Improved Steiner Tree Algorithms for Bounded Treewidth

Markus Chimani[1,⋆], Petra Mutzel[2], and Bernd Zey[2]

[1] Institute of Computer Science, Friedrich-Schiller-University of Jena
markus.chimani@uni-jena.de
[2] Department of Computer Science, TU Dortmund
{petra.mutzel,bernd.zey}@tu-dortmund.de

Abstract. We propose a new algorithm that solves the Steiner tree problem on graphs with vertex set V to optimality in $\mathcal{O}(B_{tw+2}^2 \cdot tw \cdot |V|)$ time, where tw is the graph's treewidth and the *Bell number* B_k is the number of partitions of a k-element set. This is a linear time algorithm for graphs with fixed treewidth and a polynomial algorithm for $tw = \mathcal{O}(\log |V| / \log \log |V|)$.

While being faster than the previously known algorithms, our thereby used coloring scheme can be extended to give new, improved algorithms for the prize-collecting Steiner tree as well as the k-cardinality tree problems.

1 Introduction

In this paper we consider the well-known *Steiner tree problem* (STP), as well as the related problems *prize-collecting Steiner tree* (PCST) and *k-cardinality tree* (KCT), all defined on graphs. Our central results are new exact algorithms to solve these problems in the case of graphs with bounded treewidth: the *treewidth* *tw* of a graph (see below for a concise definition) can be seen as a measure of how similar the given graph is to a tree.

Let $G = (V, E)$ be a given edge-weighted graph and $T \subseteq V$ a set of *terminals*. The Steiner tree problem is to find a minimum-weight tree $\mathcal{S}$ in G which contains all terminals T and possibly also some non-terminal (*Steiner*) vertices of $V \setminus T$. Note that while often the edge weights are considered to be only positive, we do not require any such restriction. The corresponding decision problem is strongly $\mathcal{NP}$-complete, even when restricted to edge weights 1 and 2 [23], or when G is planar [18]. The traditional algorithm by Dreyfus and Wagner [17] solves the STP exactly in $\mathcal{O}(3^t \cdot |V|)$ time—recently improved to $\mathcal{O}(2^t \cdot |V|)$ [8]—where $t := |T|$ is the number of terminals.

Regarding G's treewidth *tw*, the oldest but yet strongest result is due to Korach and Solel [20]; yet this technical report has never been officially published and has been cited only rarely, e.g., in [7, 19, 22]. Their algorithm achieves a runtime of $\mathcal{O}(tw^{4tw} \cdot |V|)$ but the paper's description is very sketchy and leaves

⋆ Funded via a juniorprofessorship by the Carl-Zeiss-Foundation.

many details unclear; it does not contain a formal proof of either the running time nor of its correctness. More recent publications, in particular those dealing with PTASes (see next paragraph) where the STP on bounded-treewidth-graphs arises as a subproblem, instead propose their own, yet weaker, results.

For the unweighted STP, i.e., the objective is to minimize the number of edges of $\mathcal{S}$, a very recent and surprising result by Cygan et al. [15] gives a Monte Carlo algorithm for the decision problem with a one-sided error—false negatives occur with probability of at most 1/2—requiring only $\mathcal{O}(3^{tw}|V|^{\mathcal{O}(1)})$ time. While the result is of course directly applicable to integer weighted STP where the maximum edge weight is bounded by a constant, we cannot see how to generalize the algorithm to arbitrary edge weights, and its derandomization is considered an open problem.

Recently, the STP and related problems for graphs with bounded treewidth achieved more attention due to their applicability to approximate network problems in planar graphs: In multiple papers [2,3,4,12,13,14], PTASes (polynomial time approximation schemes) are proposed which transform the given planar graph into a graph with bounded treewidth (via edge removals), solve the problem optimally (or within $1+\varepsilon$) on this modified graph, and then use this solution to construct a $(1+\varepsilon)$ solution to the original graph. Hence, the development of faster algorithms for the problem on bounded treewidth directly leads to faster PTASes for the corresponding problem on planar graphs.

For the STP, the approximation scheme of [13] uses an algorithm for solving the problem on graphs with bounded carving-width (a relative of treewidth) as a black box. Chekuri et al. [14] (later merged into [2]) give an algorithm for the prize-collecting Steiner tree problem (cf. Section 3) with running time $\mathcal{O}(B_k^3 \cdot s_k \cdot |V|)$, where $k := tw+1$, B_k is the number of partitions of a set with k elements (k-th *Bell number*), and s_k is the number of subgraphs of a k-vertex graph. Since $s_k = \mathcal{O}(2^{(k^2)})$, this leads to a running time of $\mathcal{O}(2^{(tw^2)} \cdot B_{tw+1}^3 \cdot |V|)$ for a graph with treewidth tw. This algorithm then allows PTASes for PCST and prize-collecting Steiner forest problems. Independently, Bateni et al. [4] (also later merged into [2]) describe PTASes for prize-collecting network design problems on planar graphs by using a similar approach. They investigate the PCST (the solution is a tree), prize-collecting TSP (the solution is a cycle), and the prize-collecting Stroll (the solution is a path). To this end they describe a $(1+\varepsilon)$-approximation for the PCST problem (that can be adapted to solve the other two considered problems as well) with a running time of order $\mathcal{O}(tw^{tw} \cdot 2^{tw} \cdot |V|)$.

Furthermore, Polzin and Daneshmand [22] introduced an algorithm with running time $\mathcal{O}(2^{3b} \cdot |V|)$ where b (the size of a "border" obtained throughout the algorithm) is a parameter similar to pathwidth. Yet note that even for simple trees—with natural treewidth 1—the pathwidth is unbounded.

Note that all these exact algorithms (not the approximations) fall into the category of FPT (fixed parameter tractable) algorithms w.r.t. the considered parameters (e.g., treewidth). An introduction to this research field can be found in [16,21].

Our Contribution. Herein, we propose a new algorithm to solve the Steiner tree problem exactly in $\mathcal{O}(B_{tw+2}^2 \cdot tw \cdot |V|)$ time. The k-th *Bell number* B_k thereby is the number of partitions of a set with k elements, and can be recursively defined as $B_0 = 1$, $B_{k+1} = \sum_{i=0}^{k} \binom{k}{i} B_i$. We can bound $B_k < (0.792k/\ln(k+1))^k$ [5] and in particular $B_k < k! < k^k$ for $k \geq 3$. Our algorithm is hence linear for graphs with fixed treewidth and requires $\mathcal{O}(|V|^3 \log |V| / \log\log |V|)$ time for $tw \in \mathcal{O}(\log |V| / \log\log |V|)$. The algorithm guarantees a running time that is smaller than the currently best proposed running times, including the works of [20]. This paper therefore also closes the unclear situation regarding the latter. We will discuss our algorithm in Section 2.

To achieve this result, we use the well-known dynamic programming paradigm over the decomposition tree (see next section), coupled with a special numbering and coloring scheme. Furthermore, our new coloring scheme shows to be versatile enough to also allow new, faster algorithms to solve the prize-collecting Steiner tree problem in the same time complexity, as well as the k-cardinality tree problem in $\mathcal{O}(B_{tw+2}^2 \cdot (tw + k^2) \cdot |V|)$ time. We discuss these extensions in Sections 3 and 4, respectively.

Preliminaries: Tree Decompositions. The concept of treewidth was introduced by Robertson and Seymour [25] by the term tree decomposition. See [9,11] for an in-depth introduction to this topic:

Let $G = (V, E)$ be the given graph. Its tree decomposition $(\mathcal{T}, \mathcal{X})$ is a pair of a tree $\mathcal{T} = (I, F)$ and a collection $\mathcal{X} = \{X_i\}_{i \in I}$ of vertex subsets (called *bags*) with the following properties:

td/1: Every vertex $v \in V$ is contained in at least one bag X_i, $i \in I$. For every edge $(u, v) \in E$ there is at least one bag X_i, $i \in I$, containing both vertices u, v.

td/2: For every vertex $v \in V$, the nodes i with $v \in X_i$ form a subtree of $\mathcal{T}$.

To avoid confusion, we speak of *vertices* V in the graph G, and of *nodes* I in the tree $\mathcal{T}$. The width of a tree decomposition $(\mathcal{T}, \mathcal{X})$ is the size of the largest bag in $\mathcal{X}$ minus 1. The *treewidth* of a graph is the smallest width over all possible tree decompositions. Hence, the treewidth measures how similar the decomposed graph is to a tree: trees have treewidth 1, (generalized) series-parallel graphs have treewidth 2, etc. On the other side of the spectrum, complete graphs have treewidth $|V| - 1$, by putting all vertices in one bag. Determining whether a graph has treewidth k, for a given integer k, is $\mathcal{NP}$-complete [1] but polynomial (i.e., in FPT) for any constant k [10].

Most importantly, we note that the size of $(\mathcal{T}, \mathcal{X})$ is only linear, even when considering *nice* tree decompositions. Such tree decompositions always exist even for the optimal treewidth and have the following properties:

1. The tree $\mathcal{T}$ is considered to be rooted at some $r \in I$.
2. Each node is either a *leaf* (0 children), or has exactly 1 or 2 children.
3. Let $i \in I$ be a leaf, then $|X_i| = 1$.
4. Let $j \in I$ be the only child of a node $i \in I$, then either (a) X_j contains all vertices of X_i except for one ($X_j \subset X_i$, $|X_j|+1 = |X_i|$), or (b) X_j contains all

vertices of X_i plus one additional one ($X_i \subset X_j$, $|X_i|+1 = |X_j|$). Considering the tree in a bottom-up fashion, the node i is then called an *introduce* or *forget* node, respectively.

5. Let $j, j' \in I$ be the two children of a node $i \in I$, then all three corresponding bags are identical ($X_j = X_{j'} = X_i$), and i is called a *join* node.

Overall, given any tree decomposition, we can easily transform it into a nice tree decomposition where we pick the root r such that its bag X_r contains at least one terminal vertex. While the latter property is not ultimately necessary, it allows us to give a simpler description of our algorithm. We will discuss this in more detail at the end of Section 2.2.

2 Steiner Tree Algorithm

Our algorithm follows the classical bottom-up approach for algorithms based on tree decompositions: Starting from the leaves of a nice tree decomposition $(\mathcal{T} = (I, F), \mathcal{X})$, we enumerate a sufficient number of possible sub-solutions per tree node $i \in I$, using only the information previously computed for the children of i. Such information is stored in a table tab_i, for the node $i \in I$. The final optimal solution of the original problem can then be read from the table tab_r of $\mathcal{T}$'s root node r.

Since the tree traversal requires only $O(|V|)$ time, the algorithm's time complexity is mainly dependent on the amount of information to be stored per node (i.e., the size of tab_i which can be estimated by the number of sub-solutions times the size per sub-solution), as well as on the necessary effort to establish the sub-solutions at a node, based on its children's data.

In Section 2.1, we will concentrate on the first question, i.e., how to represent the necessary solutions efficiently. In fact, this modeling (based on coloring) is the main result of this paper, which subsequently allows us to obtain stronger memory and runtime bounds than the previous approaches. Section 2.2 then describes how to efficiently combine our coloring with the bottom-up traversal to solve the Steiner tree problem. Finally, Section 2.3 formally establishes the correctness and running time of our approach.

2.1 Representing Sub-solutions

The general idea of using the (rooted) tree decomposition is the following: Let i be any node in $\mathcal{T}$ with the corresponding bag X_i. We define X_i^+ to be the set of all vertices in X_j for all nodes $j \in I$ that are either i itself or any of its descendants. Then, let G_i (G_i^+) describe the subgraph of G induced by the vertices X_i (X_i^+, respectively). Let T_i (T_i^+) be the set of terminals in X_i (X_i^+, respectively).

When we consider any node $i \in I$, we observe, based on property td/2 of a tree decomposition, that no vertex of $X_i^+ \setminus X_i$ will appear in any other bag than the ones descending from node i. For our bottom-up approach this means that these vertices are not considered in other parts of the algorithm and will never be considered again. Hence, the sub-solutions at node i have to ensure that all terminals $T_i^+ \setminus T_i$ are properly connected with other vertices to allow a feasible solution in the end. Consider the optimal Steiner tree $\mathcal{S}$ in G. The subgraph of $\mathcal{S}$ in G_i^+ then forms a forest, with the property that any terminal $T_i^+ \setminus T_i$ is connected to some vertex in X_i.

Our table tab_i hence stores multiple rows, each row representing a solution. Observe that we do not have to consider all possible subgraphs of a bag X_i but can use the fact that a forest in G_i contains at most $|X_i| - 1$ edges. It remains how to uniquely, succinctly, and compactly describe these forests (and allow for fast merging operations within the bottom-up approach). We show that it (coarsely) is sufficient to consider all possible *partitions* of the (at most $tw+1$ many) vertices X_i by assigning colors to them. Each color then indicates the set of vertices that lie in a connected component (tree, in fact) in G_i^+. We will see that by careful enumeration we only require a table with at most B_{tw+2} different partitions, instead of the straight-forward $\mathcal{O}((tw+1)^{tw+1})$.

To obtain such a description scheme, we first consider some arbitrary but fixed total numbering $\Phi : V \xrightarrow{1:1} \{1, \ldots, |V|\}$ of all vertices of the given graph. Based thereon, we assign—locally for each bag X_i—the unique secondary index $\varphi_i : X_i \xrightarrow{1:1} \{1, \ldots, |X_i|\}$ which satisfies $\Phi(v) < \Phi(w) \Leftrightarrow \varphi_i(v) < \varphi_i(w)$ for all $v, w \in X_i$. We now introduce a coloring function $\gamma_i : X_i \to \{0, \ldots, |X_i|\}$; thereby any vertex $v \in X_i$ may only be colored by a color at most as large as its local index, i.e., $\gamma_i(v) \leq \phi_i(v)$. Our interpretation is that all vertices of color 0 are not contained in the represented sub-solution. All vertices with a common color > 0 are connected in the graph G_i^+. Note that these connections do not have to exist in G_i. Finally, in order to be a feasible coloring, we require all terminals T_i in X_i to be colored > 0.

Note that, by the above coloring properties, the color of a connected component C of the sub-solution is exactly the smallest secondary index of all vertices contained in C. We observe that a vertex v with $\varphi_i(v) = z$ has $z+1$ possible colors. Hence the number of possible colorings for a bag X_i (and therefore of rows in tab_i) can trivially be bounded by $\prod_{z=1}^{|X_i|}(z+1) = (|X_i|+1)! = \mathcal{O}((tw+2)!)$. This would already allow better overall bounds for the algorithm than previously known. Yet, we can observe that when we conceptually add an additional "ghost" element to an $|X_i|$-element set, and consider all possible partitions thereof, we can interpret these resulting partitions as all possible colorings: The partition that contains the "ghost" element is considered to be the partition with color 0. All other partitions get the color of the smallest secondary index among its elements. It is straight-forward to efficiently enumerate all $B_{|X_i|+1}$ possible partitions (hence rows in tab_i) of a $|X_i|+1$-element set.

In each row, we store the unique corresponding coloring of the solution, i.e., a color index for each vertex of X_i, which we can trivially compute in $\mathcal{O}(tw)$ time. Additionally, we will store a solution value for each row, see below. Hence, the size of any table tab_i can be bounded by $\mathcal{O}(B_{tw+2} \cdot tw)$.

2.2 Processing the Decomposition Tree

Having our coloring concept at hand, we can now describe how to ensure its properties when computing the actual sub-solution tables in a bottom-up fashion. Our recursion can be described by distinguishing between the different currently considered nodes of $\mathcal{T}$. Recall that for each row, representing some coloring γ, we store the cost $val(\gamma)$ of the represented sub-solution.

Leaf Node. Let $i \in I$ be a leaf, and hence a (trivial) base case for our algorithm. The table tab_i contains only two rows corresponding to the two possible colors 0 and 1, respectively, for the only vertex $v \in X_i$. If $v \in T$ but is colored 0, the sub-solution's cost is $+\infty$; in all other cases the cost is 0.

Introduce Node. Let $i \in I$ be an introduce node, and $j \in I$ its only child. We have $X_j \subset X_i$, $|X_j| + 1 = |X_i|$, and let v be the additional vertex.

As a preprocessing, we initialize tab_i and modify tab_j as follows: We generate all $B_{|X_i|+1}$ possible rows of tab_i and set their value entries to $+\infty$. In tab_j we add an additional column for v (which remains uncolored, say color -1) and modify the other color numbers to match the coloring scheme of i, instead of j: By the fact that both secondary indices stem from a common primary index Φ, this means that precisely all colors $\geq \phi_i(v)$ have to be increased by one. We observe that this preprocessing takes only $\mathcal{O}(B_{tw+2} \cdot tw)$ time.

The cost for any coloring γ_i of X_i with $\gamma_i(v) = 0$ is straight-forward: Let γ_j be the unique coloring in tab_j that agrees with γ_i on all vertices except for v. If $v \in T$, i.e., v is a terminal vertex, $val(\gamma_i) = +\infty$, otherwise $val(\gamma_i) = val(\gamma_j)$.

Now, we consider all *compatible* combinations of rows of tab_j and tab_i with the intuition that several connected components of a solution at j may become connected via the newly inserted, > 0-colored vertex v. Therefore, a coloring γ_j of X_j is *compatible* with a coloring γ_i of bag X_i with $\gamma_i(v) \neq 0$ if and only if the color partitions agree for all colors except for the color to which v belongs. More formally, let c be the color of v in γ_i, then any vertex partition induced by some color in γ_j is either also a vertex partition with the same color in γ_i, or a (proper) subset of the vertex partition of color c in γ_i. Intuitively, the vertex v connects with some formerly separated color partitions, coloring them all with a common color.

We can compute the cost $val(\gamma_i)$ for this solution at the introduce node i by adding the costs of these new connections to the precomputed cost $val(\gamma_j)$ of γ_j. For the former, we simply have to find, for each *formerly* separate color partition W, the cheapest edge in G_i connecting v with any vertex in W, and sum over these costs. If no such edge exists, the corresponding connection cost is $+\infty$. If the so computed cost of γ_i is smaller than the current $val(\gamma_i)$ entry for this coloring in tab_i, we update $val(\gamma_i)$ accordingly. Hence, processing an introduce node takes $\mathcal{O}(B_{tw+2}^2 \cdot tw)$ time.

Forget Node. Let $i \in I$ be a forget node, and $j \in I$ its only child. We have $X_i \subset X_j$, $|X_i| + 1 = |X_j|$, and let v be the additional (discarded, in fact) vertex.

As a preprocessing, we generate all rows of tab_i and set their solution costs to $+\infty$. We then look at the rows of tab_j one by one; let γ_j be the corresponding coloring, and $c := \gamma_j(v)$. We say γ_j *induces* a coloring γ_i of the vertices X_i, by simply dropping the vertex v and shifting the color index by -1 for all colors $> \phi_j(v)$; the vertices colored with color $\phi_j(v)$ in γ_j obtain the color matching the smallest secondary index $\phi_i(.)$ among themselves. Note that we can look up the row of the induced coloring in tab_i in $\mathcal{O}(tw)$ by exploiting the enumeration scheme.

If $c > 0$ but there is no other vertex with color c, we cannot easily remove this vertex from the solution, as it represents a component (containing, in general, terminals) that has to be connected to the final Steiner tree $\mathcal{S}$ (recall that we can safely assume that the decomposition tree's root node contains a terminal). Hence we cannot use this sub-solution to improve the solution value of the induced coloring of X_i. Otherwise, we can safely drop the vertex and set $val(\gamma_i) := val(\gamma_j)$ if the current value of $val(\gamma_i)$ is not already smaller.

Join Node. Let $i \in I$ be a join node, and $j, j' \in I$ its two children. We have $X_j = X_{j'} = X_i$.

Again, we first construct all rows of tab_i and set the solution values to $+\infty$. Then we consider all possible combinations of solutions from X_j and $X_{j'}$. Let γ_j and $\gamma_{j'}$ be colorings (rows) of tab_j and $tab_{j'}$, respectively. We want to construct a merged solution γ_i that resembles the combined connectivities of both solutions, i.e., two vertices $v_s, v_t \in X_i$ should be in the same color partition if and only if there is a vertex sequence $\langle v_s := v_1, v_2, \ldots, v_\beta := v_t\rangle$ in X_i such that, for all $1 \leq \alpha < \beta$, the vertices $v_\alpha, v_{\alpha+1}$ have the same color in γ_j or $\gamma_{j'}$.

Note that, a priori, such a merge might lead to cycles in the solution: assume two vertices v_1, v_2 are colored with identical color c_j in γ_j. Furthermore, they have a (probably different but) common color $c_{j'}$ in $\gamma_{j'}$. Hence the vertices are connected in both sub-solutions, but the connection paths do not need to coincide. Even if the paths do coincide, we would have to identify them to not count their cost twice for the combined solution. Hence, we only want to combine solutions with the property that any pair of vertices has a common color > 0 in at most one of the two colorings $\gamma_j, \gamma_{j'}$. Then, the value of the combined solution can be given as $val(\gamma_i) := val(\gamma_j) + val(\gamma_{j'})$, which we can store into tab_i (unless the stored value for this solution is already smaller). Again, observe that we can identify the row index in tab_i of any given solution γ_i in $\mathcal{O}(tw)$ by exploiting the enumeration scheme.

It would be trivial to perform the check whether to merge, as well as the actual merge, in $\mathcal{O}(tw^2)$ time, for any given pair of sub-solutions. Yet, we can do better and perform the merge operation, including the check of the precondition, in linear time $\mathcal{O}(tw)$: Consider a helper array $recol : \{1, \ldots, |X_i|\} \to \{1, \ldots, |X_i|\}$ and construct a graph C with a vertex c_r per possible color r. Then, for each $v \in X_i$, add an edge $(c_{\gamma_j(v)}, c_{\gamma_{j'}(v)})$. Clearly, the graph has only $\mathcal{O}(tw)$ vertices and edges. Remove the vertex c_0 together with its incident edges, and mark all other vertices in C as unvisited. Then, for increasing $r \in \{1, \ldots, |X_i|\}$, start a depth-first search (DFS) in C at any unvisited c_r: set $recol(c_{r'}) := r$ for any vertex $c_{r'}$ visited in this DFS run. Hence, in the end, $recol$ gives the new color for any color in either γ_j or $\gamma_{j'}$. Whenever a DFS run revisits an already visited vertex (within the same run), we identified a cycle (including the special case of multiple edges), and the merge operation should be aborted. If no cycles are detected, we can finally again consider each $v \in X_i$ and set $\gamma_i(v) := 0$ if $\gamma_j(v) = \gamma_{j'}(v) = 0$, and $\gamma_i(v) := \mathit{recol}(\max\{\gamma_j(v), \gamma_{j'}(v)\})$ otherwise.

Remark. Note that if the given graph G has only positive edge weights, we do not need to actively identify cycles or multiedges: the merged solution's objective value will be greater than the alternative cycle/multiedge-free combination, which will, at some point, also be considered. Since we store only the best solution for any coloring in tab_i, the stored solutions will always be cycle- and multiedge-free.

Extracting the Solution at the Root Node. From the described construction process it is clear that each solution of a bag X_i describes the (minimum) costs of a forest where all terminals from X_i^+ are (probably indirectly) connected to some vertex of X_i. Also recall that it can be safely assumed that at least one terminal is contained in the root bag X_r of $\mathcal{T}$. Hence the optimum solution value for the whole graph can be found in the root bag X_r of $\mathcal{T}$, identifying a cheapest solution where all vertices with color $\neq 0$ are contained in the same connected component (i.e., have the same color).

Computing the optimum solution, i.e., the set of edges, is possible by backtracking or by storing the set of edges for each row and each bag. The latter increases the required memory but has no negative impact on the running time since these sets are simple linked lists that can be concatenated in $\mathcal{O}(1)$.

Remark. We can—with the same time complexity—also run the algorithm on a tree decomposition where the root node does not contain any terminal vertex. In this case, whenever we process a tree node i where $T \subseteq X_i^+$ (i.e., all terminals are within the subtree induced by i), we check for the best solution where all vertices with color $\neq 0$ belong to the same color partition, and store a reference to it. After processing the root node, this reference gives the optimal solution.

2.3 Analysis

In the following, we will discuss the algorithm's running time and prove that it correctly computes an optimal solution.

Lemma 1. *The above algorithm requires $\mathcal{O}(B_{tw+2}^2 \cdot tw \cdot |V|)$ time.*

Proof. The running time mainly depends on the size of the tables and the combination of tables during the bottom-up traversal of the decomposition tree. We already established that each table tab_i at some tree node i stores $\mathcal{O}(B_{tw+2})$ rows and requires overall $\mathcal{O}(B_{tw+2} \cdot tw)$ storage.

During the bottom-up traversal of $\mathcal{T}$ we consider all possible row combinations for two tables in the case of the introduce and the join node. For each such combination, we perform a merge operation in $\mathcal{O}(tw)$, and we hence require overall $\mathcal{O}(B_{tw+2}^2 \cdot tw)$ time. This bound dominates the time required for the other tree node types (forget and leaf nodes), as well as all other extra effort—feasibility tests, shifting of indices, etc.—which is only linearly dependent on the treewidth. Due to the linear size of $\mathcal{T}$, we can deduce the overall running time. □

Lemma 2. *The above algorithm correctly computes an optimal solution to the given Steiner tree problem.*

Proof. The algorithm's correctness can be shown by a straight-forward inductive proof on the decomposition tree. Let $\Gamma_i^c := \{v \in X_i \mid \gamma_i(v) = c\}$ be the vertices colored c in a coloring γ_i. Our induction hypothesis (IH) states that, for each processed bag X_i, the cost of each solution γ_i corresponds to a minimum forest $F_i \subseteq G_i^+$ with the properties

- F_i consists of (pairwise disconnected) trees F_i^c, one for each color $c > 0$ with $\Gamma_i^c \neq \emptyset$, with $\Gamma_i^c \subseteq V(F_i^c)$, and $\Gamma_i^{c'} \cap V(F_i^c) = 0$ for all $c' \neq c$. I.e., each tree connects only vertices of the same color partition.
- F_i contains all terminals of G_i^+, i.e., $T_i^+ \subseteq V(F_i)$.

The base cases are leaf nodes where the hypothesis clearly holds. Now, let the induction hypothesis be true for all descendants of a bag X_i.

Forget node. Each coloring of a forget bag X_i is induced by $|X_i| + 1$ many colorings in the child table—one for each possible color of the forget vertex. Our algorithm picks the minimal among them that remains feasible after the removal of the forget vertex, and does not change its solution value.

Assume the minimum solution γ_i at X_i would be smaller then this identified sub-solution. Then we could add the forget vertex to the solution γ_i of X_i, coloring it as required by F_i. This is a feasible coloring for the child node, and stays feasible after removing the forget vertex. Hence, it would have been considered by our algorithm (without modifying its solution value).

Introduce node. For an introduce node i, the solution table contains a copy of its child table, when coloring the new vertex either 0 or with its own secondary index. Furthermore, the new vertex allows the connection of several components.

Assume some optimal solution at an introduce bag X_i would be smaller than the one obtained by the algorithm. If the introduced vertex v is colored 0 or has a unique color, the otherwise identical coloring (up to index shifting) was stored in the child table (IH). As the algorithm would not have changed the solution value, we arrive at a contradiction. Now assume v belongs to some color class Γ_i^c with more than one element, and let F_i^c be the corresponding solution tree. When we remove v from F_i^c, it decomposes into several components. Our algorithm considered all possible such components in the child table, including their optimal costs (IH), and attached them to v via the minimal edges. That means that our algorithm considered this solution and would have computed its costs correctly.

Join node. Similar to above, assume that we would have a solution γ_i at a join node i which is strictly smaller than the one computed by our algorithm, and consider the forest F_i. Observe that the vertex set X_i of i is identical to those of its children j, j'. We can partition F_i into two sub-forests: Let F_i^1 be the forest restricted to the edges of G_j^+, and let F_i^2 be the forest restricted to the edges $E(G_{j'}^+) \setminus E(G_{j'})$, i.e., it does not contain any edges already contained in F_i^1. Observe that F_i^1 induces a feasible coloring solution at node j, and F_i^2 a feasible coloring solution at node j, and that both are disjoint. Hence, by (IH), our algorithm would have considered to merge the corresponding optimal subsolutions to obtain γ_i, with the correct objective value based on summing the costs of F_i^1 and F_i^2. □

Finally, the following theorem summarizes the above lemmas.

Theorem 1. *Given a graph with vertex set V and a tree decomposition with treewidth tw, the Steiner tree problem can be solved to optimality in $\mathcal{O}(B_{tw+2}^2 \cdot tw \cdot |V|)$ time.*

3 Prize-Collecting Steiner Tree

The prize-collecting Steiner tree problem (PCSTP) is an extension of the STP. Thereby, instead of being required to connect all terminals, we get a (vertex-specific) *prize* for each vertex we connect. We are hence given a function $p : V \to \mathbb{R}_{>0}$ and want to find a tree $\mathcal{S} = (V_\mathcal{S}, E_\mathcal{S})$ that minimizes $\sum_{e \in E_\mathcal{S}} c(e) - \sum_{v \in \mathcal{S}} p(v)$, where c is the edge-cost function.[1]

Our algorithm for the STP can be adapted by introducing the profits into the cost calculations (at the introduce nodes) and removing the necessity that terminals are assigned a color $\neq 0$. Because terminals may be omitted, the optimum solution need not necessarily be captured by the table at the decomposition tree's root node. Hence, during the bottom-up traversal each row of each table is a potential global solution if the corresponding coloring induces a feasible tree. The remaining part of the algorithm remains identical and after the previous discussion on the running time and optimality we conclude the following theorem.

Theorem 2. *Given a graph with vertex set V and a tree decomposition with treewidth tw, the prize-collecting Steiner tree problem can be solved to optimality in $\mathcal{O}(B_{tw+2}^2 \cdot tw \cdot |V|)$ time.*

4 *k*-Cardinality Tree

The k-cardinality tree (KCT) problem is defined on an edge-weighted, undirected graph and asks for a minimum-cost tree containing exactly k edges. Betzler [7] introduced an FPT algorithm with parameter k and time complexity $\mathcal{O}(2^{\mathcal{O}(k)} k \cdot |E| \cdot \log |V|)$. Ravi et al. [24] sketched a general FPT strategy for any decomposable graph [6] (including graphs with bounded treewidth) with time complexity $\mathcal{O}(f(tw) \cdot k^2 \cdot |V|)$. As their general description considers any

[1] Sometimes, the objective function is also described as $\min \sum_{v \notin V_\mathcal{S}} p(v) + \sum_{e \in E_\mathcal{S}} c(e)$. From the point of view of optimal solutions, both problems are equivalent as $\sum_{v \in V} p(v)$ is a constant. We prefer the former definition to be able to locally evaluate the objective function at each node of the decomposition tree.

dependence on the decomposition's parameter (e.g., treewidth) a constant, there are no more details on the non-polynomial function $f(tw)$. In the following, we describe how to extend our previous algorithm for the STP to obtain an exact algorithm for the KCT problem with a running time that increases by less than a factor of k^2, compared to the STP. In fact, our extension follows the concept of [24], although in a less abstract way. Our obtained runtime bound is equivalent to their result, and, to our knowledge, constitutes the first published bound for $f(tw)$.

As for the (PC)STP, we enumerate all possible partitions of the vertices of each bag by assigning colors, and propagate optimal solutions to the root bag in a bottom-up traversal of $\mathcal{T}$. Yet, in contrast to the (PC)STP, holding a single solution per partition and choosing minima is not sufficient as we have to take the overall number of chosen edges into account.

Therefore the algorithm maintains, for each possible coloring at node i, a solution value of a minimal forest with exactly k' edges—establishing the color-induced partition—for each possible $0 \leq k' \leq k$. All vertices of such a *k'-forest* are either from X_i or are (indirectly) connected to vertices in X_i; all vertices of a tree of such a forest are colored identically. Clearly, for solutions with ℓ non-0-colored vertices and c different colors > 0, the solution value for $k' < \ell - c$ is $+\infty$, as any feasible forest requires more edges. The main observation is that these k'-forests are always disjoint from any solution considered at any node not in the subtree rooted at i, except for the vertices and edges in G_i. Overall, the size of each row at any table tab_i is $\mathcal{O}(tw + k)$.

Trivially, both possible colorings at a *leaf node* have cost 0 for the 0-forest and $+\infty$ otherwise. For a *forget node* the component of the forget vertex v has to be considered: if v is the only vertex with color > 0 in the child table tab_j, the attached size-k forest (tree, in fact) might be the optimum k-cardinality tree; hence, we compare and update the global optimum (similar as for the PCSTP). If v is the only vertex with color γ_j but there are also other vertices colored > 0, we cannot deduce a feasible solution and obtain forest values $+\infty$. Otherwise (i.e., v does not define its own color class, or is colored 0), v can be simply discarded without changing the costs of the k'-forests. Analogously to the STP—and in the following also for the cases of the other inner nodes—we always store the smallest solution value for each k' that is achievable by a reduction from any compatible coloring of the child bag.

The new vertex v in an *introduce node* might connect several connected components, say c many. Similar to the STP, the cheapest edges connecting v with each component are chosen; the cost of a k'-forest at the child node, together with the cost-sum of the new edges, gives the cost of a $(k' + c)$-forest for the considered coloring at node i. As we consider all compatible colorings at the child node and store the minimum per k'', we will, in the end, know the minimally achievable k''-forest for any possible cardinality $1 \leq k'' \leq k$ at i.

For a *join node* observe that k'-forests from two combined solutions are pairwise disjoint, as long as their coloring does not induce cycles or multiedges, as we discussed for the STP. Hence, as for the STP, we combine only two solutions with this property, reusing our DFS sub-algorithm. To compute the new minimal k'-forest, for each k', we consider all combinations of a k_1-forest of the first, and a k_2-forest of the second solution with $k_1 + k_2 = k'$. These are $\mathcal{O}(k^2)$ computations.

After processing the root node, we may update the globally stored optimum by the k-forests (trees, in fact) arising from colorings with a single non-0 color.

Analyzing the running time, we again require tables with $\mathcal{O}(B_{tw+2})$ rows, each row of size $\mathcal{O}(tw + k)$. In case of a join and an introduce node, two tables are combined by considering all possible $\mathcal{O}(B_{tw+2}^2)$ combinations; the largest effort of $\mathcal{O}(k^2 + tw)$ per combination arises at a join node. Due to space restrictions and obvious analogies to the STP we omit the correctness proof and close the discussion on the KCT problem with the following theorem.

Theorem 3. *Given a graph with vertex set V and a tree decomposition with treewidth tw, the k-cardinality tree problem can be solved optimally in $\mathcal{O}(B_{tw+2}^2 \cdot (tw + k^2) \cdot |V|)$ time.*

5 Conclusions

We showed new, currently fastest treewidth-based exact algorithms for the STP, the PCSTP, and the KCT problem. For the former two problems, these algorithms also directly speed-up current PTASes for planar STP and PCSTP, as those use algorithms for bounded treewidth as their most time-consuming subroutines.

References

1. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM J. Algebraic Discrete Methods 8(2), 277–284 (1987)
2. Bateni, M., Chekuri, C., Ene, A., Hajiaghayi, M., Korula, N., Marx, D.: Prize-collecting Steiner problems on planar graphs. In: SODA, pp. 1028–1049. SIAM (2011)
3. Bateni, M., Hajiaghayi, M., Marx, D.: Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth. In: STOC, pp. 211–220. ACM (2010)
4. Bateni, M., Hajiaghayi, M., Marx, D.: Prize-collecting network design on planar graphs. CoRR, abs/1006.4339 (2010)
5. Berend, D., Tassa, T.: Improved bounds on Bell numbers and on moments of sums of random variables. Probability and Mathematical Statistics 30, 185–205 (2010)
6. Bern, M.W., Lawler, E.L., Wong, A.L.: Linear-time computation of optimal subgraphs of decomposable graphs. J. Algorithms 8, 216–235 (1987)
7. Betzler, N.: Steiner tree problems in the analysis of biological networks. Master's thesis, Universität Tübingen (2006)
8. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Fourier meets Möbius: Fast subset convolution. In: STOC, pp. 67–74. ACM (2007)
9. Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybernetica 11(1-2), 1–22 (1993)

10. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. 25(6), 1305–1317 (1996)
11. Bodlaender, H.L.: Treewidth: Structure and Algorithms. In: Prencipe, G., Zaks, S. (eds.) SIROCCO 2007. LNCS, vol. 4474, pp. 11–25. Springer, Heidelberg (2007)
12. Borradaile, G., Kenyon-Mathieu, C., Klein, P.: A polynomial-time approximation scheme for Steiner tree in planar graphs. In: SODA, pp. 1285–1294. SIAM (2007)
13. Borradaile, G., Klein, P., Mathieu, C.: An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. ACM Transactions on Algorithms 5, 1–31 (2009)
14. Chekuri, C., Ene, A., Korula, N.: Prize-collecting Steiner tree and forest in planar graphs. CoRR, abs/1006.4357 (2010)
15. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time. CoRR, abs/1103.0534 (2011)
16. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
17. Dreyfus, S.E., Wagner, R.A.: The Steiner problem in graphs. Networks 1, 195–207 (1972)
18. Garey, M.R., Johnson, D.S.: The rectilinear Steiner tree problem is NP-complete. SIAM Journal on Applied Mathematics 32(4), 826–834 (1977)
19. Gassner, E.: The Steiner forest problem revisited. J. Discrete Algorithms 8(2), 154–163 (2010)
20. Korach, E., Solel, N.: Linear time algorithm for minimum weight Steiner tree in graphs with bounded treewidth. Technical Report 632, Israel Institute of Technology (1990)
21. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
22. Polzin, T., Daneshmand, S.: Practical Partitioning-Based Methods for the Steiner Problem. In: Àlvarez, C., Serna, M. (eds.) WEA 2006. LNCS, vol. 4007, pp. 241–252. Springer, Heidelberg (2006)
23. Prömel, H.J., Steger, A.: The Steiner Tree Problem. A Tour Through Graphs, Algorithms and Complexity. Vieweg Verlag (2002)
24. Ravi, R., Sundaram, R., Marathe, M.V., Rosenkrantz, D.J., Ravi, S.S.: Spanning trees short or small. In: SODA, pp. 546–555. SIAM (1994)
25. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. J. Algorithms 7(3), 309–322 (1986)