Abstract

Bayesian networks are widely used graphical models which represent uncertainty relations between the random variables in a domain compactly and intuitively. The first step of applying Bayesian networks to real-word problems typically requires building the structure of the networks. Among others, score-based exact structure learning has become an active research topic in recent years. In this context, a scoring function is used to measure the goodness of the data fitting a structure. The goal is to find the structure which optimizes the scoring function, and it has been shown a NP hard problem. This paper presents a comprehensive survey of the score-based Bayesian networks structure learning. We begin with the basic concepts such as Bayesian networks, structure learning. We then review the recent research works in Bayesian networks structure learning, and analyze their characteristics.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>2</td>
</tr>
<tr>
<td>1.1 Notation and Terminology</td>
<td>3</td>
</tr>
<tr>
<td>2 Bayesian Network Structure Learning</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Scoring Functions</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Local Search Strategies</td>
<td>7</td>
</tr>
<tr>
<td>2.2.1 Hill Climbing</td>
<td>8</td>
</tr>
<tr>
<td>2.2.2 Max-Min Hill Climbing</td>
<td>8</td>
</tr>
<tr>
<td>2.2.3 Stochastic Search</td>
<td>9</td>
</tr>
<tr>
<td>2.3 Optimal Search Strategies</td>
<td>9</td>
</tr>
<tr>
<td>2.3.1 Branch-and-Bound</td>
<td>10</td>
</tr>
<tr>
<td>2.3.2 Dynamic Programming</td>
<td>10</td>
</tr>
<tr>
<td>2.3.3 Integer Linear Programming</td>
<td>11</td>
</tr>
<tr>
<td>2.3.4 Heuristic Search</td>
<td>11</td>
</tr>
<tr>
<td>3 Exact Structure Learning</td>
<td>13</td>
</tr>
<tr>
<td>3.1 Integer Linear Programming</td>
<td>13</td>
</tr>
<tr>
<td>3.2 Heuristic Search</td>
<td>14</td>
</tr>
<tr>
<td>3.2.1 Finding the Shortest Path: heuristic search algorithms</td>
<td>16</td>
</tr>
<tr>
<td>3.2.2 Heuristics</td>
<td>17</td>
</tr>
<tr>
<td>3.3 Integer Linear Programming VS Heuristic Search</td>
<td>19</td>
</tr>
<tr>
<td>4 Conclusion and Future Work</td>
<td>21</td>
</tr>
</tbody>
</table>
1 Introduction

Bayesian networks, also known as belief networks, are widely-used graphical models that represent uncertain relations between the random variables in a uncertain domain compactly and intuitively [36]. In particular, each node in the graph represents a random variables, while edges between nodes represent probabilistic dependencies among the corresponding random variables. These conditional dependencies are often estimated by using known statistical and computaitonal methods. Hence, Bayesian Networks combine principles from graph theory, probability theory, computer science and statistics.

A Bayesian network is a directed acyclic graph (DAG) in which nodes represent random variables. The structure of a DAG is defined by two sets, the set of nodes and the set of directed edges. Each node represents one random variable. An edge from node $X_i$ to node $X_j$ represents a statistical dependency between the corresponding variables. Thus, the arrow indicates that a value taken by $X_j$ depends on the value taken by $X_i$, or roughly speaking that $X_i$ influences $X_j$. $X_j$ is then referred to as a parent of $X_j$, and, similarly, $X_j$ is then referred to as the child of $X_i$. The structure of DAG guarantees that there is no cycle in the Bayesian networks; in other words, there is no node that can be its own ancestor or its own descendent. Such a condition is of vital importance to the factorization of the joint probability of a collection of nodes.

In addition to the DAG structure, which often considered as the qualitative part of the Bayesian Networks, one needs to specify the quantitative parameters of Bayesian networks. These parameters are quantified by a set of conditional probability distributions or conditional probability tables (CPTs), one for each variable conditioning on its parents. Overall, a Bayesian network represents a joint probability distribution over the variables.

Figure 1 show an example of Bayesian Network, the qualitative part is indicated by the DAG structure, and the quantitative part is given by four CPTs.
Applying Bayesian networks to real-world problems typically requires building graphical representations of relationships among variables. When these relationships are not known \textit{a priori}, the structure of Bayesian networks must be learned. The topic of this survey is to review the state of the art works in Bayesian networks structure learning.

The remainder of this section first introduce the Bayesian networks as well as the notation and terminology used throughout the rest of this paper.

1.1 Notation and Terminology

A Bayesian network is a directed acyclic graph (DAG) $G$ that represents a joint probability distribution over a set of random variables $\mathbf{V} = \{X_1, X_2, \ldots, X_n\}$. A directed arc from $X_i$ to $X_j$ represents the dependence between the two variables; we say $X_i$ is a parent of $X_j$. We use $\text{PA}_j$ to stand for the parent set of $X_j$. The dependence relation between $X_j$ and $\text{PA}_j$ are quantified using a conditional probability distribution, $P(X_j|\text{PA}_j)$. The joint probability distribution represented by $G$ is factorized as the product of all the conditional probability distributions in the network, i.e., $P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\text{PA}_i)$.
In addition to the compact representation, Bayesian networks also provide principled approaches to solving various inference tasks, including belief updating, most probable explanation, maximum a Posteriori assignment [36], and most relevant explanation [49, 47, 48].

Given a dataset $D = \{D_1, ..., D_N\}$, where each data point $D_i$ is a vector of values over variables $V$, learning a Bayesian network is the task of finding a network structure that best fits $D$. In this work, we assume that each variable is discrete with a finite number of possible values, and no data point has missing values.

The remainder of the paper is organized as follow: in section 2, we introduce the basic concepts of Bayesian Network structure learning and different learning algorithms. In section 3, we elaborate two most efficient exact structure learning methods. We conclude the review in section 4.
Bayesian Network Structure Learning

Bayesian networks structure learning problem takes the data as inputs and produces an directed acyclic graph as the output. Before we dive into the algorithmic details in the next section, let us review these basic concepts and briefly introduce different learning methods in this section.

There are roughly three main approaches to the learning problem: score-based learning, constraint-based learning, and hybrid methods. Score-based learning methods evaluate the quality of Bayesian network structures using a scoring function and selects the one that has the best score [12, 24]. These methods basically formulate the learning problem as a combinatorial optimization problem. They work well for datasets with not too many variables, but may fail to find optimal solutions for large datasets. We will discuss this approach in more detail in the next section, as it is the approach we take. Constraint-based learning methods typically use statistical testings to identify conditional independence relations from the data and build a Bayesian network structure that best fits those independence relations [36, 42, 8, 18, 46]. Constraint-based methods mostly rely on results of local statistical testings, so they can often scale to large datasets. However, they are sensitive to the accuracy of the statistical testings and may not work well when there are insufficient or noisy data. In comparison, score-based methods work well even for datasets with relatively few data points. Hybrid methods aim to integrate the advantages of the previous two approaches and use combinations of constraint-based and/or score-based methods for solving the learning problem [15, 1, 45, 37]. One popular strategy is to use constraint-based learning to create a skeleton graph and then use score-based learning to find a high-scoring network structure that is a subgraph of the skeleton [45, 37].

In this paper, we mainly focus on the score-based approach. We do not consider Bayesian model averaging methods which aim to estimate the posterior probabilities of structural features such as edges rather than model selection [24, 20, 14]. Score-based learning methods rely on a scoring function $Score(.)$ in evaluating the quality
of a Bayesian network structure. A search strategy is used to find a structure $G^*$ that optimizes the score. Therefore, score-based methods have two major elements, *scoring functions* and *search strategies*. We first introduce the scoring functions, then review two classes of search strategies, local search strategies and optimal search strategies.

### 2.1 Scoring Functions

Many scoring functions can be used to measure the quality of a network structure. Some of them are Bayesian scoring functions which define a posterior probability distribution over the network structures conditioning on the data, and the structure with the highest posterior probability is presumably the best structure. These scoring functions are best represented by the Bayesian Dirichlet score (BD) [25] and its variations, e.g., K2 [12], Bayesian Dirichlet score with score equivalence (BDe) [25], and Bayesian Dirichlet score with score equivalence and uniform priors (BDeu) [7]. Other scoring functions often have the form of trading off the goodness of fit of a structure to the data and the complexity of the structure. The goodness of fit is measured by the likelihood of the structure given the data or the amount of information that can be compressed into a structure from the data. Scoring functions belonging to this category include minimum description length (MDL) (or equivalently Bayesian information criterion, BIC) [38, 43, 28], Akaike information criterion (AIC) [3, 6], (factorized) normalized maximum likelihood function (NML/fNML) [40], and the mutual information tests score (MIT) [17]. All of these scoring functions have some general properties. The most important property is *decomposable*, that is, the score of a network can be decomposed into a sum of node scores [24].

It is worth noting that the optimal structure $G^*$ may not be unique because multiple Bayesian network structures may share the same optimal score.\[1\] Two network structures are said to belong to the same *equivalence class* [9] if they represent the same set of probability distributions with all possible parameterizations. Score-equivalent scoring

---

\[1\] That is why we often use “an optimal” instead of “the optimal” throughout this paper.
functions assign the same score to structures in the same equivalence class. Most of the above scoring functions are score equivalent.

Due to the limit of the space, we mainly introduce the MDL score in this survey. Let $r_i$ be the number of states of $X_i$, $N_{pa_i}$ be the number of data points consistent with $PA_i = pa_i$, and $N_{x_i,pa_i}$ be the number of data points further constrained by $X_i = x_i$. MDL is defined as follows [28].

$$\text{MDL}(G) = \sum_i \text{MDL}(X_i|PA_i), \quad (1)$$

where

$$\text{MDL}(X_i|PA_i) = H(X_i|PA_i) + \frac{\log N}{2}K(X_i|PA_i), \quad (2)$$

$$H(X_i|PA_i) = -\sum_{x_i,pa_i} N_{x_i,pa_i} \log \frac{N_{x_i,pa_i}}{N_{pa_i}}, \quad (3)$$

$$K(X_i|PA_i) = (r_i - 1) \prod_{X_i \in PA_i} r_i. \quad (4)$$

The goal is then to find a Bayesian network that has the minimum MDL score. Any other decomposable scoring function, such as BIC, BDeu, or fNML, can be used instead without affecting the search strategy. One slight difference between MDL and the other scoring functions is that the latter scores need to be maximized in order to find an optimal solution. But it is rather straightforward to translate between maximization and minimization problems by simply changing the sign of the scores. Also, we sometimes use costs to refer to the scores.

### 2.2 Local Search Strategies

Given $n$ variables, there are $O(n2^{n(n-1)})$ directed acyclic graphs (DAGs). The size of the solution space grows exponentially in the number of variables. It is not surprising that score-based structure learning has been shown to be NP-hard [10]. Due to the complexity,
early research focused mainly on developing local search algorithms [24, 5]. Popular local
search strategies that were used include greedy hill climbing, Min Min Hill Climbing,
stochastic search, etc.

2.2.1 Hill Climbing

Greedy hill climbing methods have been proposed. They start from an initial solu-
tion, which typically is an empty network without any edges, or a randomly generated
structure, and iteratively apply single edge operations, including addition, deletion
and reversal, looking for the choice that locally maximizing the score improvement. Extens-
ions to this approach include tabu search with random restarts [22], limiting the number
of parents or parameters for each variable [21], searching in the space of equivalence
classes [11], searching in the space of variable orderings [44], and searching under the
constraints extracted from data [45].

2.2.2 Max-Min Hill Climbing

Max-min hill-climbing (MMHC) [45], a double step algorithm, was proposed. After first
executing a statistical conditional independence test to find a reduced set of candidate
parents sets, it applies a greedy hill-climbing step to find a local optimal solution. MMHC
is classified as a hybrid methods between scoring-based methods and constraint based on
methods.

The Max-min step, based on the Max-min parent-children (MMPC), is a heuristic that
looks for a set of candidates for each node in the graph. It aims to find an undirected
graph representing the skeleton of original DAG by looking for subsets of variables $Z$
conditionally separating pairs of variables $X, Y$. Such test is denoted as $T(X, Y|Z)$, and
is $G^2$ test in [45].

MMPC works as follows. Let $PC_X^G$ be the set of parents and children for node $X$ in
a Bayesian Network over $P$ that is Markov equivalent with respect to $G$, it holds that
$PC_X^C = PC_X^{C'}$, and we can write only $PC_X$. Thus, the set of parents and children of a node is the same for all the Markov equivalent Bayesian Networks over the same probability distribution $P$. MMPC performs the conditional independence test $T(X,Y|Z)$ in a subroutine called $MinAssoc$, The logarithm of the so obtained $p-$value is called the association between variables.

The Max-min heuristic, for each variable $X$, iteratively constructs a set of variables with high association with $X$ (the CPC set), picking up at each iteration the variable $Y$ with the largest association, until such association falls under a minimum value. The set of CPCs found by the Max-min heuristic form the skeleton of the Bayesian Network.

The hill-climbing algorithm is then applied to the skeleton to perform a local search, by using three possible edge operations over DAGs (edge insertion, edge removal and edge reversal) and greedily choosing the one that increases the score the most until no improvement. The search space for edges is limited to the ones allowed by skeleton found by Max-min heuristic.

### 2.2.3 Stochastic Search

Stochastic search methods such as Markov Chain Monte Carlo and simulated annealing have also been applied to find a high-scoring structure [24, 19, 33]. These methods explore the solution space using non-deterministic transitions between neighboring network structures while favoring better solutions. The stochastic moves are used in hope to escape local optima and find better solutions.

### 2.3 Optimal Search Strategies

Local search methods are quite robust in the face of large learning problems with many variables. However, they do not guarantee to find an optimal solution. What is worse, the quality of their solutions is typically unknown. Recently multiple exact algorithms
have been developed for learning optimal Bayesian networks.

2.3.1 Branch-and-Bound

A branch and bound algorithm (BB) was proposed in [16] for learning Bayesian networks. The algorithm first creates a cyclic graph by allowing each variable to obtain optimal parents from all the other variables. A best-first search strategy is then used to break the cycles by removing one edge at a time. The algorithm uses an approximation algorithm to estimate an initial upper bound solution for pruning. The algorithm also occasionally expands the worst nodes in the search frontier in hope to find better networks to update the upper bound. At completion, the algorithm finds an optimal network structure that is a subgraph of the initial cyclic graph. If the algorithm ran out of memory before finding the solution, it will switch to using a depth-first search strategy to find a suboptimal solution. Authors claim their algorithms to be optimal. Such optimality, however, is under the constraint that each variable can only select parents from optimal parents set.

2.3.2 Dynamic Programming

Several dynamic programming algorithms are proposed based on the observation that a Bayesian network has at least one leaf [34, 41]. A leaf is a variable with no child variables in a Bayesian network. In order to find an optimal Bayesian network for a set of variables V, it is sufficient to find the best leaf. For any leaf choice X, the best possible Bayesian network is constructed by letting X choose an optimal parent set PA_X from V \{X\} and letting V \{X\} form an optimal subnetwork. Then the best leaf choice is the one that minimizes the sum of Score(X, PA_X) and Score(V \{X\}) for a scoring function Score(.).

More formally, we have:

\[
Score(V) = \min_{X \in V} \{Score(V \setminus \{X\}) + BestScore(X, V \setminus \{X\})\}, \tag{5}
\]
where
\[
\text{BestScore}(X, V \setminus \{X\}) = \min_{\text{PA}_X \subseteq V \setminus \{X\}} \text{Score}(X, \text{PA}_X).
\] (6)

Given the above recurrence relation, a dynamic programming algorithm works as follows. It first finds optimal structures for single variables, which is trivial. Starting with these base cases, the algorithm builds optimal subnetworks for increasingly larger variable sets until an optimal network is found for $V$. The dynamic programming algorithms can find an optimal Bayesian network in $O(n2^n)$ time and space [27, 34, 39, 41]. Recent algorithms have improved the memory complexity by either trading longer running times for reduced memory consumption [35] or taking advantage of the layered structure present within the dynamic programming lattice [32].

### 2.3.3 Integer Linear Programming

Integer linear programming (ILP) has been used to learn optimal Bayesian network structures [13, 26]. The BNSL problem is formulated as an integer linear program over a polytope with an exponential number of facets. An outer bound approximation to the polytope is then solved. If the solution of the relaxed problem is integral, it is guaranteed to be the optimal structure. Otherwise, cutting planes and branch and bound algorithms are subsequently applied to find the optimal structure. More details will be covered in next section.

### 2.3.4 Heuristic Search

Several heuristic search algorithms have been developed for solving this problem by formulating the problem as a shortest path problem [52], e.g., A*, breadth-first branch and bound (BFBnB) [31], and anytime Window A* [29]. Most of these algorithms need a lower bound function in estimating the quality of a search path so that they can prune
paths that are guaranteed to lead to suboptimal solutions and focus on exploring the most promising search spaces.

A lower bound was first computed by a simple heuristic function which completely relaxes the acyclicity constraint of Bayesian networks; then a better lower bound was proposed by using static $k$-cycle conflict heuristic \cite{50}, which demonstrated excellent performance when used by the search algorithms to solve many benchmark data sets. It takes as input a partition of the random variables of a data set, and computes the heuristic value of a path by enforcing the acyclicity constraint between the variables within each group and relaxing the acyclicity between different groups. More details will be covered in next section.
3 Exact Structure Learning

In this survey, we focus on the exact Bayesian network structure learning. We briefly reviewed some algorithms which adopted optimal search strategies in the last section. Previous study [51] showed that integer linear programming and heuristic search methods are more efficient than branch-and-bound and dynamic programming methods. We highlight and elaborate these two methods in this section.

3.1 Integer Linear Programming

In [13], BN structure learning was encoded as an integer programming. For each variable \( x \) and candidate parent set \( PA \), a binary variable \( I(PA \rightarrow x) \) is created. \( I(PA \rightarrow x) = 1 \) if and only if \( PA \) are the parents of \( x \) in an optimal BN. These variables were called family variables. Table 1 shows coding of family variables. For each variable \( X_i \), there are \( k_i \) potential optimal parents candidates. The total number of family variables is \( \sum_i k_i \).

| \( I(PA_{1,1} \rightarrow X_1) \) | \( I(PA_{1,2} \rightarrow X_1) \) | ... | \( I(PA_{1,k_1} \rightarrow X_1) \) |
| \( I(PA_{2,1} \rightarrow X_2) \) | \( I(PA_{2,2} \rightarrow X_2) \) | ... | \( I(PA_{2,k_2} \rightarrow X_2) \) |
| \( I(PA_{3,1} \rightarrow X_3) \) | \( I(PA_{3,2} \rightarrow X_1) \) | ... | \( I(PA_{3,k_3} \rightarrow X_3) \) |
| ... | ... | ... | ... |
| \( I(PA_{n,1} \rightarrow X_n) \) | \( I(PA_{n,2} \rightarrow X_n) \) | ... | \( I(PA_{n,k_n} \rightarrow X_n) \) |

Table 1: coding for family variables

Then BN structure learning can be formulated as the following constrained optimisation problem:

\[
\sum_{x,PA} s(x, PA) I(PA \rightarrow x)
\]

To use an integer linear programming approach it is necessary to use linear constraints to ensure that only valid DAGs are feasible.

Jaakkola etc. [26] use a family of cluster constraints which impose that, for every subset \( C \subset V \) of the nodes of the graph \( G \), there must be at least one node whose parent set...
either completely lies outside C, or is the empty set. Formally, the cluster constraints is defined as follows: for every set of nodes $C \subset V$,

$$\sum_{x \in C} \sum_{PA: PA \cap C = \emptyset} I(PA \rightarrow x) \geq 1$$  \hspace{1cm} (8)

This class of constraints stems from the observation that any subset $C$ of nodes in a DAG must contain at least one node who has no parent in that subset. Clearly, there is an exponential number of cluster constrains. The method to this problem is to remove them, solve the linear relaxation of the problem and look for violated constraints.

Cussens etc. [13, 4] inherited Jaakola approach, but treated problem with branch-and-cut strategy: they consider the relaxated problem obtained by removing a more general version of the cluster constraints from the mode and solve it, andding the most effective cluster constraints as cutting planes when needed, getting a solution $x^\ast$. If $x^\ast$ does not violate any cluster constraints and is integer-value, the problem is solved; otherwise, $x^\ast$ is used to be branched on, creating two new subproblems.

### 3.2 Heuristic Search

In [52], BN structure learning was cast as a shortest path search problem. The state space graph for learning Bayesian networks is basically a Hasse diagram containing all of the subsets of the variables in a domain. Figure 2 visualizes the state space graph for a learning problem with four variables.

Figure 2 shows the implicit search graph for four variables. The top-most node with the empty set at layer 0 is the start search node, and the bottom-most node with the complete set at layer $n$ is the goal node, where $n$ is the number of variables in a domain. An arc from $U$ to $U \cup \{X_i\}$ represents generating a successor node by adding a new variable $\{X_i\}$ to an existing set of variables $U$; $U$ is called a predecessor of $U \cup \{X_i\}$. the cost of the arc is equal to the score of the optimal parent set for $X_i$ out of $U$, which is computed
by considering all subsets of the variables in $PA \subseteq U, PA \in P_i$, i.e.,

$$cost(U \rightarrow U \cup \{X_i\}) = BestScore(X_i, U)$$

$$= \min_{PA_i \subseteq U, PA_i \in P_i} s_i(PA_i).$$

For example, the arc $\{X_1, X_2\} \rightarrow \{X_1, X_2, X_3\}$ has a cost equal to $BestScore(X_3, \{X_1, X_2\})$.

Each node at layer $i$ has $n - i$ successors as there are this many ways to add a new variable, and $i$ predecessors as there are this many leaf choices. We define expanding a node $U$ as generating all successors nodes of $U$.

With the search graph thus defined, a path from the start node to the goal node is defined as a sequence of nodes such that there is an arc from each of the nodes to the next node in the sequence. Each path also corresponds to an ordering of the variables in the order of their appearance. For example, the path traversing nodes $\emptyset, \{X_1\}, \{X_1, X_2\}, \{X_1, X_2, X_3\}, \{X_1, X_2, X_3, X_4\}$ stands for the variable ordering $X_1, X_2, X_3, X_4$. That is why the graph is called order graph. The cost of a path is defined as the sum of the costs of all the arcs on the path. The shortest path is then the path with the minimum total cost in the order graph.

Given the shortest path, we can reconstruct a Bayesian network structure by noting that
each arc on the path encodes the choice of optimal parents for one of the variables out of the preceding variables, and the complete path represents an ordering of all the variables. Therefore, putting together all the optimal parent choices generates a valid Bayesian network. By construction, the Bayesian network structure is optimal.

### 3.2.1 Finding the Shortest Path: heuristic search algorithms

This shortest path problem has been solved using several heuristic search algorithms, including A* [53], anytime window A* (AWA*) [30] and breadth-first branch and bound (BFBnB) [31].

![Figure 3: Finding Shortest Path by Heuristic Search.](image)

In A* [23], an admissible heuristic function is used to calculate a lower bound on the cost from a node \( U \) in the order graph to goal. An f-cost is calculated for \( U \) by summing the cost from start to \( U \) (called \( g(U) \)) and the lower bound from \( U \) to goal (called \( h(U) \)). Figure 3 illustrates the basics of the heuristic search methods for finding shortest path of BNSL problem. \( g(U) \) corresponds to the score of the subnetwork over the variables \( U \), and \( h(U) \) estimates the score of the remaining variables. So \( f(U) = g(U) + h(U) \).

The f-cost provides an optimistic estimation on how good a path through \( U \) can be. The search maintains a list of nodes to be expanded sorted by f-costs called open and a list...
of already-expanded nodes called *closed*. Initially, *open* contains just *start*, and *closed* is empty. Nodes are then expanded from *open* in best-first order according to f-costs. Expanded nodes are added to *closed*. As better paths to nodes are discovered, they are added to *open*. Upon expanding *goal*, the shortest path from *start* to *goal* has been found.

In AWA* [2], a sliding window search strategy is used to explore the order graph over a number of iterations. During each iteration, the algorithm uses a fixed window size, *w*, and tracks the layer *l* of the deepest node expanded. For the order graph, the layer of a node corresponds to the number of variables in its subnetwork. Nodes are expanded in best-first order as usual by A*; however, nodes selected for expansion in a layer less than *l* − *w* are instead frozen. A path to *goal* is found in each iteration, which gives an upper bound solution. After finding the path to *goal*, the window size is increased by 1 and the frozen nodes become *open*. The iterative process continues until no nodes are frozen during an iteration, which means the upper bound solution is optimal. Alternatively, the search can be stopped early if a resource bound, such as running time, is exceeded; the best solution found so far is output.

In BFBnB [54], nodes are expanded one layer at a time. Before beginning the BFBnB search, a quick search strategy, such as AWA* for a few iterations or greedy hill climbing, is used to find a “good” network and its score. The score is used as an upper bound. During the BFBnB search, any node with an f-cost greater than the upper bound can safely be pruned.

### 3.2.2 Heuristics

The effectiveness of the heuristic function is a critical factor in the effectiveness of the search.

The following simple heuristic function was introduced in [53] for computing lower bounds for A* search.
Figure 4: Two pattern databases for a 8 variables problem. 8 variables are partitioned into two groups, $V_1 = \{X_1, X_2, X_3, X_4\}$ and $V_2 = \{X_5, X_6, X_7, X_8\}$. (Left) The pattern database for group $V_1$; bold arrows show the path corresponding to the score $P_1$ for pattern $\{X_2, X_3\}$, where $P_1 = \text{BestScore}(X_3, \{X_1, X_2, X_4\} \cup V_2) + \text{BestScore}(X_2, \{X_1, X_4\} \cup V_2)$. (Right) The pattern database for group $V_2$; bold arrows show the path corresponding to the score $P_2$ for pattern $\{X_5, X_7\}$, where $P_2 = \text{BestScore}(X_7, \{X_5, X_6, X_8\} \cup V_1) + \text{BestScore}(X_5, \{X_6, X_8\} \cup V_1)$. The heuristic value for pattern $\{X_2, X_3, X_5, X_7\}$ is $P_1 + P_2$.

**Definition 1** Let $U$ be a node in the order graph, its heuristic value is

$$h(U) = \sum_{X \in V \setminus U} \text{BestScore}(X, V \setminus \{X\}). \quad (11)$$

The above heuristic function allows each remaining variable to choose optimal parents from all of the other variables. Therefore it completely relaxes the acyclicity constraint of Bayesian networks in the estimation. The heuristic was proven to be admissible, meaning it never overestimates the future distance [53]. Admissible heuristics guarantee the optimality. However, because of the complete relaxation of the acyclicity constraint, the simple heuristic may generate loose lower bounds.

In [50], an improved heuristic function called static k-cycle conflict heuristic was proposed by reducing the amount of relaxation. The idea is to partition the variables $V$ into multiple groups $V_i$ (typically two), i.e. $V = \bigcup_i V_i$, and enforce acyclicity within each group while still allowing cycles between the groups. For the partition, we need to compute a pattern database for each group $V_i$. For this particular problem, a pattern
database for group $V_i$ is basically a full order graph containing all subsets of $V_i$. We will use a backward breadth first search to create the graph layer by layer starting from the node $V_i$. The cost for any reverse arc from $U \cup \{X\}$ to $U$ in this order graph will be $BestScore(X, (\bigcup_{j \neq i} V_j) \cup U)$. We then enumerate all subsets of each group $V_i$ as the patterns, which can be done by a reverse breadth-first search in an order graph containing only $V_i$ [50]. The patterns from different groups are guaranteed to be mutually exclusive, so we simply pick out the maximum-size pattern for each group that is a subset of $V \setminus U$ and add them together as the lower bound. Figure 4 shows two pattern databases for a 8-variable problem, as well as the procedure of calculating the heuristic value of node \{X_2, X_3, X_5, X_7\}.

The tightness of the static k-cycle conflict heuristic depends highly on the partition being used. The heuristic can avoid directed cycles for the patterns within the same group, but cannot avoid cycles between different groups. In the example shown in Figure 4, $X_3$ selects parents \{X_1, X_3\} as parents (subset of \{X_1, X_4\} $\cup$ $V_2$ ) and $X_5$ selects \{X_3, X_6\} as parents; there is a cycle between \{X_3\} and \{X_5\}. In current works so far, now simple partition are used, e.g, first half variables fall into one group and second half variables fall into the other group. How to form a good partition could be one future research direction.

3.3 Integer Linear Programming VS Heuristic Search

Integer linear programming approach formulates the BNSL problem as an integer linear program whose variables correspond to the potential optimal parent sets of all variables (called famility variables), thus the efficiency depends of the number of family variables, while the heuristic search approach formulates the BNSL problem as the graph search problem who efficiency depends on complexity of the search space, which is $2^n$, where $n$ is the number of variables.

Yuan etc. [51] compared the two approaches. They compared GOBNILP (a public soft-
ware using integer linear programming method) to A* algorithm (which guarantees to expand the minimum number of nodes in heuristic search algorithms). The comparison between GOBNILP and A* showed that they each has its own advantages. A* was able to find optimal Bayesian networks for all the datasets well within the time limit. GOBNILP failed to learn optimal Bayesian networks for some benchmark datasets. The reason is that GOBNILP formulates the learning problem as an integer linear program whose variables correspond to the optimal parent sets of all variables. Even though these datasets do not have many variables, they have many optimal parent sets, so the integer programs for them have too many variables to be solvable within the time limit. On the other hand, the results also showed that GOBNILP was quite efficient on many benchmark datasets. Even though a dataset may have many variables, GOBNILP can solve it efficiently as long as the number of optimal parent sets is small.

These insights are quite important, as they provide a guideline for choosing a suitable algorithm given the characteristic of a dataset. If there are many optimal parent sets but not many variables, heuristic search is the better algorithm; if the other way around is true, integer linear programming is better. In comparison to integer linear programming, heuristic search is less sensitive to the number of optimal parent sets, number of data points, or scoring functions, but is more sensitive to the number of variables in the datasets.
4 Conclusion and Future Work

Bayesian Networks are frequently employed for decision support system, such as in medicine and finance. Learning a Bayesian network from data is an important problem that has been studied extensively during the last two decades. Due to the NP-hard characteristic of the Bayesian network structure learning problem, the early works focused on local search methods which provided non-optimal results. Along with the rapid development of computer software and hardware technology, current research on learning optimal Bayesian networks, once thought impractical, have made strides in recent years. This survey first gives a comprehensive suvey of bayesian network structure learning algorthms. We introduce the basic concepts of bayesian network structure learning problem. We review the widely used greedy and local search non-optimal learning algirthms; then highlight recent exact structure learning methods, integer linear programming and heuristic search. The characeristics of both methods are reviewed. The main challenge for exact structure learning problems is how to scale up the learning problem. These two methods leads to many possible future directions worth futher exploration.
References


