On Finding Optimal Polytrees[☆]

Serge Gaspers[a], Mikko Koivisto[b], Mathieu Liedloff[c], Sebastian Ordyniak[d], Stefan Szeider[e]

[a] *The University of New South Wales and NICTA*
[b] *University of Helsinki*
[c] *Université d'Orléans*
[d] *Masaryk University, Botanick 68a, 60200 Brno, Czech Republic, Tel. +420549495699*
[e] *Institute of Information Systems, Vienna University of Technology*

Abstract

We study the NP-hard problem of finding a directed acyclic graph (DAG) on a given set of nodes so as to maximize a given scoring function. The problem models the task of inferring a probabilistic network from data, which has been studied extensively in the fields of artificial intelligence and machine learning. Several variants of the problem, where the output DAG is constrained in several ways, are NP-hard as well, for example when the DAG is required to have bounded in-degree, or when it is required to be a polytree. Polynomial-time algorithms are known only for rare special cases, perhaps most notably for branchings, that is, polytrees in which the in-degree of every node is at most one.

In this paper, we generalize this polynomial-time result to polytrees that can be turned into a branching by deleting a constant number of arcs. Our algorithm stems from a matroid intersection formulation. As the order of the polynomial time bound depends on the number of deleted arcs, the algorithm does not establish fixed-parameter tractability when parameterized by that number. We show that certain additional constraints on the sought polytree render the problem fixed-parameter tractable. We contrast this positive result by showing that if we parameterize by the number of deleted nodes, a somewhat more powerful parameter, the problem is not fixed-parameter tractable, subject to a complexity-theoretic assumption.

Keywords: Directed acyclic graphs, branchings, polytrees, parameterized complexity, matroids, probabilistic networks

1. Introduction

There has been extensive research on learning probabilistic networks from data by maximizing some suitable scoring function. In this setting, we are given a set of nodes and a scoring function that assigns a real number to every directed graph (digraph) on the node set, and the goal is to find (or construct) a directed *acyclic* graph (DAG) that maximizes the score. As the problem is NP-hard in general [2], researchers have sought for natural problem variants that would be computationally tractable. One approach to this end is to tighten the combinatorial constraints for the sought DAG. This approach has two main advantages: (1) the found more constrained DAG provides a lower bound on the optimum score over less constrained DAGs, which can be used to guide the search in the less constrained space; and (2) a more constrained DAG often simplifies the reasoning tasks that will be performed on the learned probabilistic network. It is important to note that the constraints in question restrict the output of the problem, not the input. Thus,

[☆]This paper appeared in a preliminary and shortened form in the Proceedings of AAAI 2012, the 26th Conference on Artificial Intelligence [1].

Email addresses: sergeg@cse.unsw.edu.au (Serge Gaspers), mikko.koivisto@cs.helsinki.fi (Mikko Koivisto), mathieu.liedloff@univ-orleans.fr (Mathieu Liedloff), sordyniak@gmail.com (Sebastian Ordyniak), stefan@szeider.net (Stefan Szeider)

tighter constraints do not necessarily amount to easier problem variants. This makes a systematic search for positive and negative results more challenging than in the case where the restrictions concern the input.

The first positive result in this direction was given by Edmonds [3], who gave an efficient algorithm for the class of *branchings*. A branching is a directed forest with in-degree at most one, equivalently, a DAG with in-degree at most one. The algorithm was discovered independently by Chu and Liu [4], and it has been later simplified and expedited by others [5, 6, 7, 8, 9, 10, 11]. On the other hand, Chickering [2] showed—by a reduction from the Feedback Arc Set problem—that the problem becomes NP-hard as soon as the in-degree bound on the DAG is raised to two (or any larger constant). Motivated by this gap, Dasgupta [12] asked for a class of DAGs that is more general than branchings yet admitting provably good structure-learning algorithms. In particular, he studied *polytrees*, that is, digraphs whose underlying undirected graph is a tree. The results were, however, rather negative, showing that the optimization problem is NP-hard when the in-degree bound is two (or any larger constant).

Given the recent advances in exact exponential algorithms in general (see, e.g., the dedicated book [13]), and in finding optimal DAGs in particular, it is natural to ask, whether "fast" exponential-time algorithms exist for finding optimal polytrees. For unconstrained DAGs the fastest known algorithms run in time within a polynomial factor of 2^n, where n is the number of nodes [14, 15, 16, 17]. Currently, we do not know whether these bounds can be achieved for polytrees—a brute-force algorithm would visit each polytree one by one, whose number scales as the number of directed labeled trees $n^{n-2}2^{n-1}$ [18]. Do significantly faster algorithms exist? What if we restrict our search for DAGs that are "almost branchings": Does the problem become easier if only a small number of nodes are allowed an in-degree larger than one?

Results

This article takes a step towards answering these questions by considering polytrees that differ from branchings by only a few arcs. More precisely, we study the problem of finding an optimal *k-branching*, defined as a polytree that can be turned into a branching by deleting k arcs. In the problem formulation, we make the assumption—which is standard in the context of learning probabilistic networks—that the score of a DAG is obtained as the sum of local scores, each of which is a single real number assigned to a node and its in-neighbors; we give precise definitions in the next section.

Our main result is an algorithm that finds an optimal k-branching in polynomial time for every constant k. Our overall approach is straightforward: we search exhaustively over all possible sets of at most k "extra arcs", fix the guessed arcs, and solve the induced optimization problem for branchings. Implementing this seemingly innocent algorithm, however, requires a successful treatment of certain complications that arise when applying the existing matroid machinery for finding optimal branchings. In particular, one needs to control the interaction of the extra arcs with the solution from the induced subproblem.

While our algorithm for finding an optimal k-branching is polynomial for fixed k, the degree of the polynomial depends on k. Thus the algorithm does not scale well with respect to k. We therefore investigate variants of the problem that admit *fixed-parameter tractability* in the sense of Downey and Fellows [19]: the running time is given by $O(f(k)p(n))$, where p is a polynomial depending only on the input size n and f is a computable function that depends only on the parameter k. In particular, we show that finding an optimal k-branching is NP-hard but fixed-parameter tractable under the following additional constraint on the sought k-branching: the set of arcs incident to nodes with more than one parent are required to form a connected polytree with exactly one sink; in addition, we assume that each node has a bounded number of possible incoming arcs. We complement the fixed-parameter tractability result by showing that finding an optimal polytree is not fixed-parameter tractable with respect to another natural parameter, subject to complexity theoretic assumptions. Namely, we show that the problem is not fixed-parameter tractable when parameterized by the *number of nodes* whose deletion produces a branching. We want to point out that deleting nodes instead of arcs enlarges the class of potential solution networks and therefore potentially produces higher quality networks. This is what makes this parameterization interesting in its own right.

v	P	$f(v,P)$
1	$\emptyset$	1.0
2	$\emptyset$	1.0
3	$\{1\}$	1.0
4	$\emptyset$	0.1
4	$\{1\}$	0.2
5	$\{1\}$	0.5
5	$\{1,2\}$	1.0
6	$\{3\}$	0.8
6	$\{3,4\}$	1.0
7	$\{5\}$	0.9
7	$\{4,5\}$	1.0

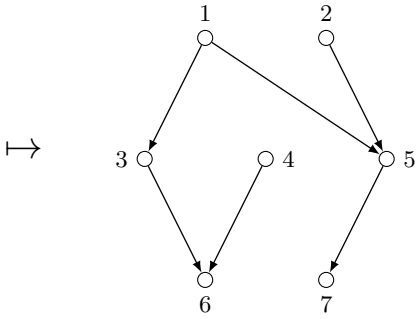

Figure 1: An optimal polytree for a given scoring function.

2. The k-Branching Problem

A (directed) probabilistic network, also called a Bayesian network or directed graphical model, is a multivariate probability distribution that obeys a structural representation in terms of a directed acyclic graph (DAG) and a collecion of univariate conditional probability distributions, one for each node of the graph. In the structure learning problem we consider, the DAG will be the protagonist, whereas the conditional distributions will only play an implicit role in defining the scoring function [20, 21, 22, 12].

We use the following terminology and notation. A DAG is a pair (N, A), where N is the *node set* and $A \subseteq N \times N$ is the *arc set*. We shall identify the graph with the arc set A when there is no ambiguity about the node set. We call a node u a *parent* of v in the graph if (u, v) is an arc in A, and denote by A_v the set of parents of v, that is, the in-neighborhood of v. When our interest is in the undirected structure of the graph, we may denote by $\overline{A}$ the *skeleton* of A, that is, the set of *edges* $\{\,\{u, v\} : (u, v) \in A\,\}$. Specifically, we call A a *polytree* if $\overline{A}$ is acyclic, and a *branching* if additionally each node has at most one parent.

When learning a probabilistic network from data it is customary to introduce a scoring function that assigns each graph A a *score* $f(A)$, which is a real number (or $-\infty$), measuring how well A fits the data. While there are plenty of alternative scoring functions, derived under different statistical paradigms and assumptions [20, 21, 22, 12], the most popular ones share one important property: they are *decomposable*, that is,

$$f(A) = \sum_{v \in N} f(v, A_v)\,,$$

with some *local scoring functions* $f(v, A_v)$. The generic computational problem is to maximize the scoring function over some appropriate class of graphs given the local scoring functions as input. Note that the score $f(v, A_v)$ need not be a sum of any individual arc weights, and that the parent set A_v may be empty. Figure 1 shows a table representing local scoring functions, together with an optimal polytree.

We study this problem by constraining the output to a graph class that is a subclass of polytrees but a superclass of branchings. We call a polytree A a *k-branching* if there exists a set of at most k arcs $D \subseteq A$ such that in $A \setminus D$ every node has at most one parent. Thus, for example, any branching is a 0-branching, and any polytree is a k-branching for $k = |N| - 1$. Following Dasgupta [12], we reserve the term *k-polytree* for polytrees where each node has at most k parents. The k-branching problem is to find a k-branching A that maximizes $f(A)$, given the values $f(v, A_v)$ for each node v and set A_v in a collection of potential parent sets for v:

k-Branching

Input: A set N of n nodes, collections $\mathcal{C}_v$ of potential parent sets (subsets of $N \setminus \{v\}$) for each $v \in N$, real-valued local scores $f(v, A_v)$ for each $v \in N$ and $A_v \in \mathcal{C}_v$, and a nonnegative integer k. We make the convention that $f(v, A_v) = -\infty$ if $A_v \notin \mathcal{C}_v$.

Output: A k-branching A on N that maximizes $f(A)$.

The size of the input in this problem formulation is, in essence, determined by the total number of potential parent sets. We will mostly be interested in cases where this number grows at most polynomially in the number of nodes, which is true particularly when the in-degree of the graph is bounded by some constant. Since any 2-polytree is a k-branching for some $k < |N|$, the NP-hardness of finding an optimal 2-polytree [12, Theorem 6] implies the following (by simply setting $k = |N| - 1$ and removing all sets of cardinality greater than 2 from all collections of parent sets $\mathcal{C}_v$):

Theorem 1. *The k-*Branching *problem is NP-hard.*

We will also consider problem variants where we tighten or loosen the constraints on the sought polytree. Because the constraints concern the output of the problem, we introduce the problem variants under modified names. Specifically, in Section 4 we will introduce the k-Intree-Branching problem and the k-Node-Branching problem.

We note that k-branchings generalize branchings in a different direction than the Tree-augmented Naive Bayes classifier (TAN) due to Friedman [23]. Namely, in a TAN the in-degree of each node is at most two, and there is a designated class node of in-degree zero, removing of which leaves a spanning tree; the tree is undirected in the sense that the symmetric conditional mutual information is employed to score arcs.

3. An Algorithm for the k-Branching Problem

Throughout this section we consider a fixed instance of the k-Branching problem, that is, a node set N and a scoring function f. Thus all arcs will refer to elements of $N \times N$. We will use the following additional notation. If A is an arc set, then $H(A)$ denotes the *heads* of the arcs in A, that is, the set $\{\, v : (u, v) \in A \,\}$. If C is a set of edges, then $N(C)$ denotes the induced node set $\{\, u, v : \{u, v\} \in C \,\}$.

We present an algorithm that finds an optimal k-branching by implementing the following approach. First, we guess an arc set D of size at most k. Then we search for an optimal polytree A that contains D such that in $A \setminus D$ every node has at most one parent; in other words, $B = A \setminus D$ is an optimal branching with respect to an induced scoring function. Clearly, the set $\overline{D}$ must be acyclic. The challenge is in devising an algorithm that finds an optimal branching B that is disjoint from D while guaranteeing that the arcs in D will not create undirected cycles in the union $B \cup D$. To this end, we will employ an appropriate weighted matroid intersection formulation that extends the standard formulation for branchings.

We will need some basic facts about matroids. A *matroid* is a pair $(E, \mathcal{I})$, where E is a set of *elements*, called the *ground set*, and $\mathcal{I}$ is a collection of subsets of E, called the *independent sets*, such that

(M1) $\emptyset \in \mathcal{I}$;

(M2) if $A \subseteq B$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$; and

(M3) if $A, B \in \mathcal{I}$ and $|A| < |B|$ then there exists an $e \in B \setminus A$ such that $A \cup \{e\} \in \mathcal{I}$.

The *rank* of a matroid is the cardinality of its maximal independent sets. Any subset of E that is not independent is called *dependent*. Any minimal dependent set is called a *circuit*.

The power of matroid formulations is much due to the availability of efficient algorithms [24, 25, 26, 27, 28, 29] for the Weighted Matroid Intersection problem, defined as follows. Given two matroids $M_1 = (E, \mathcal{I}_1)$ and $M_2 = (E, \mathcal{I}_2)$, and a weight function $w\colon E \to \mathbb{R}$, find an $I \subseteq E$ that is independent in both matroids and maximizes the total weight of I, that is, $w(I) = \sum_{e \in I} w(e)$. The complexity of the fastest algorithm for the Weighted Matroid Intersection problem that we are aware of is summarized as follows:

Theorem 2 ([24])**.** *The* Weighted Matroid Intersection *problem can be solved in $O(mr(r+c+\log m))$ time, where $m = |E|$, r is the minimum of the ranks of M_1 and M_2, and c is the time needed for finding the circuit of $I \cup \{e\}$ in both M_1 and M_2 where $e \in E$ and I is independent in both M_1 and M_2.*

We now proceed to the specification of two matroids, $M_1(S) = (N \times N, \mathcal{I}_1(S))$ and $M_2(N \times N, \mathcal{I}_2(S))$, parametrized by an arbitrary arc set S such that $\overline{S}$ is acyclic:

1. The *in-degree matroid* $M_1(S)$: Let $\mathcal{I}_1(S)$ consist of all arc sets B such that no arc in B has a head in $H(S)$ and every node outside $H(S)$ is the head of at most one arc in B.

2. The *acyclicity matroid* $M_2(S)$: Let $\mathcal{I}_2(S)$ consist of all arc sets B such that $\overline{B \cup S}$ is acyclic.

We observe that the standard matroid intersection formulation of branchings is obtained as the special case of $S = \emptyset$: then an arc set is seen to be branching if and only if it is independent in both the in-degree matroid and the acyclicity matroid.

The next two lemmas show that $M_1(S)$ and $M_2(S)$ are indeed matroids whenever $\overline{S}$ is acyclic.

Lemma 3. *$M_1(S)$ is a matroid (whenever $\overline{S}$ is acyclic).*

Proof. Fix the arc set S and denote $\mathcal{I}_1(S)$ by $\mathcal{I}_1$ for short. Clearly, $\emptyset \in \mathcal{I}_1$ and if $A \subseteq B$ and $B \in \mathcal{I}_1$ then also $A \in \mathcal{I}_1$. Consequently, $M_1(S)$ satisfies (M1) and (M2). To see that $M_1(S)$ satisfies (M3) let $A, B \in \mathcal{I}_1$ with $|A| < |B|$. Because of the definition of $M_1(S)$ the sets A and B contain at most one arc with head v, for every $v \in N \setminus H(S)$. Because $|A| < |B|$ there is a node $v \in N \setminus H(S)$ such that v is the head of an arc in B but v is not the head of an arc in A. Let $e \in B$ be the arc with head v. Then $e \in B \setminus A$ and $A \cup \{e\} \in \mathcal{I}_1$. Hence, $M_1(S)$ satisfies (M3). □

Lemma 4. *$M_2(S)$ is a matroid (whenever $\overline{S}$ is acyclic).*

Proof. Fix the arc set S and denote $\mathcal{I}_2(S)$ by $\mathcal{I}_2$ for short. Because the skeleton $\overline{S}$ is acyclic and acylicity is a hereditary property (a graph property is called hereditary if it is closed under taking induced subgraphs) it follows that $\emptyset \in \mathcal{I}_2$ and if $A \subseteq B$ and $B \in \mathcal{I}_2$ then also $A \in \mathcal{I}_2$. Consequently, $M_2(S)$ satisfies (M1) and (M2). To see that $M_2(S)$ satisfies (M3) let $A, B \in \mathcal{I}_2$ with $|A| < |B|$. Consider the sets $A' = \overline{A \cup S}$ and $B' = \overline{B \cup S}$. Let C be a connected subset of A'. Because both A' and B' are acyclic, it follows that the number of edges of B' with both endpoints in $N(C)$ is at most the number of edges of A' with both endpoints in $N(C)$. Because every edge in $A' \setminus \overline{S}$ corresponds to an arc in A and similarly every edge in $B' \setminus \overline{S}$ corresponds to an arc in B and $|A| < |B|$, it follows that there is an arc $e \in B \setminus A$ whose endpoints are contained in two distinct components of A'. Consequently, the set $A' \cup \overline{\{e\}}$ is acyclic and hence $A \cup \{e\} \in \mathcal{I}_2$. □

We now relate the common independent sets of these two matroids to k-branchings. If A is a k-branching, we call an arc set D a *deletion set* of A if D is a subset of A, contains at most k arcs, and in $A \setminus D$ every node has at most one parent.

Lemma 5. *Let A be an arc set and D a subset of A of size at most k such that no two arcs from $D' = \{ (u, v) \in A \setminus D : v \in H(D) \}$ have the same head and such that $\overline{S}$ is acyclic, where $S = D \cup D'$. We have that A is a k-branching with deletion set D if and only if $A \setminus S$ is independent in both $M_1(S)$ and $M_2(S)$.*

Proof. ($\Rightarrow$) : Suppose A is a k-branching with deletion set D. Then $A \setminus D$ is a branching, which shows that every node v outside $H(S)$ has in-degree at most one in $A \setminus S$. Since by definition all arcs with a head in $H(S)$ are contained in S, no arc in $A \setminus S$ has a head in $H(S)$. Therefore, $A \setminus S$ is independent in $M_1(S)$. Since every k-branching is a polytree, $\overline{(A \setminus S) \cup S} = \overline{A}$ is acyclic, and therefore $A \setminus S$ is independent in $M_2(S)$.

($\Leftarrow$) : Since $A \setminus S$ is independent in $M_2(S)$, we have that $\overline{(A \setminus S) \cup S} = \overline{A}$ is acyclic. Thus, A is a polytree. As $A \setminus S$ is independent in $M_1(S)$, every node outside $H(S)$ has in-degree at most one in $A \setminus S$ and every node from $H(S)$ has in-degree zero in $A \setminus S$. Since the head of every arc from D' is in $H(S)$ and no two arcs from D' have a common head, $(A \setminus S) \cup D' = A \setminus D$ has maximum in-degree at most one. Because $|D| \leq k$, we have that A is a k-branching with deletion set D. □

The characterization of Lemma 5 enables the following algorithm for the k-Branching problem. Define the weight function by letting $w(u, v) = f(v, \{u\}) - f(v, \emptyset)$ for all arcs (u, v). Guess the arc sets D and D', put $S = D \cup D'$, check that $\overline{S}$ is acyclic, find a maximum-weight set B that is independent in both $M_1(S)$

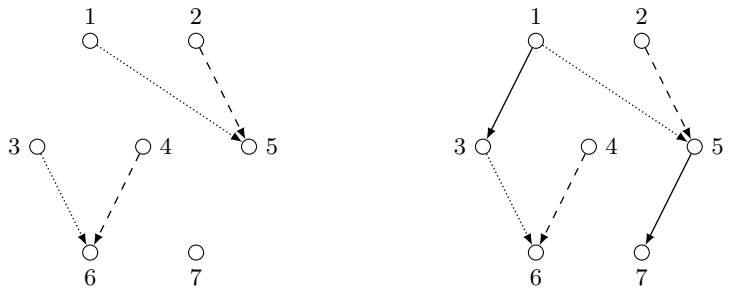

Figure 2: Left: the two guessed arc sets D (dotted) and D' (dashed). Right: the arc set A (solid) that is a heaviest common independent set of the two matroids $M_1(S)$ and $M_2(S)$.

and $M_2(S)$; output a k-branching $A = B \cup S$ that yields the maximum weight over all guesses D and D', where the weight of A is obtained as

$$w(A) = w(B) + \sum_{v \in H(S)} \big(f(v, S_v) - f(v, \emptyset)\big)\,.$$

It is easy to verify that $w(A) = f(A) - \sum_{v \in N} f(v, \emptyset)$, and thus maximizing the weight $w(A)$ is equivalent to maximizing the score $f(A)$. Figure 2 illustrates the algorithm for the scoring function of Figure 1.

It remains to analyze the complexity of the algorithm. Denote by n the number of nodes. For a moment, consider the arc set S fixed. To apply Theorem 2, we bound the associated key quantities: the size of the ground set is $O(n^2)$; the rank of both matroids is clearly $O(n)$; circuit detection can be performed in $O(n)$ time, by a depth-first search for $M_1(S)$ and by finding a node that has higher in-degree than it is allowed to have in $M_2(S)$. Thus, by Theorem 2, a maximum-weight set that is independent in both matroids can be found in $O(n^4)$ time. Then consider the number of possible choices for the set $S = D \cup D'$. There are $O(n^{2k})$ possibilities for choosing a set D of at most k arcs such that $\overline{D}$ is acyclic. For a fixed D, there are $O(n^k)$ possibilities for choosing a subset $D' \subseteq N \times H(D)$ such that $\overline{D \cup D'}$ is acyclic and no two arcs from D' have the same head. Thus there are $O(n^{3k})$ relevant choices for the set S.

We have shown the following:

Theorem 6. *The k-*Branching *problem can be solved in $O(n^{3k+4})$ time.*

4. Fixed-Parameter Tractability

Theorem 6 shows that the k-Branching problem can be solved in "non-uniform polynomial time" as the order of the polynomial time bound depends on k. In this section we study the question of whether one can get k "out of the exponent" and obtain a uniform polynomial-time algorithm.

The framework of *Parameterized Complexity* [19] offers the suitable tools and methods for such an investigation, as it allows us to distinguish between uniform and non-uniform polynomial-time tractability with respect to a parameter. An instance of a parameterized problem is a pair (I, k) where I is the *main part* and k is the *parameter*; the latter is usually a non-negative integer. A parameterized problem is *fixed-parameter tractable* if there exist a computable function f and a constant c such that instances (I, k) of size n can be solved in time $O(f(k)n^c)$. FPT is the class of all fixed-parameter tractable decision problems. Fixed-parameter tractable problems are also called *uniform polynomial-time tractable* because if k is considered constant, then instances with parameter k can be solved in polynomial time where the order of the polynomial is independent of k (in contrast to non-uniform polynomial-time running times such as n^k).

Parameterized complexity offers a completeness theory similar to the theory of NP-completeness. One uses *parameterized reductions* which are many-one reductions where the parameter for one problem maps into the parameter for the other. More specifically, problem L reduces to problem L' if there is a mapping R from instances of L to instances of L' such that (i) (I, k) is a yes-instance of L if and only if $(I', k') = R(I, k)$ is a

yes-instance of L', (ii) $k' \leq g(k)$ for a computable function g, and (iii) R can be computed in time $O(f(k)n^c)$ where f is a computable function, c is a constant, and n denotes the size of (I, k). The parameterized complexity class W[1] is considered as the parameterized analog to NP. For example, the parameterized Maximum Clique problem (given a graph G and a parameter $k \geq 0$, does G contain a complete subgraph on k vertices?) is W[1]-complete under parameterized reductions. Note that there exists a trivial non-uniform polynomial-time n^k algorithm for the Maximum Clique problems that checks all sets of k vertices. FPT $\neq$ W[1] is a widely accepted complexity theoretic assumption [19]. For example, FPT = W[1] implies the (unlikely) existence of a $2^{o(n)}$ algorithm for n-variable 3-Sat [30, 31]. A first parameterized analysis of probabilistic network structure learning using structural parameters such as treewidth has recently been carried out by Ordyniak and Szeider [32].

The algorithm from Theorem 6 considers $O(n^{3k})$ relevant choices for the set $S = D \cup D'$, and for each fixed choice of S the running time is polynomial. Thus, for variants of the problem for which the enumeration of all relevant sets S is fixed-parameter tractable, one obtains an FPT algorithm. In one such variant we require that $S = D \cup D'$ is an in-tree, that is, a directed tree where every arc is directed towards a designated root. We call a polytree a *k-intree-branching* if deleting such an arc set D leaves a branching.

k-Intree-Branching

Input: (N, f, k) as in k-Branching.

Output: A k-intree-branching A on N that maximizes $f(A)$.

To obtain an FPT algorithm, we further restrict ourselves to inputs where each node has a bounded number of potential parent sets:

Theorem 7. *The k-Intree-Branching problem is fixed-parameter tractable if each node has a bounded number of potential parent sets.*

Proof. To compute a k-intree-branching A, the algorithm guesses its deletion set D and the set $D' = \{(u, v) \in A \setminus D : v \in H(D)\}$. As A is a k-branching, $|D| \leq k$ and for every $v \in H(D)$ there is at most one arc in D' with head v. The algorithm first guesses the root r for the in-tree S. Then it goes over all possible choices for D and D' as follows, until D has at least k arcs.

Guess a leaf ℓ of S (initially, r is the unique leaf of S), and guess a non-empty parent set P for ℓ in A. If $|D| + |P| + 1 > k$, then backtrack. Otherwise, choose at most one arc (p, ℓ) to add to D', where $p \in P$, and add all other arcs from a node from P to ℓ to D (if $|P| = 1$, no arc is added to D'). Now, check whether the current choice for $S = D \cup D'$ leads to a k-branching by checking whether $\overline{S}$ is acyclic and using the matroids $M_1(S)$ and $M_2(S)$ as in Theorem 6.

There are at most n choices for r. The in-tree S is expanded in at most k steps, as each step adds at least one arc to D. In each step, ℓ is chosen among at most $k + 1$ leaves, there is a constant number of choices for its parent set P and at most $k + 2$ choices for adding (or not) an arc (p, ℓ), with $p \in P$, to D' (as $|P| \leq k + 1$). The acyclicity check for $\overline{S}$ and the weighted matroid intersection can be computed in time $O(n^4)$, leading to a total running time of $O(k^{2k}c^k n^5)$, where c is such that every node has at most c potential parent sets. □

We remark that the intree-requirement may be replaced by other conditions requiring the connectivity of D or a small distance between arcs from D, giving other fixed-parameter tractable variants of the k-Branching problem.

The following theorem shows that a superpolynomial dependency on k or some other parameter is necessary, since the k-Intree-Branching problem is NP-hard:

Theorem 8. *The k-Intree-Branching problem is NP-hard even if each node has at most 3 potential parent sets.*

Proof. We devise a polynomial reduction from 3-Sat-2 a version of 3-Satisfiability where every literal occurs at most in two clauses. 3-Sat-2 is well known to be NP-hard [33].

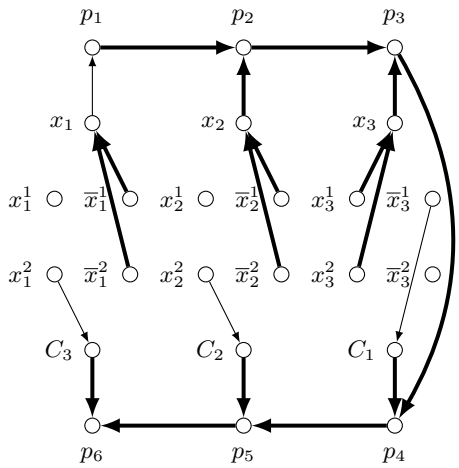

Figure 3: An optimal k-intree-branching A for the formula $\Phi = C_1 \wedge C_2 \wedge C_3$ with $C_1 = x_1 \vee x_2 \vee \overline{x}_3$, $C_2 = \overline{x}_1 \vee x_2 \vee x_3$, and $C_3 = x_1 \vee \overline{x}_2 \vee \overline{x}_3$ according to the construction in the proof of Theorem 8. The bold arcs are all the arcs in the set $D \cup D'$, where D is any deletion set for A and $D' = \{(u, v) \in A \setminus D : v \in H(D)\}$.

Our reduction uses the same ideas as Dasgupta's proof of a related result [12, Theorem 6]. Let Φ be an instance of 3-SAT-2 with clauses $C_1, \ldots, C_m$ and variables $x_1, \ldots, x_n$. We define the set N of nodes as follows. For every variable x_i in Φ the set N contains the nodes p_i, x_i, x_i^1, x_i^2, $\overline{x}_i^1$ and $\overline{x}_i^2$. Furthermore, for every clause C_j the set N contains the nodes p_{n+j} and C_j. Let $1 \leq i \leq n$, $1 \leq j \leq m$, and $1 \leq l \leq 2$. We set

- $f(C_j, \{x_i^l\}) = 1$ if the clause C_j is the l-th clause that contains the literal x_i;
- $f(C_j, \{\overline{x}_i^l\}) = 1$ if the clause C_j is the l-th clause that contains the literal $\overline{x}_i$;
- $f(x_i, \{x_i^1, x_i^2\}) = 1$ and $f(x_i, \{\overline{x}_i^1, \overline{x}_i^2\}) = 1$ for every i with $1 \leq i \leq n$;
- $f(p_1, \{x_1\}) = 1$ and $f(p_i, \{x_i, p_{i-1}\}) = 1$ for every i with $1 < i \leq n$;
- $f(p_{n+j}, \{C_j, p_{n+j-1}\}) = 1$ for every $1 \leq j \leq m$.

Furthermore, we set $f(v, P) = 0$ for all the remaining combinations of $v \in N$ and $P \subseteq N$, and we set $k = 2n + m - 1$.

This completes the construction of N, f, and k. Figure 3 shows an optimal k-branching A for a 3-SAT-2 formula. Observe that every node of N has at most 3 potential parent sets. It remains to prove the following claim.

Claim: Φ *is satisfiable if and only if there is a* k*-intree-branching* A *such that* $f(A) \geq 2(m + n)$.

($\Rightarrow$) : Suppose that the formula Φ is satisfiable and let β be a satisfying assignment for Φ. Furthermore, for every $1 \leq j \leq m$ let l_j be a literal of C_j that is set to true by β. We construct a k-intree-branching A as follows. For every $1 \leq j \leq m$ the digraph A contains an arc (x_i^l, C_j) if $l_j = x_i$ and C_j is the l-th clause that contains x_i and an arc $(\overline{x}_i^l, C_j)$ if $l_j = \overline{x}_i$ and C_j is the l-th clause that contains $\overline{x}_i$ for some $1 \leq i \leq n$ and $1 \leq l \leq 2$. Furthermore, for every $1 \leq i \leq n$ the digraph A contains the arcs (x_i^1, x_i) and (x_i^2, x_i) if $\beta(x_i) =$ false and the arcs $(\overline{x}_i^1, x_i)$ and $(\overline{x}_i^2, x_i)$ if $\beta(x_i) =$ true. Finally, A contains the arcs (x_i, p_i), (C_j, p_{n+j}) and (p_l, p_{l+1}) for every $1 \leq i \leq n$, $1 \leq j \leq m$, and $1 \leq l < m + n$. Figure 3 shows an optimal k-intree-branching A for some 3-SAT-2 formula. It is easy to see that A is a k-intree-branching with $f(A) = 2(m + n)$.

($\Leftarrow$) : Suppose there is a k-intree-branching A such that $f(A) \geq 2(m + n)$. Because $f(A) \geq 2(m + n)$ it follows that every node of N achieves its maximum score in A. Hence, A has to contain the arcs (x_i, p_i), (C_j, p_{n+j}), (p_l, p_{l+1}), for every $1 \leq i \leq n$, $1 \leq j \leq m$, and $1 \leq l < m + n$. For the same reasons A has to

contain either the arcs (x_i^1, x_i) and (x_i^2, x_i) or the arcs $(\overline{x}_i^1, x_i)$ and $(\overline{x}_i^2, x_i)$ for every $1 \leq i \leq n$. Furthermore, for every $1 \leq j \leq m$ the k-intree-branching A has to contain one arc of the form (x_i^l, C_j) or $(\overline{x}_i^l, C_j)$ where C_j is the l-th clause that contains x_i or $\overline{x}_i$, respectively, for some $1 \leq i \leq n$ and $1 \leq l \leq 2$. Let $1 \leq i \leq n$, $1 \leq j \leq m$, and $1 \leq l \leq 2$. We first show that whenever A contains an arc (x_i^l, x_i), then A contains no arc of the form (x_i^l, C_j) and similarly if A contains an arc $(\overline{x}_i^l, x_i)$, then A contains no arc of the form $(\overline{x}_i, C_j)$. Suppose for a contradiction that A contains an arc (x_i^l, x_i) together with an arc (x_i^l, C_j) or an arc $(\overline{x}_i^l, x_i)$ together with an arc $(\overline{x}_i^l, C_j)$. In the first case A contains the undirected cycle $(x_i^l, x_i, p_i, \ldots, p_{n+j}, C_j, x_i^l)$ and in the second case A contains the cycle $(\overline{x}_i^l, x_i, p_i, \ldots, p_{n+j}, C_j, \overline{x}_i^l)$ contradicting our assumption that A is a k-intree-branching. It now follows that the assignment β with $\beta(x_i) = \text{true}$ if A does not contain the arcs (x_i^1, x_i) and (x_i^2, x_i) and $\beta(x_i) = \text{false}$ if A does not contain the arcs $(\overline{x}_i^1, x_i)$ and $(\overline{x}_i^2, x_i)$ is a satisfying assignment for Φ. □

So far we have measured the difference of a polytree to branchings in terms of the number of arcs to be deleted. Next we investigate the consequences of measuring the difference by the number of nodes to be deleted.

We use the following additional notation. If A is an arc set, we denote by $N(A)$ its induced node set $\{ u, v : (u, v) \in A \}$ and for a node set X we denote by $A - X$ the set of arcs $\{ (u, v) \in A : u, v \notin X \}$. We call a polytree A a *k-node-branching* if there exists a set of at most k nodes $X \subseteq N(A)$ such that $A - X$ is a branching. Clearly every k-branching is a k-node-branching, but the reverse does not hold. In other words, the class of k-node-branchings properly contains the class of k-branchings. This means that the *k-node-branching* problem has the potential to produce higher quality networks than the *k-branching* problem.

k-Node-Branching

Input: (N, f, k) as in k-Branching.

Output: A k-node-branching A on N that maximizes $f(A)$.

We next show that the k-Node-Branching problem is hard for the parameterized complexity class W[1]. This provides strong evidence that the problem is not fixed-parameter tractable.

Theorem 9. *The k-Node-Branching problem is* W[1]*-hard for parameter k.*

Proof. We devise a parameterized reduction from the following problem, called Partitioned Clique, which is well-known to be W[1]-complete for parameter p [34].

Partitioned Clique

Instance: An integer p and an undirected graph $G = (V, E)$ with a partition of V into p sets $V_1, \ldots, V_p$ of equal size.

Question: Are there p vertices $v_1 \in V_1, \ldots, v_k \in V_p$ such that $\{v_i, v_j\} \in E$ for all $1 \leq i < j \leq p$? (The graph $K = \big(\{v_1, \ldots, v_p\}, \{ \{v_i, v_j\} : 1 \leq i < j \leq p \}\big)$ is a *p-clique* of G.)

Fix an arbitrary instance of Partitioned Clique. We construct an instance (N, f, k) of k-Node-Branching, with $k = \binom{p}{2} + p$, as follows.

For each $i = 1, \ldots, p$ we introduce a node c_i. For each vertex $v \in V$ we introduce p nodes $v^1, \ldots, v^p$. Furthermore, we introduce a node h_{ij} for each $1 \leq i < j \leq p$. In summary, we set $N = \{c_1, \ldots, c_p\} \cup \{ v^1, \ldots, v^p : v \in V \} \cup H$, where $H = \{ h_{ij} : 1 \leq i < j \leq p \}$. For every i with $1 \leq i \leq p$ and $w \in V_i$, we denote $H_i = \{ h_{lp} \in H : l = i \text{ or } p = i \}$ and $V_i^w = \{ v^1, \ldots, v^p : v \in V_i \text{ and } v \neq w \}$.

We define the scoring function f as follows. We set $f(c_i, H_i \cup V_i^w) = 1$ for every $1 \leq i \leq p$ and $w \in V_i$, and $f(h_{ij}, \{u^j, w^i\}) = 1$ for every $1 \leq i < j \leq p$, $u \in V_i$, $w \in V_j$, and $\{u, w\} \in E(G)$. Furthermore, we set $f(v, P) = 0$ for all the remaining combinations of v and P.

Because this reduction can be computed in polynomial-time and k is polynomial in p, the given construction is a parameterized reduction. It remains to prove the following claim.

Claim: G has a p-clique if and only if there is a k-node-branching A such that $f(A) \geq k$.

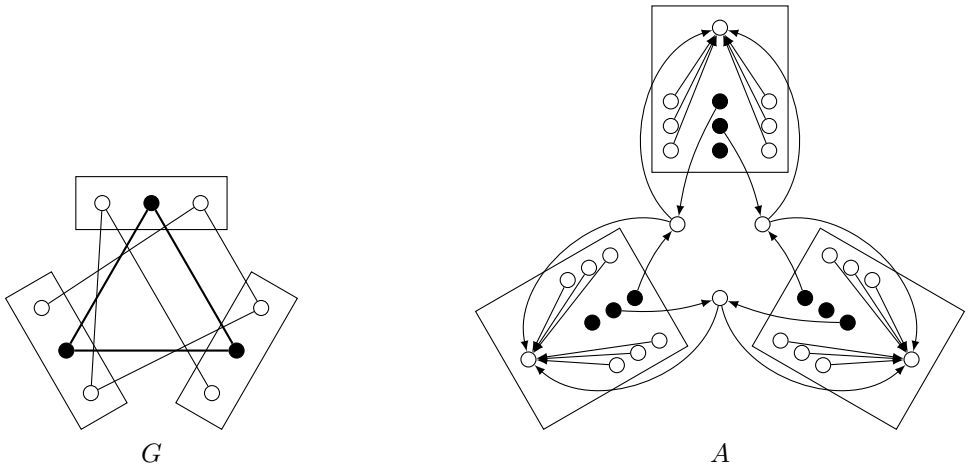

Figure 4: A graph G with a partitioning of the vertices into $p = 3$ sets and a corresponding optimal k-node-branching A with $k = \binom{p}{2} + p$ and $f(A) \geq k$, as constructed in the proof of Theorem 9.

($\Rightarrow$) : Suppose that G has a p-clique K with vertices $v_1, \ldots, v_p$. Let A be the directed graph on N such that for every i with $1 \leq i \leq p$, the node c_i has parents $H_i \cup V_i^{v_i}$ and for every i and j with $1 \leq i < j \leq p$ the node h_{ij} has parents v_i^j and v_j^i, and every other node of A has no parents. Figure 4 shows an optimal k-node-branching A constructed from an example graph G. It is straightforward to verify that A is a DAG and moreover A is a k-node-branching. Furthermore, because K is a p-clique of G, it follows that $f(A) = k$, as required.

($\Leftarrow$) : Suppose there is a k-node-branching A with $f(A) \geq k$. It follows that every node of A achieves its maximum score. In particular, for every $1 \leq i \leq p$ the nodes c_i must have score 1 in A and hence there is a node $w_i \in V_i$ such that c_i is adjacent to all nodes in $V_i^{w_i} \cup H_i$. Furthermore, for every $1 \leq i < j \leq p$ the node h_{ij} is adjacent to exactly one node v_i^j in V_i and to exactly one node v_j^i in V_j. Then, $w_i = v_i$ and $w_j = v_j$ because otherwise the skeleton of A would contain the cycle (v_i, h_{ij}, c_i) or the cycle (v_j, h_{ij}, c_j). Consequently, the edges represented by the parents of h_{ij} in A for all $1 \leq i < j \leq p$ form a p-clique of G. □

5. Concluding Remarks

We have investigated a natural approach to extend the efficient optimization algorithms known for branchings to classes of polytrees that differ from branchings by only a few extra arcs. Our main result is a polynomial time algorithm for finding an optimal polytree that contains a constant number of extra arcs. The algorithm allows us to efficiently find optimal probabilistic networks in a class that is more expressive than the class based on branchings. Moreover, adjusting the number of extra arcs provides us with a trade-off between the power of the model and the running time of the algorithm. Our algorithm also provides better lower bounds on the score over polytrees or any other larger class networks, which bounds can be used to guide the search for an optimal solution.

At first glance, one might expect that our result can be obtained by simply guessing the extra arcs and solving the remaining problem for branchings. However, we do not know whether such a reduction is possible in the strict sense. Indeed, we had to take a slight detour and modify the two matroids in a way that guarantees a control for the interactions caused by the presence of high-in-degree nodes. As a result, we obtained an algorithm that runs in time polynomial in the input size for any constant bound k on the number of extra arcs. The running time bound $O\left(n^{3k+4}\right)$ is, in fact, less than cubic in the size of the input, supposing the local scores are given explicitly for all possible parent sets—namely, each of the n nodes has $\binom{n-1}{k+1}$ relevant input values.

While our result answers one question in the affirmative, it also raises several further questions. First, our complexity analysis relied on a result concerning the general weighted matroid intersection problem. Do our

two specific matroids admit significantly faster specialized algorithms? One might expect such algorithms exist, since the related problem for branchings can be solved in $O(n^2)$ time by Tarjan's algorithm [11]. Second, even if we could solve the matroid intersection problem faster, our algorithm would remain practical only for very small values of k. Can one find an optimal k-branching significantly faster, especially if allowing every node to have at most two parents? As the current algorithm makes around n^{3k} mutually overlapping guesses, there might be a way to considerably reduce the time requirement. Specifically, we ask whether the restricted problem is fixed-parameter tractable with respect to the parameter k, that is, solvable in $O(f(k)p(n))$ time for some computable function f and polynomial p [19]. The fixed-parameter algorithm given in Section 4 can be seen as a first step towards an answer to this question. Third, can our approach be extended to k-node-branchings? That is, is there a polynomial time algorithm for the k-Node-Branching problem for every fixed k?

Finally, it remains an open question, whether finding optimal polytrees is easier or harder than finding optimal DAGs. In this light, it would be reasonable to also investigate DAGs that need not be polytrees but that can be turned into branchings by deleting some k arcs.

Acknowledgments

We would like to thank an anonymous reviewer for the suggestion to highlight the difference between restricting the input and restricting the output of a problem. Serge Gaspers, Sebastian Ordyniak, and Stefan Szeider acknowledge support from the European Research Council (COMPLEX REASON, 239962). Serge Gaspers is the recipient of an Australian Research Council Discovery Early Career Researcher Award (project number DE120101761) and a Future Fellowship (project number FT140100048). Mikko Koivisto acknowledges the support from the Academy of Finland (Grant 125637). Mathieu Liedloff acknowledges the support from the French Agence Nationale de la Recherche (ANR AGAPE ANR-09-BLAN-0159-03). Sebastian Ordyniak acknowledges support from the Employment of Newly Graduated Doctors of Science for Scientific Excellence (CZ.1.07/2.3.00/30.0009). NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

[1] S. Gaspers, M. Koivisto, M. Liedloff, S. Ordyniak, S. Szeider, On finding optimal polytrees, in: J. Hoffmann, B. Selman (Eds.), Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada, AAAI Press, 2012.

[2] D. M. Chickering, Learning Bayesian networks is NP-complete, in: Learning from data (Fort Lauderdale, FL, 1995), Vol. 112 of Lecture Notes in Statist., Springer Verlag, 1996, pp. 121–130.

[3] J. R. Edmonds, Optimum branchings, Journal of Research of the National Bureau of Standards 71B (4) (1967) 233–240.

[4] Y. J. Chu, T. H. Liu, On the shortest arborescence of a directed graph, Science Sinica 14 (1965) 1396–1400.

[5] F. C. Bock, An algorithm to construct a minimum directed spanning tree in a directed network, in: B. Avi-Itzak (Ed.), Developments in Operations Research, Gordon and Breach, 1971, pp. 29–44.

[6] P. M. Camerini, L. Fratta, F. Maffioli, A note on finding optimum branchings, Networks 9 (1979) 309–312.

[7] D. R. Fulkerson, Packing rooted directed cuts in a weighted directed graph, Mathematical Programming 6 (1974) 1–13.

[8] H. N. Gabow, Z. Galil, T. Spencer, R. E. Tarjan, Efficient algorithms for finding minimum spanning trees in undirected and directed graphs, Combinatorica 6 (2) (1986) 109–122.

[9] H. N. Gabow, Z. Galil, T. H. Spencer, Efficient implementation of graph algorithms using contraction, Journal of the ACM 36 (3) (1989) 540–572.

[10] R. M. Karp, A simple derivation of Edmonds' algorithm for optimum branchings, Networks 1 (3) (1971) 265–272.

[11] R. E. Tarjan, Finding optimum branchings, Networks 7 (1977) 25–35.
[12] S. Dasgupta, Learning polytrees, in: K. B. Laskey, H. Prade (Eds.), UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, July 30-August 1, 1999, Morgan Kaufmann, 1999, pp. 134–141.
[13] F. V. Fomin, D. Kratsch, Exact Exponential Algorithms, Texts in Theoretical Computer Science. An EATCS Series, Springer, 2010.
[14] M. Koivisto, K. Sood, Exact Bayesian structure discovery in Bayesian networks, J. Mach. Learn. Res. 5 (2004) 549–573.
[15] S. Ott, S. Miyano, Finding optimal gene networks using biological constraints, Genome Informatics 14 (2003) 124–133.
[16] P. Parviainen, M. Koivisto, Exact structure discovery in Bayesian networks with less space, in: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI 2009), 2009, pp. 436–443.
[17] T. Silander, P. Myllymäki, A simple approach for finding the globally optimal Bayesian network structure, in: Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI 2006), 2006, pp. 445–452.
[18] A. Cayley, A theorem on trees, Quart. J. Math. 23 (1889) 376–378.
[19] R. G. Downey, M. R. Fellows, Parameterized Complexity, Monographs in Computer Science, Springer Verlag, New York, 1999.
[20] W. Lam, F. Bacchus, Learning Bayesian belief networks: An approach based on the MDL principle, Computational Intelligence 10 (1994) 269–293.
[21] D. M. Chickering, A transformational characterization of equivalent Bayesian network structures, in: Uncertainty in artificial intelligence (Montreal, PQ, 1995), Morgan Kaufmann, San Francisco, CA, 1995, pp. 87–98.
[22] D. Heckerman, D. Geiger, D. M. Chickering, Learning Bayesian networks: The combination of knowledge and statistical data, Machine Learning 20 (3) (1995) 197–243.
[23] N. Friedman, D. Geiger, M. Goldszmidt, Bayesian network classifiers, Machine Learning 29 (1997) 131–163.
[24] C. Brezovec, G. Cornuéjols, F. Glover, Two algorithms for weighted matroid intersection, Mathematical Programming 36 (1) (1986) 39–53.
[25] J. R. Edmonds, Submodular functions, matroids and certain polyhedra, in: Proceedings of the Calgary International Conference on Combinatorial Structures and their Applications, 1970, pp. 69–87.
[26] J. R. Edmonds, Matroid intersection, Annals of Discrete Mathematics 4 (1979) 39–49.
[27] A. Frank, A weighted matroid intersection algorithm, Journal of Algorithms 2 (1981) 328–336.
[28] M. Iri, N. Tomizawa, An algorithm for finding an optimal 'independent' assignment, Journal of the Operations Research Society of Japan 19 (1976) 32–57.
[29] E. L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.
[30] R. Impagliazzo, R. Paturi, F. Zane, Which problems have strongly exponential complexity?, J. of Computer and System Sciences 63 (4) (2001) 512–530.
[31] J. Flum, M. Grohe, Parameterized Complexity Theory, Vol. XIV of Texts in Theoretical Computer Science. An EATCS Series, Springer Verlag, Berlin, 2006.
[32] S. Ordyniak, S. Szeider, Parameterized complexity results for exact bayesian network structure learning, J. Artif. Intell. Res. 46 (2013) 263–302.
[33] M. R. Garey, D. R. Johnson, Computers and Intractability, W. H. Freeman and Company, New York, San Francisco, 1979.
[34] K. Pietrzak, On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems, J. of Computer and System Sciences 67 (4) (2003) 757–771.