Finding Branch-Decompositions of Matroids, Hypergraphs, and More

Jisu Jeong[1]
Department of Mathematical Sciences, KAIST, Daejeon, Korea
jisujeong89@gmail.com

Eun Jung Kim[2]
Université Paris-Dauphine, PSL Research University, CNRS, Paris, France
eun-jung.kim@dauphine.fr

Sang-il Oum[3]
Department of Mathematical Sciences, KAIST, Daejeon, Korea
sangil@kaist.edu

Abstract

Given n subspaces of a finite-dimensional vector space over a fixed finite field $\mathbb{F}$, we wish to find a "branch-decomposition" of these subspaces of width at most k, that is a subcubic tree T with n leaves mapped bijectively to the subspaces such that for every edge e of T, the sum of subspaces associated to the leaves in one component of $T - e$ and the sum of subspaces associated to the leaves in the other component have the intersection of dimension at most k. This problem includes the problems of computing branch-width of $\mathbb{F}$-represented matroids, rank-width of graphs, branch-width of hypergraphs, and carving-width of graphs.

We present a fixed-parameter algorithm to construct such a branch-decomposition of width at most k, if it exists, for input subspaces of a finite-dimensional vector space over $\mathbb{F}$. Our algorithm is analogous to the algorithm of Bodlaender and Kloks (1996) on tree-width of graphs. To extend their framework to branch-decompositions of vector spaces, we developed highly generic tools for branch-decompositions on vector spaces. For this problem, a fixed-parameter algorithm was known due to Hliněný and Oum (2008). But their method is highly indirect. Their algorithm uses the non-trivial fact by Geelen et al. (2003) that the number of forbidden minors is finite and uses the algorithm of Hliněný (2006) on checking monadic second-order formulas on $\mathbb{F}$-represented matroids of small branch-width. Our result does not depend on such a fact and is completely self-contained, and yet matches their asymptotic running time for each fixed k.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability, Mathematics of computing → Graph algorithms

Keywords and phrases branch-width, rank-width, carving-width, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.80

Related Version A full version of the paper is available at `https://arxiv.org/abs/1711.01381`.

[1] Supported by the National Research Foundation of Korea (NRF) grant (No. NRF-2017R1A2B4005020).
[2] Supported by the National Research Agency (ANR) grant (No. ANR-17-CE40-0028).
[3] Supported by the National Research Foundation of Korea (NRF) grant (No. NRF-2017R1A2B4005020).

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 80; pp. 80:1–80:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Let $\mathbb{F}$ be a finite field and r be a positive integer. A *subspace arrangement* $\mathcal{V}$ is a (multi)set of subspaces of $\mathbb{F}^r$, which can be represented by an $r \times m$ matrix M with an ordered partition $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ of $\{1, 2, \ldots, m\}$ such that for every $1 \leq i \leq n$, the i-th element of $\mathcal{V}$ is the column space of the submatrix of M induced by the columns in I_i.[4] Here, an *ordered partition* $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ of $\{1, 2, \ldots, m\}$ is a partition of $\{1, 2, \ldots, m\}$ such that $x < y$ for all $x \in I_i, y \in I_j$ with $i < j$.

Robertson and Seymour [12] introduced the notion of branch-width for graphs, hypergraphs, and more generally, for connectivity functions. We are going to define the branch-width of a subspace arrangement as follows. First, a tree is *subcubic* if every node has degree at most 3. We define a *leaf* of a tree as a node of degree at most 1. A *branch-decomposition* of $\mathcal{V}$ is a pair $(T, \mathcal{L})$ of a subcubic tree T with no degree-2 nodes and a bijective function $\mathcal{L}$ from the set of all leaves of T to $\mathcal{V}$. For a node v of T and an edge e incident with v, let us write $A_v(T - e)$ to denote the set of all leaves of T in the component of $T - e$ containing v. For a branch-decomposition $(T, \mathcal{L})$ of $\mathcal{V}$ and each edge $e = uv$ of T, we define the *width* of e to be $\dim\left(\sum_{x \in A_u(T-e)} \mathcal{L}(x)\right) \cap \left(\sum_{y \in A_v(T-e)} \mathcal{L}(y)\right)$. The *width* of $(T, \mathcal{L})$ is the maximum width of all edges of T. (If T has no edges, then the width of $(T, \mathcal{L})$ is 0.) The *branch-width* of $\mathcal{V}$ is the minimum k such that there exists a branch-decomposition of $\mathcal{V}$ having width at most k.

We aim to solve the following problem, and Theorem 1.1 is our main theorem.

BRANCH-WIDTH
Parameters: A finite field $\mathbb{F}$ and an integer k.
Input: An $r \times m$ matrix M over $\mathbb{F}$ with an ordered partition $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ of $\{1, 2, \ldots, m\}$ and an integer k.
Output: A branch-decomposition $(T, \mathcal{L})$ of width at most k of a subspace arrangement $\mathcal{V}$ consisting of the column space of the submatrix of M induced by the columns in I_i for each i or a confirmation that the branch-width of $\mathcal{V}$ is larger than k.

▶ **Theorem 1.1.** *Let $\mathbb{F}$ be a finite field, let r be a positive integer, and let k be a nonnegative integer. Let $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$ be a subspace arrangement of subspaces of $\mathbb{F}^r$ where each V_i is given by its spanning set of d_i vectors and $m = \sum_{i=1}^{n} d_i$. In time $O(rm^2 + (k+1)rmn + k^3n^3 + f(|\mathbb{F}|, k)n^2)$ for some function f, one can either find a branch-decomposition of $\mathcal{V}$ having width at most k or confirm that no such branch-decomposition exists.*

Various width parameters of discrete structures have been introduced and used for algorithmic and structural applications. One popular way of creating a width parameter of a discrete structure is to define it as the branch-width of some connectivity function defined on that discrete structure. Theorem 1.1 immediately gives rise to analogous algorithms for many of them, such as carving-width of graphs, rank-width of graphs, and branch-width of graphs, hypergraphs, and matroids. We will give a brief overview of each application.

Branch-width of matroids represented over a finite field $\mathbb{F}$. Let $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$ be a subspace arrangement of subspaces of $\mathbb{F}^r$. If each V_i is the span of a vector v_i in $\mathbb{F}^r$ for each $i = 1, 2, \ldots, n$, then $\mathcal{V}$ can be identified with the matroid M represented by the vectors v_1, v_2, …, v_n. Furthermore, branch-width and branch-decompositions of M are precisely branch-width and branch-decompositions of $\mathcal{V}$, respectively.

[4] Subspace arrangements can be regarded as representable *partitioned matroids* used in [6]. A partitioned matroid is a matroid equipped with a partition of its ground set.

Rank-width of graphs. Rank-width, introduced by Oum and Seymour [10], is a width parameter of graphs expressing how easy it is to decompose a graph into a tree-like structure, called a *rank-decomposition*, while keeping every edge cut to have a small 'complexity', called the *width* of a rank-decomposition, where the complexity is measured by the matrix rank function. Each vertex of a graph G can be associated with a subspace of dimension at most 2 so that the subspace arrangement $\mathcal{V}$ consisting of all subspaces associated with the vertices of G has branch-width $2k$ if and only if G has rank-width k (See appendix). Furthermore, a branch-decomposition of $\mathcal{V}$ of width $2k$ corresponds to a rank-decomposition of G of width k.

Branch-width of hypergraphs. Robertson and Seymour defined the notion of a branch-width [12] not only for graphs but also for hypergraphs. Let $\mathbb{F} = GF(2)$ be the binary field and let $\{v_1, v_2, \ldots, v_n\}$ be the standard basis of $\mathbb{F}^n$. For a hypergraph G with n vertices $v_1, v_2, \ldots, v_n$, we associate each edge e with the span of the vertices incident with e. Let $\mathcal{V}$ be the subspace arrangement consisting of all subspaces associated with the edges of G. Then it is not difficult to show that branch-width and branch-decomposition of G are precisely branch-width and branch-decomposition of $\mathcal{V}$, respectively.

Carving-width of graphs. Seymour and Thomas [13] introduced carving-width of graphs. Let $\mathbb{F} = GF(2)$ be the binary field and let $\{e_1, e_2, \ldots, e_m\}$ be the standard basis of $\mathbb{F}^m$. For a graph G with edges $e_1, e_2, \ldots, e_m$, we associate each vertex v with the span of the edges incident with v. If $\mathcal{V}$ is the subspace arrangement consisting of all subspaces associated with the vertices of G, then carving-width and carving of G are precisely branch-width and branch-decomposition of $\mathcal{V}$, respectively.

For the first two applications, the analogous theorems were proved earlier by Hliněný and Oum [6]. However, their approach was completely indirect; they use a non-trivial fact shown by Geelen et al. [3] that the class of matroids of branch-width at most k has finitely many forbidden minors, each having at most $O(6^k)$ elements. Then they use a monadic second-order formula to describe whether a matroid contains a fixed minor and use the dynamic programming algorithm to decide a monadic second-order formula aided by a given branch-decomposition of bounded width. So far this describes the decision algorithm of Hliněný [5] that decides whether branch-width is at most k. On top of this algorithm, Hliněný and Oum use a sophisticated reduction to modify the input and use the decision algorithm repeatedly to recover a branch-decomposition. Roughly speaking, this reduction attaches a gadget to the input matroid and this step requires extending the underlying finite field to an extension field, because this gadget is not representable if the underlying field is too small. As the list of forbidden minors is unknown, their algorithm should generate the list of minor-minimal matroids having branch-width larger than k. Thus, even for small values of k, it would be practically impossible to implement their algorithm. Contrary to the previous algorithm, our algorithm does not depend on the finiteness of obstructions and yet matches their asymptotic running time for each fixed k.

We do not know any previous analogous theorems for branch-width of hypergraphs. For branch-width of graphs, Thilikos and Bodlaender [14] posted a 50-page long technical report in 2000 proving that for every fixed k, one can check in linear time whether a graph has branch-width at most k and if so, output a branch-decomposition of minimum width. This work was presented at a conference in 1997 [2]. For carving-width of graphs, the conference paper of Thilikos, Serna, and Bodlaender [15] presented a linear-time algorithm for each fixed k that determines whether the carving-width of an input graph G is at most k, and if so, constructs a carving of G with minimum carving-width.

Our approach and technical ingredients. We develop a framework inspired by the approach of Bodlaender and Kloks [1] on their work on tree-width of graphs. (A similar framework was also given independently by Lagergren and Arnborg [9].) They created a linear-time algorithm that can find a tree-decomposition of width at most k or confirm that the tree-width of an input graph is larger than k for each fixed k. They used dynamic programming based on a given tree-decomposition of bounded width. For the dynamic programming, they designed a special encoding of all possible tree-decompositions of width at most k that can arise from certain parts of a graph.

We also use dynamic programming approach, taking advantage of having a tree-like structure from the given branch-decomposition. Then, how do we generate a branch-decomposition of small width in the first place? For this purpose, we use the technique called the iterative compression, which is initiated by Reed, Smith, and Vetta [11] and used by [4] for computing the branch-width of linear matroid. In Branch-Width Compression, we are given a branch-decomposition of width at most $2k$ of a subset of $\mathcal{V}$ and solve Branch-width. The obtained branch-decomposition of width at most k at each step is incremented by a new element of $\mathcal{V}$, which serves as a given branch-decomposition of width at most $2k$ for the next step.

To use a branch-decomposition for dynamic programming, we need a concept of a 'boundary', that plays the role of a bag in a tree-decomposition. For a branch-decomposition $(T, \mathcal{L})$ of a subspace arrangement $\mathcal{V}$ and an edge e of T, we consider the boundary B as the intersection of the sum of subspaces associated to the leaves in one component of $T - e$ and the sum of subspaces associated to the leaves of the other component of $T - e$. As the branch-width is at most k, the boundary B has dimension at most k. Furthermore, we restrict our attention to the finite field $\mathbb{F}$ and so the number of subspaces of B is finite. For the convenience of dynamic programming on branch-decompositions, we define *transcripts* of a branch-decomposition in Section 2, which is essentially a precomputed list of bases and linear transformations useful for computations with boundaries.

As usual, we need a compact encoding scheme to store partial solutions that may be extended to a branch-decomposition of width at most k, if it exists. We have two important aspects here.

First of all, we will restrict our search to a smaller set of branch-decompositions. Namely, if $(T', \mathcal{L}')$ is a given branch-decomposition of width at most $2k$ in Branch-Width Compression, then the algorithm will find a branch-decomposition of width at most k that is *totally pure with respect to* $(T', \mathcal{L}')$. In order to efficiently compress the partial solutions at each step of dynamic programming, it is crucial to ensure that some part of a partial solution can be forgotten (and can be retrieved from the unforgotten part later). A part of a partial solution can be ignored only when there is a guarantee that the said part does not need to be 'mixed' with another partial solution in the future. In other words, if there is a branch-decomposition of width at most k which is obtained via mixing, then there also exists as good a decomposition which can be obtained while mixing is avoided. This general idea lies at the core of every dynamic programming based on decomposition of small width. The first technical barrier to implement this principle for our problem is how to formalize what constitute those forgettable parts and what it means to avoid mixing. The 'forgettable part' is formalized as the notion of 'blocked' (plus some more) nodes introduced in Section 5. Intuitively, a totally pure branch-decomposition with respect to $(T', \mathcal{L}')$ is a decomposition which has successfully avoided the 'mixing' at every descendant so far. The two operations introduced in Section 3, fork and split, are technical tools developed in order to prove that it is possible to avoid mixing. Using these operations, we show that if there is a branch-decomposition of width at most k, then there is a branch-decomposition of width at

most k that is totally pure with respect to $(T', \mathcal{L}')$ in Section 3. The procedure of obtaining a compact encoding of a partial solution, introduced as *trimming* in Section 4, is essentially discarding the forgettable parts.

The second technical barrier is to devise an encoding of a (partial) branch-decomposition and all computational operations, necessary for dynamic programming, compatible with this encoding scheme. For this purpose, we will define a B-namu[5] in Section 4. Here, B is going to be the boundary for some edge in $(T', \mathcal{L}')$. A B-namu is, roughly speaking, a subcubic tree whose incidences are decorated by subspaces of B and whose edges are labeled by a nonnegative integer so that it represents the 'shadow' of a branch-decomposition of width at most k on B. We define an operation τ on B-namus that compresses a B-namu into a 'compact' B-namu and prove that there are only finitely many compact B-namus of width at most k, when B has bounded dimension and $\mathbb{F}$ is a finite field. This operation τ on B-namus consists of two steps: one is the aforementioned trimming, another is *compressing* to compress an integer sequence introduced by Bodlaender and Kloks [1] for their work on tree-decompositions. A part of B-namu processed by compressing step can be potentially mixed with another partial solution in the future, but a desired decomposition can be always retrieved if one exists. In contrast, a trimmed part is forgotten and never gets mixed in the future. Our computational operations on B-namus are designed so that the trimmed parts can be efficiently retrieved. Computational operations on B-namus for dynamic programming are defined including comparison.

Difference with our previous work. In the previous work [8], the authors found a similar algorithm for path-decompositions of a subspace arrangement. A path-decomposition of a subspace arrangement is a linearized variant of a branch-decomposition that restricts the subcubic trees to caterpillar trees. Here are the key technical differences.

First, the concept of totally pure branch-decompositions was not needed in [8]. In the previous work, two linear orderings are merged into another linear ordering and there is no need to consider the possibility of 'avoiding mixing'. In the end, compressing an integer sequence was sufficient to obtain a short encoding. For branch-decompositions, we sometimes insert a whole subtree into a branch-decomposition and this requests a new concept such as totally pure branch-decompositions. Also, 'summing' two partial solution encodings for join operation in dynamic programming is much more delicate in this work.

Second, we needed the concept of k-safeness in order to extend our algorithm for path-decompositions to the algorithm for branch-decompositions. When we sum two B-namus, some edges of trees in the B-namus are in common but some edges are not shared and will be forgotten. Although an edge in one B-namu is not in another B-namu, the width assigned to the edge can potentially increase. We hope the width of an edge not to exceed k even when this edge is 'forgotten', and thus we need to handle this carefully.

Lastly, we improve the running time of an algorithm computing transcripts. In [8], the idea of transcripts was used as well although the notion was not formally introduced. If we adapt our new method to the result of [8], we also get an $O(n^3)$-time algorithm for path-decompositions of subspace arrangements based on iterative compressions, saving a factor of $O(n)$.

Section 2, Section 3, and Section 4 give both definitions and some properties of transcripts, totally pure branch-decompositions, and B-namus, respectively. Section 5 presents the algorithm to solve the Branch-Width problem. We remark that many proofs are omitted because of the page limit. However, the detailed proofs are contained in the full version.

[5] 'Namu' is a tree in Korean.

2 Transcripts

Dynamic programming algorithms on tree-decomposition benefit from the small width by encoding solutions with respect to the bags. While the bags are explicit in a given tree-decomposition, a branch-decomposition of a subspace arrangement does not provide an easy-to-handle metric for encoding solutions to our problem. In order to make it more useful, we need some extra information.

Let $(T, \mathcal{L})$ be a branch-decomposition of a subspace arrangement $\mathcal{V}$. We will assume that T is a rooted binary tree by picking an arbitrary edge e and subdividing e to create a degree-2 root node r, and call $(T, \mathcal{L})$ a *rooted* branch-decomposition. For a node v of T, let $\mathcal{V}_v$ be the set of all elements of $\mathcal{V}$ associated with v and its descendants by $\mathcal{L}$. For a set X of vectors from a vector space over a field $\mathbb{F}$, the *span* $\langle X \rangle$ of X is the subspace consisting of all (finite) linear combinations of vectors in X, where the scalars are taken from $\mathbb{F}$. For two subspaces X and Y, we denote the subspace $\{x+y : x \in X, y \in Y\}$ by $X+Y$. For a set $\mathcal{X}$ of subspaces, let $\langle \mathcal{X} \rangle = \sum_{X \in \mathcal{X}} X$. The *boundary space* B_v at v is defined as $B_v = \langle \mathcal{V}_v \rangle \cap \langle \mathcal{V} - \mathcal{V}_v \rangle$. Later, we shall encode partial branch-decompositions with respect to the boundary spaces of a given branch-decomposition $(T, \mathcal{L})$. For this, we need to know B_v in advance.

A *transcript* of $(T, \mathcal{L})$ is a pair $\Lambda = (\{\mathfrak{B}_v\}_{v \in V(T)}, \{\mathfrak{B}'_v\}_{v \in V(T)})$ of sets of ordered bases $\mathfrak{B}_v$ and $\mathfrak{B}'_v$ of subspaces $B_v = \langle \mathfrak{B}_v \rangle$ and $B'_v = \langle \mathfrak{B}'_v \rangle$ of $\mathbb{F}^r$, respectively, such that

- the first $|\mathfrak{B}_v|$ elements of $\mathfrak{B}'_v$ are precisely $\mathfrak{B}_v$ for each node v,
- $\langle \mathfrak{B}'_v \rangle = \langle \mathfrak{B}_{w_1} \rangle + \langle \mathfrak{B}_{w_2} \rangle$ for each node v having two children w_1 and w_2,
- $\langle \mathfrak{B}'_v \rangle = \langle \mathfrak{B}_v \rangle$ for each leaf v.

If a node u of T is a parent of a node v of T, then $B_v = \langle \mathfrak{B}_v \rangle$ is a subspace of $B'_u = \langle \mathfrak{B}'_u \rangle$ and therefore there exists the unique $|\mathfrak{B}'_u| \times |\mathfrak{B}_v|$ matrix T_v over $\mathbb{F}$ such that $T_v[x]_{\mathfrak{B}_v} = [x]_{\mathfrak{B}'_u}$ for all $x \in \mathfrak{B}_v$. This matrix T_v is called the *transition matrix* of Λ at a node v. (For the root node r, let T_r be the null matrix. For a vector x in a vector space with a basis $\mathfrak{B}$ over a field $\mathbb{F}$, $[x]_{\mathfrak{B}}$ denotes the coordinate vector with respect to the basis $\mathfrak{B}$, which is a $|\mathfrak{B}| \times 1$ matrix over $\mathbb{F}$.)

We can compute the transcript of a given branch-decomposition as follows.

▶ **Theorem 2.1.** *Let $\mathcal{V}$ be a subspace arrangement of $\mathbb{F}^r$ represented by an $r \times m$ matrix M in reduced row echelon form with no zero rows such that each $V \in \mathcal{V}$ has dimension at most k. Let $n = |\mathcal{V}|$. Given branch-decomposition $(T, \mathcal{L})$ of $\mathcal{V}$, in time $O(k^3 n^2)$, one can correctly compute a basis of $\langle \mathcal{V}_v \rangle \cap \langle \mathcal{V} - \mathcal{V}_v \rangle$ for all nodes v of T or confirm that $(T, \mathcal{L})$ has width larger than k. In addition, if $(T, \mathcal{L})$ has width at most k, then we can compute the transcript $\Lambda = (\{\mathfrak{B}_v\}, \{\mathfrak{B}'_v\})$ of $(T, \mathcal{L})$ with its transition matrices in time $O(k^3 n^2)$.*

3 Pure branch-decompositions

We are going to assume that a subspace arrangement $\mathcal{V}$ and its rooted branch-decomposition $(T^b, \mathcal{L}^b)$ are given. For two nodes x, y of T^b, we say that $x \leq y$ if either $x = y$ or x is a descendant of y. We write $x < y$ if $x \leq y$ and $x \neq y$. For a node x of T^b, let $\mathcal{V}_x$ be the set of all subspaces $\mathcal{L}^b(\ell)$ where ℓ is a leaf of T^b with $\ell \leq x$ and let $B_x = \langle \mathcal{V}_x \rangle \cap \langle \mathcal{V} - \mathcal{V}_x \rangle$. Let $(T, \mathcal{L})$ be a branch-decomposition[6] of $\mathcal{V}_0 \subseteq \mathcal{V}$. Let x be a node of T^b such that $\mathcal{V}_x \subseteq \mathcal{V}_0$. We define $\mathcal{L}(T, u, v) = \{\mathcal{L}(w) : w \in A_v(T - uv)\}$ and write $\mathcal{L}_x(T, u, v) = \mathcal{L}(T, u, v) \cap \mathcal{V}_x$. (We remark that $\mathcal{L}(T, u, v)$ is a set of subspaces and $\langle \mathcal{L}(T, u, v) \rangle$ is the sum of members of $L(T, u, v)$.)

[6] One may consider $(T, \mathcal{L})$ as a (partial) solution whereas $(T^b, \mathcal{L}^b)$ is the given branch-decomposition over which dynamic programming is executed.

(a) T **(b)** T'

Figure 1 Constructing T' by forking at v by $\mathcal{V}_x$. ■ represents a leaf node mapped to an element of $\mathcal{V}_x$ by $\mathcal{L}$ and ○ represents a leaf node mapped to an element of $\mathcal{V}_0 - \mathcal{V}_x$ by $\mathcal{L}$.

We say that an edge uv of T *x-guards* its end v if $\langle \mathcal{L}_x(T,u,v)\rangle \cap B_x \subsetneq \langle \mathcal{L}_x(T,v,u)\rangle \cap B_x$. An edge uv of T is *x-degenerate* if $\langle \mathcal{L}_x(T,u,v)\rangle \cap B_x = \langle \mathcal{L}_x(T,v,u)\rangle \cap B_x$. A 2-edge path uvw of T is an *x-blocking path* if $\langle \mathcal{L}_x(T,u,v)\rangle \cap B_x = \langle \mathcal{L}_x(T,v,w)\rangle \cap B_x$, $\langle \mathcal{L}_x(T,w,v)\rangle \cap B_x = \langle \mathcal{L}_x(T,v,u)\rangle \cap B_x$, and neither uv nor vw is x-degenerate or x-guarding.

We give a general idea behind the notion of totally pure branch-decompositions. For $\mathcal{V}_x \subseteq \mathcal{V}_0$, one can consider the branch-decomposition 'induced' by $\mathcal{V}_x$ from $(T,\mathcal{L})$; such a branch-decomposition can be canonically defined by choosing a minimal subtree of T whose leaf set is mapped to $\mathcal{V}_x$ by $\mathcal{L}$ (and smoothing degree-2 nodes if necessary). Similarly, the branch-decomposition 'induced' by $\mathcal{V}_0 - \mathcal{V}_x$ can be obtained. Let $(T_x,\mathcal{L}_x)$ and $(T_{\bar{x}},\mathcal{L}_{\bar{x}})$ be the respective branch-decompositions. If uvw is an x-blocking path of T_x, then it can be shown that the connected component of $T_x - uv - vw$ containing v does not need to be mixed with another branch-decomposition in the future. Specifically, if the subtree of T homeomorphic to T_x is 'mixed' with some subtree of $T_{\bar{x}}$ in T, then one can 'untangle' the mixing: one 'lifts' the former subtree and 'plants' it on the x-blocking path uvw (so as to be rooted at a new node subdividing vw provided $\dim\langle \mathcal{L}_x(T,u,v)\rangle \cap \langle \mathcal{L}_x(T,v,u)\rangle \geq \dim\langle \mathcal{L}_x(T,v,w)\rangle \cap \langle \mathcal{L}_x(T,w,v)\rangle)$. This operation on $(T,\mathcal{L})$ is called the *forking* (see Figure 1). It can be proved that forking operations under above assumption do not increase the width. This is why we can 'forget' a subtree of T_x, namely the subtree of $T_x - uv - vw$ containing v. A similar observation can be made in regards to x-guarding edges, for which the related operation is *splitting*.

Then how do we know whether there is unwanted mixing in regards to an x-guarding edge or an x-blocking path? The following notions formalize this. An edge uv of T that x-guards v is called *improper* x-guarding if v has two neighbors v_1, v_2 in $T - uv$ such that $\mathcal{L}_x(T,v,v_1)$, $\mathcal{L}_x(T,v,v_2)$, and $\mathcal{L}(T,u,v) \cap (\mathcal{V}_0 - \mathcal{V}_x)$ are nonempty. An x-blocking path uvw of T is *improper* if v has a neighbor t in $T - u - w$ such that $\mathcal{L}_x(T,v,u)$, $\mathcal{L}_x(T,v,w)$, $\mathcal{L}_x(T,v,t)$, and $\mathcal{L}(T,v,t) \cap (\mathcal{V}_0 - \mathcal{V}_x)$ are nonempty.

When T has an x-degenerate edge e, it turns out that we can apply splitting operations at any x-degenerate edge and untangle T_x and $T_{\bar{x}}$ so that the new branch-decomposition is a disjoint union of T_x and $T_{\bar{x}}$ connected by a single edge (which will be incident with a subdividing node of the x-degenerate edge e). Hence, it is conceivable that we might be able to forget all nodes of T_x, possibly except for one node as a placeholder representing T_x. In this way, we request that any extension of T_x in the future shall be in the form of disjoint union plus one edge. However, it is possible that e is also a z-degenerate edge for some $z < x$. In this case, forcing the join of T_x and $T_{\bar{x}}$ at a subdividing node of e can violate the disjointness of T_z and $T_{\bar{z}}$. We want to prevent this, and it leads us to the definition of x-degenerate branch-decompositions.

For $\mathcal{S} \subseteq \mathcal{V}_0$, an edge uv of T is said to *cut* $\mathcal{S}$ if $\mathcal{L}(T,u,v) \cap \mathcal{S} \neq \emptyset$ and $\mathcal{L}(T,v,u) \cap \mathcal{S} \neq \emptyset$. We say that $(T,\mathcal{L})$ is *x-degenerate* if T has an x-degenerate edge uv such that uv cuts $\mathcal{V}_x$ and for all $z < x$, if $(T,\mathcal{L})$ is z-degenerate, then uv does not cut $\mathcal{V}_z$. Such an edge uv is called *improper* x-degenerate. Note that if x is a leaf of T^b, then $(T,\mathcal{L})$ is not x-degenerate because T has no edge cutting $\mathcal{V}_x$. We say that $(T,\mathcal{L})$ is *x-disjoint* if $\mathcal{V}_0 = \mathcal{V}_x$ or T has an edge uv such that $\mathcal{L}(T,u,v) = \mathcal{V}_x$ and v is incident with an improper x-degenerate edge.

We say that $(T, \mathcal{L})$ is *x-pure* if the following hold.

- If $(T, \mathcal{L})$ is x-degenerate, then $(T, \mathcal{L})$ is x-disjoint.
- If $(T, \mathcal{L})$ is not x-degenerate, then all x-blocking paths and all x-guarding edges of T are not improper.

We say that a branch-decomposition $(T, \mathcal{L})$ of $\mathcal{V}_0$ is *totally pure with respect to* $(T^b, \mathcal{L}^b)$ if $(T, \mathcal{L})$ is x-pure for all nodes x of T^b with $\mathcal{V}_x \subseteq \mathcal{V}_0$. We prove that if the branch-width of a subspace arrangement is at most k, then there exists a totally pure branch-decomposition of the subspace arrangement whose width is at most k. The proof strategy is to apply forking and splitting operations for every node x of T^b in a bottom-up manner. If $(T, \mathcal{L})$ is x-degenerate, then applying the operations will create a new branch-decomposition which is x-disjoint. If it is not x-degenerate, then the operations will resolve entanglements at the improper x-guarding edges and at the improper x-blocking paths so that no improper ones are left. For this approach to work, we need to ensure that applying these operations do not create new entanglements at nodes z of T^b where disentanglement already happened (or is happening now). That is, z-disjointness is preserved, and no new improper z-guarding edge or z-blocking path is created.

▶ **Proposition 3.1.** *Let $(T^b, \mathcal{L}^b)$ be a rooted branch-decomposition of a subspace arrangement $\mathcal{V}$ and let $\mathcal{V}_0 \subseteq \mathcal{V}$. If the branch-width of $\mathcal{V}_0$ is at most k, then $\mathcal{V}_0$ has a branch-decomposition of width at most k that is totally pure with respect to $(T^b, \mathcal{L}^b)$.*

4 Namus

Let $\mathbb{F}$ be a finite field and let B be a subspace of $\mathbb{F}^r$ of dimension θ. In this section, we introduce the data structure for encoding partial solutions and operations on this data structure required for dynamic programming. For a tree T, an *incidence* is a pair (v, e) of a node v of T and an edge e incident with v. Let $\mathcal{I}(T)$ be the union of $\{(*, \emptyset), (0, \emptyset)\}$ and the set of all incidences of T. A *B-namu* Γ is a quadruple (T, α, λ, U) of

- a subcubic tree T having at least one node,
- a function α from $\mathcal{I}(T)$ to the set of all subspaces of B,
- a function λ from the union of $\{\emptyset\}$ and the set of all edges of T to the set of integers, and
- a subspace U of B

such that

- **(i)** for every two-edge path v_0, e_1, v_1, e_2, v_2 in T, $\alpha(v_0, e_1)$ is a subspace of $\alpha(v_1, e_2)$,
- **(ii)** for all incidences (v, e) of T, $\alpha(v, e)$ is a subspace of U,
- **(iii)** $\alpha(*, \emptyset) = U$, $\alpha(0, \emptyset) = \{0\}$, and $\lambda(\emptyset) = 0$,
- **(iv)** for every edge $e = uv$ of T, $\lambda(e) \geq \dim \alpha(v, e) \cap \alpha(u, e)$.

The *width* of a B-namu $\Gamma = (T, \alpha, \lambda, U)$ is the maximum of $\lambda(e)$ over all edges $e = uv$ of T. (If T has no edges, then the width of Γ is defined to be 0.) For a B-namu $\Gamma = (T, \alpha, \lambda, U)$, we write $T(\Gamma) = T$.

Canonical B-namus. The *canonical B-namu* of a branch-decomposition $(T, \mathcal{L})$ of a subspace arrangement $\mathcal{V}$ is the B-namu (T, α, λ, U) such that

- $\alpha(v, e) = B \cap \sum_{x \in A_v(T-e)} \mathcal{L}(x)$ for each node v of T and an edge e incident with v,
- $\lambda(e) = \dim \sum_{x \in A_u(T-e)} \mathcal{L}(x) \cap \sum_{y \in A_v(T-e)} \mathcal{L}(y)$ for each edge $e = uv$ of T,
- $U = B \cap \sum_{x \in A(T)} \mathcal{L}(x)$ where $A(T)$ is the set of all leaves of a tree T.

Projections. For two subspaces B and B' with $B' \subseteq B$, we define the *projection* $\Gamma|_{B'}$ of a B-namu $\Gamma = (T, \alpha, \lambda, U)$ on B' as the B'-namu $(T, \alpha', \lambda', U')$ such that $U' = U \cap B'$, $\alpha'(v, e) = \alpha(v, e) \cap B'$ for all incidences (v, e) of T, and $\lambda'(e) = \lambda(e)$ for all edges e of T.

Compact B-namus. We have two operations, trimming and compressing, on B-namus, which will transform a B-namu into a 'compact' B-namu. Roughly speaking, *trimming* is an operation to remove irrelevant edges and *compressing* is an operation to suppress redundant edges. Irrelevant edges are those edges which can be 'untangled' by forking or splitting operations as described in Section 3. As addressed in Proposition 3.1, there exists a branch-decomposition of minimum width that is totally pure with respect to $(T^b, \mathcal{L}^b)$. Such a branch-decomposition is minimally 'mixed' at every node x of T^b in the sense that forking and splitting operations has been fully applied at every x without causing additional mixing. The idea behind trimming is that, for keeping track of totally pure branch-decompositions in the dynamic programming algorithm, those edges to be untangled by forking or splitting operations can be ignored. For a B-namu Γ, let $\mathrm{trim}(\Gamma)$ denote the B-namu obtained by trimming Γ.

For a node x of $(T^b, \mathcal{L}^b)$, can we bound the size of an arbitrary trimmed B_x-namu Γ, namely the size of $T(\Gamma)$? As $T(\Gamma)$ is a subcubic tree, bounding the size of $T(\Gamma)$ is equivalent to bounding the diameter of $T(\Gamma)$. By condition (i) in the definition of B-namu, it is not difficult to see that $T(\Gamma)$ has a large diameter if and only if $T(\Gamma)$ contains a long path in which any length-two path is x-blocking. Assuming that Γ is trimmed, such a long path induces a substructure in which every internal node has degree two and α maps every incidence to the same subspace. That is, the information on such a path dictated by Γ is almost uniform except that the values of λ changes over the edges and the values of λ can be viewed as an integer sequence. Now, the idea of compressing operation is to keep only the edges associated with local minimum and maximum values of this integer sequence and ignore all other edges. An integer sequence obtained in this way is called a *typical sequence* in the literature [1] and it is known to have length at most $2k + 1$ when the integers are in the range $\{0, \ldots, k\}$.

We say that a B-namu is *compact* if it contains no 'irrelevant' nodes or edges so that trimming or compressing does not affect to the B-namu. Let $U_k(B)$ be the set of all compact B-namus Γ of width at most k such that $V(T(\Gamma)) = \{1, 2, \ldots, n\}$ for some integer n. The previous discussion is summarized in the next statement, which ensures in Section 5 that the number of partial solutions stored at each node of $(T^b, \mathcal{L}^b)$ for the dynamic programming algorithm will be bounded.

▶ **Lemma 4.1.** *The set $U_k(B)$ contains at most $f(k, \theta, |\mathbb{F}|)$ elements and can be generated from B in $g(k, \theta, |\mathbb{F}|)$ steps for some functions f and g.*

Sum of two B-namus. For two B-namus $\Gamma_1 = (T_1, \alpha_1, \lambda_1, U_1)$ and $\Gamma_2 = (T_2, \alpha_2, \lambda_2, U_2)$, we define a *sum* $(T, \alpha, \lambda, U_1 + U_2)$ of Γ_1 and Γ_2. Roughly speaking, we first take a tree T such that a subdivision of T_1 is a subtree of T and a subdivision of T_2 is a subtree of T (see Figure 2). For each incidence (v, e) of T, if it corresponds to both an incidence (v_1, e_1) of T_1 and an incidence (v_2, e_2) of T_2, then $\alpha(v, e)$ is the sum of $\alpha_1(v_1, e_1)$ and $\alpha_2(v_2, e_2)$, and if it corresponds to only one of (v_1, e_1) and (v_2, e_2), say (v_1, e_1), then $\alpha(v, e) = \alpha_1(v_1, e_1)$. Similarly, we can define λ on every edge of T (with some correction term). The formal definition is given in the full version. Note that a sum of two B-namus is not unique because there are many choices of taking a tree T. Given B-namus Γ_1 and Γ_2, let us denote by $\Gamma_1 \oplus \Gamma_2$ the set of all sums of Γ_1 and Γ_2.

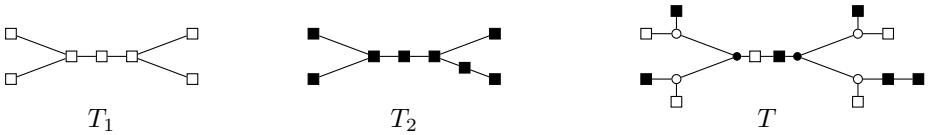

Figure 2 Obtaining a sum of two B-namus.

▶ **Lemma 4.2.** *Let Γ_1, Γ_2 be compact B-namus of width at most k. Then the set $\Gamma_1 \oplus \Gamma_2$ contains at most $2^{2^{2(2\theta k+\theta+k+3)}}$ B-namus.*

Comparing two B-namus. A B-namu $(T', \alpha', \lambda', U)$ is a *subdivision* of a B-namu $\Gamma = (T, \alpha, \lambda, U)$ if T' is a subdivision of T, $\alpha'(v', e') = \alpha(v, e)$, and $\lambda'(e') = \lambda(e)$ for every incidence (v', e') of T' and its corresponding incidence (v, e) of T.

For two B-namus $\Gamma_1 = (T_1, \alpha_1, \lambda_1, U_1)$ and $\Gamma_2 = (T_2, \alpha_2, \lambda_2, U_2)$, we say that $\Gamma_1 \leq \Gamma_2$ if $T_1 = T_2$, $\alpha_1 = \alpha_2$, $U_1 = U_2$ and $\lambda_1(e) \leq \lambda_2(e)$ for every edge e of T_1. For two B-namus Γ_1 and Γ_2, we say that $\Gamma_1 \preccurlyeq \Gamma_2$ if there exist a subdivision Γ_1' of Γ_1 and a subdivision Γ_2' of Γ_2 such that $\Gamma_1' \leq \Gamma_2'$.

▶ **Lemma 4.3.** *For two B-namus Δ and Γ, we can decide whether $\Delta \preccurlyeq \Gamma$ by executing at most $f(|V(T(\Delta))|, |V(T(\Gamma))|, \theta, |\mathbb{F}|)$ comparison operations (on integers and on subspaces of B) for some function f.*

5 The algorithm

We present an algorithm to solve the Branch-Width problem. Given a matrix M and a set Y of column indices, $M[Y]$ denotes the submatrix of M induced by columns indexed by Y.

Preprocessing. We will first describe the preprocessing steps to reduce the input size. The subspace arrangement of n subspaces is given by an $r \times m$ matrix where r and m could be arbitrary large. Our aim here is to reduce r and m or confirm that branch-width is larger than k. Eventually, we will convert the input into a smaller one. Furthermore, we will convert M into the reduced row echelon form, which is crucial for our algorithm for computing the transcript of a branch-decomposition in Theorem 2.1. In the Branch-Width problem, if a branch-decomposition of width at most k exists, then we say that $(M, \mathcal{I}, k)$ is a YES instance. Otherwise, it is a NO instance. For a matrix M, let $\mathrm{col}(M)$ be the column space of M, that is the span of all column vectors of M.

▶ **Lemma 5.1.** *Let $\mathbb{F}$ be a finite field and let k be a nonnegative integer. Let $n \geq 2$. Let M be an $r \times m$ matrix over $\mathbb{F}$ with an ordered partition $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ of $\{1, 2, \ldots, m\}$ and let V_i be the column space of $M[I_i]$ for every i. In time $O(rm^2 + (k+1)rmn)$, we can either find $i \in \{1, 2, \ldots, n\}$ such that $\dim(V_i \cap (\sum_{j \neq i} V_i)) > k$ or find an $r' \times m'$ matrix M' over $\mathbb{F}$ with an ordered partition $\mathcal{I}' = \{I_1', I_2', \ldots, I_n'\}$ of $\{1, 2, \ldots, m'\}$ such that*

- **(i)** *$r' \leq m' \leq kn$,*
- **(ii)** *M' is of the reduced row echelon form with no zero rows,*
- **(iii)** *for each i, the column vectors of $M'[I_i']$ are linearly independent and $|I_i'| \leq k$,*
- **(iv)** *for each i, $\mathrm{col}(M'[I_i']) \subseteq \mathrm{col}(M'[\{1, 2, \ldots, m'\} - I_i'])$,*
- **(v)** *$(M, \mathcal{I}, k)$ is a YES instance with a branch-decomposition $(T, \mathcal{L})$ if and only if $(M', \mathcal{I}', k)$ is a YES instance with $(T, \mathcal{L}')$ where $\mathcal{L}'$ maps a leaf v to $\mathrm{col}(M'[I_i'])$ whenever $\mathcal{L}$ maps v to $\mathrm{col}(M[I_i])$.*

The full set. Our dynamic programming algorithm constructs a set of compact B_x-namus of width at most k at each node x of the given branch-decomposition $(T^b, \mathcal{L}^b)$ in a bottom-up manner. This set is called the *full set at x of width k with respect to* $(T^b, \mathcal{L}^b)$ and written as $\mathrm{FS}_k(x; T^b, \mathcal{L}^b)$ or $\mathrm{FS}_k(x)$ for brevity. The full set needs to be defined so that at the root node $\mathfrak{r}$ of T^b, $\mathrm{FS}_k(\mathfrak{r}) \neq \emptyset$ if and only if the branch-width of $\mathcal{V}$ is at most k. Moreover, we need to ensure that the full set at every node of T^b can be constructed from the full sets of its children.

Roughly speaking, the full set at x is an upward closed set of $(U_k(B_x), \preccurlyeq)$, where $U_k(B_x)$ and $\preccurlyeq$ are defined in the previous section. Every minimal[7] element in this upward closed set is a B-namu that can be obtained from a branch-decomposition $(T, \mathcal{L})$ of $\mathcal{V}_x$ having width at most k that is totally pure with respect to $(T^b, \mathcal{L}^b)$ by discarding some subtrees of T. This concept is captured in the notion of *reduced B-namus* below. Any discarded subtrees keep the rest of T connected. The subtrees of T that will not be mixed with another partial solution shall qualify as the disposable parts. Two types of disposable parts arise: one is a subtree consisting of x-blocked nodes (which will be defined soon), and the other type is a subtree whose entire leaf set is precisely mapped with $\mathcal{V}_z$ for some $z \leq x$ such that $(T, \mathcal{L})$ is z-degenerate. In particular, if $(T, \mathcal{L})$ is x-degenerate, then we discard all nodes except one node. Because we only consider totally pure branch-decompositions $(T, \mathcal{L})$, if $(T, \mathcal{L})$ is z-degenerate for some $z < x$, then $(T, \mathcal{L})$ is a disjoint union of $\mathcal{V}_z$ and $\mathcal{V}_x - \mathcal{V}_z$ joined via a single edge. It is intuitively easy to understand that we want to keep the subtree containing $\mathcal{V}_z$ intact from any mixing in the future and thus want to discard this part. However, implementing this idea with full technical details is quite tricky.

Moreover, not every reduced B-namu obtained in this way can be a member of a full set. A technical condition called *k-safeness* must be met by any edge that gets discarded. This condition is expressed as an inequality, indicating that when the current partial solution grows into a branch-decomposition for $\mathcal{V}$, the width at the forgotten edge is at most k.

For a B-namu $\Gamma = (T, \alpha, \lambda, U)$ and a subtree T' of T, we say that a B-namu $\Gamma' = (T', \alpha', \lambda', U)$ is *induced by T' from* Γ if $\alpha'(v, e) = \alpha(v, e)$ and $\lambda'(e) = \lambda(e)$ for every incidence (v, e) of T'. For a node x of T^b and a branch-decomposition $(T, \mathcal{L})$ of $\mathcal{V}_x$ which is totally pure with respect to $(T^b, \mathcal{L}^b)$, we obtain the *reduced B_x-namu of* $(T, \mathcal{L})$ induced by a subtree T' of T from the canonical B_x-namu of $(T, \mathcal{L})$ where T' is obtained by the following rule.

- If $(T, \mathcal{L})$ is x-degenerate, then T' is a subtree having only one node of T.
- If $(T, \mathcal{L})$ is not x-degenerate, then T' is obtained by deleting every node w if
  - **(i)** there is an x-blocking path $v_1 v v_2$ centered at $v \neq w$ such that there is a path from w to v in $T - vv_1 - vv_2$, or
  - **(ii)** some edge uv x-guards $v \neq w$ such that there is a path from w to v in $T - uv$, or
  - **(iii)** w is a node of a subtree T'' of T with a root $v \neq w$ having two children such that the leaf set of T'' is precisely mapped to $\mathcal{V}_z$ for some $z < x$ and $(T, \mathcal{L})$ is z-degenerate.

A node w satisfying (i) or (ii) is said to be *x-blocked*. A branch-decomposition $(T, \mathcal{L})$ is *k-safe with respect to x* if for every edge uv of T which is not contained in T',

$$\dim \sum_{s \in A_v(T-uv)} \mathcal{L}(s) \cap \sum_{t \in A_u(T-uv)} \mathcal{L}(t) + \dim B_x - \dim B_x \cap \sum_{t \in A_u(T-uv)} \mathcal{L}(t) \leq k.$$

Now, for each node x of T^b, *the full set at x of width k with respect to* $(T^b, \mathcal{L}^b)$ is defined

[7] Technically, a minimal element in our definition might not be a compact B_x-namu but just a trimmed B_x-namu. However, a compact B-namu of the minimal element defines the same upper set due to the transitivity of $\preccurlyeq$.

as the set of all Γ in $U_k(B_x)$ such that $\Delta \preccurlyeq \Gamma$ for the reduced B_x-namu Δ of some branch-decomposition $(T, \mathcal{L})$ of $\mathcal{V}_x$ having width at most k that is k-safe with respect to x, and totally pure with respect to $(T^b, \mathcal{L}^b)$.

Dynamic programming. For computing the full sets, we assume the following are given:

- A rooted branch-decomposition $(T^b, \mathcal{L}^b)$ of $\mathcal{V}$ of width at most θ.
- A set of transition matrices $\{T_v\}_{v \in V(T^b)}$ of some transcript Λ of $(T^b, \mathcal{L}^b)$.

A B-namu $\Gamma = (T, \alpha, \lambda, U)$ is a *k-safe extension of* $\mathrm{trim}(\Gamma)$ if for every edge uv in $E(T) - E(\mathrm{trim}(T))$, we have $\lambda(uv) + \dim U - \max(\dim \alpha(v, uv), \dim \alpha(u, uv)) \leq k$. For two sets $\mathcal{R}_1$, $\mathcal{R}_2$ of B-namus, we define $\mathcal{R}_1 \oplus \mathcal{R}_2$ as the set $\bigcup_{\Gamma_1 \in \mathcal{R}_1, \Gamma_2 \in \mathcal{R}_2} \Gamma_1 \oplus \Gamma_2$. Note that for two children x_1, x_2 of a node x in T^b, when we compute $\mathrm{FS}_k(x_1) \oplus \mathrm{FS}_k(x_2)$, we regard $\mathrm{FS}_k(x_1)$, $\mathrm{FS}_k(x_2)$ as the sets of $(B_{x_1} + B_{x_2})$-namus. Thus, $\mathrm{FS}_k(x_1) \oplus \mathrm{FS}_k(x_2)$ is well defined. If B' is a subspace of B, then we define $\mathcal{R}|_{B'}$ as the set of projections $\Gamma|_{B'}$ for all $\Gamma \in \mathcal{R}$. For a subspace B of $\mathbb{F}^r$ and a set $\mathcal{R}$ of B-namus, the set $\mathsf{up}_k(\mathcal{R}, B)$ is the collection of all B-namus $\Gamma \in U_k(B)$ with $\mathrm{trim}(\Gamma') \preccurlyeq \Gamma$ for some $\Gamma' \in \mathcal{R}$ such that Γ' is a k-safe extension of $\mathrm{trim}(\Gamma')$.

▶ **Proposition 5.2.** *Let k be a nonnegative integer. Let $(T^b, \mathcal{L}^b)$ be a rooted branch-decomposition of a subspace arrangement $\mathcal{V}$ over a finite field $\mathbb{F}$ of width at most θ.*

- *For a leaf ℓ of T^b, we have $\mathrm{FS}_k(\ell) = \{\Delta_\ell\}$ where $\Delta_\ell = (T, \alpha, \lambda, B_\ell)$ is the B_ℓ-namu such that T is a tree with $V(T) = \{1\}$.*
- *For two children x_1 and x_2 of a node x in T^b, $\mathrm{FS}_k(x) = \mathsf{up}_k((\mathrm{FS}_k(x_1) \oplus \mathrm{FS}_k(x_2))|_{B_x}, B_x)$.*
- *For the root node $\mathfrak{r}$ of T^b, $\mathrm{FS}_k(\mathfrak{r}) \neq \emptyset$ if and only if the branch-width of $\mathcal{V}$ is at most k.*

Moreover, we can compute $\mathrm{FS}_k(\mathfrak{r})$, and construct a (rooted) branch-decomposition of $\mathcal{V}$ of width at most k if $\mathrm{FS}_k(\mathfrak{r}) \neq \emptyset$, in time $f(k, \theta, |\mathbb{F}|)|\mathcal{V}|$ for some function f.

Proposition 5.2 states that when $\mathrm{FS}_k(x) \neq \emptyset$, the same B-namu or a better one can be constructed by conducting operations on B-namus in the full set at its child nodes. Therefore, when $\mathrm{FS}_k(\mathfrak{r}) \neq \emptyset$, one can identify a B-namu Γ_x at each node x of T^b which participates in the construction of the element in $\mathrm{FS}_k(\mathfrak{r})$. Additionally, we have the information including how Γ_x's are combined and how the combined B-namu is related to the B-namu at its parent. We construct a branch-decomposition of $\mathcal{V}$ having width at most k by backtracking based on such information. However, proving the correctness of this backtracking algorithm is highly nontrivial. For this, we introduce the notion of *witnesses*. Details are in the full version.

Summary. Now we are ready to present the proof of Theorem 1.1.

Proof of Theorem 1.1. We preprocess the input by applying Lemma 5.1 to $(M, \mathcal{I}, k)$ in time $O(rm^2 + (k+1)rmn)$ and obtain an equivalent instance $(M', \mathcal{I}', k)$ as described in Lemma 5.1 and otherwise, we confirm that the branch-width of $\mathcal{V}$ exceeds k. We may assume that $k > 0$ because if $I'_i = \emptyset$ for all i, then every branch-decomposition has width 0. Henceforth, we assume $M = M'$, $\mathcal{I} = \mathcal{I}'$, $V_i = \mathrm{col}(M'[I'_i])$ to simplify notations.

We may also assume that $\dim V_i \neq 0$ for all i because otherwise we delete all such V_i and later we can extend a branch-decomposition of $\mathcal{V} - \{V_i\}$ to that of $\mathcal{V}$ of the same width. After the preprocessing, if $n = 1$, then an arbitrary branch-decomposition has width 0 and so we simply output an arbitrary branch-decomposition of $\mathcal{V}$. If $n = 2$, then the branch-width is at most k because $\dim V_1, \dim V_2 \leq k$ by (iii) of Lemma 5.1. So we may assume that $n \geq 3$.

We will apply iterative compression on $\mathcal{V}_i = \{V_1, \ldots, V_i\}$ for $i = 3, \ldots, n$. We initially start with a trivial branch-decomposition $(T_2, \mathcal{L}_2)$ of $\mathcal{V}_2 = \{V_1, V_2\}$ having width at most k. We carry out a COMPRESSION STEP for each $i = 3, \ldots, n$ as follows.

(1) By adding a new leaf v to T_{i-1} and extending $\mathcal{L}_{i-1}$ to map v to V_i, we create a branch-decomposition $(T'_i, \mathcal{L}'_i)$ of $\mathcal{V}_i$. Note that the width of $(T'_i, \mathcal{L}'_i)$ is at most $2k$ because $(T_{i-1}, \mathcal{L}_{i-1})$ has width at most k and V_i has dimension at most k.

(2) We use the algorithm in Theorem 2.1 to compute transition matrices $\{T_v\}_{v \in V(T'_i)}$ of the transcript for $(T'_i, \mathcal{L}'_i)$ in time $O(k^3 n^2)$. Note that the submatrix $M[I_1 \cup I_2 \cup \cdots \cup I_i]$ is in reduced row echelon form and so we can apply Theorem 2.1 by ignoring zero rows.

(3) Given a rooted branch-decomposition $(T'_i, \mathcal{L}'_i)$ of $\mathcal{V}_i$ of width at most $2k$ and a set of the transition matrices $\{T_v\}_{v \in V(T'_i)}$, we compute the full set in time $g(k, 2k, |\mathbb{F}|)i$ for some function g by Proposition 5.2. If the full set at the root is empty, then the branch-width of $\mathcal{V}_i$ is larger than k. If so, we conclude that the branch-width of $\mathcal{V}$ is larger than k and stop. If the full set at the root is nonempty, then the algorithm in Proposition 5.2 also provides a branch-decomposition $(T_i, \mathcal{L}_i)$ of $\mathcal{V}_i$ having width at most k.

If this algorithm finds $(T_n, \mathcal{L}_n)$, then $(T_n, \mathcal{L}_n)$ is a branch-decomposition of $\mathcal{V}$ having width at most k. For each i, (1)–(3) runs in at most $O(k^3 n^2) + f(k, |\mathbb{F}|)n$ time for some function f and therefore the total running time of this step is $O(k^3 n^3) + f(k, |\mathbb{F}|)n^2$. ◀

References

1 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996. doi:10.1006/jagm.1996.0049.

2 Hans L. Bodlaender and Dimitrios M. Thilikos. Constructive linear time algorithms for branchwidth. In *Automata, languages and programming (Bologna, 1997)*, volume 1256 of *Lecture Notes in Comput. Sci.*, pages 627–637. Springer, Berlin, 1997. doi:10.1007/3-540-63165-8_217.

3 James F. Geelen, A. M. H. Gerards, Neil Robertson, and Geoff Whittle. On the excluded minors for the matroids of branch-width k. *J. Combin. Theory Ser. B*, 88(2):261–265, 2003. doi:10.1016/S0095-8956(02)00046-1.

4 Petr Hliněný. A parametrized algorithm for matroid branch-width. *SIAM J. Comput.*, 35(2):259–277, 2005. doi:10.1137/S0097539702418589.

5 Petr Hliněný. Branch-width, parse trees, and monadic second-order logic for matroids. *J. Combin. Theory Ser. B*, 96(3):325–351, 2006. doi:10.1016/j.jctb.2005.08.005.

6 Petr Hliněný and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008. doi:10.1137/070685920.

7 Jisu Jeong, Eun Jung Kim, and Sang-il Oum. Constructive algorithm for path-width of matroids. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 1695–1704, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611974331.ch116.

8 Jisu Jeong, Eun Jung Kim, and Sang-il Oum. The "art of trellis decoding" is fixed-parameter tractable. *IEEE Trans. Inform. Theory*, 63(11):7178–7205, 2017. An extended abstract appeared in a conference proceeding [7]. doi:10.1109/TIT.2017.2740283.

9 Jens Lagergren and Stefan Arnborg. Finding minimal forbidden minors using a finite congruence. In *Automata, languages and programming (Madrid, 1991)*, volume 510 of *Lecture Notes in Comput. Sci.*, pages 532–543. Springer, Berlin, 1991. doi:10.1007/3-540-54233-7_161.

10 Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

11 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. doi:10.1016/j.orl.2003.10.009.

12 Neil Robertson and Paul D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Combin. Theory Ser. B*, 52(2):153–190, 1991. doi:10.1016/0095-8956(91)90061-N.

13 Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994. doi:10.1007/BF01215352.

14 Dimitrios M. Thilikos and Hans L. Bodlaender. Constructive linear time algorithms for branchwidth. Technical Report UU-CS 2000-38, Universiteit Utrecht, 2000. URL: http://archive.cs.uu.nl/pub/RUU/CS/techreps/CS-2000/2000-38.pdf.

15 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Constructive linear time algorithms for small cutwidth and carving-width. In *Algorithms and computation (Taipei, 2000)*, volume 1969 of *Lecture Notes in Comput. Sci.*, pages 192–203. Springer, Berlin, 2000. doi:10.1007/3-540-40996-3_17.