Efficient Bayesian Network Structure Learning via Parameterized Local Search on Topological Orderings

Niels Grüttemeier, Christian Komusiewicz and **Nils Morawietz**[*]
Philipps-Universität Marburg, Marburg, Germany
{niegru, komusiewicz, morawietz}@informatik.uni-marburg.de

Abstract

In Bayesian Network Structure Learning (BNSL), we are given a variable set and parent scores for each variable and aim to compute a DAG, called Bayesian network, that maximizes the sum of parent scores, possibly under some structural constraints. Even very restricted special cases of BNSL are computationally hard, and, thus, in practice heuristics such as local search are used. In a typical local search algorithm, we are given some BNSL solution and ask whether there is a better solution within some pre-defined neighborhood of the solution. We study ordering-based local search, where a solution is described via a topological ordering of the variables. We show that given such a topological ordering, we can compute an optimal DAG whose ordering is within inversion distance r in subexponential FPT time; the parameter r allows to balance between solution quality and running time of the local search algorithm. This running time bound can be achieved for BNSL without any structural constraints and for all structural constraints that can be expressed via a sum of weights that are associated with each parent set. We show that for other modification operations on the variable orderings, algorithms with an FPT time for r are unlikely. We also outline the limits of ordering-based local search by showing that it cannot be used for common structural constraints on the moralized graph of the network.

Introduction

Bayesian networks are arguably the most popular and important model for representing dependencies between variables of multivariate probability distributions (Darwiche 2010). A Bayesian network consists of a directed acyclic graph (DAG) D and a set of conditional probability tables, one for each vertex of D. The vertices of the network are the variables of the distribution and each conditional probability table specifies the probability distribution of the corresponding vertex given the values of its parents. Thus, the value of each variable depends directly on the values of its parents. The graph D is called the structure of the network. Given a set of observed data over some variable set N one needs to learn the structure D from this data. This is usually done in two steps. First, for each variable v and each possible set of parents a *parent score* $f_v(P)$ is computed. Roughly speaking, the score represents how useful it is to choose this particular set of parents in order to faithfully represent the probability distribution that created the observed data. Second, given, for each vertex v, a set of possible parent sets with their parent scores, we aim to compute the DAG that maximizes the sum of the parent scores of its vertices. This problem, called BAYESIAN NETWORK STRUCTURE LEARNING (BNSL), is NP-hard (Chickering 1995).

The current-best worst-case running time bound for exact algorithms for BNSL is $2^n \cdot \text{poly}(|I|)$ where n is the number of variables and $|I|$ is the size of the instance. Clearly, this running time is impractical for larger variable sets. Moreover, even very restricted special cases of BNSL remain NP-hard, for example the case where every possible parent set has constant size (Ordyniak and Szeider 2013). Finally, restricting the topology of the DAG to sparse classes such as graphs of bounded treewidth or bounded degree leads to NP-hard learning problems as well (Korhonen and Parviainen 2013, 2015; Grüttemeier and Komusiewicz 2020).

Due to this notorious hardness of BNSL, it is mostly solved using heuristics. One of the most successful heuristic approaches relies on local search (Tsamardinos, Brown, and Aliferis 2006). More precisely, after computing the possible parent sets and their scores, in this approach one starts with an arcless DAG D and adds, removes, or reverses an arc while this results in an increased network score. More recent local search approaches do not use DAGs as solution representations but rather orderings of the variables (Alonso-Barba, de la Ossa, and Puerta 2011; Lee and van Beek 2017; Scanagatta, Corani, and Zaffalon 2017). This approach is motivated by the fact that given an ordering τ of the variables, one may greedily find the optimal DAG D among all DAGs for which τ is a topological ordering.

Our Results. We study different versions of local search on variable orderings. In contrast to previous work that defined the local neighborhood of an ordering as all orderings that can be reached via *one* operation, we consider *parameterized* local search (Fomin et al. 2010; Marx and Schlotter 2010; Gaspers et al. 2012; Fellows et al. 2012; Ordyniak and Szeider 2013). Here, one sets a parameter r and aims to find a better network that can be reached via at most r modifications of the ordering. The hope is that, by considering such

[*]Supported by the Deutsche Forschungsgemeinschaft (DFG), project OPERAH, KO 3669/5-1.

a larger neighborhood, one may avoid being stuck in a bad local optimum. We consider three different kinds of operations in this work. We first study *insertions*, where one may move one variable to an arbitrary position in the ordering and *swaps* where two arbitrary variables may exchange their positions. We observe that the local search problem can be solved in polynomial time for both variants if r is constant. The degree of the polynomial, however, depends on r. We show that, under widely accepted complexity-theoretic assumptions, this dependence cannot be avoided. Afterwards, we consider only swaps of adjacent vertices, we call this operation *inversion*. Our main result is an algorithm with running time $2^{\mathcal{O}(\sqrt{r}\log r)} \cdot \text{poly}(|I|)$ for deciding for a given variable ordering, whether there is a better ordering that can be reached via at most r inversions.

Our algorithms work not only for the standard BNSL problem but also for some structural constraints that we may wish to impose on the final DAG. To formulate algorithms compactly, we introduce a generalization of BNSL, where each possible parent set is associated with a score and a weight and the aim is to find the highest scoring network that does not exceed a specified weight bound. We show that this captures natural types of structural constraints. As a side result, we show that a previous polynomial-time algorithm for acyclic directed superstructures (Ordyniak and Szeider 2013) can be generalized to this new more general problem.

Finally, we show that for using an ordering-based local search approach in the presence of structural constraints it is essentially necessary, that the corresponding variant of BNSL is polynomial-time solvable on instances where the directed super structure is acyclic. This implies for several important structural constraints that ordering-based local search is unlikely to be useful.

Due to lack of space, several proofs are deferred to a full version of this article.

Problem Definition

Notation. We consider directed graphs $D = (N, A)$ which consist of a vertex set N and an arc-set $A \subseteq N \times N$. If D contains no directed cycle, then D is called a *directed acyclic graph (DAG)*. An arc $(u, v) \in A$ is *incoming into* v and *outgoing from* u. A *source* is a vertex without incoming arcs, and a *sink* is a vertex without outgoing arcs. The set $P_v^A := \{u \in N \mid (u, v) \in A\}$ is the *parent set of* v, and for every $u \in P_v^A$, v is called *child of* u. Given a subset $S \subseteq N$ and a vertex v, the vertices in $S \cap P_v^A$ are called the *S-parents of v*. Throughout this work let $n := |N|$. Given $f : X \to Y$ and $X' \subseteq X$, we let $f|_{X'} : X' \to Y$ denote the *limitation of f to X'* which is defined by $f|_{X'}(x) := f(x)$ for every $x \in X'$. We let $[i, j]$ denote the set of integers ℓ such that $i \leq \ell \leq j$.

Orderings. Given a vertex set N, an *ordering of N* is an n-tuple $\tau = (v_1, \dots, v_n)$ containing every vertex of N. For $i \leq n$, we let $\tau(i)$ denote the ith vertex appearing on τ. We write $u <_\tau v$ if the vertex u appears before v on τ. A *partial ordering of τ* is an ordering σ of a subset $S \subseteq N$ such that for all $u, v \in S$ it holds that $u <_\tau v$ if and only if $u <_\sigma v$. Given a vertex set S, we let $\tau[S]$ denote the partial ordering containing exactly the vertices from S, and we let $\tau - S := \tau[N \setminus S]$ denote the partial ordering we obtain when removing all vertices of S from τ. For $i \leq j \leq n$ we define $\tau(i, j)$ as the partial ordering $\tau(i)\tau(i+1)\dots\tau(j)$. Given a partial ordering σ of τ, we let $N(\sigma)$ be the set of all elements appearing on σ. If $\sigma = \tau(i, j)$ we may write $N_\tau(i, j) := N(\tau(i, j))$. Let $D := (N, A)$ be a directed graph. An ordering τ of N is a *topological ordering of D* if $u <_\tau w$ for every arc $(u, w) \in A$. A directed graph has a topological ordering if and only if it is a DAG. A *distance d* is a mapping that assigns an integer to every pair of orderings τ and τ' of N such that $d(\tau, \tau') = d(\tau', \tau)$ and $d(\tau, \tau) = 0$. For an integer r, we say that an ordering τ' is *r-close to τ with respect to d* if $d(\tau, \tau') \leq r$. If τ and d are clear from the context we may only write *τ' is r-close*.

Bayesian Network Structure Learning with Multiweights. Let N be a set of vertices. A mapping $\mathcal{F}$ is a collection of *local multiscores for N* if $\mathcal{F}(v) \subseteq 2^{N \setminus \{v\}} \times \mathbb{N}_0 \times \mathbb{N}_0$ for each $v \in N$. Intuitively, if $(P, s, \omega) \in \mathcal{F}(v)$ for some vertex $v \in N$, then choosing P as the parent set of v may simultaneously give a local score of s and a local weight of ω. Throughout this work we assume that for every v there exists some $s \in \mathbb{N}_0$ such that $(\emptyset, s, 0) \in \mathcal{F}(v)$, that is, every vertex has a local multiscore for the empty parent set with weight zero. Given $v \in N$ and $\mathcal{F}$, the *possible parent sets* are defined by $\mathcal{P}_\mathcal{F}(v) := \{P \mid \exists (P, s, \omega) \in \mathcal{F}(v)\}$. Given N and $\mathcal{F}$, the directed graph $S_\mathcal{F} := (N, A_\mathcal{F})$ with $A_\mathcal{F} := \{(u, v) \mid \exists P \in \mathcal{P}_\mathcal{F}(v) : u \in P\}$ is called the *superstructure* of N and $\mathcal{F}$ (Ordyniak and Szeider 2013).

Definition 1. *Let N be a vertex set, let $\mathcal{F}$ be multiscores for N. An arc-set $A \subseteq N \times N$ is called* $\mathcal{F}$-valid *if (N, A) is a DAG and $P_v^A \in \mathcal{P}_\mathcal{F}(v)$ for all $v \in N$.*

We say that an $\mathcal{F}$-valid arc-set has weight at most k if for every $v \in N$ there is some $(P_v^A, s_v, \omega_v) \in \mathcal{F}(v)$ such that $\sum_{v \in N} \omega_v \leq k$. For a given integer k and an $\mathcal{F}$-valid arc-set A we define $\text{score}_\mathcal{F}(A, k) := \sum_{v \in N} s_v$ as the maximal score one can obtain from any choice of triples $(P_v^A, s_v, \omega_v) \in \mathcal{F}_v, v \in N$, with $\sum_{v \in N} \omega_v \leq k$. If $\mathcal{F}$ is clear from the context we may write $\text{score}(A, k) := \text{score}_\mathcal{F}(A, k)$. We now formally define the problem.

WEIGHTED BAYESIAN NETWORK STRUCTURE LEARNING (W-BNSL)
Input: A set of vertices N, local multiscores $\mathcal{F}$, and two integers $t, k \in \mathbb{N}_0$.
Question: Is there an $\mathcal{F}$-valid arc-set $A \subseteq N \times N$ such that $\text{score}(A, k) \geq t$?

Throughout this work we assume that for an instance $I := (N, \mathcal{F}, t, k)$ of W-BNSL it holds that $k \in \text{poly}(|I|)$. Given an instance $I := (N, \mathcal{F}, t, k)$ of W-BNSL, we call an $\mathcal{F}$-valid arc-set A with $\text{score}(A, k) \geq t$ a *solution* of I.

We consider a version of W-BNSL where one is additionally given an ordering of the vertex set and an integer r and aims to learn a DAG that has a topological ordering that is r-close to the given ordering. Given a distance d, this problem is formally defined as follows.

d-LOCAL W-BNSL
Input: A set of vertices N, local multiscores $\mathcal{F}$, an ordering τ of N, and three integers $t, k, r \in \mathbb{N}$.
Question: Is there an $\mathcal{F}$-valid arc-set A such that score$(A, k) \geq t$ and (N, A) has a topological ordering τ' that is r-close to τ with respect do d?

Given an instance I of d-LOCAL W-BNSL we call an arc-set A *feasible for I* if A is $\mathcal{F}$-valid with weight at most k and (N, A) has a topological ordering that is r-close to τ. A feasible arc-set that maximizes score(A, k) is called a *solution of I*.

W-BNSL generalizes the classical BAYESIAN NETWORK STRUCTURE LEARNING (VANILLA-BNSL). In VANILLA-BNSL one is given a set N of vertices and one local score s for each pair consisting of a vertex $v \in N$ and possible parent set $P \subseteq N \setminus \{v\}$ and the goal is to learn a DAG such that the sum of the local scores is maximal. This problem can be modeled with local multiscores $\mathcal{F}(v)$ containing triples $(P, s, 0)$. Since VANILLA-BNSL is NP-hard (Chickering 1995) and the weights ω are not used for this construction, it follows that W-BNSL is NP-hard even if $k = 0$.

The general problem W-BNSL also allows to model Bayesian network structure learning under additional sparsity constraints: One example for such a constrained version is BOUNDED ARCS-BNSL (BA-BNSL). In BA-BNSL one aims to learn a DAG that contains at most k arcs for some given integer k. This can be modeled with multiscores containing triples $(P, s, |P|)$. BA-BNSL is known to be fixed-parameter tractable when parameterized by the number k of arcs (Grüttemeier and Komusiewicz 2020).

A further example is BOUNDED INDEGREE c-BNSL (BI-c-BNSL) which is defined for every constant c. In BI-c-BNSL one aims to learn a network that contains at most k vertices that have more than c parents for a given integer k. This scenario can be modeled with triples (P, s, ω) where $\omega = 1$ if $|P| > c$ and $\omega = 0$, otherwise. Next, we observe that W-BNSL is solvable in polynomial time if the superstructure is a DAG. This generalizes algorithms for VANILLA-BNSL (Ordyniak and Szeider 2013) and BA-BNSL (Grüttemeier and Komusiewicz 2020).

Theorem 2. W-BNSL *is solvable in polynomial time if $S_{\mathcal{F}}$ is a DAG.*

Proof. Let $I = (N, \mathcal{F}, t, k)$ be an instance of W-BNSL where the superstructure $S_{\mathcal{F}}$ is a DAG and let τ be a topological ordering of $S_{\mathcal{F}}$. We describe a dynamic programming algorithm to solve I. The dynamic programming table T has entries of type $T[i, k']$ with $i \in [1, n+1]$ and $k' \in [0, k]$. Each entry stores the maximal score of an arc-set A of weight at most k' where only the vertices of $N_\tau(i, n)$ are allowed to learn a non-empty parent set and only the local multiscores of the vertices of $N_\tau(i, n)$ count towards the score and weight of A. We start to fill the table T, by setting $T[n+1, k'] := 0$ for all $k' \in [0, k]$. The recurrence to compute an entry for $i \in [1, n]$ and $k' \in [0, k]$ is

$$T[i, k'] := \max_{\substack{(P,s,\omega) \in \mathcal{F}(v) \\ \omega \leq k'}} s + T[i+1, k' - \omega].$$

where $v := \tau(i)$. Thus, to determine if I is a yes-instance of W-BNSL, it remains to check if $T[1, k] \geq t$. The corresponding network can be found via traceback. The formal correctness proof is straightforward and thus omitted.

The dynamic programming table T has $(n+1) \cdot (k+1)$ entries. Each entry can be computed in linear time. Hence, the total running time is $\mathcal{O}(n \cdot k \cdot |I|)$. □

Input Representation. For $N = \{v_1, \ldots, v_n\}$ the local multiscores $\mathcal{F}$ are given as a two-dimensional array $\mathcal{F} := [Q_1, \ldots, Q_n]$, where each Q_i is an array containing a quadruple $(s, \omega, |P|, P)$ for each $(P, s, \omega) \in \mathcal{F}(v)$. The instance size $|I|$ is the number of bits needed to store this two-dimensional array.

Parameterized Complexity. A problem is *slicewise polynomial (XP)* for some parameter k if it can be solved in time $|I|^{f(k)}$ for a computable function f. 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. If a problem is W[1]-hard, then it is assumed to be fixed-parameter *intractable*. For a detailed introduction into parameterized complexity we refer to the standard monographs (Cygan et al. 2015; Downey and Fellows 2013).

Parameterized Local Search for Insert and Swap Distances

A *swap operation* on two vertices v and w on an ordering τ interchanges the positions of v and w. The distance Swap(τ, τ') is the minimum number of swap operations needed to transform τ into τ'. An *insert operation* on an ordering τ removes one arbitrary vertex from τ and inserts it at a new position. We define Insert(τ, τ') as the minimum number of insert operations needed to transform τ into τ'. This number can be computed as Insert$(\tau, \tau') = |N| - \text{LCS}(\tau, \tau')$, where LCS$(\tau, \tau')$ is the length of the longest common subsequence of τ and τ'. That is, if Insert$(\tau, \tau') = r$, then there is a subset $S \subseteq V$ of size r such that $\tau - S = \tau' - S$.

For both distances, local search approaches for BNSL have been studied previously (Alonso-Barba, de la Ossa, and Puerta 2011; Lee and van Beek 2017; Scanagatta, Corani, and Zaffalon 2017). We now focus on the parameterized complexity regarding the parameter r which is the radius of the local search neighborhood. We first prove that there are XP algorithms for Insert-LOCAL W-BNSL and Swap-LOCAL W-BNSL when parameterized by r. That is, both problems are solvable in polynomial time if r is a constant. However, the degree of the polynomial depends on r. Afterwards we show that there is little hope that this algorithm can be improved to a fixed-parameter algorithm by showing that both problems are W[1]-hard when parameterized by r.

Theorem 3. Insert-LOCAL W-BNSL *and* Swap-LOCAL W-BNSL *are solvable in $n^{\mathcal{O}(r)} \cdot \text{poly}(|I|)$ time.*

Theorem 4. Insert-Local W-BNSL *and* Swap-Local W-BNSL *are* W[1]*-hard when parameterized by r even if $k = 0$, $|\mathcal{F}(v)| \leq 2$ for all $v \in N$, $S_{\mathcal{F}}$ is a DAG, and every potential parent set has size at most two.*

Proof. We describe a parameterized reduction from the Clique-problem which is W[1]-hard when parameterized by k (Downey and Fellows 2013).

Given an instance $I = (G = (V, E), k)$ of Clique where G has n vertices and m edges, we compute an equivalent instance $I' = (N, \mathcal{F}, \tau, t, k', r)$ of Insert-Local W-BNSL in polynomial time. Let $V = \{v_1, \ldots, v_n\}$, and let $E = \{e_1, \ldots, e_m\}$. We start with an empty set of vertices N and add for every edge $e_i \in E$ the vertices w_i and $\{w_i^1, \ldots, w_i^k\}$. Moreover, we add the vertices of V to N and k additional vertices $x_1, \ldots, x_k$.

For every $e_i \in E$, we set $\mathcal{F}(w_i) := \{(e_i, k, 0)\}$ and $\mathcal{F}(w_i^j) := \{(\{w_i\}, n^9, 0)\}$ with $j \in [1, k]$. Further, we set $\mathcal{F}(x_i) := \{(\{x_{i-1}\}, n^9, 0)\}$ for all $i \in [2, k]$ and $\mathcal{F}(v) := \{(\{x_k\}, \binom{k}{2} - 1, 0)\}$ for every vertex $v \in V$. Moreover, for each $v \in N$, we also add $(\emptyset, 0, 0)$ to $\mathcal{F}(v)$. Note that $S_{\mathcal{F}}$ is a DAG.

For each $j \in [1, m]$, we set $\tau_j := (w_j, w_j^1, w_j^2, \ldots, w_j^k)$ and $\tau := \tau_1 \cdot \tau_2 \cdot \ldots \cdot \tau_m \cdot (x_1, \ldots, x_k, v_1, \ldots, v_n)$.

Finally, we set $r := k$, $k' := 0$, and $t := (mk+k-1)n^9 + n(\binom{k}{2} - 1) + k$ which completes the construction of I'.

Since any parent set has weight zero, we abbreviate in the following $\text{score}(X) := \text{score}(X, k')$ for every $X \subseteq N \times N$. Let $A^* := \{(w_i, w_i^j) \mid e_i \in E, 1 \leq j \leq k\} \cup \{(x_{i-1}, x_i) \mid 2 \leq i \leq k\}$ be the set of arcs for parent sets of score n^9 and let $\hat{A} := A^* \cup \{(x_k, v_i) \mid v_i \in V\}$ be the set of all arcs of parent sets with positive score that do not violate the topological ordering τ. By construction, $\text{score}(\hat{A}) = t - k$.

The idea is that every arc-set A of score at least t has to contain all the arcs of A^*. Moreover, if $D = (N, A)$ has a topological ordering which is r-close to τ, then at most r of the vertices of N change their position. Intuitively, these should be the vertices of the clique S in G such that all parent sets can be learned that represent edges between the vertices of S in G. The vertices of S should be inserted at the beginning of the new ordering of N such that w_i can learn the parent set e_i for each $e_i \in E_G(S)$.

Claim 1. *I is a yes-instance of* Clique *if and only if I' is a yes-instance of* Insert-Local W-BNSL.

Next, we describe how we can modify this construction to obtain the hardness result for Swap-Local W-BNSL. We add k additional vertices $y_1, \ldots, y_k$ with $\mathcal{F}_2(y_j) := \{(\emptyset, 0, 0), (\{x_k\}, n^8, 0)\}$ for each $j \in [1, k]$ and $\mathcal{F}_2(v) := \mathcal{F}(v)$ for each $v \in N$. Moreover, we set $\sigma = (y_1, \ldots, y_k) \cdot \tau$ and $t_2 := t + k \cdot n^8$. Next, we show that I' is a yes-instance of Insert-Local W-BNSL if and only if $I_2 := (N_2, \mathcal{F}_2, \sigma, t_2, k', r)$ is a yes-instance of Swap-Local W-BNSL. By construction and the above argumentation, an arc-set A has score at least $t_2 - k = \text{score}(\hat{A}) + k \cdot n^8$ if and only if $A^* \cup \{(x_k, y_j) \mid 1 \leq j \leq k\} \subseteq A$. Consequently, each swap operation has to swap a different vertex from $\{y_1, \ldots, y_k\}$ with a vertex which is later in the current topological ordering than x_k, that is, with a vertex of V. Let S be the vertices of V that are swapped with the vertices of $\{y_1, \ldots, y_k\}$. Then, $\text{score}(A) \geq t_2$ if and only if S forms a clique in G due to the equivalence between I and I'. □

Parameterized Local Search for Inversions Distance

In this section we study parameterized local search for the *inversions* distance. An inversion on an ordering is an operation that swaps the positions of two consecutive vertices of the ordering.

We describe a randomized algorithm to solve Inv-Local W-BNSL. The algorithm has constant error probability and runs in subexponential FPT time for r, the radius of the local search neighborhood. The algorithm is closely related to a parameterized local search algorithm for Feedback Arc Set in Tournaments (FAST) (Fomin et al. 2010). In FAST the input is an arc-weighted directed graph where between every pair u, v of vertices either (u, v) or (v, u) is present. The task is to delete a minimum-weight set of arcs such that the remaining graph is acyclic. Note that FAST can be modeled as a special case of W-BNSL. The local search neighborhood considered by Fomin et al. (2010) is the number of arc-flips, which correspond to our inversions in the topological ordering. With our algorithm we generalize the local search algorithm for FAST and show that we can use it to obtain a local search algorithm for W-BNSL.

We first introduce some formalism of inversions. Let $\tau = (v_1, \ldots, v_n)$ be an ordering of a set N. An *inversion on position* $i \in \{1, n-1\}$ transforms τ into the ordering $h_i(\tau) := (v_1, \ldots, v_{i-1}, v_{i+1}, v_i, \ldots, v_n)$. A *sequence of inversions* $S = (s_1, \ldots, s_\ell)$ is a finite sequence of elements in $[1, n-1]$. Applying S on τ transforms τ into the ordering $S(\tau) := h_{s_\ell} \circ h_{s_{\ell-1}} \circ \cdots \circ h_{s_1}(\tau)$. The distance $\text{Inv}(\tau, \tau')$ is the length of the shortest sequence of inversions S such that $S(\tau) = \tau'$.

Our algorithm is based on the following intuition: If an ordering τ' is r-close to τ, there are only some windows of length at most r in τ such that the vertices inside these windows change their positions. The remaining vertices keep their positions. We exploit this and provide an algorithm that finds orderings for the vertices of these windows by solving small sub-instances of Inv-Local W-BNSL. We then combine the solutions to obtain a solution for the whole instance.

We describe the algorithm in three steps: First, we describe a randomized algorithm that solves Inv-Local W-BNSL in $n^{\mathcal{O}(\sqrt{r})} \cdot \text{poly}(|I|)$ time. Second, we argue how we may combine the solutions of the sub-instances. Third, we make use of the results from the two previous subsections and provide an FPT algorithm that solves Inv-Local W-BNSL. More precisely, we show that we can compute the solution of an instance of Inv-Local W-BNSL by solving polynomially many sub-instances with at most r vertices and combining their solutions.

A Randomized Algorithm for Inv-Local W-BNSL

We first provide a randomized algorithm that solves Inv-Local W-BNSL in $n^{\mathcal{O}(\sqrt{r})} \cdot \text{poly}(|I|)$ time with a constant error probability. The key idea behind the algorithm is as follows: If we are given a set of suborderings of τ which do not change their relative positions in the resulting network, then we have a limited space of possible orderings which can be obtained with r inversions. Our algorithm is based on color coding (Alon, Yuster, and Zwick 1995) and runs in subexponential time in r. The concrete technique is closely related to a subexponential time algorithm for Feedback Arc Set in Tournaments (Alon, Lokshtanov, and Saurabh 2009). Intuitively, we randomly color all vertices with $\mathcal{O}(\sqrt{r})$ colors in a way that vertices of the same color keep their relative positions.

Before we describe the algorithm we need some definitions: Let N be a set of vertices. A function $\chi : N \to [1, \ell]$ is called a *coloring (of N with ℓ colors)*. For each color $i \in [1, \ell]$, we call $Z^i := \{v \in N \mid \chi(v) = i\}$ the *color class of i*. We next define colorful valid arc-sets and colorful solutions which are important for the color coding algorithm.

Definition 5. *Let $I := (N, \mathcal{F}, \tau, k, r)$ be an instance of* Inv-Local W-BNSL, *let $\chi : N \to [1, \ell]$ be a coloring, and let A be an $\mathcal{F}$-valid arc-set. We say that A is a* colored arc-set *if there is a topological ordering τ' of (N, A) such that $\text{Inv}(\tau, \tau') \le r$ and $\tau[Z^i] = \tau'[Z^i]$ for every color class Z^i. A colored arc-set A that maximizes* $\text{score}(A, k)$ *is called a* colored solution *of I and χ.*

We next describe a deterministic algorithm that efficiently finds a colored solution.

Proposition 6. *Given an instance $I := (N, \mathcal{F}, \tau, k, r)$ of* Inv-Local W-BNSL *and a coloring $\chi : N \to [1, \ell]$ for I, a colored solution A can be computed in $n^\ell \cdot \text{poly}(|I|)$ time.*

Proof. We describe a dynamic programming algorithm. To this end, we first introduce some notation. For every color class Z^i consider the sub-ordering $\tau[Z^i]$ that contains only the vertices of Z^i. Given some integer $x \le |Z^i|$ we define $z_i(1), z_i(2), \dots, z_i(x)$ as the first x vertices on $\tau[Z^i]$ and set $Z^i_x := \{z_i(1), z_i(2), \dots, z_i(x)\}$. Note that $Z^i_0 = \emptyset$. Given an integer vector $\vec{p} = (p_1, \dots, p_\ell)$ with $p_i \in [0, |Z^i|]$, we define $\tau(\vec{p}) := \tau[Z^1_{p_1} \cup Z^2_{p_2} \cup \dots \cup Z^\ell_{p_\ell}]$. As a shorthand, for the set of vertices appearing on $\tau(\vec{p})$ we set $N(\vec{p}) := N(\tau(\vec{p}))$ and we define $\mathcal{F}_{\vec{p}}$ by $\mathcal{F}_{\vec{p}}(v) := \{(P, s, \omega) \in \mathcal{F}(v) \mid P \subseteq N(\vec{p})\}$ for every $v \in N(\vec{p})$ as the *limitation of $\mathcal{F}$ to $N(\vec{p})$*. Note that, given a partial ordering $\tau(\vec{p})$, the vertex $z_i(p_i)$ is the last vertex of color class Z^i appearing on $\tau(\vec{p})$. Throughout this proof we let $\vec{0}$ denote the integer vector of length ℓ where all entries equal zero, and $\vec{e}_i$ the integer vector of length ℓ where the ith entry equals one and all other entries equal zero.

Every color class can be seen as a chain of vertices that keep their relative position in the ordering. A vector $\vec{p}$ describes which prefixes of these chains we consider. Intuitively, our dynamic programming algorithm starts with empty chains and then recursively adds the next vertex of one of the chains and finds a solution of this instance.

Formally, the dynamic programming table T has entries of the type $T[\vec{p}, k', r']$ with $k' \in [0, k]$ and $r' \in [0, r]$. Each entry stores the score of a colored solution of the Inv-Local W-BNSL instance

$$I^{\vec{p}}_{k',r'} := (N(\vec{p}), \mathcal{F}_{\vec{p}}, \tau(\vec{p}), t, k', r').$$

The idea behind this algorithm is to recursively find the best sink of the current network and combine this with a colored solution of the remaining network. To specify the contribution of a sink to the score, we introduce the following definition: For given $i \in [0, \ell]$, $k' \in [0, k]$, and $\vec{p}$, we define the value $f_{\vec{p}}(i, k')$ as the maximal local score of a parent set $P \subseteq N(\vec{p})$ of $z_i(p_i)$ that simultaneously has weight at most k'. More formally,

$$f_{\vec{p}}(i, k') := \max_{\substack{(P,s,\omega) \in \mathcal{F}(z_i(p_i)) \\ P \subseteq N(\vec{p}) \\ \omega \le k'}} s.$$

The value of $f_{\vec{p}}(i, k')$ can be computed in $\text{poly}(|I|)$ time by iterating over the array representing $\mathcal{F}(v)$.

We next describe how to fill the dynamic programming table. As base case we set $T[\vec{0}, k', r'] := 0$ for all $k' \in [0, k]$ and $r' \in [0, r]$. The recurrence to compute entries for $\vec{p} \neq \vec{0}$ is

$$T[\vec{p}, k', r'] := \max_{k'' \le k'} \max_{\substack{i,\, p_i > 0 \\ R(\vec{p}, i) \le r'}} \Big(f_{\vec{p}}(i, k'') + T[\tau(\vec{p} - \vec{e}_i), k' - k'', r' - R(\vec{p}, i)] \Big),$$

where $R(\vec{p}, i) := |\{v \in N(\vec{p}) \mid z_i(p_i) <_{\tau(\vec{p})} v\}|$ is the number of elements that appear after $z_i(p_i)$ in $\tau(\vec{p})$. The score of a colored solution of I and χ can be computed by evaluating $T[(|Z^1|, |Z^2|, \dots, |Z^\ell|), k, r]$. The corresponding network can be found via traceback. We next show that the dynamic programming recurrence is correct.

Claim 2. *For each $\vec{p}$, k', and r' it holds that $T[\vec{p}, k', r']$ is the score of a colored solution of $I^{\vec{p}}_{k',r'}$ and $\chi|_{N(\vec{p})}$.*

We next consider the running time of the dynamic programming algorithm. The table T has $\mathcal{O}(n^\ell \cdot (k+1) \cdot (r+1))$ entries. Each entry can be computed in time polynomial in $|I|$. Thus, the algorithm runs in $n^\ell \cdot \text{poly}(|I|)$ time. $\square$

We now describe how to use the algorithm behind Proposition 6 to obtain a randomized algorithm for Inv-Local W-BNSL. We randomly color the vertices with $\sqrt{8r}$ colors[1] and use the algorithm from Proposition 6 to find a colored solution. Given an instance I and a coloring χ, we say that χ is a *good coloring of I* if every colored solution of χ and I is an (uncolored) solution of I. We next analyze the likelihood of randomly choosing a good coloring.

Lemma 7. *Let I be an instance of* Inv-Local W-BNSL *and let $\chi : N \to [1, \sqrt{8r}]$ be a coloring that results from randomly uniformly assigning a color to each vertex. Then, the probability that χ is a good coloring of I is at least $(2e)^{-\sqrt{r/8}}$.*

[1]Without loss of generality, assume that $\sqrt{8r}$ is an integer.

Applying the algorithm behind Proposition 6 with random colorings $(2e)^{\sqrt{r/8}}$ times gives the following result.

Proposition 8. *There exists a randomized algorithm for* Inv-Local W-BNSL *that, in time* $(2e)^{\sqrt{r/8}} \cdot n^{\sqrt{8r}} \cdot \text{poly}(|I|)$ *returns an* $\mathcal{F}$*-valid arc-set* A *with weight at most* k*. With probability at least* $1 - \frac{1}{e}$*, the set* A *is a solution.*

Combining and Splitting Solutions

Recall that we want to use the algorithm behind Proposition 8 as a subroutine and apply it on small sub-instances of Inv-Local W-BNSL. In this subsection we provide technical lemmas which specify how we may combine (Lemma 10) and split (Lemma 14) solutions of Inv-Local W-BNSL-instances. These lemmas are the main technical tools that we need to show correctness of the FPT algorithm in the next subsection.

The sub-instances contain the vertices of a *window* $\tau(a, b)$. Changing the ordering in $\tau(a, b)$, these vertices may learn a parent set containing vertices from $\tau(1, b)$. For our purpose, only the new parents in $\tau(a, b)$ are important, hence we may hide the parents in $\tau(1, a - 1)$. We next give a formal definition of these sub-instances.

Definition 9. *Let* $I := (N, \mathcal{F}, \tau, k, r)$ *be an instance of* Inv-Local W-BNSL*. Moreover, let* $k' \leq k$*,* $r' \leq r$*,* $a, b \in [1, n]$ *with* $a \leq b$*. A* window instance *is then defined as* $B^{a,b}_{k',r'} := (N_\tau(a, b), \mathcal{F}|_{N_\tau(a,b)}, \tau(a, b), k', r')$*, where the multiscores* $\mathcal{F}|_{N_\tau(a,b)}$ *are defined by* $\mathcal{F}|_{N_\tau(a,b)}(v) := \{(P \cap N_\tau(a, b), s, \omega) \mid (P, s, \omega) \in \mathcal{F}(v) \text{ and } P \subseteq N_\tau(1, b)\}$*.*

The following lemma formalizes how we may combine solutions of window instances.

Lemma 10. *Let* $I := (N, \mathcal{F}, \tau, k, r)$ *be an instance of* Inv-Local W-BNSL*, let* $a \in [1, n - 1]$*, let* $k', k'' \in [0, k]$ *with* $k' + k'' \leq k$*, let* $r', r'' \in [0, r]$ *with* $r' + r'' \leq r$*, and let* $N_1 := N_\tau(1, a)$ *and* $N_2 := N \setminus N_1 = N_\tau(a + 1, n)$*. Moreover, let* A_1 *be a solution of* $B^{1,a}_{k',r'}$ *and let* A_2 *be a solution of* $B^{a+1,n}_{k'',r''}$*. Then, there exists an arc-set* $A \subseteq N \times N$ *that is feasible for* I *with*

$$score_{\mathcal{F}}(A, k' + k'') \geq \text{score}_{\mathcal{F}|_{N_1}}(A_1, k') + \text{score}_{\mathcal{F}|_{N_2}}(A_2, k'').$$

We next focus on how to find the instances that we solve during the fixed-parameter algorithm. That is, we find positions where we may split a given solution of an instance into two solutions of window instances. To this end, we introduce the concept of *borders*.

Definition 11. *Let* N *be a set of vertices and let* τ, τ' *be two orderings of* N*. A position* $i \in [1, n]$ *is a* border of τ and τ' *if there exists a sequence* S *of inversions transforming* τ *into* τ' *such that* $|S|$ *is minimal and* i *does not occur on* S*.*

Obviously, the position n is a trivial border for all orderings τ and τ' of N. Moreover, if $i < n$ is a border of τ it holds that $N_\tau(1, i) = N_{\tau'}(1, i)$ and $N_\tau(i + 1, n) = N'_\tau(i + 1, n)$. Intuitively, a border is a position that splits τ and τ' into two parts such that the corresponding parts of the orderings contain the same vertices. We next define borders of instances of Inv-Local W-BNSL.

Definition 12. *Let* $I := (N, \mathcal{F}, \tau, k, r)$ *be an instance of* Inv-Local W-BNSL*. We say that* $i \in [1, n]$ *is a* border of I *if there exist a solution* A *of* I *and a topological ordering* τ' *of* (N, A) *with* $\text{Inv}(\tau, \tau') \leq r$ *such that* i *is a border of* τ *and* τ'*.*

We next prove two lemmas that are important for the correctness of our FPT algorithm. The first lemma states that for any position j in an ordering τ there exists a border i that is close to j.

Lemma 13. *Let* $I := (N, \mathcal{F}, \tau, k, r)$ *be an instance of* Inv-Local W-BNSL*. Then, for every* $j \in [1, n]$ *there exists an* $i \in [j, \min(n, j + r)]$ *such that* i *is a border of* I*.*

Proof. Since n is always a border, the statement trivially holds if $n \leq j + r$. Let $j + r < n$. Observe that $[j, j + r]$ contains $r+1$ elements. Let A be a solution of I and let τ' be a topological ordering of (N, A) with inversions$(\tau, \tau') \leq r$. Furthermore, let S be a minimal sequence of inversions that transforms τ into τ'. Then, $\text{Inv}(\tau, \tau') \leq r$ implies $|S| \leq r$. Hence, there exists some $i \in [j, j + r]$ that does not occur on S. Consequently, i is a border of τ and τ' and therefore, i is a border of I. □

The next lemma formalizes how we may split solutions.

Lemma 14. *Let* $I := (N, \mathcal{F}, \tau, k, r)$ *be an instance of* Inv-Local W-BNSL *that has a border at position* $i \in [1, n - 1]$*, and let* A *be a solution of* I*. Then, there exist* $r' \leq r$*,* $k' \leq k$ *and arc-sets* A_1 *and* A_2 *such that*

a) A_1 *is feasible for* $B^{1,i}_{r',k'}$*,*

b) A_2 *is feasible for* $B^{i+1,n}_{r-r',k-k'}$*, and*

c) $\text{score}(A, k) \leq \text{score}_{\mathcal{F}|_{N_1}}(A_1, k') + \text{score}_{\mathcal{F}|_{N_2}}(A_2, k-k')$*, where* $N_1 := N_\tau(1, i)$*, and* $N_2 := N_\tau(i + 1, n)$*.*

An FPT-algorithm for Inv-Local W-BNSL

We now provide a randomized fixed-parameter algorithm for Inv-Local W-BNSL when parameterized by the search radius r. The algorithm is based on dynamic programming and uses the observations from the previous two subsections.

Theorem 15. *There exists a randomized algorithm for* Inv-Local W-BNSL *that, in time* $2^{\mathcal{O}(\log(r)\sqrt{r})} \cdot \text{poly}(|I|)$*, returns an* $\mathcal{F}$*-valid arc-set with weight at most* k*. With probability at least* $1 - \frac{1}{e}$*, the returned arc-set is a solution.*

Proof. Let $I := (N, \mathcal{F}, \tau, k, r)$ be an instance of Inv-Local W-BNSL. We describe the algorithm in two steps. We first describe a deterministic algorithm that finds a solution of I in polynomial time when using an oracle that gives solutions of window instances with at most r vertices. Afterwards, we describe how to replace the oracle evaluations with the randomized algorithm from Proposition 8 to obtain a randomized algorithm for Inv-Local W-BNSL with the claimed running time and error probability.

The oracle algorithm. We fill a dynamic programming table T that has entries of the type $T[j, k', r']$ with $j \in [1, n+1], k' \in [0, k]$, and $r' \in [0, r]$. The idea is that each entry $T[j, k', r']$ with $j \leq n$ stores the score of a solution of the window instance $B^{j,n}_{k',r'}$.

We next describe how to fill T. As base case we set $T[n+1, k', r'] = 0$ for all k' and r'. The recurrence to compute an entry for $j \leq n$ is

$$T[j, k', r'] := \max_{p \in [j, \min(n, j+r)]} \max_{\substack{k'' \leq k' \\ r'' \leq r'}} \Big(\mathrm{Val}(B^{j,p}_{k'',r''}) + T[p+1, k'-k'', r'-r''] \Big),$$

where $\mathrm{Val}(B^{j,p}_{k'',r''})$ denotes the score of a solution of $B^{j,p}_{k'',r''}$. Observe that all of these window instances have at most r vertices. The score of a solution of I can be computed by evaluating $T[1, k, r]$. The ordering of the corresponding network can be found via traceback. We next show that the dynamic programming algorithm is correct.

Claim 3. *For each $j \in [1, n]$, $k' \in [0, k]$, and $r' \in [0, r]$ it holds that $T[j, k', r']$ is the score of a solution of the window instance $B^{j,n}_{k',r'}$.*

The correctness of Claim 3 relies on Lemmas 10 and 14.

Replacing the Oracle Evaluations. The dynamic programming table has $(n+1)(k+1)(r+1)$ entries. Each entry can be computed by using at most $r^2 k$ oracle evaluations. Thus, there are at most $x := (n+1)(k+1)(r+1) \cdot r^2 \cdot k \in \mathrm{poly}(|I|)$ oracle evaluations. We replace every oracle evaluation by applying the algorithm behind Proposition 8 exactly x times and keeping a result with maximal score.

Observe that the algorithm always computes a feasible arc-set for I. The probability that the correct result of one oracle evaluation is returned is at least $1 - \frac{1}{e^x}$. Consequently, the success probability of the algorithm is at least $(1 - \frac{1}{e^x})^x \geq (1 - \frac{1}{e})$. The inequality holds since we have equality in the case of $x = 1$ and the left hand side of the inequality strictly increases when $x \geq 1$ increases.

We next analyze the running time of the algorithm. As mentioned above, the dynamic programming table has $(n+1)(k+1)(r+1) \in \mathrm{poly}(|I|)$ entries. For each entry we apply the algorithm from Proposition 8 on x window instances with at most r vertices. Since $x \in \mathrm{poly}(|I|)$ we have a total running time of $2^{\mathcal{O}(\log(r)\sqrt{r})} \cdot \mathrm{poly}(|I|)$. □

Limits of Ordering-Based Local Search

As mentioned above, W-BNSL can be used to model Bayesian network learning under additional sparsity constraints like a bounded number of arcs. However, some important sparsity constraints cannot be modeled with our framework, for example sparsity constraints that are posed on the moralized graph (Elidan and Gould 2008): The *moralized graph of a DAG D* is an undirected graph $\mathcal{M}(D) := (V, E_1 \cup E_2)$ with $V := N$, $E_1 := \{\{u, v\} \mid (u, v) \in A\}$, and $E_2 := \{\{u, v\} \mid u \text{ and } v \text{ have a common child in } D\}$. The edges in E_2 are called *moral edges*.

Version	Constraint on $\mathcal{M}(D)$
B-EDGES-BNSL	bounded number of edges
B-VC-BNSL	bounded vertex cover size
B-DN-BNSL	bounded dissociation number
B-TW-BNSL	bounded treewidth

Table 1. Further sparsity constraints.

Table 1 displays versions of BNSL under sparsity constraints for the moralized graph. Since all these problem versions are NP-hard even when restricted to instances where the superstructure is acyclic (Korhonen and Parviainen 2013, 2015; Grüttemeier and Komusiewicz 2020), it is unlikely that they can be modeled in our framework since W-BNSL is solvable in polynomial time on such instances.

We now argue that there is little hope that one can efficiently find improvements of a given DAG by applying changes on its topological ordering. For all problem versions in Table 1 it is NP-hard to learn a DAG of a given score t even when $S_{\mathcal{F}}$ is a DAG and t is polynomial in n and all local scores are integers (Korhonen and Parviainen 2013, 2015; Grüttemeier and Komusiewicz 2020). Furthermore, observe that all the constraints on $\mathcal{M}(D)$ from Table 1 are true if D does not contain arcs. Assume one can find improvements of a given DAG in polynomial time if the radius r of the local search neighborhood is constant. Then, we can solve instances with acyclic superstructure by setting $r = 0$, starting with an empty DAG and a topological ordering of $S_{\mathcal{F}}$, and improve the score t times. This would be a polynomial time algorithm for instances where t is polynomial in n and $S_{\mathcal{F}}$ is a DAG which would imply P = NP.

Conclusion

To assess whether parameterized local search is in principle a viable approach to ordering-based BNSL, we performed some preliminary experiments for VANILLA-BNSL. Given an instance, we compute 20 random topological orderings. For each such ordering, we repeat the following two steps until no further improvement was found: First, apply single insert operations until no further improvement can be found. Afterwards, slide a window of given size r over the ordering and find an optimal ordering for this window. The sliding window can be seen as a permissive local search for the r-inversion neighborhood. We ran experiments for each $r \in \{3, 5, 7, 9, 11\}$ using the same 20 random orderings for a fair comparison. We used the data sets provided at the GOBNILP (Cussens and Bartlett 2013) homepage.[2] For most of the instances the best results are obtained for $r \in \{9, 11\}$; we remark that the running time bottleneck in our preliminary experiments was not the combinatorial explosion in r but rather the slow implementation of the insert operation. Overall, it seems promising to consider parameterized local search for the number of inversions. In the future, we plan to evaluate the practical usefulness of the approach further by studying it extensively from an algorithm engineering perspective.

[2]https://www.cs.york.ac.uk/aig/sw/gobnilp/

References

Alon, N.; Lokshtanov, D.; and Saurabh, S. 2009. Fast FAST. In *Proceedings of the Thirty-Sixth International Colloquium on Automata, Languages and Programming (ICALP '09)*, volume 5555 of *Lecture Notes in Computer Science*, 49–58. Springer.

Alon, N.; Yuster, R.; and Zwick, U. 1995. Color-Coding. *Journal of the ACM* 42(4): 844–856.

Alonso-Barba, J. I.; de la Ossa, L.; and Puerta, J. M. 2011. Structural learning of Bayesian networks using local algorithms based on the space of orderings. *Soft Computing* 15(10): 1881–1895.

Chickering, D. M. 1995. Learning Bayesian Networks is NP-Complete. In *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics (AISTATS '95)*, 121–130. Springer.

Cussens, J.; and Bartlett, M. 2013. Advances in Bayesian Network Learning using Integer Programming. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, (UAI'13)*. AUAI Press.

Cygan, M.; Fomin, F. V.; Kowalik, L.; Lokshtanov, D.; Marx, D.; Pilipczuk, M.; Pilipczuk, M.; and Saurabh, S. 2015. *Parameterized Algorithms*. Springer.

Darwiche, A. 2010. Bayesian networks. *Communications of the ACM* 53(12): 80–90.

Downey, R. G.; and Fellows, M. R. 2013. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer.

Elidan, G.; and Gould, S. 2008. Learning Bounded Treewidth Bayesian Networks. In *Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems (NIPS '08)*, 417–424. Curran Associates, Inc.

Fellows, M. R.; Fomin, F. V.; Lokshtanov, D.; Rosamond, F. A.; Saurabh, S.; and Villanger, Y. 2012. Local search: Is brute-force avoidable? *Journal of Computer and System Sciences* 78(3): 707–719.

Fomin, F. V.; Lokshtanov, D.; Raman, V.; and Saurabh, S. 2010. Fast Local Search Algorithm for Weighted Feedback Arc Set in Tournaments. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI '10)*. AAAI Press.

Gaspers, S.; Kim, E. J.; Ordyniak, S.; Saurabh, S.; and Szeider, S. 2012. Don't Be Strict in Local Search! In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI '12)*. AAAI Press.

Grüttemeier, N.; and Komusiewicz, C. 2020. Learning Bayesian Networks Under Sparsity Constraints: A Parameterized Complexity Analysis. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI '20)*, 4245–4251. ijcai.org.

Korhonen, J. H.; and Parviainen, P. 2013. Exact Learning of Bounded Tree-width Bayesian Networks. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics (AISTATS '13)*, 370–378. JMLR.org.

Korhonen, J. H.; and Parviainen, P. 2015. Tractable Bayesian Network Structure Learning with Bounded Vertex Cover Number. In *Proceedings of the Twenty-Eighth Annual Conference on Neural Information Processing Systems (NIPS '15)*, 622–630. MIT Press.

Lee, C.; and van Beek, P. 2017. Metaheuristics for Score-and-Search Bayesian Network Structure Learning. In *Proceedings of the Thirtieth Canadian Conference on Artificial Intelligence (CCAI '17)*, volume 10233 of *Lecture Notes in Computer Science*, 129–141. Springer.

Marx, D.; and Schlotter, I. 2010. Parameterized Complexity and Local Search Approaches for the Stable Marriage Problem with Ties. *Algorithmica* 58(1): 170–187.

Ordyniak, S.; and Szeider, S. 2013. Parameterized Complexity Results for Exact Bayesian Network Structure Learning. *Journal of Artificial Intelligence Research* 46: 263–302.

Scanagatta, M.; Corani, G.; and Zaffalon, M. 2017. Improved Local Search in Bayesian Networks Structure Learning. In *Proceedings of the Third Workshop on Advanced Methodologies for Bayesian Networks (AMBN '17)*, volume 73 of *Proceedings of Machine Learning Research*, 45–56. PMLR.

Tsamardinos, I.; Brown, L. E.; and Aliferis, C. F. 2006. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning* 65(1): 31–78.