*, *26*, 65–92.