Structure Learning of Bayesian Networks using Constraints

Cassio P. de Campos CASSIO@IDSIA.CH
Dalle Molle Institute for Artificial Intelligence (IDSIA), Galleria 2, Manno 6900, Switzerland

Zhi Zeng ZENGZ@RPI.EDU
Qiang Ji JIQ@RPI.EDU
Rensselaer Polytechnic Institute (RPI), 110 8th St., Troy NY 12180, USA

Abstract

This paper addresses exact learning of Bayesian network structure from data and expert's knowledge based on score functions that are decomposable. First, it describes useful properties that strongly reduce the time and memory costs of many known methods, such as hill-climbing, dynamic programming and sampling methods. Secondly, a branch and bound algorithm is presented that integrates parameter and structural constraints with data in a way to guarantee global optimality with respect to the score function. It is an any-time procedure because, if stopped, it provides the best current solution and an estimation about how far it is from the global solution. We show empirically the advantages of the properties and the constraints, and the applicability of the algorithm to large data sets (up to one hundred variables) that cannot be handled by other current techniques (limited to around 30 variables).

1. Introduction

A Bayesian network (BN) is a probabilistic graphical model that relies on a structured dependency among random variables to represent a joint probability distribution in a compact and efficient manner. It is composed by a directed acyclic graph (DAG) where nodes are associated to random variables and conditional probability distributions are defined for variables given their parents in the graph. Learning the graph (or structure) of a BN from data is one of the most challenging problems that arise in such models. Best exact known methods take exponential time on the number of variables and are applicable to small settings (less than 30 variables). Approximate procedures can handle somewhat larger networks, but may commonly get stuck in local maxima. Nevertheless, the quality of the structure plays a crucial role in the accuracy of the model. If the dependency among variables is not properly learned, the estimated distribution may be far from the *correct* one. In general terms, the problem is to find the best structure according to some score function that depends on the data (Heckerman et al., 1995). There are other approaches to learn a structure that are not based on scoring (for example taking some statistical similarity among variables), but we do not discuss them on this paper. The research on this topic is active (Chickering, 2002; Teyssier & Koller, 2005; Tsamardinos et al., 2006). Best exact ideas (where it is guaranteed to find the global best scoring structure) are based on dynamic programming (Koivisto et al., 2004; Singh & Moore, 2005; Koivisto, 2006; Silander & Myllymaki, 2006), and they spend time and memory proportional to $n \cdot 2^n$, where n is the number of variables. Such complexity forbids the use of those methods to a couple of tens of variables, mostly because of memory consumption.

In the first part of this paper, we present some properties of the problem that allow for a huge improvement on most cited methods. We work with the *Akaike Information Criterion* (AIC), and the *minimum description length* (MDL), which is equivalent to the *Bayesian Information Criterion* (BIC), but results extrapolate to the Bayesian Dirichlet (BD) Scoring (Cooper & Herskovits, 1992) and some derivatives. We show that the search space of possible structures can be reduced drastically without losing the global optimality guarantee and that the memory requirements are very small in many practical cases (we show empirically that only a few thousand scores are stored for a prob-

Appearing in *Proceedings of the 26^{th} International Conference on Machine Learning*, Montreal, Canada, 2009. Copyright 2009 by the author(s)/owner(s).

lem with 50 variables and one thousand instances).

As data sets with many variables cannot be efficiently handled (unless P=NP, as the problem is known to be NP-hard (Chickering et al., 2003)), a desired property of a method is to produce an *anytime* solution, that is, the procedure, if stopped at any moment, provides an approximate solution, while if run until it finishes, a global optimum solution is found. However, the most efficient exact methods are not *anytime*. We propose a new anytime exact algorithm using a branch-and-bound (B&B) approach with caches. Scores are computed during the initialization and a poll is built. Then we perform the search over the possible graphs iterating over arcs. Although iterating over orderings is probably faster, iterating over arcs allows us to work with constraints in a straightforward way. Because of the B&B properties, the algorithm can be stopped at anytime with a best current solution found so far and an upper bound to the global optimum, which gives a kind of certificate to the answer and allows the user to stop the computation when she believes the current solution is good enough. (Suzuki, 1996) has proposed a B&B method, but it is not a global exact algorithm, as the search is conducted after a node ordering is fixed. Our method does not rely on a predefined ordering, instead it finds a global optimum structure considering all possible orderings.

2. Bayesian networks

A BN represents a single joint probability density over a collection of random variables. It can be defined as a triple $(\mathcal{G}, \mathcal{X}, \mathcal{P})$, where $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$ is a DAG with $V_\mathcal{G}$ a collection of n nodes associated to random variables $\mathcal{X}$ (a node per variable), and $E_\mathcal{G}$ a collection of arcs; $\mathcal{P}$ is a collection of conditional probability densities $p(X_i|PA_i)$ where PA_i denotes the parents of X_i in the graph (PA_i may be empty), respecting the relations of $E_\mathcal{G}$. We assume throughout that variables are categorical. In a BN every variable is conditionally independent of its non-descendants given its parents (Markov condition). This structure induces a joint probability distribution by the expression $p(X_1, \dots, X_n) = \prod_i p(X_i|PA_i)$.

For convenience, we define some notation. Let $r_i \geq 2$ be the number of discrete categories of X_i, q_i the number of elements in Ω_{PA_i} (the number of configurations of the parent set, that is, $q_i = \prod_{X_t \in PA_i} r_t$) and θ be the entire vector of parameters such as $\theta_{ijk} = p(x_i^k | pa_i^j)$, where $i \in \{1, \dots, n\}$, $j \in \{1, ..., q_i\}$, $k \in \{1, ..., r_i\}$ (hence $x_i^k \in \Omega_{X_i}$ and $pa_i^j \in \Omega_{PA_i}$).

Given a complete data set $D = \{D_1, \dots, D_N\}$ of with N instances, with $D_t = \{x_{1,t}^{k_1}, \dots, x_{n,t}^{k_n}\}$ a instance of all variables, the goal of structure learning is to find the arcs $E_\mathcal{G}$ that maximizes an score function such as MDL or AIC.

$$\max_{\mathcal{G}} s_D(\mathcal{G}) = (\max_{\theta} L_D(\theta) - t \cdot W),$$

where θ represents all parameters of the model (and thus depends on the graph $\mathcal{G}$), $t = \sum_{i=1}^n (q_i \cdot (r_i - 1))$ is the number of free parameters and W depends on the criterion ($W = \frac{\log N}{2}$ in MDL and $W = 1$ in AIC), and L_D is the log-likelihood function:

$$L_D(\theta) = \log \prod_{i=1}^{n} \prod_{j=1}^{q_i} \prod_{k=1}^{r_i} \theta_{ijk}^{n_{ijk}}, \tag{1}$$

where n_{ijk} indicates how many elements of D contain both x_i^k and pa_i^j. This function can be written as $L_D(\theta) = \sum_{i=1}^n L_{D,i}(\theta_i)$, where $L_{D,i}(\theta_i) = \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} n_{ijk} \log \theta_{ijk}$. From now on, the subscript D is assumed implicit for simplicity.

An important property of such criteria is that they are decomposable, that is, can be applied to each node X_i separately: $\max_\mathcal{G} s(\mathcal{G}) = \max_\mathcal{G} \sum_{i=1}^n s_i(PA_i)$, where $s_i(PA_i) = L_i(PA_i) - t_i(PA_i) \cdot W$, with $L_i(PA_i) = \max_{\theta_i \in \Omega_{\times PA_i}} L_i(\theta_i)$, and $t_i(PA_i) = q_i \cdot (r_i - 1)$. Because of this property and to avoid computing such functions several times, we create a cache that contains $s_i(PA_i)$ for each X_i and each parent set PA_i. Note that this cache may have a exponential size on n, as there are 2^{n-1} subsets of $\{X_1, \dots, X_n\} \setminus \{X_i\}$ to be considered as possible parent set. This gives a total space and time of $O(n \cdot 2^n)$ to build the cache. Instead, we show that this number is much smaller.

Lemma 1 *Let X_i be a node of $\mathcal{G}'$, a DAG for a BN where $PA(X_i) = J'$. Suppose $J \subset J'$ is such that $s_i(J) > s_i(J')$. Then J' is not the parent set of X_i in the optimal DAG.*

Proof. As J' is the parent set of X_i in $\mathcal{G}'$, we can take a graph $\mathcal{G}$ that differs from $\mathcal{G}'$ only on $PA(X_i) = J$, which is also a DAG (as the removal of some arcs does not introduce cycles) and $s(\mathcal{G}) = \sum_{j \neq i} s_j(PA(X_j)) + s_i(J) > \sum_{j \neq i} s_j(PA(X_j)) + s_i(J') = s(\mathcal{G}')$. Hence all DAG $\mathcal{G}'$, with $PA(X_i) = J'$, have a subgraph $\mathcal{G}$ with a better score than $\mathcal{G}'$, and thus J' is not the optimal parent configuration for X_i. □

Lemma 1 is useful to discard elements from the cache of X_i, but it does not ensure anything about other supersets of J, that is, we still need to compute all the possible parent configurations and later verify which can be removed. Next theorems handle this issue.

Theorem 1 *Using MDL or AIC as score function and assuming $N \geq 4$, take $\mathcal{G}$ and $\mathcal{G}'$ DAGs such that $\mathcal{G}$ is a subgraph of $\mathcal{G}'$. If $\mathcal{G}$ is such that $\prod_{j \in PA_i} r_j \geq N$, for some X_i, and X_i has more parents in $\mathcal{G}'$ than in $\mathcal{G}$, then $\mathcal{G}'$ is not an optimal structure.*

Proof.[1] Take a DAG $\mathcal{G}$ such that $J = PA_i$ for a node X_i, and take $\mathcal{G}'$ equal to $\mathcal{G}$ except that it contains an extra node in $J^{new} = PA_i$, that is, in $\mathcal{G}'$ we have $J^{new} = J \cup \{X_e\}$. Note that the difference in the scores of the two graphs are restricted to $s_i(\cdot)$. In the graph $\mathcal{G}'$, $L_i(J^{new})$ will certainly not decrease and $t_i(J^{new})$ will increase, both with respect to the values of $\mathcal{G}$. The difference in the scores will be $s_i(J^{new}) - s_i(J)$, which equals to

$$\begin{aligned} L_i(J^{new}) - t_i(J^{new}) - (L_i(J) - t_i(J)) \leq \\ -\sum_{j=1}^{q_i}\sum_{i=1}^{r_i} n_{ijk} \log \theta_{ijk} - t_i(J^{new}) + t_i(J) \leq \\ \sum_{j=1}^{q_i} n_{ij} \left(-\sum_{i=1}^{r_i} \frac{n_{ijk}}{n_{ij}} \log \frac{n_{ijk}}{n_{ij}} \right) - t_i(J^{new}) + t_i(J) \leq \\ \sum_{j=1}^{q_i} n_{ij} H(\theta_{ij}) - t_i(J^{new}) + t_i(J) \leq \\ \sum_{j=1}^{q_i} n_{ij} \log r_i - q_i \cdot (r_e - 1) \cdot (r_i - 1) \cdot W \end{aligned}$$

The first step uses the fact that $L_i(J^{new})$ is negative, the second step uses that fact that $\hat{\theta}_{ijk} = \frac{n_{ijk}}{n_{ij}}$, with $n_{ij} = \sum_{i=1}^{r_i} n_{ijk}$, is the value that maximizes $L_i(\cdot)$, and the last step uses the fact that the entropy of a discrete distribution is less than the log of its number of categories. Finally, the last equation is negative if $q_i \cdot (r_e - 1) \cdot (r_i - 1) \cdot W \geq N \log r_i$. Because $r_i \geq 2 \Rightarrow r_i - 1 \geq \log r_i$, and $N \geq 4 \Rightarrow \frac{\log N}{2} \geq 1$ (the W of the MDL case), we have that $q_i = \prod_{j \in J} r_j \geq N$ ensures that $s_i(J^{new}) < s_i(J)$, which implies that the graph $\mathcal{G}'$ cannot be optimal ($\mathcal{G}$ is a proper subgraph and has a better score). □

Corollary 1 *In the optimal structure $\mathcal{G}$, each node has at most $O(\log_2 N)$ parents.*

Proof. It follows directly from Theorem 1 and the fact that $r_i \geq 2$, for all X_i. □

Theorem 1 and Corollary 1 ensures that the cache stores at most $O(\binom{n-1}{\log N})$ elements for each variable (all combinations up to $\log N$ parents). Although it does not help us to improve the theoretical size bound, Lemma 2 gives us even less elements.

Lemma 2 *Let X_i be a node with $J \subset J'$ two possible parent sets such that $t_i(J') + s_i(J) > 0$. Then J' and all supersets $J'' \supset J'$ are not optimal parent configurations for X_i.*

Proof. Because $L_i(\cdot)$ is a negative function, $t_i(J') + s_i(J) > 0 \Rightarrow -t_i(J') - s_i(J) < 0 \Rightarrow (L_i(J') - t_i(J')) - s_i(J) < 0 \Rightarrow s_i(J') < s_i(J)$. Using Lemma 1, we have that J' is not the optimal parent set for X_i. The result also follows for any $J'' \supset J$, as we know that $t_i(J'') > t_i(J')$. □

The idea is to check the validity of Lemma 2 every time the score of a parent set J' of X_i is about to be computed, discarding J' and all supersets whenever possible. This result allows us to stop computing scores for J' and all its supersets. Lemma 1 is even stronger (w.r.t. the scores), but regards a comparison between exactly two parent configuration. Nevertheless, Lemma 1 can be applied to the final cache to remove all certainly useless parent configurations. As we see in Section 5, the practical size of the cache after these properties is small even for large networks. Lemma 1 is also valid for other decomposable functions, including BD and derivations (e.g. BDe, BDeu), so the benefits shall apply to those scores too. The other theorems may need some assumptions about the initial N and the choice of priors. Further discussion is left for future work because of lack of space.

3. Constraints

An additional way to reduce the space of possible DAGs is to consider some constraints provided by experts. We work with two main types of constraints: constraints on parameters that define rules about the probability values inside the local distributions of the network, and structural constraints that specify where arcs may or may not be included.

3.1. Parameter Constraints

We work with a general definition of parameter constraints, where any convex constraint is allowed. If θ_{i,PA_i} is the parameter vector of the node X_i with parent set PA_i, then a *convex constraint* is defined as $h(\theta_{i,PA_i}) \leq 0$, where $h : \Omega_{\theta_{i,PA_i}} \to \mathcal{R}$ is a convex function over θ_{i,PA_i}.

This definition includes many well known constraints, for example from Qualitative Probabilistic Networks (QPN) (Wellman, 1990): *qualitative influences* define some knowledge about the state of a variable given

[1] Another proof appears in (Bouckaert, 1994), but it leads directly to the conclusion of Corollary 1. The intermediate result is algorithmically important.

the state of another, which roughly means that observing a greater state for a parent X_a of a variable X_b makes more likely to have greater states in X_b (for any parent configuration except for X_a). For example, $\theta_{bj_2 2} \geq \theta_{bj_1 2}$, where $j_k \doteq \{x_a^k, pa_b^{j_*}\}$ and j_* is an index ranging over all parent configurations except for X_a. In this case, observing x_a^2 makes more likely to have x_b^2. A *negative influence* is obtained by replacing the inequality operator $\geq$ by $\leq$, and a *zero influence* is obtained by changing inequality to an equality. Other constraints such as *synergies* (Wellman, 1990) are also linear and local to a single node.

Although the parameter constraints we allow are very general, we have the following restriction about them: if a constraint is specified for a node i and a set of parents J, then the actual parent set PA_i will be a superset of J. Furthermore, we have a peculiar interpretation for each constraint C as follows: if $J \subset PA_i$ (proper subset), then the parameter constraint must hold for all configurations of the parents of X_i that do not belong to J. For example, suppose X_1 has X_2 and X_3 as parents (all of them binary), and the following constraint h was defined on X_1: $p(x_1^2|x_2^2x_3^2) + 2 \cdot p(x_1^2|x_2^2x_3^1) \leq 1$. If a new node X_4 is included as parent of X_1, the constraint h becomes the two following constraints:

$$\begin{aligned} p(x_1^2|x_2^2x_3^2x_4^1) + 2 \cdot p(x_1^2|x_2^2x_3^1x_4^1) &\leq 1, \\ p(x_1^2|x_2^2x_3^2x_4^2) + 2 \cdot p(x_1^2|x_2^2x_3^1x_4^2) &\leq 1, \end{aligned}$$

that is, h holds for each state of X_4. For example if another parent X_5 is included, then four constraints would be enforced with all possible combinations. This interpretation for constraints is in line with the definition of qualitative constraints of QPNs, and most importantly, it allows us to treat the constraints in a principled way for each set of parents. It means that the constraint must hold for all configurations of parents not involved in the constraint, which can be also interpreted as *other parents are not relevant and the constraint is valid for each one of their configurations.*

3.2. Structural constraints

Besides probabilistic constraints, we also allow for structural constraints on the possible graphs. This constraints help reduce the search space and are available in many cases. We work with the following rules:

- $indegree(X_j, k, op)$, where $op \in \{\text{lt}, \text{eq}\}$ and k an integer, means that the node X_j must have *less than* (when $op = \text{lt}$) or *equal to* (when $op = \text{eq}$) k parents.
- $arc(X_i, X_j)$ indicates that the node X_i must be a parent of X_j.
- Operators *or* ($\vee$) and *not* ($\neg$) are used to form the rules. The *and* operator is not explicitly used as we assume that each constraint is in disjunctive normal form.

For example, the constraints $\forall_{i \neq c, j \neq c}\ \neg arc(X_i, X_j)$ and $indegree(X_c, 0, \text{eq})$ impose that only arcs from node X_c to the others are possible, and that X_c is a root node, that is, a Naive Bayes structure will be learned. Only part of the variables are included, that is, the procedure acts as a feature selection procedure by letting some variables unlinked. Note that the symbol $\forall$ employed is not part of the language but is used for easy of expose (in fact it is necessary to write down every constraint defined by such construction). On the other hand, the constraints $\forall_{j \neq c}\ indegree(X_j, 3, \text{lt})$, $indegree(X_c, 0, \text{eq})$, and $\forall_{j \neq c}\ indegree(X_j, 0, \text{eq}) \vee arc(X_c, X_j)$ ensure that all nodes have X_c as parent, or no parent at all. Besides X_c, each node may have at most one other parent. X_c is a root node. This learns the structure of a Tree-augmented Naive (TAN) classifier, also performing a kind of feature selection (some variables may end up unlinked). In fact, it learns a forest of trees, as we have not imposed that all variables must be linked.

3.3. Dealing with constraints

First note that all constraints in previous examples can be imposed during the construction of the cache, because they involve just a single node each. In essence, parent sets of a node X_i that do violate some constraint are not stored in the cache, and this can be checked during cache construction. On the other hand, constraints such as $arc(X_1, X_2) \vee arc(X_2, X_3)$ cannot be imposed in such as phase, as they impose a non-local condition (the arcs go to distinct variables, namely X_2 and X_3), as the cache is essentially a local procedure with respect to each variable. Such constraints that involve distinct nodes can be verified during the B&B phase. More details on how the structural constraints are tackled are presented in the algorithm description.

Regarding parameter constraints, we compute the scores using a constrained optimization problem, i.e. maximize the score function subject to simplex equality constraints and all parameter constraints defined by the user.

$$\begin{aligned} &\max_{\theta_i} L_i(\theta_i) - t_i(PA_i) \\ &\qquad \text{subject to} \quad \forall_{j=1 \ldots q_i} \quad g_{ij}(\theta_{ij}) = 0, \qquad (2) \\ &\qquad\qquad\qquad\quad\ \forall_{z=1 \ldots m_{hi}} \quad h_{iz}(\theta_i) \leq 0, \end{aligned}$$

where $g_{ij}(\theta) = -1 + \sum_{k=1}^{r_i} \theta_{ijk}$ imposes that distributions defined for each variable given a parent configuration sums one over all variable states, and the m_{hi} convex constraints h_{iz} defined the space of feasible parameters for the node X_i. This is possible because: (1) we have assumed that a constraint over $p(x_i^k | x_{i_1}^{k_1}, \ldots, x_{i_t}^{k_t})$ forces $X_{i_1}, \ldots, X_{i_t} \subseteq PA_i$, that is, when a parameter constraint is imposed, the parent set of the node must contain at least the variables involved in the constraint; (2) the optimization is computed for every possible parent set, that is, PA_i is known in the moment to write down the optimization problem, which is solved for each X_i and each set PA_i. We use the optimization package of (Birgin et al., 2000).

Theorem 2 *Using MDL or AIC as score function and assuming $N \geq 4$, take $\mathcal{G}$ and $\mathcal{G}'$ as DAGs such that $\mathcal{G}$ is a subgraph of $\mathcal{G}'$. Suppose that both $\mathcal{G}$ and $\mathcal{G}'$ respect the same set of parameter and structural constraints. If $\mathcal{G}$ is such that $\prod_{j \in PA_i} r_j \geq N$, for some X_i, and X_i has more parents in $\mathcal{G}'$ than in $\mathcal{G}$, then $\mathcal{G}'$ is not an optimal structure.*

Proof. Just note that all derivations in Theorem 1 are also valid in the case of constraints. The only difference that deserves a comment is $\hat{\theta}_{ijk} = \frac{n_{ijk}}{n_{ij}}$, which may be an unfeasible point for the optimization (2), because the latter contains parameter constraints that might reduce the parameter space (besides the normal constraints of the maximum log-likelihood problem). As $\hat{\theta}_{ijk}$ is just used as an upper value for the log-likelihood function, and the constrained version can just obtain smaller objective values than the unconstrained version, we conclude that $\frac{n_{ijk}}{n_{ij}}$ is an upper bound also for the constrained case. Thus, the derivation of Theorem 1 is valid even with constraints. □

Corollary 1 and Lemmas 1 and 2 are also valid in this setting. The proof of Corollary 1 is straightforward, as it only depends on Theorem 1, while for Lemmas 1 and 2 we need just to ensure that all the parent configurations that are discussed there respect the constraints.

4. Constrained B&B algorithm

In this section we describe the B&B algorithm used to find the best structure of the BN and comment on its complexity, correctness, and some extensions and particular cases.The notation (and initialization of the algorithm) is as follows: $C : (X_i, PA_i) \to \mathcal{R}$ is the cache with the scores for all the variables and their possible parent configurations (using Theorem 1 and Lemmas 1 and 2 to have a reduced size); $\mathcal{G}$ is the graph created taking the best parent configuration for each node (note that $\mathcal{G}$ is not necessarily a DAG), and s is the score of $\mathcal{G}$; $\mathcal{H}$ is an initially empty graph containing, for each possible arc between nodes, a mark stating that the arc must be present, or is prohibited, or is free (may be present or not); Q is a priority queue of triples $(\mathcal{G}, \mathcal{H}, s)$, ordered by s (initially it contains a single triple with $\mathcal{G}$, $\mathcal{H}$ and s just mentioned; and finally $(\mathcal{G}_{best}, s_{best})$ is the best DAG and score found so far (s_{best} is initialized with $-\infty$).

While Q is not empty, do

1. Remove the peek $(\mathcal{G}_{cur}, \mathcal{H}_{cur}, s_{cur})$ of Q. If $s \leq s_{best}$, then start the loop again. If $\mathcal{G}_{cur}$ is a DAG and satisfies all structural constraints, update $(\mathcal{G}_{best}, s_{best})$ with $(\mathcal{G}_{cur}, s_{cur})$ and start the loop again.

2. Take $v = (X_{a_1} \to X_{a_2} \to \ldots \to X_{a_{q+1}})$, with $a_1 = a_{q+1}$, is a directed cycle of $\mathcal{G}_{cur}$.

3. For $y = 1, \ldots, q$, do
   - Mark on $\mathcal{H}_{cur}$ that the arc $X_{a_y} \to X_{a_{y+1}}$ is prohibited.
   - Recompute $(\mathcal{G}, s)$ from $(\mathcal{G}_{cur}, s_{cur})$ such that the parents of $X_{a_{y+1}}$ in $\mathcal{G}$ cope with this restriction and with $\mathcal{H}_{cur}$. Furthermore, the subgraph of $\mathcal{G}$ formed by arcs that are demanded by $\mathcal{H}_{cur}$ (those that have a mark *must exist*) must cope with the structural constraints (it might be impossible to get such graph. In such case the loop continues). Use the values in the cache $C(X_{a_{y+1}}, PA_{a_{y+1}})$ to not have to recompute the scores.
   - Include the triple $(\mathcal{G}, \mathcal{H}_{cur}, s)$ into Q.
   - Mark on $\mathcal{H}_{cur}$ that the arc $X_{a_y} \to X_{a_{y+1}}$ must be present and that the sibling arc $X_{a_{y+1}} \to X_{a_y}$ is prohibited, and continue.

The algorithm uses a B&B search where each case to be solved is a relaxation of a DAG, that is, they may contain cycles. At each step, a graph is picked up from a priority queue, and it is verified if it is a DAG. In such case, it is a feasible structure for the network and we compare its score against the best score so far (which is updated if needed). Otherwise, there must be a directed cycle in the graph, which is then broken into subcases by forcing some arcs to be absent/present. Each subcase is put in the queue to be processed. The procedure stops when the queue is empty. Note that every time we break a cycle, the subcases that are created are independent, that is, the sets of graphs that respect $\mathcal{H}$ for each subcase are disjoint. We obtain this fact by properly breaking the cycles: when $v = (X_{a_1} \to X_{a_2} \to \ldots \to X_{a_{q+1}})$ is detected, we create q

subcases such that the first does not contain $X_{a_1} \to X_{a_2}$ (but may contain the other arcs of that cycle), the second case certainly contains $X_{a_1} \to X_{a_2}$, but $X_{a_2} \to X_{a_3}$ is prohibited (so they are disjoint because of the difference in the presence of the first arc), and so on such that the y-th case certainly contains $X_{a_{y'}} \to X_{a_{y'+1}}$ for all $y' < y$ and prohibits $X_{a_y} \to X_{a_{y+1}}$. This idea ensures that we never process the same graph twice. So the algorithm runs at most $\prod_i |C(X_i)|$ steps, where $|C(X_i)|$ is the size of the cache for X_i.

B&B can be stopped at any time and the current best solution as well as an upper bound for the global best score are available. This stopping criterion might be based on the number of steps, time and/or memory consumption. Moreover, the algorithm can be easily parallelized. We can split the content of the priority queue into many different tasks. No shared memory needs to exist among tasks if each one has its own version of the cache. The only data structure that needs consideration is the queue, which from time to time must be balanced between tasks. With a message-passing idea that avoids using process locks, the gain of parallelization is linear in the number of tasks. As far as we know, best known exact methods are not easily parallelized, do not deal with constraints, and do not provide lower and upper estimates of the best structure if stopped early. If run until it ends, the proposed method gives a global optimum solution for the structure learning problem.

Some particular cases of the algorithm are worth mentioning. If we fix an ordering for the variables such that all the arcs must link a node towards another non-precedent in the ordering (this is a common idea in many approximate methods), the proposed algorithm does not perform any branch, as the ordering implies acyclicity, and so the initial solution is already the best. The performance would be proportional to the time to create the cache. On the other hand, bounding the maximum number of parents of a node is relevant only for hardest inputs, as it would imply a bound on the cache size, which is already empirically small.

5. Experiments

We perform experiments to show the benefits of the reduced cache and search space and the gains of constraints.[2] First, we use data sets available at the UCI repository (Asuncion & Newman, 2007). Lines with missing data are removed and continuous variables are discretized over the mean into binary variables. The data sets are: *adult* (15 variables and 30162 instances), *car* (7 variables and 1728 instances) letter (17 variables and 20000 instances), lung (57 variables and 27 instances), mushroom (23 variables and 1868 instances), nursery (9 variables and 12960 instances), Wisconsin Diagnostic Breast Cancer or *wdbc* (31 variables and 569 instances), zoo (17 variables and 101 instances). No constraints are employed in this phase as we intend to show the benefits of the properties earlier discussed.

Table 1 presents the cache construction results, applying Theorem 1 and Lemmas 1 and 2. Its columns show the data set name, the number of steps the procedure spends to build the cache (a step equals to a call to the score function for a single variable and a parent configuration), the time in seconds, the size of the generated cache (number of scores stored, the spent memory is actually $O(n)$ times that number), and finally the size of the cache if all scores were computed. Note that the reduction is huge. Although in the next we are going to discuss three distinct algorithms, the benefits shown by the application of these results imply in performance gain for many other algorithms of the literature to learn BN structures. It is also possible to analyze the search space reduction implied by these results on columns 2 and 3 of Table 2.

Table 1. Cache sizes (number of stored scores) and time (in seconds) to build it for many networks and data sizes. Steps represent the number of local (single node given a parent set) score evaluations.

name	steps	time(s)	size	$n2^n$
adult	30058	182.09	672	$2^{17.9}$
car	335	0.09	24	$2^{8.8}$
letter	534230	2321.46	41562	$2^{20.1}$
lung	43592	1.33	3753	$2^{61.8}$
mushroom	140694	72.13	8217	$2^{26.5}$
nursery	1905	3.94	49	$2^{11.2}$
wdbc	1692158	351.04	7482	2^{35}
zoo	9118	0.31	1875	$2^{20.1}$

In Table 2, we show results of three distinct algorithms: the B&B described in Section 4, the dynamic programming (DP) algorithm of (Silander & Myllymaki, 2006), and an algorithm that picks variable orderings randomly and then find the best structure where all arcs link a node towards another that is not precedent in the ordering. This last algorithm (named OS) is similar to K2 algorithm with random orderings, but it is always better because a global optimum is found for each ordering.[3] The scores obtained by each algorithm (in percentage against the value obtained by B&B) and

[2] The software is available online through the address *http://www.ecse.rpi.edu/∼cvrl/structlearning.html*

[3] We have run a hill-climbing approach (which is also benefited by ideas presented in this paper), but its accuracy was worse than OS. We omit it because of lack of space.

Table 2. Comparison of MDL scores amongst B&B and dynamic programming (both running at most 10 million steps), and ordering sampling (one thousand times). *Fail* means it could not solve the problem within the maximum number of steps. The scores in percentage are w.r.t. the score of the B&B procedure.

	search	reduced	B&B			DP		OS	
network	space	space	score	gap	time(s)	score	time(s)	score	time(s)
adult	2^{210}	2^{71}	-286902.8	5.5%	150.3	0.0%	0.77	0.1%	0.17
car	2^{42}	2^{10}	-13100.5	0.0%	0.01	0.0%	0.01	0.0%	0.01
letter	2^{272}	2^{188}	-173716.2	8.1%	574.1	-0.6%	22.8	1.0%	0.75
lung	2^{3192}	2^{330}	-1146.9	2.5%	907.1	*Fail*	*Fail*	1.0%	0.13
mushroom	2^{506}	2^{180}	-12834.9	15.3%	239.8	*Fail*	*Fail*	1.0%	0.12
nursery	2^{72}	2^{17}	-126283.2	0.0%	0.04	0.0%	0.04	0.0%	0.04
wdbc	2^{930}	2^{216}	-3053.1	13.6%	333.5	*Fail*	*Fail*	0.8%	0.13
zoo	2^{272}	2^{111}	-773.4	0.0%	5.2	0.0%	3.5	1.0%	0.03

the corresponding time spent are presented (excluding the cache construction). A limit of ten million steps is given to each method (steps are considered as the number of queries to the cache). It is also presented the reduced space where B&B performs its search, as well as the maximum gap of the solution. This gap is obtained by the relaxed version of the problem. So we can guarantee that the global optimal solution is within this gap (even though the solution found by the B&B can already be the best, as shown in the first line of the table). With the reduced cache presented here, finding the best structure for a given ordering is very fast, so it is possible to run OS for millions of orderings in short time.

Table 3. B&B procedure learning TANs. Time (in seconds) to find the global optimum, cache size (number of stored scores) and (reduced) space for B&B search.

network	time(s)	cache size	space
adult	0.26	114	2^{39}
car	0.01	14	$2^{6.2}$
letter	0.32	233	2^{61}
lung	0.26	136	2^{51}
mushroom	0.71	398	2^{88}
nursery	0.06	26	2^{12}
wdbc	361.64	361	2^{99}

Some additional comments are worth. DP can solve the *mushroom* set in less than 10 minutes if we drop the step limit. The expectation for *wdbc* is around four days. Hence, we cannot expect to obtain an answer in larger cases, such as *lung*. It is clear that, in worst case, the number of steps of DP is smaller than that of B&B. Nevertheless, B&B eventually bounds some regions without processing them, provides an upper bound at each iteration, and does not suffer from memory exhaustion as DP. This makes the method applicable even to very large settings. Still, DP seems a good choice for small n. When n is large (say more than 35), DP will not finish in reasonable time, and hence will not provide any solution, while B&B still gives an approximation and a bound to the global optimum. About OS, if we sample one million times instead of one thousand as described before, its results improve and the global optimum is found also for *adult* and *mushroom* sets. Still, OS provides no guarantee or estimation about how far is the global optimum (here we know it has achieved the optimum because of the exact methods). Finally, both DP and OS are benefited by the smaller cache.

Table 3 shows the results when we employ constraints to force the final network to be a Tree-augmented Naive Bayes (*zoo* was run, but it is not included because the unconstrained learned network was already TAN). Here the class is isolated in the data set and constraints are included as described in Section 3.2. Note that the cache size, the search space and consequently the time to solve the problems have all decreased. Finally, Table 4 has results for random data sets with predefined number of nodes and instances. A randomly created BN with at most $3n$ arcs is used to sample the data. Because of that, we are able to generate random parameter and structural constraints that are certainly valid for this *true* BN (approximately $n/2$ constraints for each case). The table contains the total time to run the problem and the size of the cache, together with the percentage of gain when using constraints. Note that the code was run in parallel with a number of tasks equals to n, otherwise an increase by a factor of n must be applied to the results in the table. We can see that the gain is recurrent in all cases (the constrained version has also less gap in all cases, although such number is not shown).

6. Conclusions

This paper describes a novel algorithm for learning BN structure from data and expert's knowledge. It integrates structural and parameter constraints with

Table 4. Results on random data sets generated from random networks. Time to solve (10 million steps) and size of the cache are presented for the *normal* unconstrained case and the percentage of gain when using constraints.

nodes(n)/	unconstrained			constrained gain	
instances	gap	time(s)	cache	time	cache
30/100	0%	0.06	125	67%	11.6%
30/500	0%	2.7	143	47.5%	26.5%
50/100	0%	0.26	310	31.4%	16.1%
50/500	0%	20.66	231	57.2%	29.8%
70/100	0%	4.58	1205	36.9%	18.8%
70/500	1.1%	356.9	666	38.4%	21.9%
100/100	0.5%	9.05	2201	47.5%	23.5%
100/500	1.4%	1370.4	726	50.2%	33.0%

data through a B&B procedure that guarantees global optimality with respect a decomposable score function. It is an any-time procedure in the sense that, if stopped early, it provides the best current solution found so far and a maximum error of such solution.

We also describe properties of the structure learning problem based on scoring DAGs that enable the B&B procedure presented here as well as other methods to work over a reduced search space and memory. Such properties allow the construction of a cache with all possible local scores of nodes and their parents without large memory consumption.

Because of the properties and the characteristics of the B&B method, even without constraints it is more efficient than state-of-the-art exact methods for large domains. We show through experiments with randomly generated data and public data sets that problems with up to 70 nodes can be exactly processed in reasonable time, and problems with 100 nodes are handled within a small worst case error. These results surpass by far current methods, which treat less than 35 variables. Described ideas may also help to improve other approximate methods and may have interesting practical applications, which we will pursue in future work.

Acknowledgments

This work is supported in part by the grant W911NF-06-1-0331 from the U.S. Army Research Office.

References

Asuncion, A., & Newman, D. (2007). UCI machine learning repository.

Birgin, E. G., Martínez, J. M., & Raydan, M. (2000). Nonmonotone spectral projected gradient methods on convex sets. *SIAM J. on Optimization*, *10*, 1196–1211.

Bouckaert, R. (1994). Properties of bayesian belief network learning algorithms. *Conf. on Uncertainty in Artificial Intelligence* (pp. 102–109). Morgan Kaufmann.

Chickering, D., Meek, C., & Heckerman, D. (2003). Large-sample learning of bayesian networks is np-hard. *Conf. on Uncertainty in Artificial Intelligence* (pp. 124–13). Morgan Kaufmann.

Chickering, D. M. (2002). Optimal structure identification with greedy search. *J. of Machine Learning Research*, *3*, 507–554.

Cooper, G., & Herskovits, E. (1992). A bayesian method for the induction of probabilistic networks from data. *Mach. Learning*, *9*, 309–347.

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

Koivisto, M. (2006). Advances in exact bayesian structure discovery in bayesian networks. *Conf. on Uncertainty in Artificial Intelligence.* AUAI Press.

Koivisto, M., Sood, K., & Chickering, M. (2004). Exact bayesian structure discovery in bayesian networks. *J. of Machine Learning Research*, *5*, 2004.

Silander, T., & Myllymaki, P. (2006). A simple approach for finding the globally optimal bayesian network structure. *Conf. on Uncertainty in Artificial Intelligence.* AUAI Press.

Singh, A. P., & Moore, A. W. (2005). *Finding optimal bayesian networks by dynamic programming* (Technical Report). Carnegie Mellon University. CMU-CALD-05-106.

Suzuki, J. (1996). Learning bayesian belief networks based on the minimum description length principle: An efficient algorithm using the B&B technique. *Int. Conf. on Machine Learning* (pp. 462–470).

Teyssier, M., & Koller, D. (2005). Ordering-based search: A simple and effective algorithm for learning bayesian networks. *Conf. on Uncertainty in Artificial Intelligence.*

Tsamardinos, I., Brown, L. E., & Aliferis, C. (2006). The max-min hill-climbing bayesian network structure learning algorithm. *Mach. Learning*, *65*, 31–78.

Wellman, M. P. (1990). Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence*, *44*, 257–303.