A Parametrized Algorithm for Matroid Branch-Width

Petr Hliněný[⋆]

Department of Computer Science FEI
VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava, Czech Republic

E-mail: `hlineny@member.ams.org`

April 7, 2005

Abstract. Branch-width is a structural parameter very closely related to tree-width, but branch-width has an immediate generalization from graphs to matroids. We present an algorithm that, for a given matroid M of bounded branch-width t which is represented over a finite field, finds a branch decomposition of M of width at most $3t$ in cubic time. Then we show that the branch-width of M is a uniformly fixed-parameter tractable problem. Other applications include recognition of matroid properties definable in the monadic second-order logic for bounded branch-width, or [Oum] a cubic approximation algorithm for graph rank-width and clique-width.

Keywords: representable matroid, parametrized algorithm, branch-width, rank-width.

2000 Math Subjects Classification: 05B35, 68R05.

1 Introduction

We assume that the reader is familiar with basic concepts of graph theory, for example [7]. In the past decade, the notion of a *tree-width* of graphs [21] attracted plenty of attention, both from graph-theoretical and computational points of view. This attention followed the pioneer work of Robertson and Seymour on the Graph Minor Project [20], and results of various researchers using tree-width in dynamic programming and parametrized complexity, see [2] for a brief introduction. We postpone formal definitions until the next sections.

The theory of *parametrized complexity* provides a background for analysis of difficult algorithmic problems which is finer than classical complexity theory. For an overview, we suggest [8]. Briefly saying, a problem is called "fixed-parameter tractable" if there is an algorithm having running time with the (possible) super-polynomial part separated in terms of some natural "parameter", which is supposed to be small even for large inputs in practice. Successful practical applications of this concept are known, for example, in computational biology or in

[⋆] This paper follows author's original research carried out at the Victoria University of Wellington in New Zealand, in 2001–2002.

database theory: Imagine a query of a small size k to a large database of size $n \gg k$; then an $O(2^k \cdot n)$ parametrized algorithm may be better in practice than, say, an $O(n^k)$ algorithm, or even than an $O((kn)^c)$ polynomial algorithm.

Generaly speaking, we are interested in algorithmic problems that are parametrized by a tree-like structure of the input objects, or the tree-width parameter. However, for matroid theorists, it is the (very similar, but less known) parameter called branch-width [21] that has proved to be the more useful tool. This is because, unlike tree-width, branch-width does not refer to vertices and so it extends directly from graphs to matroids, and hence also to matrices and vector configurations. We refer to [15] for a closer discussion on matroid tree-width.

The tree-width of a graph is hard to compute in general. It has been shown by Bodlaender [3] that graph tree-width is a uniformly fixed-parameter tractable property, and, moreover, an optimal tree-decomposition of a graph can be found in parametrized linear time. That algorithm is used in [4] to find an optimal branch-decomposition of a graph in parametrized linear time. However, these algorithms are so closely tied with graphs that it seems impossible to generalize them to matroids. Recent advances concerning matroid connectivity, on the other hand, lead to a simple and practical polynomial algorithm [11] for deciding whether a given matroid has branch-width at most 3. Unfortunately, there seems to be no generalization of that algorithm to higher values of branch-width, too.

In this paper we show (Algorithm 4.1, Theorem 4.12) how to construct a near-optimal branch-decomposition of a matroid represented over any finite field in parametrized (with respect to branch-width) cubic time. We also prove (Theorem 5.4) that matroid branch-width (and tree-width) are fixed-parameter tractable properties for matroids represented over finite fields.

These algorithms and their consequences are formulated in the language of matroid theory since it is natural and convenient, and since it shows the close relations of this research to well-known graph structural and computational concepts. Our work could be, as well, viewed in terms of matrices, point configurations, or linear codes over a finite field $\mathbb{F}$. The key to the subject is the notion of parse trees for bounded-width $\mathbb{F}$-represented matroids, defined in Section 3 as an analogue of parse trees for graphs of bounded tree-width.

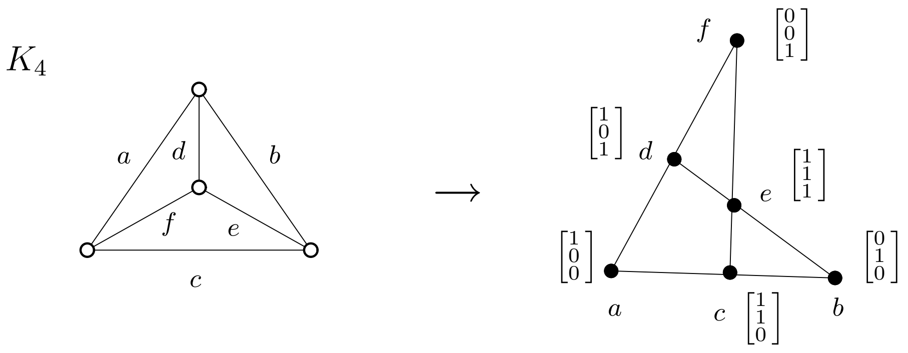
In [13] we prove a result analogous to so called "MS_2-theorem" by Courcelle [5] for matroids represented by matrices over a finite field $\mathbb{F}$: If $\mathcal{M}$ is a family of matroids described by a sentence in the monadic second-order logic of matroids, then the "parse trees" of bounded-branch-width $\mathbb{F}$-represented members of $\mathcal{M}$ are recognizable by a finite tree automaton. So in connection of [13] with Algorithm 4.1, we prove that all matroid properties expressible in the monadic second-order logic are uniformly fixed-parameter tractable for $\mathbb{F}$-represented matroids of bounded branch-width. Another application [12] gives an efficient algorithm for computing the Tutte polynomial of a matroid, the critical index, and the Hamming weight or the weight enumerator of a linear code, when the branch-width is bounded. Moreover, a recent algorithm of Oum [18] uses Algorithm 4.1 to approximate the clique-width of a graph in parametrized cubic time, via a new parameter related to matroid branch-width – the rank-width [17].

In order to make the paper accessible to a wide audience of computer scientists, we provide sufficient introductory definitions for relevant concepts of structural matroid theory.

2 Basics of Matroids

We refer to Oxley [19] for matroid terminology. A *matroid* is a pair $M = (E, \mathcal{B})$ where $E = E(M)$ is the ground set of M (elements of M), and $\mathcal{B} \subseteq 2^E$ is a nonempty collection of *bases* of M, no two of which are in an inclusion. Moreover, matroid bases satisfy the "exchange axiom"; if $B_1, B_2 \in \mathcal{B}$ and $x \in B_1 - B_2$, then there is $y \in B_2 - B_1$ such that $(B_1 - \{x\}) \cup \{y\} \in \mathcal{B}$. Subsets of bases are called *independent sets*, and the remaining sets are *dependent*. Minimal dependent sets are called *circuits*. The *rank function* $\mathrm{r}_M(X)$ in M is the maximal cardinality of an independent subset of a set $X \subseteq E(M)$.

If G is a graph, then its *cycle matroid* on the ground set $E(G)$ is denoted by $M(G)$. The independent sets of $M(G)$ are the forests of G, and the circuits of $M(G)$ are the cycles of G. In fact, a lot of matroid terminology is inherited from graphs. Another typical example of a matroid is a finite set of vectors with usual linear dependency. The two examples are both illustrated in Figure 1.



K_4
a
d
b
f
e
c
$\longrightarrow$
f
$\begin{bmatrix}0\\0\\1\end{bmatrix}$
$\begin{bmatrix}1\\0\\1\end{bmatrix}$
d
$\begin{bmatrix}1\\1\\1\end{bmatrix}$
e
$\begin{bmatrix}1\\0\\0\end{bmatrix}$
$\begin{bmatrix}0\\1\\0\end{bmatrix}$
a
c
$\begin{bmatrix}1\\1\\0\end{bmatrix}$
b

Fig. 1. An example of a vector representation of the cycle matroid $M(K_4)$. The matroid elements are depicted by dots, and their (linear) dependency is shown using lines.

The *dual* matroid M^* of M is defined on the same ground set E, and the bases of M^* are the set-complements of the bases of M. An element e of M is called a *loop* (a *coloop*), if $\{e\}$ is dependent in M (in M^*). The matroid $M \setminus e$ obtained by *deleting* a non-coloop element e is defined as $(E - \{e\}, \mathcal{B}^-)$ where $\mathcal{B}^- = \{B : B \in \mathcal{B}, e \notin B\}$. The matroid M/e obtained by *contracting* a non-loop element e is defined using duality $M/e = (M^* \setminus e)^*$. (This corresponds to contracting an edge in a graph.) A *minor* of a matroid is obtained by a sequence of deletions and contractions of elements.

The connectivity function λ_M of a matroid M is defined for all $A \subseteq E$ by

$$\lambda_M(A) = \mathrm{r}_M(A) + \mathrm{r}_M(E - A) - \mathrm{r}(M) + 1\,.$$

Here $\mathrm{r}(M) = \mathrm{r}_M(E)$. Notice that $\lambda_M(A) = \lambda_M(E - A)$. A subset $A \subseteq E$ is *k-separating* if $\lambda_M(A) \leq k$. When equality holds here, A is said to be *exactly* k-separating. An arbitrary partition $(A, E - A)$ of M is called a *separation* in M. A partition $(A, E - A)$ is called a *k-separation* if A is k-separating and both $|A|, |E - A| \geq k$. Geometrically, the affine closures of the two sides of an exact k-separation intersect in a subspace of rank $k - 1$ (such as, in a line if $k = 3$).

2.1 Branch-Decomposition

A *sub-cubic tree* is a tree in which all nodes have degree at most three. (We do not use the word ternary because such trees are actually sub-binary in the sense of the next section.) Let $\ell(T)$ denote the set of leaves of a tree T. The next definition of branch-width for matroids directly extends branch-width of graphs.

Let M be a matroid on the ground set $E = E(M)$. A *branch-decomposition* of M is a pair (T, τ) where T is a sub-cubic tree, and τ is an injection of E into $\ell(T)$, called *labeling*. Let e be an edge of T, and T_1, T_2 be the connected components of $T - e$. We say the e *displays* the partition (A, B) of E where $A = \tau^{-1}(\ell(T_1))$, $B = \tau^{-1}(\ell(T_2))$. The *width* of an edge e in T is $\omega_T(e) = \lambda_M(A) = \lambda_M(B)$. The width of the branch-decomposition (T, τ) is maximum of the widths of all edges of T, and the *branch-width* of M is the minimal width over all branch-decompositions of M. If T has no edge, then we take its width as 0.

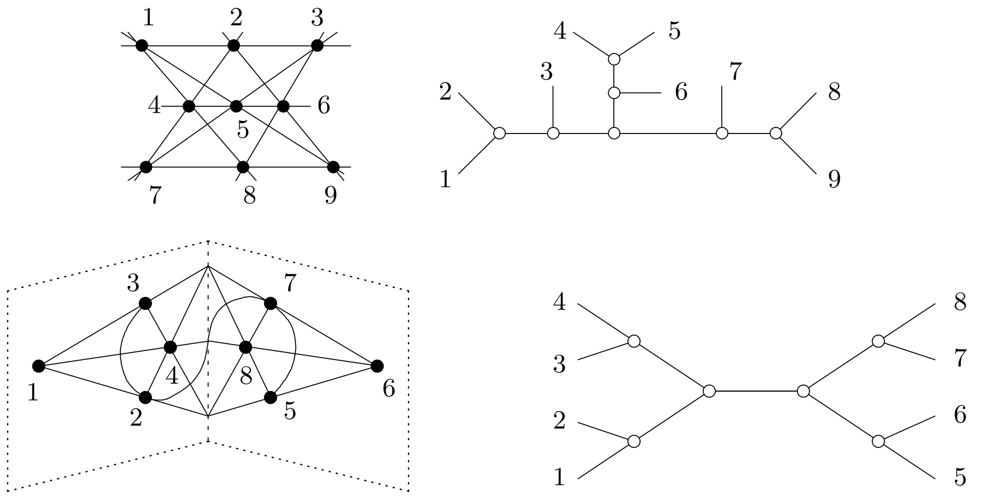


Fig. 2. Two examples of width-3 branch decompositions of the Pappus matroid (top left, in rank 3) and of the binary affine cube (bottom left, in rank 4). The lines in matroid pictures show dependencies among elements.

Examples of branch-decompositions are presented in Fig. 2. We remark that the branch-width of a graph is defined analogously, using the connectivity function λ_G where $\lambda_G(F)$ is the number of vertices incident both with F and $E(G) - F$. Clearly, the branch-width of a graph G is never smaller than the branch-width of its cycle matroid $M(G)$. It is still an open conjecture that these numbers are actually equal. On the other hand, branch-width is within a constant factor of tree-width on graphs [21], and also on matroids [15].

2.2 Represented Matroids

We now turn our attention to matroids represented over a fixed finite field $\mathbb{F}$. This is a crucial part of our introductory definitions. A *representation* of a matroid M is a matrix $\boldsymbol{A}$ whose column vectors correspond to the elements of M, and maximal linearly independent subsets of columns form the bases of M. We denote by $M(\boldsymbol{A})$ the matroid represented by a matrix $\boldsymbol{A}$.

We denote by $PG(n, \mathbb{F})$ the *projective geometry (space)* obtained from the vector space $\mathbb{F}^{n+1}$. See [19, Section 6.1,6.3] for an overview of projective spaces and of (in)equivalence of matroid representations. For a set $X \subseteq PG(n, \mathbb{F})$, we denote by $\langle X \rangle$ the span (affine closure) of X in the space. The (projective) rank $\mathrm{r}(X)$ of X is the maximal cardinality of a linearly independent subset of X. A projective transformation is a mapping between two projective spaces over $\mathbb{F}$ that is induced by a linear transformation between the underlying vector spaces. Clearly, the matroid $M(\boldsymbol{A})$ represented by a matrix $\boldsymbol{A}$ is unchanged when column vectors are scaled by non-zero elements of $\mathbb{F}$. Hence we may view a loopless matroid representation $M(\boldsymbol{A})$ as a multiset of points in the finite projective space $PG(n, \mathbb{F})$, where n is the rank of $M(\boldsymbol{A})$.

Definition. We call a finite multiset of points in a projective space over $\mathbb{F}$ a *point configuration*; and we represent a loop in a point configuration by the empty subspace $\emptyset$. Two point configurations P_1, P_2 in projective spaces over $\mathbb{F}$ are *equivalent* if there is a non-singular projective transformation between the projective spaces that maps P_1 onto P_2 bijectively. (Loops are mapped only to loops.) We define an $\mathbb{F}$-*represented matroid* to be such an equivalence class of point configurations over $\mathbb{F}$.

One may think that we do not have to include the word "bijectively" in the previous definition since non-singular projective transformations are always injective on the points, but, in fact, we have to do this to handle multiple-elements in multisets. Two labeled point configurations over $\mathbb{F}$ are equivalent in our sense if and only if, in the language of [19, Chapter 6], the matrix representations are equivalent without use of $\mathbb{F}$-automorphisms, otherwise called *strongly equivalent* in matroid theory. However, notice that our represented matroids are unlabeled in general.

Standard matroidal terms are inherited from matroids to represented matroids. Obviously, all point configurations in one equivalence class belong to the same isomorphism class of matroids, but the converse is not true in general since matroids often have inequivalent representations. When we want to deal with an $\mathbb{F}$-represented matroid, we actually pick an arbitrary point configuration from the equivalence class.

3 Parse Trees for Matroids

In this section we introduce our basic formal tool — the parse trees for represented matroids of bounded branch-width. Loosely speaking, a parse tree shows how to "build up" a matroid along the tree using only fixed amount of information at each tree node, and so it forms a suitable background for dynamic programming.

We are inspired by analogous boundaried graphs and parse trees known for handling graphs of bounded tree-width (see for example [1] or [8, Section 6.4]): A boundaried graph is a graph with a distinguished subset of labeled vertices. (The purpose is that only the boundary vertices can be "accessed from outside".) Then, simply speaking, a graph has tree-width at most $t-1$ iff it can be composed from small pieces by gluing them on boundaries of size at most t. We similarly define boundaried represented matroids, in which the boundary is a distinguished subspace of the representation, and composition operators that are used to glue representations together. (The role of composition operators, however, differs between tree-width and branch-width.)

A *rooted ordered sub-binary tree* is such that each of its nodes has at most two sons that are ordered as "left" and "right". (If there is one son, then it may be either left or right.) A *rooted subtree* T_0 of a rooted tree T is a subgraph of T such that T_0 is the connected component of $T - e$ not containing the root for some $e \in E(T)$. Let Σ be a finite alphabet. We denote by Σ^{**} the class of rooted ordered sub-binary trees with nodes labeled by symbols from Σ.

3.1 Boundaried Matroids

All matroids throughout this section are $\mathbb{F}$-represented for some fixed finite field $\mathbb{F}$. Hence, for simplicity, if we say a "(represented) matroid", then we mean an $\mathbb{F}$-represented matroid. If we speak about a projective space, we mean a projective geometry over the field $\mathbb{F}$. Let $[s,t]$ denote the set $\{s, s+1, \dots, t\}$.

The following definition presents a possible way of formalizing the notion of a "matroid with a boundary". (Since matroids have no vertices unlike graphs, we have to introduce some special elements that define the matroid boundary.)

Definition. A pair $\bar{N} = (N, \delta)$ is a *t-boundaried (represented) matroid* if the following holds: $t \geq 0$ is an integer, N is a represented matroid, and $\delta : [1,t] \to E(N)$ is an injective mapping such that $\delta([1,t])$ is an independent set in N.

We call $J(\bar{N}) = E(N) - \delta([1,t])$ the *internal elements* of $\bar{N}$, elements of $\delta([1,t])$ the *boundary points* of $\bar{N}$, and t the *boundary rank* of $\bar{N}$. The represented matroid $N \setminus \delta([1,t])$, which is the restriction of N to $J(\bar{N})$, is called the *internal matroid* of $\bar{N}$. We denote by $\partial(\bar{N})$ the boundary subspace spanned by $\delta([1,t])$.

In particular, the boundary points are not loops. The basic operation we use is the *boundary sum* $\bar{\oplus}$ of the next definition, illustrated in Fig. 3.

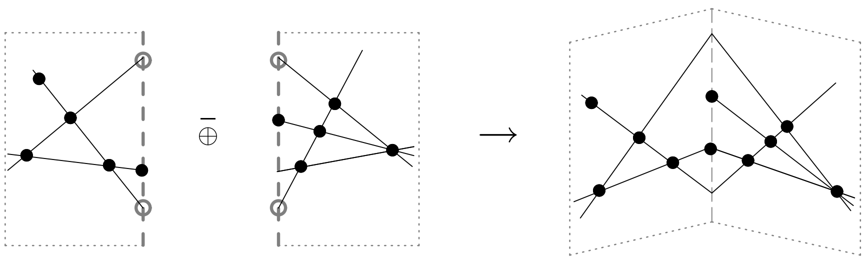


Fig. 3. An example of a boundary sum of two 2-boundaried matroids. The internal matroid elements are drawn as solid dots, the boundary points and the boundary subspace of rank 2 are drawn in gray. Solid lines show matroid dependencies. The resulting sum is a matroid represented on two intersecting planes in rank 4 (aka "3-dimensional picture" on the right).

Definition. Let $\bar{N}_1 = (N_1, \delta_1)$, $\bar{N}_2 = (N_2, \delta_2)$ be two t-boundaried represented matroids. We denote by $\bar{N}_1 \bar{\oplus} \bar{N}_2 = N$ the represented matroid defined as follows: Let Ψ_1, Ψ_2 be projective spaces such that the intersection $\Psi_1 \cap \Psi_2$ has rank exactly t. Suppose that, for $i = 1, 2$, $P_i \subset \Psi_i$ is a point configuration representing N_i, such that $P_1 \cap P_2 = \delta_1([1,t]) = \delta_2([1,t])$, and $\delta_2(j) = \delta_1(j)$ for $j \in [1,t]$. Then N is the matroid represented by $(P_1 \cup P_2) - \delta_1([1,t])$.

Informally, the boundary sum $\bar{N}_1 \bar{\oplus} \bar{N}_2 = N$ on the ground set $E(N) = J(\bar{N}_1) \dot{\cup} J(\bar{N}_2)$ is obtained by gluing the representations of N_1 and N_2 on a common subspace (the boundary) of rank t, so that the boundary points of both are identified in order and then deleted. Keep in mind that a point configuration is a multiset. It is a matter of elementary linear algebra to verify that the boundary sum is well defined with respect to equivalence of point configurations.

We write "$\leq t$-boundaried" to mean t'-boundaried for some $0 \leq t' \leq t$. We now define a composition operator (over the field $\mathbb{F}$) which will be used to generate large boundaried matroids from smaller pieces (Fig. 4).

Definition. A $\leq t$*-boundaried composition* operator is defined as a quadruple $\odot = (R, \gamma_1, \gamma_2, \gamma_3)$, where R is a represented matroid, $\gamma_i : [1, t_i] \to E(R)$ is an injective mapping for $i = 1, 2, 3$ and some fixed $0 \leq t_i \leq t$, each $\gamma_i([1, t_i])$ is an independent set in R, and $\big(\gamma_i([1, t_i]) : i = 1, 2, 3\big)$ is a partition of $E(R)$.
The $\leq t$-boundaried composition operator $\odot$ is a binary operator applied to a t_1-boundaried represented matroid $\bar{N}_1 = (N_1, \delta_1)$ and to a t_2-boundaried represented matroid $\bar{N}_2 = (N_2, \delta_2)$. The result of the composition is a t_3-boundaried represented matroid $\bar{N} = (N, \gamma_3)$, written as $\bar{N} = \bar{N}_1 \odot \bar{N}_2$, where a matroid N is defined using boundaried sums: $N' = \bar{N}_1 \bar{\oplus} (R, \gamma_1)$, $N = (N', \gamma_2) \bar{\oplus} \bar{N}_2$.

Speaking informally, a boundaried composition operator is a bounded-rank configuration with three boundaries distinguished by $\gamma_1, \gamma_2, \gamma_3$, and with no other internal points. For reference we denote by $t_i(\odot) = t_i$, by $R(\odot) = R$, and by $\gamma_i(\odot) = \gamma_i$. The meaning of a composition $\bar{N} = \bar{N}_1 \odot \bar{N}_2$ is that, for $i = 1, 2$, we glue the represented matroid N_i to R, matching $\delta_i([1, t_i])$ with $\gamma_i([1, t_i])$ in order. The result is a t_3-boundaried matroid $\bar{N}$ with boundary $\gamma_3([1, t_3])$. One may shortly write the composition as $\bar{N} = \big((\bar{N}_1 \bar{\oplus} (R, \gamma_1), \gamma_2\big) \bar{\oplus} \bar{N}_2, \gamma_3\big)$.

3.2 Parse Trees

The main purpose of introducing parse trees is in that they allow to formally define how to construct a represented matroid of branch-width at most $t + 1$ using $\leq t$-boundaried composition operators.

Let $\bar{\Omega}_t$ denote the *empty* t-boundaried matroid (Ω, δ_0) where $t \geq 0$ and $\delta_0([1, t]) = E(\Omega)$ (t will often be implicit in the context). If $\bar{N} = (N, \delta)$ is an arbitrary t-boundaried matroid, then $\bar{N} \bar{\oplus} \bar{\Omega}_t$ is actually the internal matroid of $\bar{N}$. Let $\bar{\Upsilon}$ denote the *single-element* 1-boundaried matroid (Υ, δ_1) where $E(\Upsilon) = \{x, x'\}$ are two parallel elements, and $\delta_1(1) = x'$. Let $\bar{\Upsilon}_0$ denote the *loop* 0-boundaried matroid (Υ_0, δ_0) where $E(\Upsilon_0) = \{z\}$ is a loop, and δ_0 is an empty mapping. Let $\mathcal{R}_t^{\mathbb{F}}$ denote the finite set of all $\leq t$-boundaried composition operators over the field $\mathbb{F}$. We set $\Pi_t = \mathcal{R}_t^{\mathbb{F}} \cup \{\bar{\Upsilon}, \bar{\Upsilon}_0\}$.

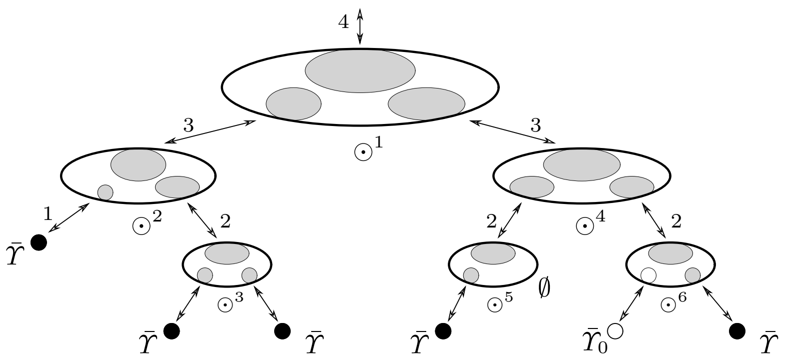


Fig. 4. An example of a boundaried parse tree. The ovals represent composition operators, with shaded parts for the boundaries and edge-numbers for the boundary ranks. (E.g. $\odot^4 = (R^4, \gamma_1^4, \gamma_2^4, \gamma_3^4)$ where $\gamma_1^4, \gamma_2^4 : [1, 2] \to E(R^4)$, $\gamma_3^4 : [1, 3] \to E(R^4)$.)

Let $T \in \Pi_t^{**}$ be a rooted ordered sub-binary tree with nodes labeled by the alphabet Π_t. Considering a node v of T; we set $\varrho(v) = 1$ if v is labeled by $\bar{\Upsilon}$, $\varrho(v) = 0$ if v is labeled by $\bar{\Upsilon}_0$, and $\varrho(v) = t_3(\odot)$ if v is labeled by $\odot$. We call T a *$\leq t$-boundaried parse tree* if the following are true:

– Only leaves of T are labeled by $\bar{\Upsilon}$ or $\bar{\Upsilon}_0$.
– If a node v of T labeled by a composition operator $\odot$ has no left (no right) son, then $t_1(\odot) = 0$ ($t_2(\odot) = 0$).

– If a node v of T labeled by $\odot$ has left son u_1 (right son u_2), then $t_1(\odot) = \varrho(u_1)$ ($t_2(\odot) = \varrho(u_2)$).

Informally, the boundary ranks of composition operators and/or single-element terminals must "agree" across each edge. Notice that $\bar{\Upsilon}$ or $\bar{\Upsilon}_0$ are the only labels from Π_t that "create" elements of the resulting represented matroid $P(T)$ in the next definition. See an illustration example in Fig. 4.

Definition. Let T be a $\leq t$-boundaried parse tree. The $\leq t$-boundaried represented *matroid $\bar{P}(T)$ parsed by T* is recursively defined as follows:

– If T is an empty tree, then $\bar{P}(T) = \bar{\Omega}_0$.
– If T has one node labeled by $\bar{\Upsilon}$ (by $\bar{\Upsilon}_0$), then $\bar{P}(T) = \bar{\Upsilon}$ ($= \bar{\Upsilon}_0$).
– If the root r of T is labeled $\odot^r$, and r has a left rooted subtree T_1 and a right rooted subtree T_2 (possibly empty), then $\bar{P}(T) = \bar{P}(T_1) \odot^r \bar{P}(T_2)$.

The composition is well defined according to the parse-tree description in the previous paragraph. The represented matroid parsed by T is $P(T) = \bar{P}(T) \bar{\oplus} \bar{\Omega}$.

We say that a t-boundaried represented matroid $\bar{M}$ is spanning if the boundary subspace $\partial(\bar{M})$ is contained in the span $\langle J(\bar{M}) \rangle$ of the internal points of $\bar{M}$. We say that a $\leq t$-boundaried parse tree T is *spanning* if, for each nonempty rooted subtree T_1 of T, the boundaried matroid $\bar{P}(T_1)$ is spanning and nonempty. The following natural result about parse trees of matroids is proved in [13].

Theorem 3.1. (PH [13]) *An $\mathbb{F}$-represented matroid M has branch-width at most $t+1$ if and only if M is parsed by some spanning $\leq t$-boundaried parse tree.*

It is easy to turn a boundaried parse tree into a branch-decomposition. Conversely, the proof of Theorem 3.1 shows a way how to construct a boundaried parse tree from a given branch-decomposition. It is, however, more efficient to construct a boundaried parse tree directly from scratch, and that is what we are going to do in this paper.

4 Constructing a Parse Tree

We start with an overview of our algorithm for construction of a matroid parse tree. We assume that a matroid M is given by a matrix $\boldsymbol{A} \in \mathbb{F}^{r \times n}$ of rank r over a finite field $\mathbb{F}$. The size of M is expressed in terms of the number n of elements, i.e. the number of columns of $\boldsymbol{A}$. Clearly, $r \leq n$. Additionally, there are two (so called) *parameters* – a finite field $\mathbb{F}$ and an integer $t > 0$. These parameters form a separate part of the input in our algorithm, but they are considered as constants for the purpose of complexity analysis of the algorithm. A more formal description in the scope of parametrized complexity can be found in Section 5.2.

We call a labeled tree T a *partial $\leq t$-boundaried parse tree* if T satisfies all properties of the parse tree definition (Subsection 3.2) except that some leaves of T may be labeled by arbitrary t'-boundaried represented matroids for $t' \leq t$. The matroid $\bar{P}(T)$ parsed by T is defined analogously to ordinary parse trees.

Notice that a 0-boundaried represented matroid is essentially an ordinary represented matroid, and that the boundary sum of 0-boundaried represented matroids is the ordinary direct sum of matroids. Let $M \restriction X$ denote the restriction of a represented matroid M to a subset of elements $X \subseteq E(M)$. Let $\boldsymbol{I}_r$ denote the $r \times r$ identity submatrix.

Algorithm 4.1. Computing a spanning boundaried parse tree of a matroid:

Parameters: A given finite field $\mathbb{F}$, and an integer $t \geq 1$.
Input: A matrix $\boldsymbol{A} = [\boldsymbol{I}_r \,|\, \boldsymbol{A}'] \in \mathbb{F}^{r \times n}$ given in the standard form over $\mathbb{F}$, such that the matroid $M(\boldsymbol{A})$ represented by $\boldsymbol{A}$ has branch-width at most $t + 1$.

1. Let $M = M(\boldsymbol{A})$, and let $E(M) = B \cup F$ where a basis B marks the columns of $\boldsymbol{I}_r$ and F the columns of $\boldsymbol{A}'$. We initially set $X = B$, and T to be an arbitrary 0-boundaried parse tree for the independent matroid $M \restriction B$.
2. For an arbitrary element $f \in F - X$, we set $X = X \cup \{f\}$, and we add a new single-element leaf representing f in $M \restriction X$ to the parse tree T.
3. If the width of the parse tree T exceeds, say, $10t$, or if $X \supseteq F$, then we compute a new spanning $\leq 3t$-boundaried parse tree T' for $M \restriction X$ using T:
   (a) We start with T' equal to the trivial 0-boundaried partial parse tree having one node labeled by $M \restriction X$.
   (b) Let ℓ be a leaf of T' labeled by a boundaried matroid $\bar{N}$ with more than one internal elements. If the boundary rank of $\bar{N}$ is less than $3t$, then we choose an arbitrary internal element e in $\bar{N}$. We add to T' a new leaf ℓ_1 representing a single-element e, another new leaf ℓ_2 labeled by $\bar{N} \setminus e$, and we re-label ℓ with the corresponding composition operator.
   (c) If the boundary rank of $\bar{N}$ equals $3t$, then there are two $\leq 3t$-boundaried matroids $\bar{N}_1, \bar{N}_2$ such that $\bar{N} = \bar{N}_1 \odot \bar{N}_2$ for some $\leq 3t$-boundaried composition operator $\odot$. Using the decomposition T, we can find $\bar{N}_1, \bar{N}_2$ and $\odot$ efficiently. Then we add two new leaves ℓ_1, ℓ_2 to T' labeled by $\bar{N}_1, \bar{N}_2$, respectively, and we re-label ℓ with $\odot$.
   (d) We repeat steps 3b–3c until T' is an ordinary parse tree.
   Finally, we set $T = T'$.
4. We repeat steps 2–3 until $X \supseteq F$.

Output: A spanning $\leq 3t$-boundaried parse tree $T \in \Pi_{3t}^{**}$ such that the represented matroid $P(T)$ parsed by T is equal to $M(\boldsymbol{A})$.

Remark. If the above promise that the matroid $M(\boldsymbol{A})$ has branch-width at most $t + 1$ is false, then the step (3c) may fail to find $\bar{N}_1$ and $\bar{N}_2$. In such a case, Algorithm 4.1 ends up with an error. The algorithm may be formulated in a way that it always finds a spanning $\leq 3t$-boundaried parse tree T for $M(\boldsymbol{A})$ if branch-width of $M(\boldsymbol{A})$ is at most $t + 1$, but the output is (possibly) empty if branch-width of $M(\boldsymbol{A})$ exceeds $t + 1$.

Remark. Another note concerns the given finite field $\mathbb{F}$: A finite field is uniquely determined by the number of its elements $q = |\mathbb{F}|$. Moreover, it is easy to construct addition and multiplication tables for $\mathbb{F}$ algorithmically from given q.

We claim that Algorithm 4.1 finds the $\leq 3t$-boundaried parse tree T for $M(\boldsymbol{A})$ in time $O(n^3)$, where n is the number of columns of $\boldsymbol{A}$. If the matrix $\boldsymbol{A}$ is not in the standard form, we can easily get the standard form in time $O(n^3)$. The step (2) can be implemented in time $O(nr) \leq O(n^2)$. Each iteration of the cycle in step (3) can be implemented in time $O(n)$, and there are at most $n-1$ iterations. Altogether, the main cycle in the algorithm is repeated $n - r$ times.

We also note that the running time $O(n^3)$ of our algorithm refers to the number n of columns of $\boldsymbol{A}$, not to the real size of the input. The real input size of the matrix $\boldsymbol{A}$ is $O(n \cdot r)$ which is typically of order up to n^2. We do not attempt to determine how Algorithm 4.1 running time depends on the parameters $\mathbb{F}$ and t (which is at least an exponential function). However, we do care that the algorithm is recursive in $\mathbb{F}$ and t — there is one algorithm running for all values of the parameters, *not* a sequence of algorithms (cf. Section 5.2).

4.1 Step (2): Adding an Element to a Parse Tree

Step (1) in Algorithm 4.1 is easy to implement. We are going to describe the implementation of step (2) of Algorithm 4.1. This is a surprisingly nontrivial task despite that we are adding the element f to the parse tree T arbitrarily. The main complication comes from necessity to recompute all boundary subspaces of the separations in T — a straightforward implementation of which would require us to solve systems of linear equations. All coming algorithms are parametrized by a finite field $\mathbb{F}$ and integers t and $t' \leq 10t$, where $\mathbb{F}$, t are as in Algorithm 4.1.

Let $M = M(\boldsymbol{D})$ be a matroid represented by a matrix $\boldsymbol{D} \in \mathbb{F}^{r \times n}$, and consider a separation (E_1, E_2) in M (i.e. a partition of $E(M)$). The projective subspace $\langle E_1 \rangle \cap \langle E_2 \rangle$ spanned by both sides of the separation is called the *guts of the separation* (E_1, E_2). The rank of the guts equals $\lambda_M(E_1) - 1$ by modularity. Naturally, one may compute a spanning set of *generator vectors* (with respect to $\boldsymbol{D}$) for the guts of (E_1, E_2) from the vectors in $\boldsymbol{D}$. If the rank of the guts is less than t' which is a constant, then the combined size of its independent generator vectors is $O(r)$.

Algorithm 4.2. Adding a vector to a separation guts in a represented matroid.

Input: A matrix $\boldsymbol{D} = [\boldsymbol{I}_r \,|\, \boldsymbol{D}'] \in \mathbb{F}^{r \times n}$ representing the matroid $M = M(\boldsymbol{D})$; a separation (F_1, F_2) of the matroid $M \setminus f$ where f is the element represented by the last column of $\boldsymbol{D}$, and where $\lambda_{M \setminus f}(F_1) \leq t' + 1$; and independent generator vectors for the guts of the separation (F_1, F_2) with respect to $\boldsymbol{D} \setminus f$.

Output: Independent generator vectors for the guts of the separation $(F_1 \cup \{f\}, F_2)$ of M with respect to $\boldsymbol{D}$, computed in time $O(r)$.

Proof. Notice that we cannot even read the whole matrix $\boldsymbol{D}$ in time $O(r)$. This is, however, not a big problem since we are going to use only the unit vectors of $\boldsymbol{I}_r$ and the generator vectors of the guts in the algorithm. Let $\Psi = \langle F_1 \rangle \cap \langle F_2 \rangle$ be the given guts, and $\Psi' = \langle F_1 \cup \{f\} \rangle \cap \langle F_2 \rangle$ be the guts we have to generate. Let $\boldsymbol{f}$ denote the last column of $\boldsymbol{D}$ representing the element f, and let $\boldsymbol{f}_1$ denote the vector obtained from $\boldsymbol{f}$ by setting to 0 the coordinates corresponding to the unit vectors of $\boldsymbol{I}_r$ appearing in F_1. Then $\boldsymbol{f}_1$ belongs to the span $\langle F_2 \rangle$ by definition.

We may easily compute $\boldsymbol{f}_1$ in time $O(r)$. Moreover, we may decide whether $\boldsymbol{f}_1 \in \Psi$ in time $O(r)$ since there is a bounded number of generator vectors for Ψ. If $\boldsymbol{f}_1 \in \Psi$, then $\boldsymbol{f}$ belongs to the span $\langle F_1 \rangle$ and so $\Psi' = \Psi$. Otherwise, since $\boldsymbol{f}_1$ belongs to the span $\langle F_1 \cup \{f\} \rangle$, we get $\Psi' = \langle \Psi \cup \boldsymbol{f}_1 \rangle$. ∎

Let $M = M(\boldsymbol{D})$ be a matroid represented by a matrix $\boldsymbol{D} \in \mathbb{F}^{r \times n}$, and let T be a spanning $\leq t'$-boundaried parse tree for M. A *coordinatization* (with respect to $\boldsymbol{D}$) of the parse tree T is assignment of the appropriate vectors to the boundary points of the matroids $\bar{P}(T_1)$ for all rooted subtrees T_1 of T, as computed from the vectors of $\boldsymbol{D}$. Similarly as above, the combined size of the vectors for each boundary subspace is $O(r)$.

Algorithm 4.3. Computing the coordinatization of a spanning $\leq t'$-boundaried parse tree T for the represented matroid $M(\boldsymbol{D})$ over $\mathbb{F}$.

Input: A matrix $\boldsymbol{D} \in \mathbb{F}^{r \times n}$, and a spanning $\leq t'$-boundaried parse tree T for the represented matroid $M(\boldsymbol{D})$.

Output: The coordinatization of T with respect to $\boldsymbol{D}$, computed in time $O(nr)$.

Proof. If T_0 is a rooted subtree of T with one node, then the coordinatization of T_0 is trivial. So assume that the root r_0 of T_0 has the left and right rooted subtrees T_1 and T_2, and that r_0 is labeled by a composition operator $\odot$. (Hence $\bar{P}(T_0) = \bar{P}(T_1) \odot \bar{P}(T_2)$. One of the subtrees may be empty.) Since T is a spanning parse tree, the boundary of $\bar{P}(T_0)$ is spanned by the boundaries of $\bar{P}(T_1)$ and $\bar{P}(T_2)$.

Thus the vectors assigned to the boundary points of $\bar{P}(T_0)$ with respect to $\boldsymbol{D}$ are linear combinations of the vectors of the boundary points of $\bar{P}(T_1)$ and $\bar{P}(T_2)$. There are finitely many $\leq t'$-boundaried composition operators for a fixed t', and so the scalars of these linear combinations can be precomputed for the parameter t' and each $\odot$. Then, at each node of T, we have to compute a bounded number of linear combinations from a bounded number of vectors of length r, which can be accomplished in time $O(r)$. There are $O(n)$ nodes in T. ∎

Consider three sequences $U_1, U_2, U_3 \subset \mathbb{F}^s$ of vectors where each U_i, $i = 1, 2, 3$ is formed by independent vectors. Then these sequences represent the composition operator $(U, \gamma_1, \gamma_2, \gamma_3)$; where $U = U_1 \cup U_2 \cup U_3$ is the point configuration, and γ_i maps the elements of the sequence in order $U_i = \{\gamma_i(1), \gamma_i(2), \dots, \gamma_i(t_i)\}$ for $i = 1, 2, 3$.

Algorithm 4.4. Computing the composition operator from given vectors.

Input: Three independent sequences $U_1, U_2, U_3 \subset \mathbb{F}^s$ of vectors where each sequence has at most t' members.

Output: The $\leq t'$-boundaried composition operator represented by the vectors of U_1, U_2, U_3, computed in time $O(s)$.

Proof. There is a bounded number of $\leq t'$-boundaried composition operators for each t'; and since each one is nothing else than a labeled point configuration, it may be identified by a bounded number of homogeneous linear vector equations and inequations. (The equations are invariant under nonsingular linear transformations.) Hence we find the composition operator represented by (U_1, U_2, U_3) in parametrized time $O(s)$ even by brute force. ∎

Lemma 4.5. *Step (2) in Algorithm 4.1 is implemented in time $O(n \cdot r)$ for fixed parameters t' and $\mathbb{F}$.*

Proof. Let $N = M \restriction X$ in the step (2) of Algorithm 4.1 (before adding f to X). We first compute the coordinatization of the parse tree T for N using Algorithm 4.3. Let T_1 be a rooted subtree of T, and let $F_1 = J\big(\bar{P}(T_1)\big)$, $F_2 = E(N) - F_1$. Then, since the parse tree T is spanning, the boundary subspace of $\bar{P}(T_1)$ is the guts of the separation (F_1, F_2). We denote by T' the rooted sub-binary tree obtained from T by adding arbitrarily a new leaf ℓ representing the element f. To turn T' into a parse tree for $N' = M \restriction X \cup \{f\}$, we have to re-compute all composition operators in T'.

Let T_1' be the rooted subtree of T' corresponding to T_1 above, and let $F_1' = J\big(\bar{P}(T_1')\big)$, $F_2' = E(N') - F_1'$. Then $F_1' = F_1 \cup \{f\}$ and $F_2' = F_2$ for $\ell \in V(T_1')$, or $F_2' = F_2 \cup \{f\}$ and $F_1' = F_1$ otherwise. We denote by $\boldsymbol{D} = [\boldsymbol{I}_r \mid \boldsymbol{D}']$ the submatrix of the matrix $\boldsymbol{A}$ from Algorithm 4.1 representing the matroid N'. We apply Algorithm 4.2 to $\boldsymbol{D}$, the separation (F_1, F_2) and f, and to the generator vectors of the boundary points of $\bar{P}(T_1)$ computed above, obtaining generator vectors of the boundary of $\bar{P}(T_1')$. We repeat the same procedure for all $O(n)$ rooted subtrees of T'. Finally, we use Algorithm 4.4 to determine the new composition operators to label all internal nodes of T'. ∎

4.2 Step (3): Getting a better Parse Tree

The heart of the implementation of step (3) in Algorithm 4.1 is the next claim. (We remark that the proof of this claim implicitly uses a so called "tangle" [21] – a notion dual to a branch-decomposition.)

Lemma 4.6. *Let $t \geq 1$ and $\bar{N}$ be a spanning $3t$-boundaried represented matroid such that the branch-width of the internal matroid $\bar{N} \oplus \bar{\Omega}$ is at most $t+1$. Then there are two $\leq 3t$-boundaried matroids $\bar{N}_1, \bar{N}_2$ such that $\bar{N} = \bar{N}_1 \odot \bar{N}_2$ for some $\leq 3t$-boundaried composition operator $\odot$.*

Proof. Recall that the internal matroid $N' = \bar{N} \oplus \bar{\Omega}$ is the restriction of $\bar{N}$ to the internal elements $J(\bar{N}) = E(N')$. We define a function $g : 2^{E(N')} \to \mathbb{Z}$, for a subset $F \subseteq E(N')$, as $g(F) = \mathrm{r}\left(\partial(\bar{N}) \cap \langle F \rangle\right)$, i.e. as the projective rank of the intersection of the span of F with the boundary of $\bar{N}$. Clearly, $g(E(N')) = 3t$ since $\bar{N}$ is spanning.

Let (U, τ) be a width-$(t+1)$ branch-decomposition of N'. For an edge $e \in E(U)$, we define $F_i(e) = \tau^{-1}\big(V(U_i)\big)$ for $i = 1, 2$ where U_1 and U_2 are the connected components of $U - e$. Note that $g\big(F_1(e)\big) + g\big(F_2(e)\big) \geq g\big(E(N')\big) = 3t$.

Claim. There is an edge $e \in E(U)$ such that both $g\big(F_1(e)\big), g\big(F_2(e)\big) \geq t$.

For a contradiction, we assume that no such edge e exists in U. Let (arbitrary) $e = w_1w_2$ where $w_i \in U_i$, $i = 1, 2$ as above, and $a = \min\big\{g(F_1(e)), g(F_2(e))\big\}$. If $a < t$, then, say, $a = g\big(F_1(e)\big) < g\big(F_2(e)\big)$. In such a case we direct the edge e from w_1 to w_2. Since U is a cubic tree, there is a node w_0 of U such that all three edges incident with w_0 are directed towards it. Then, denoting by U_1', U_2', U_3' the connected components of $U - w_0$ and by $F_i' = \tau^{-1}\big(V(U_i')\big)$, we get $g\big(F_1'\big) + g\big(F_2'\big) + g\big(F_3'\big) < t + t + t$ which is a contradiction to $g\big(E(N')\big) = 3t$.

So we consider further the edge e from the claim. The rank of $\langle F_1(e)\rangle \cap \langle F_2(e)\rangle$ is at most t since (U, τ) is a width-$(t + 1)$ branch-decomposition. Together for $i = 1, 2$ we get, by modularity of the rank in projective spaces,

$$\mathrm{r}\left[\big\langle F_i(e)\big\rangle \cap \big\langle F_{3-i}(e) \cup \partial(\bar{N})\big\rangle\right] = \mathrm{r}\left[\big\langle F_i(e)\big\rangle \cap \big\langle F_{3-i}(e)\big\rangle\right] + \\ + \mathrm{r}\left[\big\langle F_i(e)\big\rangle \cap \partial(\bar{N})\right] - \mathrm{r}\left[\big\langle F_i(e)\big\rangle \cap \big\langle F_{3-i}(e)\big\rangle \cap \partial(\bar{N})\right] \leq t + 2t - 0 = 3t\,.$$

Hence the partition $\big(F_1(e), F_2(e)\big)$ decomposes the internal elements into two $\leq$3t-boundaried matroids $\bar{N}_1, \bar{N}_2$ which are glued together by a suitable $\leq$3t-boundaried composition operator $\odot$. ∎

The task now is to find the partition (F_1, F_2) of $J(\bar{N})$ inducing the boundaried matroids $\bar{N}_1, \bar{N}_2$, and the composition operator $\odot$ from Lemma 4.6 efficiently. We use the $\leq t'$-boundaried parse tree T previously constructed in Algorithm 4.1. Unlike in the implementation of step (2), we do not work with the vectors representing $M(\boldsymbol{A})$ — instead, we obtain all necessary information from the parse tree T. This results in a more complicated but faster implementation. Again, all algorithms in this section are parametrized by a finite field $\mathbb{F}$ and integers t and $t' \leq 10t$. We first present the following two simple algorithms.

Algorithm 4.7. Computing the connectivity function of a separation.

Input: An $\leq t'$-boundaried parse tree T parsing the matroid $N = P(T)$, and a partition (F_1, F_2) of $E(N)$.

Output: The connectivity value $\lambda_N(F_1) - 1$ of the separation (F_1, F_2), computed in time $O\big(|V(T)|\big)$.

Proof. In other words, we are computing the projective rank of the guts of separation (F_1, F_2) in N. This is a straightforward application of dynamic programming over T. Let x be a node of T, and let T_x be the rooted subtree of T with the root x. Denote by $\bar{M}_x = \bar{P}(T_x)$, and by T_x', T_x'' the left and right rooted subtrees of x in T.

For $\bar{M}_x$ as above and for $E_i = F_i \cap J(\bar{M}_x)$, we call *boundary data* the triple (Σ_1, Σ_2, g) where $\Sigma_i = \partial(\bar{M}_x) \cap \langle E_i\rangle$ for $i = 1, 2$ is the intersection of the span $\langle E_i\rangle$ with the boundary of $\bar{M}_x$, and where g is the projective rank of the guts $\langle E_1\rangle \cap \langle E_2\rangle$ of the subseparation (E_1, E_2). Clearly, one may compute boundary data of $\bar{M}_x = \bar{P}(T_x)$ from boundary data of $\bar{P}(T_x')$ and $\bar{P}(T_x'')$, and from the composition operator labeling x in (parametrized) constant time. Hence the whole algorithm is implemented in linear time. ∎

Algorithm 4.8. Computing a good partition (F_1, F_2) for Lemma 4.6.

Input: A $\leq t'$-boundaried parse tree T parsing the represented matroid $N = P(T)$, and a subset $F_0 \subseteq E(N)$ such that $\lambda_N(F_0) = 3t$.
Output: A partition (F_1, F_2) of the set $E(N) - F_0$ such that $\lambda_N(F_1), \lambda_N(F_2) \leq 3t$, or an answer NO if no such partition exists; computed in time $O\big(|V(T)|\big)$.

Proof. This is a straightforward extension of the dynamic program implemented in Algorithm 4.7. Moreover, for each instance of boundary data in this case, we record one representative of the partition which we compute. Since the ground set of N can be easily implemented so that the operation of a set union (of the representatives) takes constant time, the overall computing time is still linear in T. We leave details to the reader. ∎

Now we get to the hard part — computing a valid composition operator for the new node of the parse tree T' in step (3b) or (3c). We (implicitly) know the three boundaries of the composition operator from a defining tri-partition of the matroid elements. We, however, have to choose independent spanning sets of generator points for the boundaries in such a way that the two matching boundaries of adjacent composition operators in the parse tree T' get the same generator points. For a parse tree U, we call a *virtual point* of U any point which is contained in the span of some composition operator in the tree U. The concept of virtual points of the parse tree U allows us to determine relative positions of certain points with respect to the elements of the matroid $P(U)$, without necessity to use absolute vectors in a particular matroid representation.

If (F_1, F_2) is a separation in the represented matroid $P(U)$ parsed by U, then we are going to express spanning generator points for the guts of (F_1, F_2) as a sequence of virtual points of the parse tree U. We naturally say that a set Z of virtual points in U is independent if Z is linearly independent in the point configuration parsed by U. Next is the corresponding extension of Algorithm 4.7: The fact that the guts is always spanned by some virtual points of U is implicitly proved in the algorithm.

Algorithm 4.9. Computing virtual points spanning the guts of a given separation over a parse tree.

Input: An $\leq t'$-boundaried parse tree T parsing the matroid $N = P(T)$, and a partition (F_1, F_2) of $E(N)$ such that $\lambda_N(F_1) \leq 3t$. (The partition is symmetric, i.e. (F_1, F_2) is considered the same as (F_2, F_1).)
Output: A uniquely determined sequence $\{x_1, x_2, \dots, x_k\}$, $k = \lambda_N(F_1) - 1$, of independent virtual points of the parse tree T which span the guts of the separation (F_1, F_2). This is computed in time $O\big(|V(T)|\big)$.

Proof. Let x be a node of T labeled by $\odot$, and let T_x be the rooted subtree of T with the root x. Denote by $\bar{M}_x = \bar{P}(T_x)$, and by T'_x, T''_x the left and right rooted subtrees of x in T. Analogously to Algorithm 4.7, we call *boundary data* of $\bar{M}_x$ the quadruple $(\Sigma_1, \Sigma_2, g, Z)$ where $E_i = F_i \cap J(\bar{M}_x)$ and $\Sigma_i = \partial(\bar{M}_x) \cap \langle E_i \rangle$

for $i = 1, 2$, where $g = \mathrm{r}\big(\langle E_1\rangle \cap \langle E_2\rangle\big)$, and where $Z = \{z_1, \dots, z_g\}$ is a sequence of independent virtual points of T which spans the guts of (E_1, E_2) in $\bar{M}_x$. Moreover, we require that the points of $Z \cap \partial(\bar{M}_x)$ contained in the boundary span the set $\Sigma_1 \cap \Sigma_2$ (boundary-span condition).

We denote by $\bar{M}'_x = \bar{P}(T'_x)$, $\bar{M}''_x = \bar{P}(T''_x)$, and by $E'_i = F_i \cap J(\bar{M}'_x)$, $E''_i = F_i \cap J(\bar{M}''_x)$. The guts $\langle E_1\rangle \cap \langle E_2\rangle$ is clearly spanned by the four sets $\langle E'_1\rangle \cap \langle E'_2\rangle$, $\langle E''_1\rangle \cap \langle E''_2\rangle$, $\langle E''_1\rangle \cap \langle E'_2\rangle$, $\langle E'_1\rangle \cap \langle E''_2\rangle$. The latter two sets $\langle E''_1\rangle \cap \langle E'_2\rangle$ and $\langle E'_1\rangle \cap \langle E''_2\rangle$ are contained in the span of the composition operator $\odot$ labeling x in the parse tree T by definition. Hence, using the boundary-span condition above, we can compute boundary data of $\bar{M}_x$ from boundary data of $\bar{M}'_x$ and of $\bar{M}''_x$, and from the composition operator $\odot$ in a canonical way.

Since the guts of (E_1, E_2) has bounded rank $g \le 3t$ in $\bar{M}_x$, boundary data carry only limited amount of information. One node x of the parse tree T is processed in time depending on $\mathbb{F}$ and t, but not depending on the size of T. So the whole algorithm is implemented in (parametrized) linear time. ∎

Lastly, we present an analogue of Algorithm 4.4 computing the composition operator from sequences of virtual points. Consider three sequences Z_1, Z_2, Z_3 of virtual points in a common parse tree T, each Z_i, $i = 1, 2, 3$ formed by g_i independent points. Then these sequences represent the composition operator $(Z, \gamma_1, \gamma_2, \gamma_3)$; where $Z = Z_1 \cup Z_2 \cup Z_3$ is the ground point configuration as parsed by T, and γ_i maps the elements of the sequence in order $Z_i = \{\gamma_i(1), \gamma_i(2), \dots, \gamma_i(g_i)\}$ for $i = 1, 2, 3$.

Algorithm 4.10. Computing the composition operator from the given virtual points in a parse tree.

Input: Three independent sequences Z_1, Z_2, Z_3 of virtual points in a common $\le t'$-boundaried parse tree T, where each sequence has at most $3t$ members.

Output: The $\le 3t$-boundaried composition operator represented by the virtual points of Z_1, Z_2, Z_3, computed in time $O\big(|V(T)|\big)$.

Proof. We argue similarly as in Algorithm 4.4. There is a bounded number of $\le t'$-boundaried composition operators, and each one of them may be identified by a bounded number of linear vector equations and inequations. Moreover, these equations are homogeneous and so invariant under nonsingular vector transformations by the definition of a composition operator. Thus we can decide their validity for the input from information given in the parse tree T.

Let x be a node of T labeled by $\odot^x$, and let T_x be the rooted subtree of T with the root x. Denote by T'_x, T''_x the left and right rooted subtrees of x in T, by $\bar{M}_x = \bar{P}(T_x)$, and by $Z_x \subseteq Z_1 \cup Z_2 \cup Z_3$ those of given virtual points of T which are contained in the composition operators in the subtree T_x. We use the following dynamic program:

We call *boundary-combination data* of $\bar{M}_x$ the list of all linear combinations of the virtual points Z_x which result in a point in the boundary $\partial(\bar{M}_x)$. This is a well defined notion according to the definition of a parse tree and to elementary linear algebra; and one can determine boundary-combination data of $\bar{M}_x = \bar{P}(T_x)$ from boundary-combination data of $\bar{P}(T'_x)$ and of $\bar{P}(T''_x)$, and from the composition operator $\odot^x$ labeling x. Since boundary-combination data carry limited amount of information at each tree node, the whole parse tree T can be processed in linear parametrized time.

Consider a homogeneous linear (in)equation EQ over the points $Z_1 \cup Z_2 \cup Z_3$. Then, obviously, we can decide validity of EQ from boundary-combination data at the node y of T in which the last point involved in EQ gets encountered in the set Z_y. This takes constant time. Therefore, for any $\leq 3t$-boundaried composition operator $\odot$, we can decide whether the virtual points of Z_1, Z_2, Z_3 represent $\odot$. In this way we find out the $\leq 3t$-boundaried composition operator represented by Z_1, Z_2, Z_3 in total linear (parametrized) time $O\big(|V(T)|\big)$. ∎

Lemma 4.11. *Step (3) in Algorithm 4.1 is correctly implemented in time $O(n^2)$ for fixed parameters t, $t' \leq 10t$, and $\mathbb{F}$.*

Proof. Let $M = M(\boldsymbol{A})$ be the given matroid on n elements, and let the set X and the $\leq t'$-boundaried parse tree T be as in Algorithm 4.1. (So $P(T) = M \restriction X$.) Notice that the size of the partial parse trees considered in the algorithm is at most linear in $|X| \leq n$. The important point of the implementation of step (3) in Algorithm 4.1 is that the boundaried matroids labeling the partial spanning tree T' are not explicitly described: Instead of a boundaried matroid $\bar{N}$, we record only the set $J(\bar{N})$ of its internal elements. The boundary subspace of $\bar{N}$ is then implicitly given by the guts of the separation $\big(J(\bar{N}), X - J(\bar{N})\big)$, and the boundary points are handled as virtual points in the parse tree T.

Part (3a) of Algorithm 4.1 is trivial to implement. Let us look at part (3b): The boundary rank of the boundaried matroid $\bar{N}$ labeling a chosen leaf ℓ of T' is computed in time $O(n)$ by Algorithm 4.7. Suppose that the computed rank is less than $3t$. Let $e \in J(\bar{N})$ be an arbitrary element. Then the new composition operator being added to the parse tree T' corresponds to a tri-partition $X_1 = J(\bar{N}) - \{e\}$, $X_2 = \{e\}$, and $X_3 = X - J(\bar{N})$. Rest is described below.

Suppose that the above computed boundary rank of $\bar{N}$ equals $3t$. Then we are in part (3c): We call Algorithm 4.8 for $F_0 = X_3 = X - J(\bar{N})$ over the parse tree T. The result is a partition (X_1, X_2) of the set $X - X_3 = J(\bar{N})$, which does exist by Lemma 4.6 if branch-width of M is at most $t+1$. The whole tri-partition of X to X_1, X_2, X_3 then determines the new composition operator and the new leaf labels which will be added to T'. On the other hand, if branch-width of M exceeds $t+1$, then this part may possibly fail, and then Algorithm 4.1 ends with no parse tree and an error message. Computing time is $O(n)$ here.

So far, we have constructed a tri-partition (X_1, X_2, X_3) of the set X such that $\lambda_{M \restriction X}(X_i) \leq 3t$ for $i = 1, 2, 3$, and we are going to find the composition operator that would "glue" these three parts together in an enlarged parse tree T'_1: We call Algorithm 4.9 for the separation $(X_i, X - X_i)$ over the parse tree T, for each $i = 1, 2, 3$. Then we call Algorithm 4.10 for the resulting three sequences Z_i, $i = 1, 2, 3$ of virtual points over the parse tree T to construct a composition operator $\odot'$. We construct the new tree T'_1 from T' by adding two new leaves

ℓ_1, ℓ_2 as sons of ℓ. We label ℓ_i, $i = 1, 2$ by the boundaried matroid $\bar{N}_i$ induced on the elements X_i and the boundary Z_i, and we re-label ℓ with $\odot'$. (Rest of T'_1 is unchanged.) This is all done in time $O(n)$ again.

It remains to prove correctness of the above construction by induction on $|V(T')|$. At the beginning, when T' has one node, the matroid $\bar{N}$ has boundary rank 0, and $X_3 = \emptyset$ so Z_3 is an empty sequence. Otherwise, the sequence Z_3 coincides with the set of boundary points of $\bar{N}$ since they were both determined uniquely by Algorithm 4.9 for the same separation $\big(X_3, J(\bar{N})\big)$. Therefore, $\bar{N}_1 \odot' \bar{N}_2 = \bar{N}$, and so $\bar{P}(T'_1) = \bar{P}(T')$ and $P(T'_1) = P(T) = M \restriction X$.

Since each iteration of step (3) adds a new leaf to the constructed parse tree T', there are at most $|X| - 1 = O(n)$ iterations. Thus step (3) is finished in time $O(n^2)$. We remark that our implementation works for an arbitrary finite field $\mathbb{F}$ and an integer $t \geq 1$ given as parameters. ∎

4.3 Conclusion: Implementation of Algorithm 4.1

Using Lemmas 4.5 and 4.11, and the preceeding description, we immediately conclude:

Theorem 4.12. *Let us fix an integer $t \geq 1$ and a finite field $\mathbb{F}$, and consider a given n-element $\mathbb{F}$-represented matroid $M = M(\mathbf{A})$.*

(a) If branch-width of M is at most $t + 1$, then Algorithm 4.1 computes a spanning $\leq 3t$-boundaried parse tree T for M in time $O(n^3)$.

(b) If branch-width of M exceeds $t + 1$, then Algorithm 4.1 (possibly) ends with no output parse tree, but the computation is also finished in time $O(n^3)$. ∎

5 Parametrized Complexity of Branch-Width

There are many connections between tree-width of graphs and parametrized complexity of hard graph problems [8, Chater 6]. A large class of natural graph problems, including those notoriously hard ones like hamiltonicity or 3-colouring, can be expressed in monadic second-order logic. By a classical result of Courcelle [5, 6], et al., all such MSO-definable problems can be solved quickly for (incidence) graphs with a tree-decomposition of bounded width. A similar phenomenon occurs for represented matroids, and we can use that to determine the exact value of branch-width of a represented matroid, in addition to an approximation following from Algorithm 4.1.

5.1 MSO Logic of Matroids

The *monadic second-order logic (MSOL) of matroids* is defined as follows: The syntax includes variables for matroid elements and element sets, the quantifiers $\forall, \exists$ applicable to these variables, the logical connectives $\wedge, \vee, \neg$, and the next predicates:

1. $=$, the equality for elements and their sets,
2. $e \in F$, where e is an element and F is an element set variables,
3. $\mathrm{indep}(F)$, where F is an element set variable, and the predicate tells whether F is independent in the matroid.

In our paper, we follow a tree-automata formalization of Courcelle's result, as in [1].

Theorem 5.1. (PH [13]) *Let $t \geq 1$, and let $\mathbb{F}$ be a finite field. Assume $\mathcal{M}$ is a set of represented matroids over $\mathbb{F}$ described by a sentence in the monadic second-order logic of matroids. Then there is a finite tree automaton accepting exactly the $\leq(t-1)$-boundaried parse trees of members of $\mathcal{M}$ (of branch-width bounded by t).*

We remark that the proof [13] of Theorem 5.1 is constructive — there is an algorithm that computes the accepting tree automaton for the given field $\mathbb{F}$, the formula ϕ describing $\mathcal{M}$, and t.

5.2 Parametrized Complexity

When speaking about parametrized complexity, we closely follow [8]. Here we present only the basic definition of parametrized tractability. For simplicity, we restrict the definition to decision problems, although an extension to computation problems is straightforward. Let Σ be the input alphabet. A *parametrized problem* is an arbitrary subset $A^p \subseteq \Sigma^* \times \mathbb{N}$. For an instance $(x, k) \in A^p$, we call k the *parameter* and x the input for the problem. (The parameter is sometimes implicit in the context.)

Definition. We say that a parametrized problem A^p is *(non-uniformly) fixed-parameter tractable* if there is a sequence of algorithms $\{\mathcal{A}_i : i \in \mathbb{N}\}$, and a constant c; such that $(x, k) \in A^p$ iff the algorithm $\mathcal{A}_k$ accepts (x, k), and that the running time of $\mathcal{A}_k$ on (x, k) is $O(|x|^c)$ for each k.
We say that a parametrized problem A^p is *uniformly fixed-parameter tractable* if there is an algorithm $\mathcal{A}$, a constant c, and an arbitrary function $f : \mathbb{N} \to \mathbb{N}$; such that $(x, k) \in A^p$ iff the algorithm $\mathcal{A}$ accepts (x, k), and that the running time of $\mathcal{A}$ on (x, k) is $O(f(k) \cdot |x|^c)$.

In our context, the input X is an $\mathbb{F}$-represented matroid, and the parameter k is an upper bound on the branch-width of X. (Notice that correctness of the assumed branch-width bound is implicitly checked in Algorithm 4.1.) In a more general setting, we may even consider the parameter as a pair $(\mathbb{F}, k)$ encoded as an integer.

5.3 Computing Branch-Width Exactly

We use Theorem 5.1 to determine branch-width in the following way. We remark that, unlike for graph minors, it is not known how to test for a fixed matroid minor in polynomial time.

Lemma 5.2. *For every matroid N there is an MSOL formula ψ_N such that $\psi_N \models M$ (i.e. ψ_N is true on a matroid M) if and only if N is a minor of M.*

Proof. We include a short proof here, more details can be found in [14]. Matroid N is a minor of M if there are two sets C, D such that $N = M \setminus D/C$. Suppose that $N = M \setminus D/C$ holds. Then a set $X \subseteq E(N)$ is dependent in N if and only if there is a dependent set $Y \subseteq E(M)$ in M such that $Y - X \subseteq C$.

Since N is fixed, we may identify the elements of an N-minor in M by variables $x_1, \ldots, x_n$ in order, where $n = |E(N)|$. For each $J \subseteq [1, n]$, we write

$$\mathit{mdep}(x_j : j \in J; C) \equiv \exists Y \left[\neg \operatorname{indep}(Y) \wedge \forall y \left(y \notin Y \vee y \in C \vee \bigvee\nolimits_{j \in J} y = x_j \right) \right].$$

Now, $M \setminus D/C$ is isomorphic to N iff the dependent subsets of $\{x_1, \ldots, x_n\}$ exactly match the dependent sets of N. Hence we express ψ_N as

$$\psi_N \equiv \exists C \, \exists x_1, \ldots, x_n \left[\bigwedge\nolimits_{1 \le i < j \le n} x_i \ne x_j \wedge \right.$$
$$\left. \bigwedge\nolimits_{J \in \mathcal{J}_+} \neg \mathit{mdep}(x_j : j \in J; C) \wedge \bigwedge\nolimits_{J \in \mathcal{J}_-} \mathit{mdep}(x_j : j \in J; C) \right],$$

where $\mathcal{J}_+$ is the set of all $J \subseteq [1, n]$ such that $\{x_j : j \in J\}$ actually is independent in N, and where $\mathcal{J}_-$ is the complement of $\mathcal{J}_+$. ∎

The class $\mathcal{B}_k$ of matroids of branch-width at most k is closed under taking minors, and so membership in $\mathcal{B}_k$ can be tested by looking for the so called excluded (or forbidden) minors for $\mathcal{B}_k$. By the result of [10], the excluded minors for the class $\mathcal{B}_k$ have size at most $(6^{k+1} - 1)/5$, and hence their number is finite and they can all be found by a brute force algorithmic search.

Corollary 5.3. *For every $k \ge 1$, there is a computable MSOL formula ϕ_k such that $\phi_k \models M$ if and only if M has branch-width at most k (i.e. shortly $\phi_k(M) \equiv M \in \mathcal{B}_k$).*

∎

Theorem 5.4. *Let $\mathbb{F}$ be a finite field, and let $t \ge 1$ be a constant. There is an algorithm that, given a rank-r matrix $\boldsymbol{A} \in \mathbb{F}^{r \times n}$ such that the branch-width of the matroid $M(\boldsymbol{A})$ is at most $t + 1$, finds the exact branch-width of $M(\boldsymbol{A})$ in time $O(n^3)$.*

Proof. For $k = 2, 3, \ldots, t + 1$, the algorithm first pre-computes all the excluded minors for the class $\mathcal{B}_k$, and the formulas ϕ_k from Corollary 5.3. Then the algorithm computes the finite tree automaton $\mathcal{A}_k$ from Theorem 5.1, which accepts the parse trees for represented matroids described by ϕ_k. (This pre-computation is done in time not depending on M.)

Given an n-element matroid $M(\boldsymbol{A})$ represented over $\mathbb{F}$, the algorithm calls Algorithm 4.1 to produce an $\le 3t$-boundaried parse tree T of $M(\boldsymbol{A})$ in time $O(n^3)$. Then it finds the smallest $k_0 \le t + 1$ such that the parse tree T is accepted by $\mathcal{A}_{k_0}$, where each automaton $\mathcal{A}_k$ is emulated in time $O(n)$. The branch-width of $M(\boldsymbol{A})$ is k_0. ∎

Corollary 5.5. *For a finite field $\mathbb{F}$, the branch-width of an $\mathbb{F}$-represented matroid is a uniformly fixed-parameter tractable problem.*

∎

Remark. We may analogously argue in the case of matroid tree-width, which has been defined in [15]: The tree-width of a matroid represented over a finite field is non-uniformly fixed-parameter tractable. However, we do not have a size-bound analoguous to [10] at hand, and so we have to use a non-constructive well-quasi-ordering argument of [9] to establish existence of a finite list of excluded minors for represented matroids of tree-width at most k. Since the definition of matroid tree-width is not easy, we include no formal statements here.

6 Concluding Remarks

Using Theorem 4.12, one may easily derive the following corollary of Theorem 5.1.

Corollary 6.1. *Let $t \geq 1$, let $\mathbb{F}$ be a finite field, and let ϕ be a sentence in the monadic second-order logic of matroids. Consider a given n-element $\mathbb{F}$-represented matroid $M = M(\boldsymbol{A})$ of branch-width at most t. The question whether ϕ is true for the matroid $M(\boldsymbol{A})$ is uniformly fixed-parameter tractable with respect to the combined parameter $(\mathbb{F}, t, \phi)$. If $\mathbb{F}$, t, and ϕ are fixed, then the answer can be computed from the matrix $\boldsymbol{A}$ in time $O(n^3)$.*

More similar algorithmic applications, including recognition of any minor-closed matroid family, can be found in [14]. Besides applications based directly on Theorem 5.1, we may use the machinery of matroid parse trees from Sections 3,4 for solving other problems. For example, we provide a straightforward recursive formula and an algorithm for computing the Tutte polynomial of a represented matroid in [12].

Theorem 6.2. (PH [12]) *Let $t \geq 1$, let $\mathbb{F}$ be a finite field. Consider a given n-element $\mathbb{F}$-represented matroid $M = M(\boldsymbol{A})$ of branch-width at most t. Then the Tutte polynomial $T(M(\boldsymbol{A}); x, y)$ of $M(\boldsymbol{A})$ can be computed in time $O(n^6 \log n \log\log n)$.*

Morever, a recent research of Oum shows that our matroid results are of interest also in graph theory — he uses Algorithm 4.1 to approximate the clique-width of a graph in parametrized cubic time [18]. That application uses important notion of rank-width [17], defined by the matrix rank function on adjacency matrices of graphs. (There had been no efficient approximation algorithms for graph clique-width known before introduction of rank-width.) In particular, the clique-width of a graph of rank-width r is between r and $2^{r+1} - 1$, and the rank-width of a bipartie graph G equals the branch-width minus one of the matroid represented over $GF(2)$ by the bipartite adjacency matrix of G. Oum's result is shortly stated as follows:

Theorem 6.3. (Oum [18]) *For every fixed $r > 0$, there is an algorithm checking whether the rank-width of a given graph G is at most r in time $O(n^3)$. Moreover, the algorithm outputs a rank-decomposition of with at most $24r$ in the "yes" case.*

It is interesting to watch the radical structural change when we move from represented matroids over finite fields to general abstract matroids, or even to matroids over infinite fields. For example, Theorem 5.1 [13] is provably false even for matroids that are represented over the integers by matrices with entries from $\{-1, 1, 3\}$. Also, the problems of 6.1 and 6.2 become NP-hard for matroids of branch-width 3 over the integers. The computational borderline is even more clear when considering decidability of matroid theories [16]: While MSO theories of the matroids of bounded branch-width are decidable over finite fields (and, conversely, decidability of such a theory implies a bound on branch-width), the MSO theory of all matroids of branch-width 3 is undecidable.

The situation seems to be slightly different for the problem of branch-width itself, at least when branch-width is 3. We present in [11] an easy algorithm that decides whether a matroid has branch-width at most 3 in polynomial time. (The algorithm also has a fast practical implementation.) This algorithm is not restricted to represented matroids — it works for all matroids for which the rank function can be efficiently determined. Unfortunately, there seems to be no straightforward way how to extend the algorithm to higher values of branch-width.

Problem 6.4. What is the parametrized complexity of the problem to determine the branch-width of a matroid M;
(a) if $M = M(\boldsymbol{A})$ is given by a matrix representation over an infinite field,
(b) if M is given by a rank oracle?

Acknowledgments

We would like to thank Geoff Whittle for valuable discussions about matroid representations and branch-width, and the anonymous referee for helpful suggestions to improve the presentation of our results. We also acknowledge generous research support by the NZ Marsden Fund (a research grant to Geoff Whittle) in 2001–2002, and partial support by Czech research grant GAČR 201/05/050.

References

1. K.A. Abrahamson, M.R. Fellows, *Finite Automata, Bounded Treewidth, and Well-Quasiordering*, In: Graph Structure Theory, Contemporary Mathematics 147, American Mathematical Society (1993), 539–564.
2. H.L. Bodlaender, *A Tourist Guide through Treewidth*, Acta Cybernetica 11 (1993), 1–21.
3. H.L. Bodlaender, *A Linear Time Algorithm for Finding Tree-Decompositions of Small Treewidth*, SIAM J. Computing 25 (1996), 1305–1317.

4. H.L. Bodlaender, D.M. Thilikos, *Constructive Linear Time Algorithms for Branch-width*, Proceedings 24th ICALP, Lecture Notes in Computer Science 1256, 1997, 627–637.
5. B. Courcelle, *The decidability of the monadic second order theory of certain sets of finite and infinite graphs*, LICS'88, Logic in Computer Science, Edinburg, 1988.
6. B. Courcelle, *The Monadic Second-Order Logic of Graphs I. Recognizable sets of Finite Graphs* Information and Computation 85 (1990), 12–75.
7. R. Diestel, Graph theory, Graduate Texts in Mathematics 173, Springer-Verlag, New York 1997, 2000.
8. R.G. Downey, M.R. Fellows, Parametrized Complexity, Springer-Verlag New York, 1999, ISBN 0-387-94833-X.
9. J.F. Geelen, A.H.M. Gerards, G.P. Whittle, *Branch-Width and Well-Quasi-Ordering in Matroids and Graphs*, J. Combin. Theory Ser. B 84 (2002), 270–290.
10. J.F. Geelen, A.H.M. Gerards, N. Robertson, G.P. Whittle, *On the Excluded Minors for the Matroids of Branch-Width k*, J. Combin. Theory Ser. B 88 (2003), 261–265.
11. P. Hliněný, *On the Excluded Minors for Matroids of Branch-Width Three*, Electronic Journal of Combinatorics 9 (2002), `http://www.combinatorics.org`, #R32.
12. P. Hliněný, *The Tutte Polynomial for Matroids of Bounded Branch-Width*, Combinatorics, Probability and Computing, to appear (2005).
13. P. Hliněný, *Branch-Width, Parse Trees, and Monadic Second-Order Logic for Matroids*, submitted.
    Extended abstract in: STACS 2003, Lecture Notes in Computer Science 2607, Springer Verlag (2003), 319–330.
14. P. Hliněný, *On Matroid Properties Definable in the MSO Logic*, In: Math Foundations of Computer Science MFCS 2003, Lecture Notes in Computer Science 2747, Springer Verlag Berlin (2003), 470–479.
15. P. Hliněný, G.P. Whittle, *Matroid Tree-Width*, submitted (2003).
    Extended abstract in: Eurocomb'03, ITI Series 2003–145, Charles University, Prague, Czech Republic, 202–205.
16. P. Hliněný, D. Seese, *Trees, Grids, and MSO Decidability: from Graphs to Matroids*, submitted (2004).
    Extended abstract in: IWPEC 2004, Lecture Notes in Computer Science 3162, Springer Verlag Berlin (2004), 96–107.
17. Sang-Il Oum, P. Seymour, *Approximation Algorithm to the Clique-Width of a Graph*, manuscript, 2003.
18. Sang-Il Oum, *Approximating Rank-width and Clique-width Quickly*, manuscript, 2005.
19. J.G. Oxley, Matroid Theory, Oxford University Press, 1992,1997, ISBN 0-19-853563-5.
20. N. Robertson, P.D. Seymour, *Graph Minors – A Survey*, Surveys in Combinatorics, Cambridge Univ. Press 1985, 153–171.
21. N. Robertson, P.D. Seymour, *Graph Minors X. Obstructions to Tree-Decomposition*, J. Combin. Theory Ser. B 52 (1991), 153–190.