Time–Approximation Trade-Offs for Learning Bayesian Networks

Madhumita Kundu MADHUMITA.KUNDU@UIB.NO
University of Bergen
Pekka Parviainen PEKKA.PARVIAINEN@UIB.NO
University of Bergen
Saket Saurabh SAKET@IMSC.RES.IN
University of Bergen, The Institute of Mathematical Sciences

Editors: J.H.P. Kwisthout & S. Renooij

Abstract

Bayesian network structure learning is an NP-hard problem. Furthermore, the problem remains hard even for various subclasses of graphs. Motivated by the hardness of exact learning, we study approximation algorithms for learning Bayesian networks. First, we propose a moderately exponential time algorithm with running time $\mathcal{O}(2^{\frac{\ell}{k}n})$ that has an approximation ratio $\frac{\ell}{k}$ where n is the number of vertices and ℓ and k are user-defined parameters with $\ell \leq k$. That is, we give time–approximation trade-offs for learning Bayesian networks. Second, we present a polynomial time algorithm with an approximation ratio $\frac{1}{d}$ to find an optimal graph whose connected components have size at most d.

Keywords: Bayesian Network Structure Learning, Parameterized Algorithms, Approximation Algorithms.

1. Introduction

Bayesian networks are widely-used to represent joint distributions of several random variables. Bayesian networks have two parts: the structure and the parameters. The structure of a Bayesian network is a directed acyclic graph (DAG) and it expresses conditional independencies in the joint distribution. The parameters, on the other hand, specify conditional probability distributions associated with each vertex.

One is often interested in learning the structure of a Bayesian network from data. Two main approaches for learning Bayesian networks are constraint-based and score-based. In this work, we take the score-based approach in which one assigns each DAG a score based on how well it fits the data and the goal is to find a DAG that maximizes the score. Typically, one uses a scoring function that is decomposable, that is, the score of a DAG is a sum of local scores for vertex-parent set pairs.

Finding an optimal DAG is a challenging task. Indeed, finding an optimal DAG is an NP-hard problem (Chickering et al., 2004; Chickering, 1996). The NP-hardness remains in more restricted settings such as finding an optimal polytree (Dasgupta, 1999) or chain (Meek, 2001). A notable exception is finding an optimal tree which can be found in polynomial time (Chow and Liu, 1968).

Given that the structure learning problem is NP-hard, there has been lots of studies developing heuristic algorithms. However, research in approximation algorithms that give quality guarantees has been surprisingly scarce. To our knowledge, only Ziegler (2008) has studied this problem. He presents a polynomial time algorithm that provides a $\frac{1}{m}$-approximation where m is the maximum

in-degree of the DAG. Motivated by this scarcity, we develop approximation algorithms for learning Bayesian network structures and open avenues for research in two directions.

First, we study structure learning without any additional structural constraints. The fastest known (in the worst-case) exact algorithm for this problem runs in time $O(2^n)$ (Silander and Myllymäki, 2006). We present a moderately exponential time algorithm with running time $O(2^{\frac{\ell}{k}n})$ that has an approximation ratio $\frac{\ell}{k}$ where n is the number of vertices and ℓ and k are user-defined parameters with $\ell \leq k$ (Theorem 5). By selecting ℓ and k, one can trade between speed and approximation ratio. This algorithm is based on the algorithm by Parviainen and Koivisto (2013). While the proposed algorithm is slower than Ziegler's algorithm, it can achieve a better approximation ratio and it does not require an assumption of bounded indegree.

Our second contribution is an approximation algorithm for more constrained Bayesian networks. Finding Bayesian networks whose structure satisfies some structural constraints can be an interesting task. For example, inference in Bayesian networks is NP-hard in general but tractable in networks with bounded treewidth[1]. Thus, if one desires to perform exact inference fast, one should find a high-scoring network whose treewidth is bounded by a small constant. Unfortunately, structure learning remains NP-hard for all treewidth bounds larger than 1 (Korhonen and Parviainen, 2013). On the positive side, many constrained graph classes admit naturally parameterised algorithms. It has been shown that learning an optimal Bayesian network with bounded vertex cover number can be done in polynomial time for any constant vertex cover number (Korhonen and Parviainen, 2015). Recently, this result has been extended by proving that finding an optimal Bayesian network with bounded dissociation number is polynomial time for any constant bound (Grüttemeier and Komusiewicz, 2020). Both problems are W[1]-hard.

However, extending these results seems difficult as Grüttemeier and Komusiewicz (2020) have shown that finding an optimal Bayesian network whose connected components have size at most d is NP-hard for any $d \geq 3$. We provide an approximation algorithm for finding an optimal Bayesian network whose moralised graph can be transformed into connected components of size at most d by deleting at most k vertices. Note that while this graph class may sound artificial, the above-mentioned graphs with bounded vertex cover number and bounded dissociation number are special cases of this class with $d = 1$ and $d = 2$, respectively. Our algorithm is based on maximum weighted d-set packing and has approximation ratio $\frac{1}{d}$ (Theorem 6, Corollary 7).

One challenge with approximation algorithms for Bayesian network structure learning is that it is difficult the assess whether the approximation ratios give meaningful bounds in practice. In our results, we assume that the local scores are non-negative. However, a typical score is a logarithm of some probability and thus negative. Technically, this does not pose problems as the negative local scores can be transformed into non-negative ones by adding a sufficiently large constant to each of them and in a sense "shifting" them. However, the approximation ratios apply to the shifted scores and it is not straightforward to see what it means in terms of the original score.

Let us consider a simple example to illustrate the interpretation of approximation ratios in this context. Suppose we have five vertices and local scores vary between -10 and -20. Now the local scores can be made non-negative by adding the constant 20 to each of them. After the shift, shifted local scores are between 0 and 10. Suppose further that we run an algorithm with approximation ratio $1/2$ and find a DAG with shifted score 10. As the algorithm has on approximation ratio $1/2$,

1. To be exact, the moralised graph of the Bayesian network has bounded treewidth.

we know that the DAG we found has a score that is at least $1/2$ of the score of the optimal DAG. That is, the shifted score of the optimal DAG is at most 20.

Now, we can invert the shift and get bounds in terms of the original score. As we have added a constant 20 to each local score and there are five vertices, we know that the found DAG has score $10 - (5 * 20) = -90$ using the original local scores. Similarly, the optimal DAG has a score at most $20 - (5 * 20) = -80$.

2. Preliminaries

2.1. Notation

A directed graph $D = (V, A)$ is defined as a set of vertices V with $|V| = n$ and a set of arcs $A \subseteq V \times V$. An arc (v_1, v_2) in D is called incoming at v_2 and outgoing at v_1. Also, v_2 is called child of v_1. The total number of incoming arcs of any vertex v in D is called *in-degree* and total number of outgoing arcs is called *out-degree*. The set $Pa_v^D := \{v_i \in D : (v_i, v) \in A\}$ is called the *parent set* of v in D. A directed graph D is called a directed acyclic graph (DAG) if it has no directed cycles. A directed graph is called a *tournament* if for every pair of vertices v_1, v_2, precisely one of the arcs (v_1, v_2) or (v_2, v_1) is present. Let $G = (V, E)$ be an undirected graph with n vertices in V and $|E|$ edges. The graph G is termed as *connected* if there exists a path between every pair of vertices v_1 and v_2 in G. A subgraph of G is a graph formed by deleting some vertices or edges from G. We represent the subgraph obtained by removing the vertex set V' (or the edge set E') from G as $G' = G - V'$ (or $G - E'$ respectively). A connected component C of G is a maximal connected subgraph, which means that adding any additional vertex to C would disconnect the subgraph.

2.2. Bayesian Network Structural Learning (BNSL)

We consider the score-based approach to learn Bayesian networks: we assign a score to each DAG and try to find the one that maximises the score. The score of a DAG D is defined as $f(D) = \sum_{v \in V} f_v(Pa_v^D)$ where $f_v : 2^{V \setminus \{v\}} \to \mathbb{R}$ is called a *local score*. In other words, the local score of a vertex depends on its parent set and the score of a DAG is the sum of local scores of all the vertices. Now the learning problem can be defined as follows:

> Vanilla-BNSL
> **Input:** A family of local scores $\mathcal{F} = \left\{ f_v : 2^{V \setminus \{v\}} \to \mathbb{R} | v \in V \right\}$
> **Question:** Find a DAG $D = (V, A)$ that maximises the score $f(D)$.

Input Representation. Throughout this work, we let $n := |V|$ denote the number of vertices given in an instance of a BNSL problem. Furthermore, we assume that for $V = \{v_1, \ldots, v_n\}$, the local scores $\mathcal{F}$ are given as a two-dimensional array $\mathcal{F} := [Q_1, Q_2, \ldots, Q_n]$, where each Q_i is an array containing all triples $(f_{v_i}(Pa), |Pa|, Pa)$ where $f_{v_i}(Pa) > 0$ or $Pa = \emptyset$. Note that typically the sets Q_i do not contain all possible parent sets; one may, for example, compute scores for only parent sets with bounded number of parent sets or prune scores. Thus, only some of the possible triples $(f_{v_i}(Pa), |Pa|, Pa)$ are part of the input. This input representation is known as non-zero representation (Ordyniak and Szeider, 2013)[2]. The size $|\mathcal{F}|$ is defined as the number of bits we need to store this two-dimensional array. Hence the size of input, I we define $|I| := |\mathcal{F}|$.

2. This is a slightly misleading naming convention as the scores for empty parent sets may be zeros.

Learning with graph classes. In this paper, we are interested in learning an optimal DAG within a specific graph class. More precisely, we add constraints on the *moralised graph* of the DAG. For a given DAG $D = (V, A)$, the undirected graph $\mathcal{M}(D) := \mathcal{M}(V, E = E_1 \cup E_2)$ is called as *moralised graph* of D where $E_1 := \{\{u, v\} | (u, v) \in A\}$ and $E_2 = \{\{u, v\} \mid \exists\, w (\neq u, v)$ such that $\{u, v\} \subseteq Pa_w^D\}$.

We define a graph class to be a family of undirected graphs, denoted as Π. We are going to work with the following graph class: A graph G is called a *bounded component graph* (in short *bcg*), if its every connected component has size bounded by d. We denote this class by $\Pi_d := \{G : G$ is a bcg$\}$. We also extend our study to graphs that can be made members of another graph class by deleting a small number of vertices or edges. Let $\Pi + kv := \{G \mid \exists V' \subseteq V$ such that $|V'| \leq k$ and $G - V' \in \Pi\}$ be the family of graphs that can be transformed into a graph in Π by deleting no more than k vertices.

Now we define the general problem as follows:

$(\Pi + kv)$-BNSL **Parameter:** k
Input: A set of vertices V of size n, family of local scores $\mathcal{F}$
Question: Find a DAG $D = (V, A)$ that maximises $f(D) = \sum_{v \in V} f_v(Pa_v^D)$ such that $\mathcal{M}(D) \in \Pi + kv$.

Intuitively, $(\Pi + kv)$-BNSL is the VANILLA-BNSL problem with an additional constraint to the moralised graph of the resulting network i.e. its goal is to find a directed acyclic graph which maximises the total score and whose moralised graph can be transformed into graph of Π by removing at most k vertices.

2.3. Parameterised Complexity

The framework of parameterised complexity was introduced by Downey and Fellows (1995). Parameterised complexity is an area where in addition to the overall input size n, one studies how a relevant secondary measurement affects the computational complexity of problem instances. Parameterised decision problems are defined by specifying the input, the parameter, and the question to be answered. A problem is called slicewise polynomial $(\mathcal{XP})$ for a parameter k if it can be solved in time $\mathcal{O}(|I|^{f(k)})$ for a computable function f where $|I|$ is the size of the input. That is, the problem is solvable in polynomial time when k is constant. A problem is called *fixed parameter tractable (FPT)* for a parameter k if it can be solved in time $f(k) \cdot |I|^{\mathcal{O}(1)}$ for a computable function f.

3. Moderately exponential-time approximation algorithms for BNSL

We will present an approximation algorithm that allows us to trade between time and approximation ratio. Our algorithm is based on the partial order approach by Koivisto and Parviainen (Koivisto and Parviainen, 2010; Parviainen and Koivisto, 2013) that was used to trade between space and time. They showed that, given a partial order, a highest-scoring DAG compatible with the order can be found "fast". Then an optimal DAG can be found by constructing a family of partial orders that "covers" all linear orders and learning a DAG for each order. We drop the requirement of covering all linear orders which speeds up the algorithm but in the same time we lose the guarantee of finding an optimal DAG. However, we prove that our algorithm still has approximation guarantees.

Let us start by reviewing the partial order approach.

3.1. Partial Orders

To get started, we need definitions of some concepts related to partial orders. A *partial order* P on a base-set V is a subset of $V \times V$ such that for all $x, y, z \in V$ it holds that

1. $xx \in P$ (reflexivity)
2. $xy \in P$ and $yx \in P$ imply $y = x$ (antisymmetry), and
3. $xy \in P$ and $yz \in P$ imply $xz \in P$ (transitivity).

If $xy \in P$ we say that x precedes y. A partial order P is a linear order, if in addition, $xy \in P$ or $yx \in P$(but not both) for all $x, y \in V, x \neq y$ (totality). A linear order L is a linear extension of a partial order P if $P \subseteq L$. Recall that every permutation is in a one-to-one relationship with a linear order. Thus, we say that a permutation σ is a linear extension of a partial order P if the corresponding linear order of σ is a linear extension of P. So, we can use linear order and permutation interchangeably when they represent each other.

For a DAG D, a *topological order* of D is extending D to a tournament, say T. For a set $Y \subseteq V$, an *induced sub-topological order* $T[Y]$ is a topological order inside Y i.e. $T[Y] = T \cap (Y \times Y)$.

A DAG D and a partial order P are said to be *compatible* with each other if there exists a linear order L such that $P \subseteq L$ and $A \subseteq L$. In other words, some topological order of D is a linear extension of the partial order P.

A partial order B on base-set V is a *bucket order* if V can be partitioned into nonempty sets $B_1, B_2, \ldots, B_l$ called *buckets*, such that $xy \in B$ if and only if $x = y$ or $x \in B_i$ and $y \in B_j$ for some $i < j$.

It is known (Parviainen and Koivisto, 2013) that an optimal DAG compatible with a bucket order can be found efficiently using a dynamic programming algorithm.

Lemma 1 ((Parviainen and Koivisto, 2013)) *Let P be a bucket order over a set of n elements, with bucket sizes $b_1, b_2, \ldots, b_t$. Then, there exists an algorithm to find an optimal DAG compatible with P with running time $O^*(|I| + 2^{b_{\max}})$, where $b_{\max}$ is the maximum bucket size, $|I|$ is the size of the input and $O^*()$ hides polynomial factors in n.*

3.2. Approximation Algorithm for BNSL

We are now ready to present the algorithm. The intuition behind our algorithm is as follows. We define a bucket order P and find an optimal DAG compatible with P. Note that being compatible with P limits the choice of potential parent set for the vertices. However, the vertices in the last group are free to choose their parents as long as the induced subgraph within the bucket remains acyclic. Now, with sufficiently large last bucket and covering enough partial orders, it is possible to guarantee an approximation ratio.

Next, we present the algorithm. Integers ℓ and k ($1 \leq \ell \leq k \leq n$) are user-defined parameters that can be used to control the trade-off between the running time and the approximation ratio of the algorithm. We assume that the local scores are non-negative.

At a high-level, the algorithm is simple. First, the nodes are divided randomly into k equally-sizes sets. Then a bucket order is created by merging ℓ sets to be the last bucket in the order and remaining sets are placed in buckets in arbitrary order. Then an optimal DAG compatible with the bucket order

is found. This procedure is repeated for all combinations of ℓ sets and the highest scoring DAG found is returned.

Algorithm 1

1. Partition the vertex set V into k equally-sized sets $V_1, \ldots, V_k$.[3]
2. Let $\mathcal{W}_1, \mathcal{W}_2, \ldots, \mathcal{W}_t$ be the set of all combinations of ℓ sets (merged together into a set of vertices) from $V_1, \ldots, V_k$, where $t = \binom{k}{\ell}$. For each $\mathcal{W}_i$,
   - Define a bucket order such that $\mathcal{W}_i$ is the last bucket and each V_j that is not contained in $\mathcal{W}_i$ is placed in its own bucket in an arbitrary order. Now let $P_{\mathcal{W}_i}$ be the resulting partial order on V.
   - Learn an optimal DAG $G_{\mathcal{W}_i}$ compatible with $P_{\mathcal{W}_i}$ using the dynamic programming algorithm from Parviainen and Koivisto (2013).
3. Return the highest scoring DAG found during Step 2

Next, we analyze the approximation ratio and the running time of Algorithm 1.

Let D_{OPT} be an optimal DAG. Let $s_1, s_2, \ldots, s_k$ be the scores corresponding to vertex sets $V_1, V_2, \ldots V_k$. That is, for each $i \in \{1, 2, \ldots, k\}$, we have

$$s_i = \sum_{v \in V_i} f_v(Pa_v^{D_{OPT}}).$$

It follows that

$$f(D_{OPT}) = \sum_{i=1}^{k} s_i.$$

Let us sort scores $s_1, s_2, \ldots, s_k$ in a non-increasing order. Without loss of generality we can assume that $s_1, s_2, \ldots, s_k$ be the sorted order. We pick the first ℓ buckets in this order, and let $\mathcal{W}_{OPT} = \bigcup_{i=1}^{\ell} V_i$. Because $s_1, \ldots s_\ell$ are the highest scores, the following observation follows via a simple averaging argument.

Observation 2 *Assuming that the local scores are non-negative,* $\sum_{i=1}^{\ell} s_i \geq \frac{\ell}{k} \cdot f(D_{OPT})$.

Let D^* be the graph obtained from D_{OPT} by doing the following operations.

- For every $v \in \mathcal{W}_{OPT}$, $Pa_v^{D^*} = Pa_v^{D^{OPT}}$
- For every $v \in V \setminus \mathcal{W}_{OPT}$, $Pa_v^{D^*} = \emptyset$.

Because the parent set of every $v \in V \setminus \mathcal{W}_{OPT}$ is the empty set, we observe the following.

Observation 3 *D^* is a subgraph of D^{OPT} and D^* is compatible with every bucket order $P_{\mathcal{W}_{OPT}}$ where $\mathcal{W}_{OPT}$ is the last bucket.*

3. If n is not divisible with k, then the sets $V_1, \ldots, V_k$ are "almost equally sized" in the sense that their sizes differ by at most 1, that is, $|V_i| = \lceil n/k \rceil$ or $|V_i| = \lfloor n/k \rfloor$.

Specifically, D^* is compatible with the partial order $P_{\mathcal{W}_{OPT}}$ defined in Algorithm 1 when $\mathcal{W}_{OPT}$ is the last bucket.

Let $s_1^*, s_2^*, \ldots, s_k^*$ be the scores corresponding to $V_1, V_2, \ldots V_k$ respectively in D^*. Since the parent set of every vertex $v \in \mathcal{W}_{OPT}$ in D^* is same as that in D^{OPT},

$$\sum_{i=1}^{\ell} s_i^* = \sum_{i=1}^{\ell} s_i.$$

The following lemma shows that the total score of D^* is at least $\frac{\ell}{k}$ times the score of the optimal DAG.

Lemma 4 *Assuming that the local scores are non-negative,* $f(D^*) \geq \frac{\ell}{k} \cdot f(D_{OPT})$.

Proof

$$\begin{aligned} f(D^*) &= \sum_{i=1}^{\ell} s_i^* + \sum_{j=\ell+1}^{k} s_j^* \\ &\geq \sum_{i=1}^{\ell} s_i^* && \text{(Since } \textstyle\sum_{j=\ell+1}^{k} s_j^* \geq 0\text{)} \\ &= \sum_{i=1}^{\ell} s_i && \text{(Since } \textstyle\sum_{i=1}^{\ell} s_i^* = \sum_{i=1}^{\ell} s_i\text{)} \\ &\geq \frac{\ell}{k} \cdot f(D_{OPT}). && \text{(Via Observation 2)} \end{aligned}$$

■

Now, we are ready to prove our main result.

Theorem 5 *Assume that the local scores* $f_v(S) \geq 0$ *for all* $v \in V$ *and* $S \subseteq V \setminus \{v\}$ *and the size of the input is* $|I|$. *Algorithm 1 has an approximation ratio* $\frac{\ell}{k}$ *and running time* $\mathcal{O}^*(k^\ell(|I| + 2^{n\ell/k}))$.

Proof **Approximation ratio.** Algorithm 1 considers all buckets $\mathcal{W}_i$ that are unions of ℓ sets out of $V_1, \ldots, V_k$. Therefore, Algorithm finds a DAG D' which is compatible with a partial order whose last bucket is $\mathcal{W}_{OPT}$. In other words, $\mathcal{W}_i = \mathcal{W}_{OPT}$ for some i; Let us denote the partial order by $P_{\mathcal{W}_{OPT}}$. By Lemma 1, D' is an optimal DAG compatible with $P_{\mathcal{W}_{OPT}}$. By Lemma 4, the score $f(D') \geq \frac{\ell}{k} \cdot f(D_{OPT})$. Algorithm 1 returns a DAG D that is the highest scoring DAG found while looping over partial orders. Thus, $f(D) \geq f(D') \geq \frac{\ell}{k} \cdot f(D_{OPT})$.

Running Time. There are $\binom{k}{\ell} = \mathcal{O}(k^\ell)$ sets $\mathcal{W}_i$. For a fixed $\mathcal{W}_i$, the maximum bucket size is $\ell \cdot n/k$, therefore, by Lemma 1, an optimal DAG compatible with $P_{\mathcal{W}_i}$ can be found in $\mathcal{O}^*(|I| + 2^{n\ell/k})$ time. Therefore, the overall running time is $\mathcal{O}^*(k^\ell(|I| + 2^{n\ell/k}))$. ■

We notice that the algorithm does not require us to bound the number of parents. As long as the number size of $\mathcal{F}$ is moderately exponential or less (that is, less than $2^{n\ell/k}$), the running time of the algorithm is dominated by the dynamic programming part.

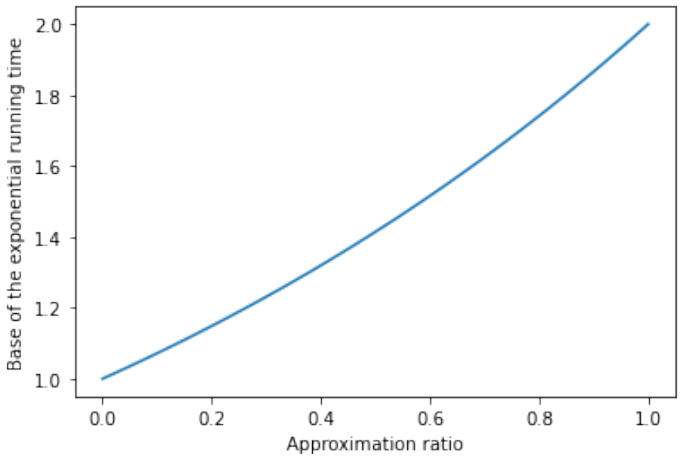

Figure 1: Running times and approximation ratios. The x-axis presents the approximation ratio ℓ/k which varies between 0 (no guarantees) and 1 (exact algorithm). The running time of the algorithm is $O(r^n)$ where $r = \ell/k$. The y-axis of the figure presents the base r.

Given the values of ℓ and k, we get approximation ratio $\frac{\ell}{k}$ and running time $\mathcal{O}^*(r^n)$ where $r = 2^{\frac{\ell}{k}}$. For example, if $\ell = 1$ and $k = 3$, then we get an algorithm that runs in time $\mathcal{O}^*(1.26^n)$ with an approximation ratio $1/3$. Increasing ℓ to $\ell = 2$ and keeping $k = 3$ gives us a slower algorithm that runs in time $\mathcal{O}^*(1.59^n)$ but with a better approximation ratio $2/3$. Figure 1 illustrates the trade-off between approximation ratio and running time.

4. Algorithm for $\Pi_d + kv$-BNSL

In this section, we design an approximation algorithm for $\Pi_d + kv$-BNSL, that is, finding an optimal DAG whose moralised graph can be partitioned into connected components of size at most d by removing at most k vertices.

We start by introducing an approximation algorithm for Π_d-BNSL. The algorithm is based on the observation that we can reduce Π_d-BNSL to the MAX WEIGHT d-SET PACKING problem (WD-SP). We assume that the local scores are non-negative.

WD-SP
Input: A universe $U = \{1, \ldots, n\}$, a family $\mathcal{S}$ of subsets of U with $|S| \leq d$ for all $S \in \mathcal{S}$, a weight function $w : \mathcal{S} \to \mathbb{R}_{\geq 0}$
Question: Find $\mathcal{H} \subseteq \mathcal{S}$ such that $H_i \cap H_j = \emptyset$ for all $H_i, H_j \in \mathcal{H}$, $H_i \neq H_j$ and $\sum_{H \in \mathcal{H}} w(H)$ is maximised.

Suppose we are given a Π_d-BNSL instance with vertex set V and a non-negative scoring function $f_v : 2^V \to \mathbb{R}_{\geq 0}$ for all $v \in V$. Now we can construct a WD-SP instance as follows. First, we set $U = V$ and $\mathcal{S} = \{X | X \subseteq U, |X| \leq d\}$. To compute the weight $w(X)$ for the set X, we find the highest scoring DAG on X. More precisely, let $\mathcal{X}$ be the set of all DAGs on the vertex set X. Then the weight $w(X) = \max_{D_X \in \mathcal{X}} \left(\sum_{x \in X} f_x(Pa_x^{D_X}) \right)$. This is a standard BNSL instance and can be solved using dynamic programming in $\mathcal{O}(d^2 2^d)$ time (Silander and Myllymäki, 2006).

Now it is easy to see that the instances are equivalent. Given a solution $\mathcal{H}$ to WD-SP, we can construct the solution to Π_d-BNSL by going through the sets $H \in \mathcal{H}$ and extracting the DAG D that maximised the summation in computation of $w(H)$. On the other hand, given a solution D to Π_d-BNSL, one can construct the solution to WD-SP by extracting the connected components in the moralised graph $\mathcal{M}(D)$. Now, we present a greedy algorithm that finds a solution for Π_d-BNSL via WD-SP.

Algorithm 2

1. Given an instance of Π_d-BNSL, construct an instance of WD-SP following the previous reduction.
2. Order the sets in $\mathcal{S}$ into decreasing order based on their weights and break ties arbitrarily. Let the ordering of sets in $\mathcal{S}$ be given by $S_1, \ldots, S_\beta$. Note that $\beta \leq n^d$.
3. Let Q be an array indexed with elements of U. Initialize each cell of Q with 0. Further initialise the WD-SP solution $\mathcal{H} = \emptyset$.
4. for $i = 1$ to β do as follows.
   - If for each element $e \in S_i$, we have that $Q[e] = 0$, then $\mathcal{H} = \mathcal{H} \cup \{S_i\}$. And, for each element $e \in S_i$, set $Q[e] := 1$.
   - Else, move to the next step.
5. From $\mathcal{H}$, construct a DAG $D = (V, E)$ in the following way: for every $H \in \mathcal{H}$, add the arcs $A(H)$ of the corresponding DAG that maximised the summation in computation of $w(H)$.
6. Return D

Theorem 6 *Algorithm 2 computes a $\frac{1}{d}$-approximate solution to Π_d-BNSL in time $\mathcal{O}((d2^d + \log n + 1)dn^d)$.*

Proof It is easy to observe that an approximate solution for WD-SP directly gives an approximate solution for Π_d-BNSL. So it is sufficient to prove the approximation ratio for WD-SP. To this end, let $\mathcal{H} \subseteq \mathcal{S}$ be a solution for WD-SP. Suppose H_1, H_2, …, H_q be the sets of $\mathcal{H}$. Without loss of generality, we can assume that $w(H_1) \geq w(H_2) \geq \ldots \geq w(H_q)$. Let $\mathcal{C} \subseteq \mathcal{S}$ with components C_1, C_2, …, C_ℓ be an optimal solution with total weight w^* and wlog we further assume that $w(C_1) \geq w(C_2) \geq \ldots \geq w(C_\ell)$. We can also assume that $w(C_r) = 0$ when $r > \ell$.

Because $|H_i| \leq d$, it intersects with at most d sets in C. As H_1 is the highest scoring set, it holds that

$$w(H_1) \geq w(C_1) \geq \frac{w(C_1) + w(C_2) + \ldots + w(C_d)}{d}.$$

Similarly, we have

$$\begin{aligned} w(H_2) &\geq \frac{w(C_{d+1}) + w(C_{d+2}) + \ldots + w(C_{2d})}{d} \\ &\cdots \\ w(H_q) &\geq \frac{w(C_{(q-1)d+1}) + w(C_{(q-1)d+2}) + \ldots + w(C_{qd})}{d}. \end{aligned}$$

Moreover, due to the construction ensuring that any two sets in $\mathcal{H}$ are mutually disjoint, it follows that

$$\begin{aligned} w(H) &= w(H_1) + w(H_2) + \dots w(H_q) \\ &\geq \frac{1}{d}\Big(w(C_1) + w(C_2) + \dots w(C_{qd})\Big) \\ &= \frac{1}{d}w(C) = \frac{w^*}{d}. \end{aligned}$$

As the solutions for Π_d-BNSL and WD-SP are equivalent, Π_d-BNSL has equivalent approximation ratio.

Second, let us prove time complexity. In Step 1, computing a weight takes $\mathcal{O}(d^2 2^d)$ time, and the weights are calculated for $\mathcal{O}(n^d)$ sets. Thus, Step 1 can be computed in $\mathcal{O}(d^2 2^d n^d)$ time. Step 2 involves sorting $\mathcal{O}(n^d)$ sets which can be done in $\mathcal{O}(dn^d \log n)$ time. Initialization in Step 3 can be carried out in $\mathcal{O}(n)$ time. The loop in Step 4 is executed at most $\beta \leq n^d$ times. In each iteration, we do at most d probes to the array Q, as well as change at most d entries, and thus the running time of this step is upper bounded by $\mathcal{O}(n^d d)$. This yields a total time requirement $\mathcal{O}((d2^d + \log n + 1)dn^d)$. ■

Now we are ready to present an approximation algorithm for $(\Pi_d + kv)$-BNSL. The algorithm uses the core-periphery approach introduced by Korhonen and Parviainen (2015).

Let $N_1 \subseteq V$ be the set of vertices whose removal from the moralised graph leaves an induced subgraph $\mathcal{M}[V \setminus N_1]$ whose every connected component has at most d vertices. Let $N_2 \subseteq V \setminus N_1$ be the set of vertices in $V \setminus N_1$ which are parents of N_1 in D or are in the same connected component with one of the parents in $\mathcal{M}[V \setminus N_1]$. Finally, let $N_3 = V \setminus (N_1 \cup N_2)$. By definition, $|N_1| \leq k$. Parents of a vertex v form a clique in the moralised graph. Thus, if a vertex had more than d parents outside N_1, there would be a connected components with more than d vertices in $\mathcal{M}[V \setminus N_1]$. Thus, each vertex in N_1 can have at most d parents outside N_1. As all parents of a single vertex are in the same component in $\mathcal{M}[V \setminus N_1]$ and the size of a component is at most d, we know that $|N_2| \leq dk$. Now, the algorithm will have $N_1 \cup N_2$ as the core and N_3 as the periphery. Furthermore, we divide also the arc set into core and periphery. That is, $A = A_c \cup A_p$ where A_c is the arc set for the core and A_p is the arc set for the periphery.

The size of the core is at most $(d+1)k$. For fixed N_1 and N_2, it suffices that we enumerate all DAGs over $N_1 \cup N_2$, for each of them check whether all the connected components of the induced subgraph of the moralised graph $\mathcal{M}[N_2]$ have at most d vertices, and select the highest scoring one A_c^*. There are $\mathcal{O}(((d+1)k)! 2^{((d+1)k)^2})$ DAGs which gives an upper bound for the time requirement.

The vertices in the periphery can have at most k parents from N_1 and at most $d-1$ parents from N_3. First, we do a preprocessing step and compute $f'_v(S) = \max_{T \subseteq N_1} f_v(S \cup T)$ for all $v \in N_3$ and $S \subseteq N_3 \setminus \{v\}$ with $|S| \leq d-1$. Computing one score takes $\mathcal{O}(2^k)$ time and we repeat this $\mathcal{O}(n^d)$ times. Thus, preprocessing can be done in $\mathcal{O}(2^k n^d)$ time.

After preprocessing, it suffices to choose parent sets for the vertices in N_3 using the scores f'_v. Here, we use Algorithm 2 to find a solution A_p^*.

Algorithm 3

- Initialize $f(A^*) = -\infty$.

- Iterate over all choices of N_1 and N_2. For each choice, do the following:
  1. Find an optimal arc set A_c^* for the core $N_c = N_1 \cup N_2$.
  2. Find an arc set $A_p^* = V \setminus N_c$ for the periphery N_p using Algorithm 2.
  3. If $f(A_c^* \cup A_p^*) > f(A^*)$ then set $A^* = A_c^* \cup A_p^*$ and $f(A^*) = f(A_c^* \cup A_p^*)$.

Theorem 7 *Assume that the local scores $f_v(S) \geq 0$ for all $v \in V$ and $S \subseteq V \setminus \{v\}$. Then, Algorithm 3 returns a $\frac{1}{d}$-approximate solution to $(\Pi_d + kv)$-BNSL in $\mathcal{O}(n^{dk+k+d+1})$ time.*

Proof By Theorem 6, the approximation ratio for the periphery is $\frac{1}{d}$ and the rest of the graph is learned exactly. Thus, it is clear that the algorithm guarantees approximation ratio $\frac{1}{d}$.

We iterate over all possible choices for sets N_1 and N_2 which takes $\binom{n}{k}\binom{n-k}{dk} = \mathcal{O}\left(n^{(d+1)k}\right)$ iterations. For each iteration, the core can be found in time $\mathcal{O}(((d+1)k)!2^{((d+1)k)^2})$ and the periphery in time $\mathcal{O}(n^{d+1})$ (Theorem 6). This yields total time requirement $\mathcal{O}(n^{dk+k+d+1})$. ■

In this paper we choose to give a self-contained greedy algorithm for WD-SP for completeness. However, we could have used the known algorithm by Berman (2000) for WD-SP. He gave an algorithm with an approximation factor of $2/d$ running in time $n^{\mathcal{O}(d)}$. Using this algorithm directly as a black box we can obtain a $\frac{2}{d}$-approximate solution to $(\Pi_d + kv)$-BNSL in $\mathcal{O}(n^{dk+k+d+1})$ time.

5. Conclusion

In this paper, we studied approximation algorithms for learning Bayesian networks. First, we gave a moderately exponential time algorithm with running time $O(2^{\frac{\ell}{k}n})$ that has an approximation ratio $\frac{\ell}{k}$ where n is the number of vertices and ℓ and k are user-defined parameters with $\ell \leq k$. That is, we give time–approximation trade-offs for learning Bayesian networks. Second, we present a polynomial time algorithm with an approximation ratio $\frac{1}{d}$ to find an optimal graph whose connected components have size at most d.

Acknowledgments

Parts of this work have been done in the context of CEDAS (Center for Data Science, University of Bergen, UiB).

References

P. Berman. A $d/2$ approximation for maximum weight independent set in d-claw free graphs. *Nord. J. Comput.*, 7(3):178–184, 2000.

D. M. Chickering. Learning Bayesian networks is NP-complete. In *Learning from data*, pages 121–130. Springer, 1996.

M. Chickering, D. Heckerman, and C. Meek. Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research*, 5, 2004.

C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

S. Dasgupta. Learning polytrees. In *UAI'99: Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, 1999.

R. G. Downey and M. R. Fellows. Parameterized computational feasibility. In *Feasible mathematics II*, pages 219–244. Springer, 1995.

N. Grüttemeier and C. Komusiewicz. Learning Bayesian networks under sparsity constraints: A parameterized complexity analysis. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 4245–4251, 2020.

M. Koivisto and P. Parviainen. A space–time tradeoff for permutation problems. In *Proceedings of the Twenty First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 484–492. Association for Computing Machinery, 2010.

J. H. Korhonen and P. Parviainen. Learning bounded tree-width Bayesian networks. In *16th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2013.

J. H. Korhonen and P. Parviainen. Tractable Bayesian network structure learning with bounded vertex cover number. *Advances in Neural Information Processing Systems*, 28:622–630, 2015.

C. Meek. Finding a path is harder than finding a tree. *Journal of Artificial Intelligence Research*, 15: 383–389, 2001.

S. Ordyniak and S. Szeider. Parameterized complexity results for exact Bayesian network structure learning. *Journal of Artificial Intelligence Research*, 46:263–302, 2013.

P. Parviainen and M. Koivisto. Finding optimal Bayesian networks using precedence constraints. *Journal of Machine Learning Research*, 14(1):1387–1415, 2013.

T. Silander and P. Myllymäki. A simple approach for finding the globally optimal Bayesian network structure. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2006.

V. Ziegler. Approximation algorithms for restricted Bayesian network structures. *Information Processing Letters*, 108(2):60, 2008.