A Tight Monte-Carlo Algorithm for Steiner Tree Parameterized by Clique-Width

Narek Bojikian
Humboldt-Universität zu Berlin, Germany

Stefan Kratsch
Humboldt-Universität zu Berlin, Germany

Abstract

Given a graph $G = (V, E)$, a set $T \subseteq V$, and an integer $\bar{b}$, the Steiner Tree problem asks whether G has a connected subgraph H with at most $\bar{b}$ vertices that spans all of T. This work presents a $3^k \cdot n^{\mathcal{O}(1)}$ time one-sided Monte-Carlo algorithm for solving Steiner Tree when additionally a clique-expression of width k is provided. Known lower bounds for less expressive parameters imply that this dependence on the clique-width of G is optimal assuming the Strong Exponential-Time Hypothesis (SETH). Indeed our work establishes that the parameter dependence of Steiner Tree is the same for any graph parameter between cutwidth and clique-width, assuming SETH.

Our work contributes to the program of determining the exact parameterized complexity of fundamental hard problems relative to structural graph parameters such as treewidth, which was initiated by Lokshtanov et al. [SODA 2011 & TALG 2018] and which by now has seen a plethora of results. Since the cut-and-count framework of Cygan et al. [FOCS 2011 & TALG 2022], connectivity problems have played a key role in this program as they pose many challenges for developing tight upper and lower bounds. Recently, Hegerfeld and Kratsch [ESA 2023] gave the first application of the cut-and-count technique to problems parameterized by clique-width and obtained tight bounds for Connected Dominating Set and Connected Vertex Cover, leaving open the complexity of other benchmark connectivity problems such as Steiner Tree and Feedback Vertex Set.

Our algorithm for Steiner Tree does not follow the cut-and-count technique and instead works with the connectivity patterns of partial solutions. As a first technical contribution we identify a special family of so-called complete patterns that has strong (existential) representation properties, and using these at least one solution will be preserved. Furthermore, there is a family of 3^k basis patterns that (parity) represents the complete patterns, i.e., it has the same number of solutions modulo two. Our main technical contribution, a new technique called “isolating a representative,” allows us to leverage both forms of representation (existential and parity). Both complete patterns and isolation of a representative will likely be applicable to other (connectivity) problems.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized complexity, Steiner tree, clique-width

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.29

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2307.14264` [5]

1 Introduction

This work presents a $3^k \cdot n^{\mathcal{O}(1)}$ time one-sided Monte-Carlo algorithm for Steiner Tree[cw], i.e., Steiner Tree parameterized by clique-width.[1] Under the Strong Exponential-Time Hypothesis (SETH)[2] this dependence on the clique-width is optimal, i.e., time $(3 - \varepsilon)^k \cdot n^{\mathcal{O}(1)}$

[1] Given a graph $G = (V, E)$, a set $T \subseteq V$ of terminals, a number $\bar{b}$, and a k-clique-expression of G, with k being the parameter, is there a connected subgraph H of G with at most $\bar{b}$ vertices that spans all of T?
[2] SETH posits that for each $c < 2$ there is $q \in \mathbb{N}$ such that q-CNF SAT cannot be solved in time $\mathcal{O}(c^n)$.

is known to be impossible [15]. Our algorithm relies on a new technique, called *isolating a representative*, that allows it to employ both counting modulo two and reduction to a specific smaller family of representative partial solutions. Implicitly, it thus works with a reduced compatibility matrix of $GF(2)$-rank 3^k, rather than the full matrix of $GF(2)$-rank 4^k, which is key for obtaining the claimed tight running time.

Our work fits into the program of determining the exact complexity of parameterized problems (modulo SETH), especially relative to width parameters like treewidth, which was initiated by Lokshtanov et al. [35, 36]. Since their work, there has been a plethora of such tight bounds relative to treewidth, see, e.g., [7, 13, 12, 14, 40, 26, 37, 41, 18, 22, 32, 21], but also relative to other parameters such as cutwidth [45, 30, 37, 44, 24, 4], modular treewidth [34, 28], and clique-width [29, 13, 31, 34, 23, 27]. An important breakthrough was made by Cygan et al. [15, 17], who were the first to obtain tight complexity bounds for connectivity problems such as Steiner Tree, Connected Vertex Cover, and Feedback Vertex Set (all parameterized by treewidth). In particular, their novel cut-and-count paradigm led to the surprising result that many of these problems admit $\mathcal{O}^*(c^{tw})$ time (randomized) dynamic programming (DP) algorithms. Intuitively, this is done by counting cuts of relaxed solutions modulo two and relying on cancellation of disconnected solutions. Note that this makes such algorithms inherently probabilistic as they rely on the isolation lemma to reduce, with high probability, to the case that there is a unique solution of minimum weight. In this way, they are able to reduce decision to counting modulo two.

The exact complexity of Steiner Tree[cw] is a natural and important question: Steiner Tree is central among connectivity problems, which have been key benchmarks in the exact complexity program. Clique-width is a well-studied graph parameter that, in particular, also allows to capture structure present in dense graphs (unlike treewidth and cutwidth etc.). So far, there are only two results regarding exact complexity (modulo SETH) of connectivity problems relative to clique-width: Hegerfeld and Kratsch [27] showed tight bounds of $6^k \cdot n^{\mathcal{O}(1)}$ for Connected Vertex Cover[cw] and $5^k \cdot n^{\mathcal{O}(1)}$ for Connected Dominating Set[cw], with both algorithms relying on cut-and-count. They explicitly leave open the exact complexities of Steiner Tree[cw], Connected Odd Cycle Transversal[cw], and Feedback Vertex Set[cw] as not being within range of their techniques. For the latter, the algorithm for Feedback Vertex Set[tw] [17] counts edges outside the tentative feedback vertex set to ensure that no cycles remain, but this seems infeasible relative to clique-width (cf. [1, 2]). For the former two, it is mentioned that their approach via cut-and-count would give time $4^k \cdot n^{\mathcal{O}(1)}$ for Steiner Tree[cw] and time $14^k \cdot n^{\mathcal{O}(1)}$ for Connected Odd Cycle Transversal[cw]. However, matching lower bounds were not in sight, as current lower bound methods rely on sufficiently large triangular submatrices inside certain compatibility matrices that correspond to the problem, which were lacking. Indeed, it is this gap between upper and lower bound that our technique allows us to close.

Our result and techniques. Formally, our result for Steiner Tree[cw] reads as follows.

▶ **Theorem 1.** *There is a one-sided error Monte Carlo algorithm (no false positives) that, given a graph $G = (V, E)$, a set of terminals $T \subseteq V$, a number $\overline{b}$, and a k-clique-expression of G, takes time $3^k \cdot n^{\mathcal{O}(1)}$ and determines, with high probability, whether a connected subgraph H of G of exactly $\overline{b}$ vertices exists that spans all of T.*

Since the clique-width of any graph G is at most its pathwidth plus two (folklore), i.e., $\mathrm{cw}(G) \leq \mathrm{pw}(G)+2$, the lower bound for Steiner Tree[pw] [15] rules out time $(3-\varepsilon)^k \cdot n^{\mathcal{O}(1)}$ for Steiner Tree[cw]. In fact, recent work of Bojikian et al. [4] shows that the same lower

Table 1 Tight complexity bounds (modulo SETH) for a selection of (connectivity) problems relative to cutwidth, treewidth, modular-treewidth, and clique-width. (Table adapted from [27].)

	cutwidth	treewidth	modular-tw	clique-width
q-Coloring	$\mathcal{O}^*(2^k)$	$\mathcal{O}^*(q^k)$	$\mathcal{O}^*(\binom{q}{\lfloor q/2 \rfloor}^k)$	$\mathcal{O}^*((2^q-2)^k)$
Vertex Cover	$\mathcal{O}^*(2^k)$	$\mathcal{O}^*(2^k)$	$\mathcal{O}^*(2^k)$	$\mathcal{O}^*(2^k)$
Connected Vertex Cover	$\mathcal{O}^*(2^k)$	$\mathcal{O}^*(3^k)$	$\mathcal{O}^*(5^k)$	$\mathcal{O}^*(6^k)$
Connected Dominating Set	$\mathcal{O}^*(3^k)$	$\mathcal{O}^*(4^k)$	$\mathcal{O}^*(4^k)$	$\mathcal{O}^*(5^k)$
Steiner Tree	$\mathcal{O}^*(3^k)$	$\mathcal{O}^*(3^k)$	$\mathcal{O}^*(3^k)$	$\boldsymbol{\mathcal{O}^*(3^k)}$
Feedback Vertex Set	$\mathcal{O}^*(2^k)$	$\mathcal{O}^*(3^k)$	$\mathcal{O}^*(5^k)$	?
Connected Odd Cycle Transversal	$\mathcal{O}^*(4^k)$	$\mathcal{O}^*(4^k)$	?	?
References	[4, 30, 45]	[16, 17, 36]	[28, 34]	[27, 34]

bound already holds relative to cutwidth. Thus, for all parameters between cutwidth and clique-width we get the same tight dependence of $3^k \cdot n^{\mathcal{O}(1)}$ for solving Steiner Tree, which contrasts the growing dependence exhibited by most other problems (see Table 1).

Our algorithm does not apply cut-and-count, i.e., counting consistent cuts of relaxed partial solutions. Instead, it works with the connectivity patterns of partial solutions for the graphs that correspond to parts of the given k-clique-expression. Roughly, a connectivity pattern indicates how the connected components of a partial solution connect the different label classes, noting that several components may intersect the same label class. Of course, the full set of possible patterns is much too large to be of use for us. Instead, we identify a family of patterns, called *complete patterns*, that are representative for the class of all patterns in a strong and constructive sense: For each pattern p there is a set $R_p = \{q_1, \ldots, q_\ell\}$ of complete patterns that together completes into exactly the same set of solutions as p does. Moreover, we identify a subfamily of 3^k basis patterns such that for each complete pattern there is a set of basis patterns that completes into the same number *modulo two* of solutions; again, this necessitates randomization through the use of the isolation lemma to guarantee existence of a unique minimum weight solution.

Now, if we simply combine these two means of representation, namely existential and modulo two, we will not end up with a correct algorithm: Existential representation makes no guarantee about the number, and hence the parity, of representations, even if there is a unique global solution. This is where our technique of *isolating a representative* comes in: When a step in the DP creates a partial solution whose connectivity pattern p is not complete, then each representing complete pattern has its weight increased by a globally set random weight (with four weights per DP step sufficing for our algorithm). In this way, a unique minimum weight representation is isolated with high probability. Combined with the usual setup of isolating a unique solution of minimum weight (and fast convolution), this allows our algorithm to count modulo two the partial solutions relative to the 3^k basis patterns and arrive at the correct answer with high probability. Overall, we have two applications of the isolation lemma, once for isolating a solution and once for isolating a representation of the solution. This makes our algorithm inherently probabilistic, though with low error bounds, and it may fail (like preceding work) by returning a false negative when either isolation fails. Success chance (of this one-sided error algorithm) can be boosted in the usual way.

Our approach can be easily generalized to solve the vertex-weighted version of this problem, when weights are bounded polynomially by the size of the graph, in the obvious way with weights replacing cardinalities of partial solutions. This contrasts the status of the edge-weighted version of this problem, which can be easily seen to be para-NP-hard when parameterized by clique-width: One can reduce any instance of the unweighted Steiner Tree problem to the edge-weighted version of this problem on a complete graph (of clique-width 2) by assigning weight one to existing edges, and some large weight to all other edges.

Related work. It has been observed that many results on exact parameterized complexity are closely related to properties of certain matrices that are related to the problem in question and the considered graph structure, and this was recently surveyed by Nederlof [39]: E.g., methods such as cut-and-count rely on low-rank factorizations of the corresponding matrices. Since cut-and-count works modulo two, i.e., over $GF(2)$, and since it can be verified that the relevant matrix for Steiner Tree[cw] has $GF(2)$-rank (at least) 4^k, this makes it unlikely that a pure cut-and-count approach could succeed in getting time $3^k \cdot n^{\mathcal{O}(1)}$. (Indeed, it was this fact combined with the intuition that current lower bound techniques would not stretch further than the $(3-\varepsilon)^k \cdot n^{\mathcal{O}(1)}$, known already relative to pathwidth, that motivated the present work.) This leaves open the possibility that the same time bound of $3^k \cdot n^{\mathcal{O}(1)}$ could be obtained by working over a different field and/or studying the support rank[3] of the matrix (cf. [39]). So far, there are few results that obtain tight complexity bounds that do not coincide with the rank of the corresponding matrices [30, 4].

Clique-width was introduced by Courcelle and Olariu [11] building on work of Courcelle et al. [9] and it is similar to the NLC-width of Wanke [46]. Courcelle et al. [10] showed that every graph problem expressible in MSO_1 logic (monadic second order logic of graphs with quantification over vertex sets but not edge sets) can be solved in linear time for graphs with a given k-clique-expression, i.e., in time $f(k) \cdot n$, though the function f depends on the formula capturing the problem and may be non-elementary (cf. [33]), so likely far from being tight. This tractability is not restricted to MSO_1-expressible problems but many other important problems are solvable in time $f(\mathrm{cw}) \cdot n^{\mathcal{O}(1)}$ or at least time $n^{f(\mathrm{cw})}$ (see, e.g., [19]). Nevertheless, some problems, like Disjoint Paths, are NP-complete on graphs of bounded clique-width, while being solvable in time $f(k) \cdot n^{\mathcal{O}(1)}$ with respect to treewidth (cf. [25]). The first single-exponential time algorithms for connectivity problems parameterized by clique-width were given by Bergougnoux and Kanté [1, 2], e.g., Steiner Tree, Connected Dominating Set, and Connected Vertex Cover each in time $2^{\mathcal{O}(\mathrm{cw})} \cdot n$, but building on the rank-based approach these bounds are likely not tight.

The main drawback of clique-width lies in the difficulty of finding good expressions, i.e., good bounds on the clique-width of given graphs, in reasonable time. Fellows et al. [20] showed that it is NP-complete to determine, on input of graph G and integer k, whether the clique-width of G is at most k. It is open, however, whether for each fixed value of k there is an efficient algorithm for recognizing graphs of clique-width at most k; such algorithms are known only for $k \leq 3$ [8]. In particular, there is neither an FPT-algorithm for Clique-Width[k] known, nor is it known to be W[1]-hard with respect to k. The arguably best way for using low clique-width is an exponential-ratio FPT-algorithm due to Seymour and Oum [43] (made faster by Oum [42]). That being said, better algorithms for computing clique-width are still possible, and there may be variants of clique-width that give similar complexity (used as parameters) but are easier to compute.

Organization. In Section 3 we introduce connectivity patterns and provide related notation. In Section 4 we show how to represent partial solutions through patterns. In Section 5 we show that one can represent arbitrary patterns using complete patterns only. In Section 6 we

[3] The smallest rank of any matrix with the same zero vs. non-zero pattern over any field, so this is only interesting for fields other than $GF(2)$.

show that one can count complete patterns by reducing them a smaller family of patterns. In Section 7 we show how to count these families efficiently, providing the main algorithm of this paper. Finally, we conclude with final thoughts in Section 8.

2 Preliminaries

In this work we deal with undirected graphs only. Given a natural number k, we denote by $[k] = \{1, 2, \dots k\}$ the set of natural numbers smaller than or equal to k, and $[k]_0 = [k] \cup \{0\}$. A *labeled graph* is a graph $G = (V, E)$ together with a *labeling function* $\text{lab}\colon V \to \mathbb{N}$. We usually omit the function lab and assume that it is given implicitly when defining a labeled graph G. We say that G is *k-labeled*, if it holds that $\text{lab}(v) \leq k$ for all $v \in V$. We define a *clique-expression* μ as a well-formed expression that consists only of the following operations on labeled graphs:

- *Create vertex* $i(v)$ for $i \in \mathbb{N}$. This operation constructs a graph containing a single vertex and assigns label i to this vertex.
- The *union* operation $G_1 \uplus G_2$. The constructed graph consists of the disjoint union of the labeled graphs G_1 and G_2.
- The *relabel* operation $\rho_{i\to j}(G)$ for $i, j \in \mathbb{N}$. This operations changes the labels of all vertices in G labeled i to the label j.
- The *join* operation $\eta_{i,j}(G)$ for $i, j \in \mathbb{N}, i \neq j$. The constructed graph results from G by adding all edges between the vertices labeled i and the vertices labeled j, i.e.

$$\eta_{i,j}(G) = (V, E \cup \{\{u, v\}\colon \text{lab}(u) = i \wedge \text{lab}(v) = j\}).$$

We denote the graph resulting from a clique-expression μ by G_μ, and the constructed labeling function by lab_μ. We associate with a clique-expression μ a syntax tree $\mathcal{T}_\mu$ (we omit the symbol μ when clear from the context) in the natural way, and associate with each node $x \in V(\mathcal{T})$ the corresponding operation. For $x \in V(\mathcal{T})$, the subtree rooted at x induces a subexpression μ_x. We define $G_x = G_{\mu_x}$, $V_x = V(G_x)$, $E_x = E(G_x)$ and $\text{lab}_x = \text{lab}_{\mu_x}$. Finally, we say that a clique-expression μ is a *k-expression* if G_x is a k-labeled graph for all $x \in V(\mathcal{T})$. We define the clique-width of a graph G (denoted by $\text{cw}(G)$) as the smallest value k such that there exists a k-expression μ with G_μ isomorphic to G. We can assume without loss of generality, that any given k-expression defining a graph $G = (V, E)$ uses at most $O(|V|)$ union operations, and at most $O(|V|k^2)$ unary operations [1, 11].

We define the Steiner Tree problem as follows: Given an undirected graph $G = (V, E)$, a terminal set $T \subseteq V$ and some value $\overline{b} \in \mathbb{N}$, asked is whether there exists a set of vertices $S \subseteq V$ of size at most $\overline{b}$, such that $T \subseteq S$ and $G[S]$ is connected. For $k \in \mathbb{N}$, and f some computable function, we denote by $\mathcal{O}^*(f(k))$ the running time $f(k)\,\text{poly}(n)$, where n is the size of the input.

Let $\mathbb{F}$ be a field, and let S be some set. For a vector $T \in \mathbb{F}^S$, and some value $x \in S$, we denote by $T[x]$ the element of T indexed by x. In general, we will only deal with binary vectors, i.e. $\mathbb{F} = \text{GF}(2)$. By $\overline{0}_S$ we denote the vector T with $T[x] = 0$ for all $x \in S$. We omit the subscript S when clear from the context. For two sets S, T, let $M \in \mathbb{F}^{S\times T}$ be some matrix, whose rows are indexed by S and columns by T. We denote by $M[s, t]$ the element of M indexed by s and t.

For some integer k, given sets $S_1, T_1, \dots, S_k, T_k$, let $M_i \in \mathbb{F}^{S_i \times T_i}$ for each $i \in [k]$. We define the Kronecker product of $M_1, \dots M_k$ as the matrix $M = M_1 \otimes \cdots \otimes M_k \in \mathbb{F}^{(S_1\times\cdots\times S_k)\times(T_1\times\cdots\times T_k)}$, where for $s_i \in S_i$ and $t_i \in T_i$ for all $i \in [k]$,

$$M[(s_1, \dots, s_k), (t_1, \dots, t_k)] = \prod_{i\in[k]} M_i[s_i, t_i].$$

It is well-known that $\text{rk}(M_1 \otimes \cdots \otimes M_k) = \text{rk}(M_1) \cdot \text{rk}(M_2) \cdot \dots \cdot \text{rk}(M_k)$.

Given two sets S, T, we denote by $S \triangle T$ the symmetric difference of S and T. It holds by a simple counting argument that $|S \triangle T| \equiv_2 |S| + |T|$. Given a finite set U, the power-set of U (denoted by 2^U) is the set of all subsets of U. A partition is a family of pairwise disjoint subsets of U that span all its elements.

A *lattice* is a partial order $(\mathcal{L}, \preceq)$ over a set $\mathcal{L}$, such that any two elements have a greatest lower bound (called *meet*), and a least upper bound (called *join*). We call two lattices isomorphic, if the underlying orders are isomorphic.

Let $f : U \to V$ be a mapping between two sets U and V, and $x \notin U$ a new element. Let $i \in V$. We define the extension $f|_{x \mapsto i} : U \cup \{x\} \to V$ of f in the natural way. We also define the disjoint union of two mappings defined over two disjoint sets in the natural way.

Let U, V be two sets, and $f : U^k \to V$ a mapping. Given sets $S_1, \dots S_k \subseteq U$, we define

$$f(S_1, \dots S_k) = \{f(s_1, \dots s_k) \colon (s_1, \ \dots s_k) \in S_1 \times S_2 \times \dots S_k\}.$$

For a set $S \subseteq V$ we also define

$$f^{-1}(S) = \{(s_1, \ \dots s_k) \in S_1 \times S_2 \times \dots S_k \colon f(s_1, \dots s_k) \in S\}.$$

An exception to this notation, is when we explicitly mention that $f : U \to V$ is a weight function, and $V \subseteq \mathbb{Z}$. In this case, for some set $S \subseteq U$ we define $f(S) = \sum_{u \in S} f(u)$, where we compute the sum over $\mathbb{Z}$.

Given a logical expression ρ, we denote by $[\rho]$ the Iverson bracket of ρ. We refer the reader to the full version [5] for formal definitions of all mentioned notation.

3 Connectivity patterns

3.1 Definition and terminology

Along this work, let U be some totally ordered ground set. In this section we define the main structures we build this paper on, we call them patterns. A pattern, as we shall see, represents the state of a partial solution, and carries enough information to extend it to a solution over the whole graph.

▶ **Definition 2.** *Let 0 be an element not in U. A* pattern *p is a subset of the power-set of $U \cup \{0\}$, such that there exists exactly one set of p containing the element 0. With $\mathcal{P}(U)$ we denote the family of all patterns over U. We omit U when clear from the context. By $Z_p \in p$ we denote the only set of p containing the element 0, and we call it the* zero-set *of p.*

Given a pattern $p \in \mathcal{P}$, we define label$(p) \subseteq U$ *as the set of all labels in U occurring in p, and with* singleton$(p) \subseteq U$ *the set of all labels in U appearing as singletons in p, i.e.*

$$\text{label}(p) = \bigcup_{S \in p} S \setminus \{0\}, \qquad \textit{and} \qquad \text{singleton}(p) = \{u \colon \{u\} \in p \wedge u \neq 0\}.$$

▶ **Definition 3.** *We define a more concise way to write patterns, that we use quite often along this paper. For a pattern $p = \{\{u_1^1, \dots, u_{r_1}^1\}, \dots, \{u_1^\ell, \dots, u_{r_\ell}^\ell\}\}$, we write*

$$p = [u_1^1 u_2^1 \dots u_{r_1}^1, \dots, u_1^\ell \dots u_{r_\ell}^\ell],$$

where we use square brackets to enclose the pattern, we do not use any separator between the elements of the same set, and separate different sets with comas. We also sometimes omit the symbol 0 when it appears as a singleton.

When we use this notation, we assume that each element is represented by a single symbol. Hence, such patterns admit a unique interpretation. For example, both $[12]$ and $[0, 12]$ denote the pattern $\{\{0\}, \{1, 2\}\}$, while the pattern $\{\{0, 10\}\}$ cannot be written in this concise way.

Now we introduce pattern operations. These operations will allow us to extend families of patterns over the syntax tree of a clique-expression to build larger solutions recursively. We define these operations in a similar way to the operations on partitions defined in [3].

▶ **Definition 4.** *Let $p, q \in \mathcal{P}$. We define the following operations:*

Join: $r = p \sqcup q$. *Let* $\sim_I = \{(S,T)\colon S \in p \wedge T \in q \wedge S \cap T \neq \emptyset\}$ *be a relation over* $p \cup q$, *and* $\sim_I^T$ *be its reflexive, transitive and symmetric closure. Let* $\mathcal{R}$ *be the equivalence classes of* $\sim_I^T$, *then* $r = \{\bigcup_{S \in P} S \colon P \in \mathcal{R}\}$.

Relabel: $r = p_{i \to j}$, *for* $i, j \in U$, *where* r *results from* p *by replacing* i *with* j *in each set of* p.

Union: $r = p \oplus q$, *where* $r = (p \setminus \{Z_p\}) \cup (q \setminus \{Z_q\}) \cup \{Z_p \cup Z_q\}$.

▶ **Definition 5.** *For* $i, j \in U, i \neq j$, *we define the operation* $\square_{i,j} p$ *for* $p \in \mathcal{P}$ *as*

$$\square_{i,j} p = \begin{cases} p & : \{i,j\} \not\subseteq \mathrm{label}(p), \\ p \sqcup [ij] & : \textit{otherwise}. \end{cases}$$

This combines all sets containing the labels i *or* j *into one set, if both labels appear in* p.

▶ **Definition 6.** *Let* $p, q \in \mathcal{P}$. *We say that* p *and* q *are* consistent *(denoted by* $p \sim q$*), if for* $r = p \sqcup q$ *it holds that* $r = \{Z_r\}$ *contains the zero-set only, i.e. the join operation merges all sets into one set only. We say that a pattern* p dominates *another pattern* q *(denoted by* $p \geq q$*), if it holds for all patterns* $r \in \mathcal{P}$ *that* $q \sim r$ *implies* $p \sim r$.

3.2 Characteristics of patterns

In the full version of this paper we introduce the notion of alternating walks between two patterns. Using this notion, we provide a new characterization of the join operation, upon which, we prove a multitude of properties of patterns. We use these properties to prove domination and consistency results on manipulated and derived patterns. In particular, this allows us to correctly reduce the states of a dynamic programming solution keeping the transitions correct. We present these properties here and refer the reader to the full version for a formal proof.

▶ **Lemma 7.** *Let* $p, q, r \in \mathcal{P}$ *be arbitrary patterns, and* $i, j \in U$ *two arbitrary labels, with* $i \neq j$. *Then all the following is true:*

1. *It holds that* $\square_{i,j} p \geq p$
2. *If* $\{i,j\} \subseteq \mathrm{label}(p)$, *then* $p \sqcup [ij] \sim q$ *if and only if* $p \sim q \oplus [ij, i, j]$.
3. *If* $\mathrm{singleton}(p) \setminus \mathrm{label}(q) \neq \emptyset$, *then* $p \not\sim q$.
4. *For* $S \in p$, *and* $u \in S$, *let* $S' = S \setminus \{u\}$, *and* $p' = (p \setminus \{S\}) \cup \{S'\}$. *It holds that* $p \geq p'$.
5. *For* $S \in p$, $S' \subseteq S$, *and* $p' = p \cup \{S'\}$, *it holds that* $p \geq p'$.
6. *If* $p \sim q$ *holds, then it must hold that* $\mathrm{label}(p) = \mathrm{label}(q)$ *and* $\mathrm{singleton}(p) = \mathrm{singleton}(q)$.
7. *Let* q' *be the pattern that results from* q *by removing all labels in* $\mathrm{label}(q) \setminus \mathrm{label}(p)$ *from each set of* q. *Then* $p \sim q$ *if and only if* $p \sim q'$.

▶ **Definition 8.** *Given a pattern* $p \in \mathcal{P}$ *and two different labels* $i, j \in U$. *We define the operation* $p' = p_{j \frown i}$ *over patterns, where* p' *results from* p *by removing the label* i *from all sets, and then adding* i *to each set that contains the label* j, *i.e.*

$$p_{j \frown i} = \bigcup_{\substack{S \in p \\ j \notin S}} S \setminus \{i\} \cup \bigcup_{\substack{S \in p \\ j \in S}} S \cup \{i\}.$$

▶ **Lemma 9.** *For all $p, q, r \in \mathcal{P}$ the following two equivalences hold*

1. *It holds that $p \sqcup q \sim r$ if and only if $p \sim q \sqcup r$.*
2. *For $i, j \in U$ with $i \neq j$, it holds that $p_{i \to j} \sim q$ if and only if $p \sim q_{j \frown i}$.*

▶ **Lemma 10.** *Given two patterns $p, q \in \mathcal{P}(U)$. Let U' be a new label set disjoint from U and ρ a one-to-one mapping from U to U'. Let q' be the pattern resulting from q by relabeling u to $\rho(u)$ for each $u \in U$. Then it holds for each pattern $r \in \mathcal{P}(U)$ that $p \oplus q \sim r$ if and only if $p \sim q' \sqcup r'$ for $r' \in \mathcal{P}(U \cup U')$ the pattern resulting from r by adding $\rho(u)$ to each set containing u for each $u \in U$.*

4 Partial solutions as patterns

From now on, let $(G, T, \bar{b})$ be the given instance of the Steiner Tree problem, for some graph $G = (V, E)$, a set of terminals $T \subseteq V$, and $\bar{b} \in \mathbb{N}$. Let $n = |V|$ and $m = |E|$. Let μ be a k-expression of G for some value $k \in \mathbb{N}$. We fix $U = [k]$. Let $\mathcal{T}$ be the corresponding syntax tree, and r its root. We also fix a value $W \in \mathbb{N}$, and a weight function $\mathbf{w} : V \to [W]$. Both will be chosen later. Let $v_0 \in T$ be an arbitrary but fixed terminal vertex. For a node $x \in V(\mathcal{T})$, we define $T_x = T \cap V_x$.

▶ **Definition 11.** *Let $x \in V(\mathcal{T})$. We call a set $S \subseteq V_x$ where $T_x \subseteq S$ a* partial solution. *Each partial solution defines a pattern $p = p_x(S)$ over G_x as follows: Let $C_1, \ldots C_\ell$ be the connected components of $G_x[S]$. For $i \in [\ell]$, let $S_i = \mathrm{lab}_x(C_i)$ be the set of all labels appearing in C_i. If $v_0 \notin S$, we define $p = \{S_1, \ldots, S_\ell\} \cup \{\{0\}\}$. Otherwise, assume that $v_0 \in C_\ell$ (otherwise swap C_ℓ and the component containing v_0). We define $p = \{S_1, \ldots, S_{\ell-1}\} \cup \{S_\ell \cup \{0\}\}$. That means the zero-set is defined by the component containing v_0 if $v_0 \in S$, or as a singleton otherwise. We call $p_x(S)$ the pattern corresponding to S in G_x. We denote by $p(S)$ the pattern $p_r(S)$ corresponding to a partial solution S in the whole graph G.*

▶ **Lemma 12.** *Given a set $S \subseteq V$ such that $T \subseteq S$, it holds that S is a Steiner tree in G, if and only if $p(S)$ consists of one set only.*

We refer the reader to the full version for a formal proof. Now we define families of partial solutions that allow to build recursive formulas for a dynamic programming scheme over $\mathcal{T}$.

▶ **Definition 13.** *For $x \in V(\mathcal{T}), b \in [\bar{b}]_0, c \in [n \cdot W]_0$, we define the family $\mathcal{S}_x[b, c] \subseteq \mathcal{P}$ of patterns corresponding to partial solutions of cardinality b and weight c over G_x as*

$$\mathcal{S}_x[b, c] = \{p \in \mathcal{P} \colon \exists S \subseteq V_x, \text{ where } T_x \subseteq S \wedge p_x(S) = p \wedge |S| = b \wedge \mathbf{w}(S) = c\}.$$

From now on, we always assume that $x \in V(\mathcal{T}), b \in [\bar{b}]_0$, and $c \in [n \cdot W]_0$. We skip repeating this to avoid redundancy.

▶ **Lemma 14.** *The families $\mathcal{S}_x$ can be built recursively over the nodes of $\mathcal{T}$ in a bottom-up manner using the operations defined in Definition 4 and Definition 5.*

Proof. We sketch the recursive definitions of $\mathcal{S}_x$ for different types of nodes $x \in V(\mathcal{T})$, and refer to the full version for a complete proof of correctness.

For a create node $\mu_x = i(v)$ (a leaf of $\mathcal{T}$), let $c = \mathbf{w}(v)$. Let $p = [0, i]$ if $v \neq v_0$, and $p = [0i]$ otherwise. Then we set $\mathcal{S}_x[1, c] = \{p\}$. We set $\mathcal{S}_x[0, 0]$ to $\{[0]\}$, if $v \notin T$, and to $\emptyset$ otherwise. For all other values of b and c, we set $\mathcal{S}_x[b, c] = \emptyset$. We define $\mathcal{S}_x$ for the other cases as follows:

Join node	$\mu_x = \eta_{i,j}(\mu_{x'})$,	then $\mathcal{S}_x[b,c] = \square_{i,j}\big(\mathcal{S}_{x'}[b,c]\big)$,
Relabel node	$\mu_x = \rho_{i\to j}(\mu_{x'})$,	then $\mathcal{S}_x[b,c] = \big(\mathcal{S}_{x'}[b,c]\big)_{i\to j}$,
Union node	$\mu_x = \mu_{x_1} \uplus \mu_{x_2}$,	then $\mathcal{S}_x[b,c] = \bigcup_{\substack{b_1+b_2=b \\ c_1+c_2=c}} \big(\mathcal{S}_{x_1}[b_1,c_1] \oplus \mathcal{S}_{x_2}[b_2,c_2]\big)$.

◀

▶ **Lemma 15.** *The graph G admits a Steiner tree of size $\overline{b}$ and weight $\overline{c}$, if and only if there exists a pattern $p \in \mathcal{S}_r\left[\overline{b},\overline{c}\right]$ with $p \sim [0]$.*

This lemma follows from Lemma 12 (see the full version for a proof). So far, we can solve the Steiner Tree problem by defining a dynamic programming over $\mathcal{T}$ that computes all families $\mathcal{S}_x\left[\overline{b},\overline{c}\right]$ for all values of $\overline{b}$ and $\overline{c}$. However, since there exist approximately 2^{2^k} different patterns over k labels, we seek to reduce the family of all patterns into a family of representing patterns in a sense that allow us to apply the dynamic programming scheme over the resulting families.

5 Pattern representation

Now we define a special family of patterns (called *complete patterns*), and show that we can represent any family of patterns by a family of complete patterns only.

5.1 Representation and complete patterns

▶ **Definition 16.** *Given two families $\mathcal{S}, \mathcal{R} \subseteq \mathcal{P}$, we say that $\mathcal{R}$* represents *$\mathcal{S}$, if for each $q \in \mathcal{P}$ the following holds: there exists a pattern $p \in \mathcal{S}$ such that $p \sim q$ if and only if there exists a pattern $p' \in \mathcal{R}$ such that $p' \sim q$. Given a family $\mathcal{R} \subseteq \mathcal{P}$, and a pattern $p \in \mathcal{P}$, we say that $\mathcal{R}$ represents p if $\mathcal{R}$ represents $\{p\}$.*

Clearly, it holds that representation is an equivalence relation. In the full version, we define the notion of *representation-preserving* operations, and show that the operations defined in Definition 4 preserve representation. This allows to restrict a dynamic programming scheme to representing families, preserving the correctness of the transitions.

▶ **Definition 17.** *A pattern $p \in \mathcal{P}$ is* complete*, if $\text{label}(p) = \text{singleton}(p)$, i.e. a pattern p is complete, if each label of p appears as a singleton in p as well. We denote by $\mathcal{P}_C(U)$ the family of all complete patterns over U.*

▶ **Lemma 18.** *For all $p, q \in \mathcal{P}_C$ the following holds: $p \sim q$ implies that $\text{label}(p) = \text{label}(q)$.*

We refer to the full version for a proof.

▶ **Definition 19.** *Let $p \in \mathcal{P}$. For $i \in U$, we define the operations $\text{fix}(p,i)$ and $\text{forget}(p,i)$ as*

$$\text{fix}(p,i) = p \cup \big\{\{i\}\big\} \qquad \text{,and} \qquad \text{forget}(p,i) = \big\{S \setminus \{i\} \colon S \in p\big\},$$

if $i \in \text{label}(p) \setminus \text{singleton}(p)$. Otherwise, we define $\text{fix}(p,j) = \text{forget}(p,j) = p$. We say that $p' = \text{fix}(p,i)$ results from p by fixing *the label i, and $p'' = \text{forget}(p,i)$ results from p by* forgetting *the label i. Given a pattern $p \in \mathcal{P}$, we define $\text{inc}(p) = \text{label}(p) \setminus \text{singleton}(p)$.*

▶ **Definition 20.** *Given a pattern $p \in \mathcal{P}$. Let $\ell = |\,\text{inc}(p)|$, and $i_1, \dots i_\ell$ be the elements of $\text{inc}(p)$ in an increasing order. Let $R_0 = \{p\}$, and $R_j = \text{forget}(R_{j-1}, i_j) \cup \text{fix}(R_{j-1}, i_j)$ for all $j \in [\ell]$. We define the family $R_p = R_\ell$ and call it a* complete representation *of p.*

The following lemmas show that R_p represents p, which proves that any set of patterns admits a representation of complete patterns only. We refer to the full version for proofs.

▶ **Lemma 21.** *Let $p \in \mathcal{P}$, and $i \in U$. It holds that $\{\mathrm{fix}(p,i), \mathrm{forget}(p,i)\}$ represents p.*

▶ **Lemma 22.** *It holds for all $p \in \mathcal{P}$ that $R_p \subseteq \mathcal{P}_C$, $|R_p| \le 2^{|\operatorname{inc}(p)|}$, and R_p represents p.*

▶ **Corollary 23.** *Let $\mathcal{S} \subseteq \mathcal{P}$. Then the family $\mathcal{R} = \bigcup_{p \in \mathcal{S}} R_p \subseteq \mathcal{P}_C$ represents $\mathcal{S}$.*

▶ **Observation 24.** *It holds that the family of complete patterns is closed under both relabel operation $i \to j$ for all $i, j \in U$, and union operation $\oplus$. On the other hand, it holds for $i, j \in U, i \ne j$, $R \subseteq \mathcal{P}_C$, $R' = \square_{i,j} R$ and for each $p \in R'$ that $\operatorname{inc}(p) \in \{\emptyset, \{i,j\}\}$. If $\operatorname{inc}(p) = \{i,j\}$, then R_p contains at most four patterns, that result from p by fixing or forgetting either labels independently.*

5.2 Counting representations

▶ **Definition 25.** *Let $\mathrm{Op} : \mathcal{P}^k \to \mathcal{P}$ be a k-ary pattern operation. We define the operation $\overset{\triangle}{\mathrm{Op}} : \left(2^{\mathcal{P}}\right)^k \to 2^{\mathcal{P}}$ over subsets of $\mathcal{P}$ as $\overset{\triangle}{\mathrm{Op}}(S_1, \dots, S_k) = \underset{\substack{(p_1,\dots,p_k)\in \\ S_1\times\cdots\times S_k}}{\triangle} \{\mathrm{Op}(p_1, \dots, s_k)\}$, for all $S_1, \dots, S_k \subseteq \mathcal{P}$, and we call it the* exclusive *version of* Op. *We compare this operation to the notion $\mathrm{Op}(S_1, \dots, S_k)$ given by the union over all resulting patterns from the operation. Given two sets of patterns $S_1, S_2 \subseteq \mathcal{P}$, we define $S_1 \overset{\triangle}{\sqcup} S_2, S_1 \overset{\triangle}{\oplus} S_2, S_1{}_{\overset{\triangle}{i \to j}}$ and $\overset{\triangle}{\square}_{i,j} S_1$ as the exclusive version of join, union, relabel and $\square_{i,j}$ of S_1 (and S_2) respectively.*

The following lemma can be proven by a simple counting argument. We refer to the full version for a formal proof.

▶ **Lemma 26.** *Let $\mathrm{Op} : \mathcal{P}^k \to \mathcal{P}$ be a k-ary pattern operation, and $S_1, \dots S_k \subseteq \mathcal{P}$. It holds for $q \in \mathcal{P}$, that*

$$|\{p \in \overset{\triangle}{\mathrm{Op}}(S_1, \dots S_k) \colon p \sim q\}| \equiv_2 |\{(p_1, \dots p_k) \in S_1 \times \dots \times S_k \colon \ \mathrm{Op}(p_1, \dots p_k) \sim q\}|$$

As we have already mentioned, for $p \in \mathcal{P}_C$, and $p' = \square_{i,j} p$, it holds that either $R_{p'} = p'$, or $R_{p'}$ contains four patterns that result from p' by either forgetting or fixing both labels i and j independently. We can enumerate these patterns as follows:

▶ **Definition 27.** *We define a new operation called* actions $\mathrm{Ac} \colon \mathcal{P} \times [4] \to \mathcal{P}$ *as follows: Given $p \in \mathcal{P}$, if $p \in \mathcal{P}_C$, we define $\mathrm{Ac}(p, \ell) = p$, for all $\ell \in [4]$. For $\operatorname{inc}(p) = \{i\}$, we define*

- $\mathrm{Ac}(p, 1) = \mathrm{fix}(p, i)$,
- $\mathrm{Ac}(p, 2) = \mathrm{forget}(p, i)$.

We set $\mathrm{Ac}(p, 3)$ and $\mathrm{Ac}(p, 4)$ to undefined in this case. If $\operatorname{inc}(p) = \{i, j\}$ for $i < j$, we define

- $\mathrm{Ac}(p, 1) = \mathrm{fix}(\mathrm{fix}(p, i), j)$,
- $\mathrm{Ac}(p, 2) = \mathrm{forget}(\mathrm{fix}(p, i), j)$,
- $\mathrm{Ac}(p, 3) = \mathrm{fix}(\mathrm{forget}(p, i), j)$,
- $\mathrm{Ac}(p, 4) = \mathrm{forget}(\mathrm{forget}(p, i), j)$.

In all other cases we set $\mathrm{Ac}(p, \ell)$ to be undefined for all $\ell \in [4]$.

Using this enumeration, we can assign to each action at a create or a join node in the subtree of $\mathcal{T}_x$ a weight, which defines a different weight to each pattern in a complete pattern representation of the patterns in $\mathcal{S}_x\left[\overline{b}, \overline{c}\right]$.

▶ **Definition 28.** *For $x \in V(\mathcal{T})$ let $\pi : V(\mathcal{T}_x) \to [4]$ be some mapping, such that $\pi(x') \in [2]$ whenever x' is a create node, and $\pi(x') = 1$ whenever x' is a union node or a relabel node. We call π an* action sequence. *Let $D \in \mathbb{N}$ be some value that will be fixed later, and let $\mathbf{d} : V(\mathcal{T}) \times [4] \to [D]$ be a weight function. We define the weight of π as*

$$\mathbf{d}(\pi) = \sum_{\substack{x' \in V(\mathcal{T}_x),\\ x' \text{ is a join or create node}}} \mathbf{d}(x', \pi(x')).$$

Given a partial solution $S \subseteq V_x$ where $T_x \subseteq S$, the action sequence π defines a pattern p^π_S, that can be defined recursively over $\mathcal{T}_x$, where for a create node $i(v)$, let v be the vertex introduced at this node. If $v \notin S$, then we set $p^\pi_S = [0]$. Otherwise, we set $p^\pi_S = \mathrm{Ac}(p_x(\{v\}), \pi(x))$. For a relabel or union node, p^π_S results by applying the corresponding operation on the patterns resulting at the children of x in the natural way. For a join node $\mu_x = \eta_{i,j}(\mu_{x'})$, let $p' = \square_{i,j} p^{\pi'}_S$, where π' is the restriction of π to the nodes in $V(\mathcal{T}_{x'})$. We set $p^\pi_S = \mathrm{Ac}(p', \pi(x))$. We say that π is an action sequence generating the pattern p^π_S from S in G_x.

The proof of the following lemma follows by induction over $\mathcal{T}$ using transitivity of representation. We refer the reader to the full version for more details.

▶ **Lemma 29.** *Given a node $x \in V(\mathcal{T})$, and a partial solution $S \subseteq V_x$, where $T_x \subseteq S$. Let Π_x be the set of all action sequences at x. Let $P^S_x = \{p^\pi_S : \pi \in \Pi\}$. Then P^S_x represents $p_x(S)$.*

▶ **Definition 30.** *For $x \in V(\mathcal{T}), \overline{b} \in [n], \overline{c} \in [n \cdot W], \overline{d} \in [|V(\mathcal{T})| \cdot D]$, we define the families $\mathcal{D}_x\left[\overline{b}, \overline{c}, \overline{d}\right]$ as follows:*

- *Create node $i(v)$, let $c = \mathbf{w}(v)$. For $\ell \in [2]$, we set $\mathcal{D}_x[1, c, \mathbf{d}(x, \ell)] = \left\{\mathrm{Ac}\left(p_x(\{v\}), \ell\right)\right\}$, and set $\mathcal{D}_x[0, 0, \mathbf{d}(x, \ell)]$ to $\{[0]\}$ if $v \notin T$, or to $\emptyset$ otherwise. For all other values of $\overline{b}, \overline{c}$ and $\overline{d}$, we set $\mathcal{D}_x\left[\overline{b}, \overline{c}, \overline{d}\right] = \emptyset$.*
- *Relabel node $\mu_x = \rho_{i \to j}(\mu_{x'})$, then $\mathcal{D}_x\left[\overline{b}, \overline{c}, \overline{d}\right] = \left(\mathcal{D}_{x'}\left[\overline{b}, \overline{c}, \overline{d}\right]\right)_{\overset{\triangle}{i \to j}}$.*
- *Union node $\mu_x = \mu_{x_1} \uplus \mu_{x_2}$, then $\mathcal{D}_x\left[\overline{b}, \overline{c}, \overline{d}\right] = \triangle_{\substack{b_1+b_2=\overline{b}\\ c_1+c_2=\overline{c}\\ d_1+d_2=\overline{d}}} \left(\mathcal{D}_{x_1}[b_1, c_1, d_1] \overset{\triangle}{\oplus} \mathcal{D}_{x_2}[b_2, c_2, d_2]\right)$.*
- *Join node $\mu_x = \eta_{i,j}(\mu_{x'})$, we define the families $\mathcal{D}'_x\left[\overline{b}, \overline{c}, \overline{d}\right] = \overset{\triangle}{\square}_{i,j}\left(\mathcal{D}_{x'}\left[\overline{b}, \overline{c}, \overline{d}\right]\right)$. Now we set $\mathcal{D}_x\left[\overline{b}, \overline{c}, \overline{d}\right] = \underset{\ell \in [4]}{\triangle} \overset{\triangle}{\mathrm{Ac}}\left(\mathcal{D}'_x\left[\overline{b}, \overline{c}, \overline{d} - \mathbf{d}(x, \ell)\right], i\right)$.*

The following lemma shows that the families $\mathcal{D}_x$ correctly count action sequences. A formal proof of the lemma and the corollary following it can be found in the full version.

▶ **Lemma 31.** *For $x \in V(\mathcal{T})$, and all values $\overline{b}, \overline{c}$ and $\overline{d}$, it holds for $p \in \mathcal{P}_C$ that $p \in \mathcal{D}_x\left[\overline{b}, \overline{c}, \overline{d}\right]$ if and only if there exist odd many pairs (S, π) where $S \subseteq V_x$ with $T_x \subseteq S, |S| = \overline{b}, \mathbf{w}(S) = \overline{c}$ and π is an action sequence of weight $\overline{d}$ generating p from S in G_x.*

▶ **Corollary 32.** *If G does not admit a Steiner tree of size $\overline{b}$, then it holds for all values $\overline{c} \in [W \cdot n]$, $\overline{d} \in [D \cdot |V(\mathcal{T})|]$ that $[0] \notin \mathcal{D}_r\left[\overline{b}, \overline{c}, \overline{d}\right]$.*

5.3 Isolation Lemma

Now we fix $W = (2 + \sqrt{2})|V|$, and $D = 4(2 + \sqrt{2})|V(\mathcal{T})|$. We choose $\mathbf{w} \in [W]^V$ and $\mathbf{d} \in [D]^{V(\mathcal{T}) \times [4]}$ uniformly and independently at random. We use the isolation lemma to show that, if a Steiner tree of size $\overline{b}$ exists, then there exist two positive integers $\overline{c}$ and $\overline{d}$, such that, with high probability, there exists a unique solution S of size $\overline{b}$ and weight $\overline{c}$, and a unique action sequence of weight $\overline{d}$ that generates the pattern $[0]$ from S.

▶ **Definition 33.** *A function $\omega : U \to \mathbb{Z}$ isolates a set family $\mathcal{F} \subseteq 2^U$ if there exists a unique $S' \in \mathcal{F}$ with $\omega(S') = \min_{S \in \mathcal{F}} \omega(S)$.*

▶ **Lemma 34** ([38, 17])**.** *Let $\mathcal{F} \subseteq 2^U$ be a set family over a universe U with $|\mathcal{F}| > 0$, and let $N > |U|$ be an integer. For each $u \in U$, choose a weight $\omega(u) \in \{1, 2, \dots N\}$ uniformly and independently at random. Then it holds that $P[\omega$ isolates $\mathcal{F}] \geq 1 - |U|/N$.*

We use the following lemmas to show that, if a solution exists, then, with high probability, there exists a unique minimum weight representation of a minimum weight solution. We refer the reader to the full version for a formal proof.

▶ **Lemma 35.** *If a Steiner tree of size $\overline{b}$ exists, let $\overline{c}$ be the smallest weight of such a tree. Then there exists a unique Steiner tree of size $\overline{b}$ and weight $\overline{c}$ with probability at least $1 - \frac{1}{2+\sqrt{2}}$.*

▶ **Lemma 36.** *Let S be a Steiner tree in G, and π be a minimum weight action sequence generating $[0]$ from S in G. Then π is unique with probability at least $1 - \frac{1}{2+\sqrt{2}}$.*

▶ **Lemma 37.** *If G admits a Steiner tree of size $\overline{b}$, then with probability at least $1/2$ it holds that $[0] \in \mathcal{D}_r\left[\overline{b}, \overline{c}, \overline{d}\right]$ for some values of $\overline{c}$ and $\overline{d}$.*

6 Parity-representation

Now we define a new kind of representation, called the parity-representation. This allows to count complete patterns (modulo 2) that are consistent with a specific pattern.

▶ **Definition 38.** *We define the family of CS-patterns $\mathcal{P}_{CS}(U) \subseteq \mathcal{P}_C(U)$ as the family of complete patterns containing only a zero-set and singletons, i.e.*

$$\mathcal{P}_{CS}(U) = \Big\{ \big\{X \cup \{0\}\big\} \cup \big\{\{u\} \colon u \in Y\big\} \colon X \subseteq Y \subseteq U \Big\}.$$

▶ **Observation 39.** *For each ground set U it holds that $|\mathcal{P}_{CS}(U)| = 3^{|U|}$.*

▶ **Observation 40.** *For $p, q \in \mathcal{P}_{CS}$ and all $i, j \in U$, it holds that $p_{i \to j}, p \oplus q \in \mathcal{P}_{CS}$.*

▶ **Definition 41.** *Given two sets of patterns $D, C \subseteq \mathcal{P}$, we say that C* parity-represents *D, if and only if for each $q \in \mathcal{P}_C$ it holds*

$$|\{p \in C \colon p \sim q\}| \equiv_2 |\{p \in D \colon p \sim q\}|.$$

For a set of patterns $C \subseteq \mathcal{P}$ and a single pattern $p \in \mathcal{P}$, we say that C parity-represents p if it holds that C parity-represents $\{p\}$.

▶ **Observation 42.** *Parity-representation is an equivalence relation.*

Now we show how to build a concrete parity-representation of an arbitrary complete pattern using CS-patterns only. We refer the reader to the full version for formal proofs.

▶ **Lemma 43.** *Given a pattern $p \in \mathcal{P}_C \setminus \mathcal{P}_{CS}$. Let $S \in p \setminus \{Z_p\}$ be an inclusion-wise minimal set, such that $|S| > 1$. Let $A = A_p^S$ be the family of patterns defined from p as follows $A_p^S = \big\{(p \setminus \{Z_p, S\}) \cup (Z_p \cup S') \colon S' \subset S\big\}$. Then A parity-represents of p.*

▶ **Lemma 44.** *Given a complete pattern $p \in \mathcal{P}_C$. Let $S_1, \dots S_r$ be the sets in p such that $S_i \neq Z_p$ and $|S_i| \geq 2$ for each $i \in [r]$. Let $A_0 = \{p\}$. For $i \in [r]$, we define $A_i = \triangle_{q \in A_{i-1}} A_q^{S_i}$. Let $A_p = A_r$. Then it holds that $A_p \subseteq \mathcal{P}_{CS}$, and that it parity-represents p.*

▶ **Corollary 45.** *For each family of complete patterns $D \subseteq \mathcal{P}_C$, there exists a family of CS-patterns $C \subseteq \mathcal{P}_{CS}$ that parity-represents D.*

Similar to pattern representation, in the full version of this paper we define the notion of *parity-representation-preserving* operations. We show that all defined operations (including actions and symmetric difference) preserve parity-representation. This allows to restrict a dynamic programming algorithm to a parity-representation of the underlying families.

Now we define families of partial solutions over $\mathcal{P}_{CS}$ that parity-represent the families $\mathcal{D}_x\left[\overline{b}, \overline{c}, \overline{d}\right]$. In the next section, we show that we can efficiently compute these families, and hence, solve the decision version of the Steiner Tree problem with high probability.

▶ **Definition 46.** *For $p \in \mathcal{P}_C$ let A_p be the set defined in Lemma 44. We define the operator $\mathrm{Ac}^* : \mathcal{P} \times 4 \to 2^{\mathcal{P}_{CS}}$ as $\mathrm{Ac}^*(p, \ell) = A_{\mathrm{Ac}(p,\ell)}$ if $\mathrm{Ac}(p, \ell) \in \mathcal{P}_C$ or as undefined otherwise. We denote the* exclusive *version of Ac^* by $\overset{\triangle}{\mathrm{Ac}}{}^*$.*

▶ **Definition 47.** *For $x \in V(\mathcal{T}), \overline{b} \in [n], \overline{c} \in [n \cdot W], \overline{d} \in [|V(\mathcal{T})| \cdot D]$, we define the families $\mathcal{C}_x\left[\overline{b}, \overline{c}, \overline{d}\right]$ as follows: For a create, relabel or union nodes x, we define the families $\mathcal{C}_x$ in an analogous way to $\mathcal{D}_x$, where we replace each $\mathcal{C}$ with $\mathcal{D}$ in the definition. For a join node $\mu_x = \eta_{i,j}(\mu_{x'})$, we define the families*

$$\mathcal{C}'_x\left[\overline{b}, \overline{c}, \overline{d}\right] = \overset{\triangle}{\square}_{i,j}\left(\mathcal{C}_{x'}\left[\overline{b}, \overline{c}, \overline{d}\right]\right) \quad \text{,and set} \quad \mathcal{C}_x\left[\overline{b}, \overline{c}, \overline{d}\right] = \bigtriangleup_{\ell \in [4]} \overset{\triangle}{\mathrm{Ac}}{}^*\left(\mathcal{C}'_x\left[\overline{b}, \overline{c}, \overline{d} - \mathbf{d}(x, \ell)\right], \ell\right).$$

In the full version, we show that $\mathcal{C}_x\left[\overline{b}, \overline{c}, \overline{d}\right]$ parity-represents $\mathcal{D}_x\left[\overline{b}, \overline{c}, \overline{d}\right]$ for all $x \in V(\mathcal{T})$ and all values $\overline{b}, \overline{c}, \overline{d}$. The following corollary follows

▶ **Corollary 48.** *It holds for all values $\overline{b}, \overline{c}, \overline{d}$ that $[0] \in \mathcal{C}_r\left[\overline{b}, \overline{c}, \overline{d}\right] \iff [0] \in \mathcal{D}_r\left[\overline{b}, \overline{c}, \overline{d}\right]$.*

7 Algorithm

In this section, we define the final dynamic programming table that computes $\mathcal{C}_x\left[\overline{b}, \overline{c}, \overline{d}\right]$ for $x \in V(\mathcal{T})$ and all values $\overline{b}, \overline{c}$ and $\overline{d}$ in time $O^*(3^k)$.

▶ **Definition 49.** *For all $x \in V(\mathcal{T})$ and all values $\overline{b} \in [k]_0, \overline{c} \in [|V| \cdot W]_0$ and $\overline{d} \in [|V(\mathcal{T})| \cdot D]_0$, we define the vectors $T = T\left[x, \overline{b}, \overline{c}, \overline{d}\right] \in \{0,1\}^{\mathcal{P}_{CS}}$ as follows:*

- *Create node $i(v)$. Let $p = [0, i]$ if $v \neq v_0$, and $p = [0i]$ otherwise. First we initialize $T[x, \overline{b}, \overline{c}, \overline{d}] = \overline{0}$ for all values $\overline{b}, \overline{c}, \overline{d}$. Now we set $T[x, 0, 0, 0][[0]] = [v \notin T]$, and for $\ell \in [2]$ we set $T[x, 1, \mathbf{w}(v), \mathbf{d}(x, \ell)][\mathrm{Ac}(p, \ell)] = 1$.*
- *Relabel node $\mu_x = \rho_{i \to j}(\mu_{x'})$, we define $T\left[x, \overline{b}, \overline{c}, \overline{d}\right][p] = \sum_{\substack{q \in \mathcal{P}_{CS} \\ q_{i \to j} = p}} T\left[x', \overline{b}, \overline{c}, \overline{d}\right][q]$.*
- *Join node $\mu_x = \eta_{i,j}(\mu_{x'})$. For all values $\overline{b}, \overline{c}, \overline{d}$, we define $T'\left[x, \overline{b}, \overline{c}, \overline{d}\right] \in \{0,1\}^{\mathcal{P}}$ as*

$$T'\left[x, \overline{b}, \overline{c}, \overline{d}\right][p] = \sum_{\substack{q \in \mathcal{P}_{CS}, \\ \square_{ij} q = p}} T\left[x', \overline{b}, \overline{c}, \overline{d}\right][q],$$

for all patterns $p \in \mathcal{P}$. Now for all $p \in \mathcal{P}_{CS}$ we set

$$T\left[x, \overline{b}, \overline{c}, \overline{d}\right][p] = \sum_{\ell \in [4]} \sum_{\substack{q \in \mathcal{P}, \\ p \in \mathrm{Ac}^*(q, \ell)}} T'\left[x, \overline{b}, \overline{c}, \overline{d} - \mathbf{d}(x, \ell)\right][q].$$

- *Union node $\mu_x = \mu_{x_1} \uplus \mu_{x_2}$. For all $p \in \mathcal{P}_{CS}$, we define*

$$T\big[x, \overline{b}, \overline{c}, \overline{d}\big][p] = \sum_{\substack{b_1+b_2=\overline{b}\\ c_1+c_2=\overline{c}\\ d_1+d_2=\overline{d}}} \sum_{\substack{p_1,p_2\in\mathcal{P}_{CS}\\ p_1\oplus p_2=p}} T[x_1, b_1, c_1, d_1][p_1] \cdot T[x_2, b_2, c_2, d_2][p_2].$$

The following lemmas show the correctness of the algorithm (proof in the full version).

▶ **Lemma 50.** *It holds that $T\big[x, \overline{b}, \overline{c}, \overline{d}\big][p] = \big[p \in \mathcal{C}_x\big[\overline{b}, \overline{c}, \overline{d}\big]\big]$ for all $x, \overline{b}, \overline{c}, \overline{d}$, and $p \in \mathcal{P}_{CS}$.*

▶ **Corollary 51.** *It holds for all $\overline{b}, \overline{c}, \overline{d}$ that $T\left[r, \overline{b}, \overline{c}, \overline{d}\right][[0]] \equiv_2 1 \iff [0] \in \mathcal{D}_r\big[\overline{b}, \overline{c}, \overline{d}\big]$.*

Although computing the tables $T\big[x, \overline{b}, \overline{c}, \overline{d}\big]$ for a union node x in the naive way yields a running time polynomial in $|\mathcal{P}_{CS}|^2$ which exceeds the bound we seek by far, we make use of a result by Hegerfeld and Kratsch [27] to compute convolutions over lattices more efficiently.

▶ **Definition 52.** *Given a lattice $(\mathcal{L}, \preceq)$, and two tables $A, B : \mathcal{L} \to \mathbb{F}$ for some field $\mathbb{F}$, we define the* join-product *(or $\vee$-product) $A \otimes_{\mathcal{L}} B$ as follows: for each $x \in \mathcal{L}$ we define*

$$A \otimes_{\mathcal{L}} B(x) = \sum_{\substack{y,z\in\mathcal{L}\\ y\vee z=x}} A(y) \cdot B(z).$$

To avoid unnecessary notation, we state a modified, more restrictive version of the mentioned result by Hegerfeld and Kratsch [27, Corollary A.10], and refer the reader to the full version for an exact statement and background definitions.

▶ **Corollary 53.** *Let $(\mathcal{L}, \preceq)$ be a finite lattice such that the join operation over $\mathcal{L}$ can be computed in polynomial time. Let k be a natural number. Given two tables $A, B\colon \mathcal{L}^k \to \mathbb{Z}_2$, the $\vee$-product $A \otimes_{\mathcal{L}^k} B$ can be computed in time $\mathcal{O}^*(|\mathcal{L}|^k)$.*

▶ **Definition 54.** *We define a new ordering $\preceq_{CS}$ over $\mathcal{P}_{CS}$ where $p_1 \preceq_{CS} p_2$, if for each $i \in U$ it holds that $i \in Z_{p_1}$ if $i \in Z_{p_2}$ and $i \in \text{label}(p_1)$ if $i \in \text{label}(p_2)$.*

▶ **Observation 55.** *The join operation over the lattice $(\mathcal{P}_{CS}, \preceq_{CS})$ is given by $p \vee q = r$, where $Z_r = Z_p \cup Z_q$ and $\text{label}(r) = \text{label}(p) \cup \text{label}(q)$. This corresponds exactly to the union operation over CS-patterns $r = p \oplus q$. Hence, it holds for a union node that*

$$T\big[x, \overline{b}, \overline{c}, \overline{d}\big] = \sum_{\substack{b_1+b_2=\overline{b}\\ c_1+c_2=\overline{c}\\ d_1+d_2=\overline{d}}} T[x', b_1, c_1, d_1] \otimes_{\mathcal{P}_{CS}} T[x'', b_2, c_2, d_2].$$

In the full version, we show that the lattice $(\mathcal{P}_{CS}(U), \preceq_{CS})$ is isomorphic to the lattice $\mathcal{L}^k$, where $\mathcal{L} = (\{1, 2, 3\}, \leq)$ is defined in the natural way. Using Corollary 53, it follows that $T\big[x, \overline{b}, \overline{c}, \overline{d}\big]$ for a union node x can be computed in time $O^*(3^k)$. The following lemma states that this is indeed the case for all nodes x of $\mathcal{T}$. We refer the reader to the full version for formal proofs.

▶ **Lemma 56.** *The families $T\big[x, \overline{b}, \overline{c}, \overline{d}\big]$ for all $x \in V(\mathcal{T})$ and all values $\overline{b}, \overline{c}, \overline{d}$ can be computed in time $\mathcal{O}^*(3^k)$.*

Now we are ready to prove the main theorem of this work (Theorem 1).

Proof (Theorem 1). The algorithm first computes all families $T\left[x, b, \overline{c}, \overline{d}\right]$ for all $x \in V(\mathcal{T})$ and all values of $b, \overline{c}, \overline{d}$. The algorithm states that there exists a Steiner tree of size $\overline{b}$, if and only if there exist two values $\overline{c}, \overline{d}$ with $T\left[r, \overline{b}, \overline{c}, \overline{d}\right][[0]] = 1$.

By Lemma 56, we can compute all families $T\left[x, b, \overline{c}, \overline{d}\right]$ in time $\mathcal{O}^*(3^k)$. By Lemma 50, it holds for all values $\overline{b}, \overline{c}, \overline{d}$ that

$$T\left[r, \overline{b}, \overline{c}, \overline{d}\right][[0]] \equiv_2 1 \iff [0] \in \mathcal{D}_r\left[\overline{b}, \overline{c}, \overline{d}\right].$$

By Corollary 32 it holds that if no Steiner tree of size $\overline{b}$ exists, then the algorithm will always return false. On the other hand if a Steiner tree of size $\overline{b}$ exists, by choosing $W = (2+\sqrt{2})|V|$, and $D = 4(2+\sqrt{2})|V(\mathcal{T})|$, and by choosing both function $\mathbf{w} \in [W]^V$, and $\mathbf{d} \in [D]^{V(\mathcal{T})\times[4]}$ uniformly and independently at random, it holds by Lemma 37 that the algorithm will correctly decide the existence of such a tree with probability at least one half. ◀

8 Conclusion

We have presented a $3^k \cdot n^{\mathcal{O}(1)}$ time one-sided Monte-Carlo algorithm for Steiner Tree[cw]. Under the Strong Exponential-Time Hypothesis, lower bounds for Steiner Tree relative to less expressive parameters [15, 4] rule out time $(3-\varepsilon)^k \cdot n^{\mathcal{O}(1)}$, and show that this parameter dependence is optimal for all parameters between cutwidth and clique-width.

Rather than using cut-and-count, which is the go-to technique for connectivity problems, our algorithm works with connectivity patterns. Two technical contributions, make this work: The family of complete connectivity patterns has strong existential representation properties, so that each connectivity pattern has an explicit existential replacement by complete patterns. This avoids spending more time for getting the representative partial solutions (such as for Gaussian elimination in the rank-based approach [3]). The second technical contribution, the technique of *isolating a representative*, allows to combine this with working via a 3^k-sized family of basis patterns, that represents complete patterns modulo two. We expect that both techniques will be of use for settling the complexity of other (connectivity) problems.

Very recently, the present authors announced a tight bound of $12^k \cdot n^{\mathcal{O}(1)}$ (modulo SETH) for Connected Odd Cycle Transversal[cw] [6].

References

1 Benjamin Bergougnoux and Mamadou Moustapha Kanté. Fast exact algorithms for some connectivity problems parameterized by clique-width. *Theor. Comput. Sci.*, 782:30–53, 2019. doi:10.1016/j.tcs.2019.02.030.

2 Benjamin Bergougnoux and Mamadou Moustapha Kanté. More applications of the d-neighbor equivalence: Acyclicity and connectivity constraints. *SIAM J. Discret. Math.*, 35(3):1881–1926, 2021. doi:10.1137/20M1350571.

3 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

4 Narek Bojikian, Vera Chekan, Falko Hegerfeld, and Stefan Kratsch. Tight bounds for connectivity problems parameterized by cutwidth. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 14:1–14:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.14.

5 Narek Bojikian and Stefan Kratsch. A tight monte-carlo algorithm for steiner tree parameterized by clique-width. *CoRR*, abs/2307.14264, 2023. doi:10.48550/arXiv.2307.14264.

6 Narek Bojikian and Stefan Kratsch. Tight algorithm for connected odd cycle transversal parameterized by clique-width, 2024. arXiv:2402.08046.

7 Glencora Borradaile and Hung Le. Optimal dynamic program for r-domination problems over tree decompositions. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.8.

8 Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce A. Reed, and Udi Rotics. Polynomial-time recognition of clique-width ≤ 3 graphs. *Discret. Appl. Math.*, 160(6):834–865, 2012. doi:10.1016/j.dam.2011.03.020.

9 Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci.*, 46(2):218–270, 1993. doi:10.1016/0022-0000(93)90004-G.

10 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

11 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discret. Appl. Math.*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

12 Radu Curticapean, Nathan Lindzey, and Jesper Nederlof. A tight lower bound for counting hamiltonian cycles via matrix rank. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1080–1099. SIAM, 2018. doi:10.1137/1.9781611975031.70.

13 Radu Curticapean and Dániel Marx. Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1650–1669. SIAM, 2016. doi:10.1137/1.9781611974331.ch113.

14 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.

15 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.23.

16 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *CoRR*, abs/1103.0534, 2011. arXiv:1103.0534.

17 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

18 Baris Can Esmer, Jacob Focke, Dániel Marx, and Pawel Rzazewski. List homomorphisms by deleting edges and vertices: tight complexity bounds for bounded-treewidth graphs. *CoRR*, abs/2210.10677, 2022. doi:10.48550/arXiv.2210.10677.

19 Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve np-hard graph problems on clique-width bounded graphs in polynomial time. In Andreas Brandstädt and Van Bang Le, editors, *Graph-Theoretic Concepts in Computer Science, 27th International Workshop, WG 2001, Boltenhagen, Germany, June 14-16, 2001, Proceedings*, volume 2204 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2001. doi:10.1007/3-540-45477-2_12.

20 Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is np-complete. *SIAM J. Discret. Math.*, 23(2):909–939, 2009. doi:10.1137/070687256.

21 Jacob Focke, Dániel Marx, Fionn Mc Inerney, Daniel Neuen, Govind S. Sankar, Philipp Schepper, and Philip Wellnitz. Tight complexity bounds for counting generalized dominating sets in bounded-treewidth graphs. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3664–3683. SIAM, 2023. doi:10.1137/1.9781611977554.ch140.

22 Jacob Focke, Dániel Marx, and Pawel Rzazewski. Counting list homomorphisms from graphs of bounded treewidth: tight complexity bounds. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 431–458. SIAM, 2022. doi:10.1137/1.9781611977073.22.

23 Robert Ganian, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. The fine-grained complexity of graph homomorphism parameterized by clique-width. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 66:1–66:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.66.

24 Carla Groenland, Isja Mannens, Jesper Nederlof, and Krisztina Szilágyi. Tight bounds for counting colorings and connected edge sets parameterized by cutwidth. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 36:1–36:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.36.

25 Frank Gurski and Egon Wanke. Vertex disjoint paths on clique-width bounded graphs. *Theor. Comput. Sci.*, 359(1-3):188–199, 2006. doi:10.1016/j.tcs.2006.02.026.

26 Tesshu Hanaka, Yasuaki Kobayashi, and Taiga Sone. A (probably) optimal algorithm for bisection on bounded-treewidth graphs. *Theor. Comput. Sci.*, 873:38–46, 2021. doi:10.1016/j.tcs.2021.04.023.

27 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by clique-width. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 59:1–59:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.59.

28 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by modular-treewidth. In Daniël Paulusma and Bernard Ries, editors, *Graph-Theoretic Concepts in Computer Science - 49th International Workshop, WG 2023, Fribourg, Switzerland, June 28-30, 2023, Revised Selected Papers*, volume 14093 of *Lecture Notes in Computer Science*, pages 388–402. Springer, 2023. doi:10.1007/978-3-031-43380-1_28.

29 Yoichi Iwata and Yuichi Yoshida. On the equivalence among problems of bounded width. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 754–765. Springer, 2015. doi:10.1007/978-3-662-48350-3_63.

30 Bart M. P. Jansen and Jesper Nederlof. Computing the chromatic number using graph decompositions via matrix rank. *Theor. Comput. Sci.*, 795:520–539, 2019. doi:10.1016/j.tcs.2019.08.006.

31 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discret. Appl. Math.*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.

32 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structurally parameterized d-scattered set. *Discret. Appl. Math.*, 308:168–186, 2022. doi:10.1016/j.dam.2020.03.052.

33 Michael Lampis. Model checking lower bounds for simple graphs. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 673–683. Springer, 2013. doi:10.1007/978-3-642-39206-1_57.

34 Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.

35 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 777–789. SIAM, 2011. doi:10.1137/1.9781611973082.61.

36 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

37 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Degrees and gaps: Tight complexity results of general factor problems parameterized by treewidth and cutwidth. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 95:1–95:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.95.

38 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Comb.*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

39 Jesper Nederlof. Algorithms for np-hard problems via rank-related parameters of matrices. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms - Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 145–164. Springer, 2020. doi:10.1007/978-3-030-42071-0_11.

40 Karolina Okrasa, Marta Piecyk, and Pawel Rzazewski. Full complexity classification of the list homomorphism problem for bounded-treewidth graphs. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 74:1–74:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.74.

41 Karolina Okrasa and Pawel Rzazewski. Fine-grained complexity of the graph homomorphism problem for bounded-treewidth graphs. *SIAM J. Comput.*, 50(2):487–508, 2021. doi:10.1137/20M1320146.

42 Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):10:1–10:20, 2008. doi:10.1145/1435375.1435385.

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

44 Marta Piecyk and Pawel Rzazewski. Fine-grained complexity of the list homomorphism problem: Feedback vertex set and cutwidth. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 56:1–56:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.56.

45 Bas A. M. van Geffen, Bart M. P. Jansen, Arnoud A. W. M. de Kroon, and Rolf Morel. Lower bounds for dynamic programming on planar graphs of bounded cutwidth. *J. Graph Algorithms Appl.*, 24(3):461–482, 2020. doi:10.7155/jgaa.00542.

46 Egon Wanke. k-nlc graphs and polynomial algorithms. *Discret. Appl. Math.*, 54(2-3):251–266, 1994. doi:10.1016/0166-218X(94)90026-4.