Cut and Count and Representative Sets on Branch Decompositions

Willem J. A. Pino[1], Hans L. Bodlaender[*2], and Johan M. M. van Rooij[3]

1 **Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands**
w.j.a.pino@students.uu.nl

2 **Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, and**
Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
H.L.Bodlaender@uu.nl

3 **Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, and**
Consultants in Quantitative Methods, Eindhoven, The Netherlands
jmmrooij@cs.uu.nl

Abstract

Recently, new techniques have been introduced to speed up dynamic programming algorithms on tree decompositions for connectivity problems: the 'Cut and Count' method and a method called the rank-based approach, based on representative sets and Gaussian elimination. These methods respectively give randomised and deterministic algorithms that are single exponential in the treewidth, and polynomial, respectively linear in the number of vertices. In this paper, we adapt these methods to branch decompositions yielding algorithms, both randomised and deterministic, that are in many cases faster than when tree decompositions would be used.

In particular, we obtain the currently fastest randomised algorithms for several problems on planar graphs. When the involved weights are $\mathcal{O}(n^{\mathcal{O}(1)})$, we obtain faster randomised algorithms on planar graphs for Steiner Tree, Connected Dominating Set, Feedback Vertex Set and TSP, and a faster deterministic algorithm for TSP. When considering planar graphs with arbitrary real weights, we obtain faster deterministic algorithms for all four mentioned problems.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory, I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases Graph algorithms, Branchwidth; Treewidth, Dynamic Programming, Planar Graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.27

1 Introduction

It is well known that many problems that are NP-hard on general graphs, become polynomial or linear time solvable on graphs where the treewidth or branchwidth is bounded by a constant. More precisely, many problems are fixed parameter tractable with treewidth or branchwidth as parameter. For an overview regarding treewidth, e.g., see [1] .

* Hans L. Bodlaender was partially supported by the Networks project, funded by the Dutch Ministry of Education, Culture and Science through NWO.

Table 1 Our results using the 'Cut and Count' (randomised) and rank-based (exact) techniques.

Problem	Randomised	Deterministic
Steiner Tree	$\mathcal{O}(3^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$	$\mathcal{O}(n((1+2^{\omega})\sqrt{5})^{bw}bw^{\mathcal{O}(1)})$
Connected Dominating Set	$\mathcal{O}(4^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$	$\mathcal{O}(n((2+2^{\omega})\sqrt{6})^{bw}bw^{\mathcal{O}(1)})$
Feedback Vertex Set	$\mathcal{O}(3^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$	$\mathcal{O}(n((1+2^{\omega})\sqrt{5})^{bw}bw^{\mathcal{O}(1)})$
Hamilton Cycle / TSP	$\mathcal{O}(4^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$	$\mathcal{O}(n(5+2^{\frac{\omega+2}{2}})^{bw}bw^{\mathcal{O}(1)})$
Planar Steiner Tree	$\mathcal{O}(2^{3.991\sqrt{n}})$	$\mathcal{O}(2^{8.039\sqrt{n}})$
Planar Connected Dominating Set	$\mathcal{O}(2^{5.036\sqrt{n}})$	$\mathcal{O}(2^{8.778\sqrt{n}})$
Planar Feedback Vertex Set	$\mathcal{O}(2^{3.991\sqrt{n}})$	$\mathcal{O}(2^{8.039\sqrt{n}})$
Planar Hamilton Cycle / TSP	$\mathcal{O}(2^{5.036\sqrt{n}})$	$\mathcal{O}(2^{6.570\sqrt{n}})$

It was long known that many graph problems with a local nature (e.g., Independent Set, Dominating Set) can be solved on graphs given with a tree decomposition of width k in time, that is single exponential in k and linear in the number of vertices n, e.g., see [17]. For several problems with a global 'connectivity' property in it, it was open whether there existed $\mathcal{O}(2^{\mathcal{O}(k)}n^{\mathcal{O}(1)})$ time algorithms. This was resolved by Cygan et al. [5] with the '*Cut and Count*' method; this approach gives fast randomised algorithms that are single-exponential in the treewidth and polynomial in the number of vertices for various problems, e.g., Feedback Vertex Set, Hamiltonian Circuit, TSP, Connected Dominating Set. At the cost of a higher constant in the base of the exponential factor, Bodlaender et al. [2] gave deterministic algorithms that are single-exponential in the treewidth and linear in the number of vertices for these connectivity problems, with a technique, based on representative sets and Gaussian elimination, called the *rank-based* approach. This algorithm was experimentally evaluated by Fafianie et al. [9], showing that in the case of the Steiner Tree problem, the method gives a significant speedup over naive dynamic programming. An alternative method that gives similar time bounds, based on representative sets and matroids, was given by Fomin et al. [11]. Later, Fomin et al. [10] showed how to use matroids to speed up the computation at join nodes in these algorithms leading, for several connectivity problems with Steiner Tree as flagship example, to the currently fastest algorithms on graphs of bounded treewidth.

Branchwidth is another well studied graph parameter, with strong relations to treewidth. The branchwidth and treewidth of a graph are bounded by each other in the following way: $bw \leq tw + 1 \leq \lfloor \frac{3}{2}bw \rfloor$. The transformation from a tree decomposition to a branch decomposition or vice versa, fulfilling these bounds can be executed in linear time. This implies that a running times of the form $\mathcal{O}(c^k n^{\mathcal{O}(1)})$ for graphs of treewidth k or branchwidth k follow from each other, except for a possibly different value for the base of the exponent c.

In this paper, we show that 'Cut and Count' and the rank-based approach can be used directly on branch decompositions. As a result, we obtain, in several cases, improvements compared to using tree decompositions instead. For an overview of our results, see Table 1.

Two other techniques to speed up dynamic programming algorithms on tree and branch decompositions are the following: Dorn [6] showed how to use matrix multiplication to speed up algorithms on branch decompositions and van Rooij et al. [3, 18] showed how to speed up algorithms on tree, branch and clique decompositions using *(generalised) subset convolutions*. In this paper, we build upon these works applying these techniques where possible.

For a comparison of our results to the current best treewidth algorithms, see Table 2 and Table 3. Here, $\omega < 2.373$ [14] is the matrix multiplication exponent. Our branch decomposition based results improve known treewidth results for parts of the range $bw \leq tw + 1 \leq \lfloor \frac{3}{2}bw \rfloor$ (note that $\frac{\omega}{2} < \frac{3}{2}$). In case of deterministic algorithms for TSP with

Table 2 Comparison of our results with best known results on treewidth [5] for randomised algorithms on problems where the weights are $\mathcal{O}(n^{\mathcal{O}(1)})$.

Problem	Treewidth	Branchwidth
Steiner Tree	$\mathcal{O}(3^{tw}n^{\mathcal{O}(1)})$	$\mathcal{O}(3^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$
Connected Dominating Set	$\mathcal{O}(4^{tw}n^{\mathcal{O}(1)})$	$\mathcal{O}(4^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$
Feedback Vertex Set	$\mathcal{O}(3^{tw}n^{\mathcal{O}(1)})$	$\mathcal{O}(3^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$
Hamilton Cycle / TSP	$\mathcal{O}(4^{tw}n^{\mathcal{O}(1)})$	$\mathcal{O}(4^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$

Table 3 Comparison of our results with best known results on treewidth for deterministic algorithms on problems with arbitrary real weights.

Problem	Treewidth	Branchwidth
Steiner Tree	$\mathcal{O}(n2^{3.134tw})$ [10]	$\mathcal{O}(n2^{3.790bw})$
Connected Dominating Set	$\mathcal{O}(n2^{3.628tw})$ [2]	$\mathcal{O}(n2^{4.137bw})$
Feedback Vertex Set	$\mathcal{O}(n2^{3.134tw})$ [10]	$\mathcal{O}(n2^{3.790bw})$
Hamilton Cycle / TSP	$\mathcal{O}(n2^{3.257tw})$ [2]	$\mathcal{O}(n2^{3.257bw})$

arbitrary real weights, our algorithms even give the advantage of using lower width branch decompositions compared to tree decompositions without the additional cost of a higher constant in the base of the exponent of the running time.

As planar graphs have branchwidth at most $2.122\sqrt{n}$, and such a branch decomposition can be constructed in polynomial time [12] (or we use the ratcatcher algorithm that exactly computes the branchwidth of planar graphs in $\mathcal{O}(n^3)$ time [15, 16]), we can apply our algorithms to solve connectivity problems on planar graphs. This leads to the currently fastest algorithms on planar graphs for several problems improving upon the best known results, due to Dorn [6, 7]. When considering randomised algorithms, we improve the currently fastest algorithms for all considered problems when weights are bounded by $\mathcal{O}(n^{\mathcal{O}(1)})$. When considering determinstic algorithms, we improve the currently fastest algorithms for all considered problems with arbitrary real weights, and the currently fastest algorithm for Hamilton Cycle and TSP when weights are bounded by $\mathcal{O}(n^{\mathcal{O}(1)})$.

2 Preliminaries

Let $G(V, E)$ be a graph with $|V| = n$ vertices and $|E| = m$ edges. For a vertex set $X \subseteq V$ the induced subgraph is denoted by $G[X]$, i.e., $G[X] = G(X, E \cap (X \times X))$. Likewise, the induced subgraph of an edge set $Y \subseteq E$ is denoted as $G[Y]$, i.e., $G[Y] = G(V(Y), Y)$ where $V(Y)$ stands for all endpoints of edges in Y. A cut in a graph is a tuple of two vertex sets (X_1, X_2) for which it holds that $X_1 \cup X_2 = V$ and $X_1 \cap X_2 = \emptyset$.

Throughout the paper the Iverson bracket notation is used. This notation denotes a number that is 1 if the condition between the brackets is satisfied and 0 otherwise, e.g., $[1 = 1]42 = 42$ and $[1 = 2]42 = 0$. We also use this notation in combination with sets S, then this denotes $[True]S = S$ and $[False]S = \emptyset$.

This paper considers dynamic programming algorithms on branch decompositions.

▶ **Definition 1** (Branch decomposition). A branch decomposition of a graph G is a tree $\mathbb{T}$ in which every internal node has degree 3 together with a bijection between the leaves of $\mathbb{T}$ and the edges of G.

As such, every leaf of $\mathbb{T}$ is assigned an edge of G and every edge of G is in exactly one leaf.

Table 4 Comparison of our results on planar graphs with best known results. The column 'Dorn ($n^{\mathcal{O}(1)}$)' states deterministic results by Dorn [6] when weights are $\mathcal{O}(n^{\mathcal{O}(1)})$; the column 'Dorn ($\mathbb{R}$)' states deterministic results by Dorn [7] for arbitrary real weights; the column 'Randomised' states our randomised results when weights are $\mathcal{O}(n^{\mathcal{O}(1)})$; and the column 'Deterministic' states our deterministic results that also apply to arbitrary real weights. We note that the mentioned results by Dorn [6, 7] have not been adjusted for the recently slightly improved matrix multiplication constant ω [14].

Problem	Dorn ($n^{\mathcal{O}(1)}$)	Dorn ($\mathbb{R}$)	Randomised	Deterministic
Planar Steiner Tree	$\mathcal{O}(2^{7.16\sqrt{n}})$	$\mathcal{O}(2^{8.49\sqrt{n}})$	$\mathcal{O}(2^{3.991\sqrt{n}})$	$\mathcal{O}(2^{8.039\sqrt{n}})$
Planar Connected Dom. Set	$\mathcal{O}(2^{8.11\sqrt{n}})$	$\mathcal{O}(2^{9.82\sqrt{n}})$	$\mathcal{O}(2^{5.036\sqrt{n}})$	$\mathcal{O}(2^{8.778\sqrt{n}})$
Planar Feedback Vertex Set	$\mathcal{O}(2^{7.56\sqrt{n}})$	$\mathcal{O}(2^{9.26\sqrt{n}})$	$\mathcal{O}(2^{3.991\sqrt{n}})$	$\mathcal{O}(2^{8.039\sqrt{n}})$
Planar Hamilton Cycle/TSP	$\mathcal{O}(2^{8.15\sqrt{n}})$	$\mathcal{O}(2^{9.86\sqrt{n}})$	$\mathcal{O}(2^{5.036\sqrt{n}})$	$\mathcal{O}(2^{5.63\sqrt{n}})$

The removal of an edge x in a branch decomposition $\mathbb{T}$ divides the edges of G in two parts E_1 and E_2, namely the edges assigned to the leaves of the resulting subtrees T_1 and T_2 of $\mathbb{T}$. For an edge x in $\mathbb{T}$, the associated *middle set* is the vertex subset $B_x \subseteq V$ consisting of all vertices both in $G[E_1]$ and in $G[E_2]$, i.e., $B_e = V_1 \cap V_2$ where V_1 and V_2 are the vertices in $G[E_1]$ and $G[E_2]$, respectively. The *width* assigned to the edge x is the size of the middle set B_x. The width of a branch decomposition $\mathbb{T}$ is the maximum width over all edges of the decomposition, and the branchwidth of a graph G is the minimum width over all possible branch decompositions of G.

To simplify the presentation, we only consider *rooted* branch decompositions. One obtains a rooted branch decomposition by splitting an arbitrary edge (u, v) in the branch decomposition into (u, w) and (w, v), adding a root node r, and adding the edge (w, r). The middle sets of these three edges are defined to be $B_{(u,w)} = B_{(w,v)} = B_{(u,v)}$ and $B_{(w,r)} = \emptyset$. On rooted branch decompositions, we can define a *leaf edge* to be an edge of $\mathbb{T}$ connected to a leaf of $\mathbb{T}$, the *root edge* to be the edge (w, r) to the root r, and an *internal edge* to be any other edge of $\mathbb{T}$. Additionally, for a non-leaf edge x of $\mathbb{T}$, we can now define its *left child* y and *right child* z in $\mathbb{T}$ by ordering the two edges below x in $\mathbb{T}$.

A dynamic programming algorithm on branch decompositions typically computes a table A_x for every edge x of the branch decomposition $\mathbb{T}$ in a bottom-up fashion. Such a table A_x usually contains a set of partial solutions (or the number of partial solutions) on $G[E_x]$ where E_x is the set of the edges assigned to the leaves below the edge x in $\mathbb{T}$. In the case that x is the root edge, the table A_x contains (the number of) complete solutions.

When considering a non-leaf edge x of a branch decomposition $\mathbb{T}$, it is convenient to define a well-known partitioning on the three middle sets involved.

▶ **Definition 2** (Partioning of middle sets)**.** Consider a non-leaf edge x in a branch decomposition $\mathbb{T}$. Let x have left child y and right child z, and let the associated middle sets be B_x, B_y, and B_z. We now define the following partitioning of $B_x \cup B_y \cup B_z$ (see Figure 1):

- *Intersection vertices:* $I = B_x \cap B_y \cap B_z$.
- *Forget vertices:* $F = (B_y \cap B_z) \setminus B_x$.
- *Left vertices:* $L = (B_x \cap B_y) \setminus B_z$.
- *Right vertices:* $R = (B_x \cap B_z) \setminus B_y$.

▶ **Lemma 3** (Constraints on size of middle set partitions)**.** *Given a branch decomposition $\mathbb{T}$ of width bw, the following inequalities on the sizes of the middle-set partitions hold for all non-leaf edges in $\mathbb{T}$:*

- $|I| + |L| + |R| \leq bw$.
- $|I| + |L| + |F| \leq bw$.
- $|I| + |F| + |R| \leq bw$.

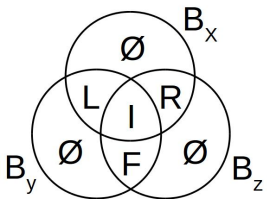

Figure 1 The partitioning of the middle sets.

Finally, to obtain our results on planar graphs, we need the following lemma that relates planar graphs to branch decompositions:

▶ **Lemma 4** (Branch decompositions of planar graphs [8, 12, 15, 16]). *Given a planar graph G, a branch decomposition $\mathbb{T}$ of G of minimal width can be computed in $\mathcal{O}(n^3)$ time. Furthermore, the computed branch decomposition $\mathbb{T}$ has width at most $2.122\sqrt{n}$, and for every non-leaf edge x in $\mathbb{T}$ the middle set partitions satisfy $|I| \leq 2$.*

3 Cut and Count and Branch Decompositions

In this section, we will discuss how to use 'Cut and Count' [5] on branch decompositions. We will illustrate this approach using the *unweighted* variant of the Steiner Tree problem. Our results on other problems use the same ideas, however these proofs are omit due to space restrictions.

The 'Cut and Count' technique of Cygan et al. [5] has two parts, the cut part and the count part. In the cut part, the problem is reformulated and transformed into a counting problem on consistently-cut candidate solutions where the connectivity constraint is relaxed. In the count part, this counting problem is solved using dynamic programming. In this paper, we summarise the cut part for Steiner Tree in Lemma 5 and refer to [5] for more details.

For a subset $X \subseteq V$, Cygan et al. [5] define a *consistent cut* of $G[X]$ to be a cut (X_1, X_2) such that there is no edge (u, v) in $G[X]$ with $u \in X_1$ and $v \in X_2$. Since we consider the unweighted version of Steiner Tree, we can let a solution be a subset of *vertices* $X \subseteq V$ such that $T \subseteq X$ and $G[X]$ is connected. A consistently-cut (possibly disconnected) candidate solution then is a pair $(X, (X_1, X_2))$ consisting of a candidate solution X and a consistent cut (X_1, X_2) of $G[X]$.

▶ **Lemma 5** (based on [5]). *Suppose we are given an algorithm* `Count` *that, given a graph G, a terminal set T, some fixed terminal $t_0 \in T$, and a weight function $w : V \to [0, ..., W]$, computes the values $A(i, w)$ defined below, for all $0 \leq i \leq k$ and $0 \leq w \leq kW$:*

$$A(i,w) = \left| \left\{ (X,(X_1,X_2)) \;\middle|\; \begin{array}{l} X \subseteq V, (X_1,X_2) \text{ a consistent cut of } G[X], \\ T \subseteq X, t_0 \in X_1, |X| = i, w(X) = w \end{array} \right\} \right| \pmod 2$$

Then, there exists a Monte-Carlo algorithm that solves Steiner Tree *on G, that cannot give false-positives and may give false negatives with probability at most $1/2$. The running time of this algorithm is dominated by the running time of the* `Count` *algorithm with $W = \mathcal{O}(n)$.*

We will omit the modulo two in the description of our counting algorithms and take the modulus afterwards, doing all computations modulo two requires slightly less time and space.

For easier exposition, we first prove the following theorem. Next, we will improve this using fast matrix multiplication in Theorem 7.

▶ **Theorem 6.** *There exist a Monte-Carlo algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw, solves* Steiner Tree *in time $\mathcal{O}(3^{\frac{3}{2}bw}n^{\mathcal{O}(1)})$.*

Proof. The result follows from Lemma 5 if we can give an algorithm that computes the required values $A(i,w)$ in $\mathcal{O}(3^{\frac{3}{2}bw}n^{\mathcal{O}(1)})$ time. We give this algorithm below.

We compute $A(i,w)$ by bottom-up dynamic programming on the branch decomposition $\mathbb{T}$. For each edge x of $\mathbb{T}$, we count partial-solution-cut pairs $(X,(X_1,X_2))$, where we call X a partial solution in $G[E_x]$ if all terminals in $G[E_x]$ are in X, and where the cut (X_1,X_2) is a consistent cut of the subgraph of $G[E_x]$ induced by X (i.e., a cut in $(G[E_x])[X]$) with additionally that if $t_0 \in X$ then $t_0 \in X_1$. To count these pairs, we define a labelling using labels 0, 1_1 and 1_2 on the vertices in the middle set B_x associated to an edge x of $\mathbb{T}$. These labels identify the situation of the vertex in a partial-solution-cut pair $(X,(X_1,X_2))$: label 0 means not in X, and labels 1_1 and 1_2 mean in X and on side X_1 and X_2 of the cut, respectively.

In a bottom-up fashion, we associate to each edge x of $\mathbb{T}$ a table $A_x(i,w,s)$ with entries for all $0 \le i \le k$, $0 \le w \le kW$, and $s \in \{0,1_1,1_2\}^{B_x}$. Such an entry $A_x(i,w,s)$ counts the number of partial-solution-cut pairs $(X,(X_1,X_2))$ as defined above that satisfy the constrains imposed by the states s on B_x and that satisfy $|X| = i$ and $w(X) = w$.

For a leaf edge x of the branch decomposition $\mathbb{T}$, we have that $B_x = \{u,v\}$ for some edge (u,v) in E. The table A_x associated to x can be filled as follows (all other entries are zero):

$$\begin{aligned}
A_x(0,0,0\ 0) &= 1[u \notin T \wedge v \notin T]\\
A_x(1,w(u),1_1\ 0) &= 1[v \notin T]\\
A_x(1,w(v),0\ 1_1) &= 1[u \notin T]\\
A_x(1,w(u),1_2\ 0) &= 1[u \neq t_0 \wedge v \notin T]\\
A_x(1,w(v),0\ 1_2) &= 1[u \notin T \wedge v \neq t_0]\\
A_x(2,w(u)+w(v),1_1\ 1_1) &= 1\\
A_x(2,w(u)+w(v),1_2\ 1_2) &= 1[u \neq t_0 \wedge v \neq t_0]
\end{aligned}$$

Here, we enforce that the cut is consistent, that every terminal $t \in T$ is in the partial solution X, that t_0 is on the correct side of the cut ($t_0 \in X_1$), and that $|X| = i$ and $w(X) = w$.

For an internal edge x of the branch decomposition $\mathbb{T}$ with children y and z, we fill the table A_x by combining the counted number of partial-solution-cut pairs from the tables for y and z. For this, we say that labellings s_x of B_x, s_y of B_y, and s_z of B_z are compatible if and only if $s_x^L = s_y^L \wedge s_x^R = s_z^R \wedge s_y^F = s_z^F \wedge s_x^I = s_y^I = s_z^I$ (where we denote by s_x^L the labelling s_x restricted to middle set partition L; for the middle set partitions see Definition 2).

We fill A_x by means of the following formula, where i^Z denotes the number of vertices with state 1 in middle set partition Z, and w^Z denotes the sum of the weights of the vertices with state 1 in middle set partition Z (for Z equals F, or I):

$$A_x(i_x,w_x,s_x) = \sum_{\substack{s_x,\,s_y,\,s_z\\ \text{compatable}\\ \text{labellings}}} \sum_{i_x = i_y + i_z - i^I - i^F} \sum_{w_x = w_y + w_z - w^I - w^F} A_y(i_y,w_y,s_y)\cdot A_z(i_z,w_z,s_z)\,.$$

This counts the total number of partial-solution-cut pairs $(X,(X_1,X_2))$ that satisfy the constraints as the summations combine all compatible entries from A_y and A_z and the multiplication combines the individual counts. To see that exactly these entries are compatible,

note that the consistency of the cut, the fact that $T \subseteq X$, and that $t_0 \in X_1$ are all enforced at the leaves and maintained by enforcing compatible labels. Furthermore, the partial-solution size i and weight w is the sum of both underlying partial solutions minus the doubling on the middle set partitions F and I.

By computing A_x for all edges in the branch decomposition $\mathbb{T}$ in the above way, we can find the required values $A(i, w)$ at the root edge r of $\mathbb{T}$ where $B_r = \emptyset$.

Consider the time required for computing table A_x. This table has at most $3^{|L|}3^{|R|}3^{|I|}k^2W$ entries, and for each entry we have to inspect at most $3^{|F|}k^2W$ combinations of entries from A_y and A_z, thus requiring $\mathcal{O}(3^{|L|+|R|+|I|+|F|}k^4n^2)$ time using $W = \mathcal{O}(n)$. This leads to a worst-case running time of $\mathcal{O}(3^{\frac{3}{2}bw}n^{\mathcal{O}(1)})$ under the constraints in Lemma 3. ◀

This result can be improved by using fast matrix multiplication similar to Dorn et al. [6].

▶ **Theorem 7.** *There exist a Monte-Carlo algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw, solves* Steiner Tree *in time $\mathcal{O}(3^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$, where ω is the matrix multiplication exponent.*

Proof. The algorithm is similar to the proof of Theorem 6, however, we evaluate the formula for the table $A_i(i_x, w_x, s_x)$ associated to an internal edge of the branch decomposition in a more efficient way. Instead of first fixing all labellings, we now first only fix compatible a labelling on I and fix i_x, i_y, w_x and w_y. Then, we can compute the contribution to $A_i(i_x, w_x, s_x)$, given the fixed values and fixed partial state, for all compatible states $s_x \in \{0, 1_1, 1_2\}^{B_x}$ using a single matrix multiplication.

To do so, we construct two matrices B and C. In matrix B there is a row for each labelling of L and a column for each labelling of F, and in matrix C there is a row for each labelling of F and a column for each labelling of R. As the labellings on I are fixed, each entry in B can be associated to a full labelling of the middle set B_y, and each entry in C can be associated to a complete labelling of the middle set B_z. Moreover, each entry in the matrix product BC can be associated to a full labelling of B_x, corresponding to the row of B (labelling of L) and column of C (labelling of R). If we fill matrix B with the corresponding values $A_y(i_y, w_y, s_y)$, and matrix C with the corresponding values $A_z(i_z, w_z, s_z)$ (note that we did not fix i_z and w_z, but these follow from all other fixed values and labellings), then matrix BC holds the contribution to $A_x(i_x, w_x, s_x)$ given the fixed labellings and values.

In the above way, we perform $3^{|I|}k^2W^2$ matrix multiplications of a $3^{|L|} \times 3^{|F|}$ matrix and a $3^{|F|} \times 3^{|R|}$ matrix. These rectangular matrices can be multiplied in $\mathcal{O}(3^{(\omega-1)|L|}3^{|F|}n^{\mathcal{O}(1)})$ time (see also [6, 13]), where we use that we can assume $|L| = |R|$ in a worst-case analysis for symmetry reasons. Under the constraints of Lemma 3, the worst-case arises when $|L| = |R| = |F| = \frac{1}{2}bw$ resulting in a running time of $\mathcal{O}(3^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$. ◀

▶ **Corollary 8.** *There exist a Monte-Carlo algorithm that, given a planar graph G, solves* Planar Steiner Tree *in time $\mathcal{O}(2^{3.991\sqrt{n}})$.*

Proof. Combine Theorem 7 with Lemma 4 and use $\omega < 2.373$ [14]. ◀

The other randomised results in Table 1 follow in a similar fashion and are omitted due to space restrictions. However, for some problems we need to use (generalised forms of) subset convolution to obtain the claimed time bounds. For the generalised subset convolution, we refer the reader to [18], and for an exposition on how to apply this in the setting of branch decompositions to [3].

4 Representative Sets and Branch Decompositions

In this section, we will discuss how to use the *rank-based approach* based on representative sets and Gaussian elimination [2] on branch decompositions. We will illustrate this approach using the *weighted* variant of the Steiner Tree problem. Our results on other problems use the same ideas, however the proofs are omitted due to space restrictions.

We need some definitions and notion regarding partitions. The set of all partitions of a set U is denoted by $\Pi(U)$. An element of a partition is also called a block. For $p \in \Pi(U)$, the term $|p|$ denotes the amount of blocks in the partition, where we let the empty partition in $\Pi(\emptyset)$ have zero blocks. For $p, q \in \Pi(U)$, $p \sqcup q$ is obtained from p and q by iteratively merging blocks in p that contain elements that are in the same block in q and vice versa. Also, $p \sqcap q$ is the partition that contains all blocks that are a non-empty intersection of a block in p and a block in q. If $X \subseteq U$, then $p_{\downarrow X} \in \Pi(X)$ is formed by removing all elements not in X from the partition p and possibly removing empty blocks. In the same way, if $U \subseteq X$, then $p_{\uparrow X} \in \Pi(X)$ is formed by adding a singleton to p for every element in $X \setminus U$.

A set of weighted partitions over U is a set $\mathcal{F} \subseteq (\Pi(U) \times \mathbb{N})$, i.e., a set of pairs consisting of a partition of U and a non-negative integer that is the weight of the partition. We use the following operators from [2] on a set of weighted partitions $\mathcal{F} \subseteq (\Pi(U) \times \mathbb{N})$:

- **Remove:** Define $\texttt{rmc}(\mathcal{F}) = \{(p, w) \in \mathcal{F} \mid \nexists (p, w') \in \mathcal{F} \wedge w' < w\}$. This operator removes non-minimal weight copies.
- **Union:** For $\mathcal{G} \subseteq (\Pi(U) \times \mathbb{N})$, define $\mathcal{F} \uplus \mathcal{G} = \texttt{rmc}(\mathcal{F} \cup \mathcal{G})$. This operator combines the two sets of weighted partitions and discards non-minimal weight copies.
- **Project:** For $X \subseteq U$, let $\bar{X} = U \setminus X$ and define $\texttt{proj}(X, \mathcal{F}) \subseteq \Pi(\bar{X}) \times \mathbb{N}$ as

$$\texttt{proj}(X, \mathcal{F}) = \texttt{rmc}(\{(p_{\downarrow \bar{X}}, w) \mid (p, w) \in \mathcal{F}, |p_{\downarrow \bar{X}}| = |p| \vee (X = \emptyset \wedge |p| = 1)\}) .$$

  This operator removes all elements from X from each partition and discards a partition if the amount of blocks in it decreases because of this, unless there is only one partition which is projected upon the empty set.
- **Join:** For a set U' and $\mathcal{G} \subseteq \Pi(U')$, let $\hat{U} = U \cup U'$, we define any pair of partitions $(p, w_1) \in \mathcal{F}$, $(q, w_2) \in \mathcal{G}$ to be compatible, unless (p, w_1) or (q, w_1) is the empty partition with non-zero weight. In that case, the pair is compatible, if and only if, the other partition is the empty partition with zero weight.
  Now we define $\texttt{join}(\mathcal{F}, \mathcal{G}) \subseteq (\Pi(\hat{U}) \times \mathbb{N})$ as:

$$\texttt{join}(\mathcal{F}, \mathcal{G}) = \texttt{rmc}(\{(p_{\uparrow \hat{U}} \sqcup q_{\uparrow \hat{U}}, w_1 + w_2) \mid (p, w_1) \in \mathcal{F}, (q, w_2) \in \mathcal{G} \text{ compatible}\}) .$$

  This operator extends all partitions to the same base set $\hat{U}$. It then combines each compatible pair of partitions by means of the $\sqcup$ operator and assigns the sum of the weights as a new weight. We will need pairs to be compatible to keep solutions connected.

We will start by giving a naive algorithm for weighted Steiner Tree on branch decompositions. Thereafter, we will show how to use representative sets and Gaussian elimination to improve the time complexity. We note that, different from Section 3, we now let (partial) solutions be sets of *edges* connecting the terminals T.

The Naive Algorithm for Steiner Tree on Branch Decompositions

In a bottom-up fashion, the naive algorithm computes a table A_x for each edge x of the branch decomposition $\mathbb{T}$. This table keeps track of all possible partial solutions $Y \subseteq E_x$ on $G[E_x]$ that can be extended to a minimal weight solution on G. These partial solutions

are subsets $Y \subseteq E_x$ such that all terminals in $G[E_x]$ are incident to an edge in Y, and all connected components in $G[Y]$ either contain an edge incident to B_x or connect all terminals T in G.

Each entry $A_x(s)$ in the table is indexed by a labelling $s \in \{0,1\}^{B_x}$ on the vertices in B_x and contains a *set* of weighted partitions. The label 1 means that the vertex will be incident to the solution edge set, which is the case when the vertex is a terminal or when the vertex is incident to an edge in the partial solution Y. The label 0 means that it will not be incident to the solution edge set. The set of weighted partitions $A_x(s)$ is a set of weighted partitions on all vertices with label 1 in s. $A_x(s)$ represents all partial solutions on $G[E_x]$ consistent with the labelling s in the following way: the weight of the partition corresponds to the weight of the partial-solution Y; and vertices are in the same block of the partition p that represents that solution Y, if and only if, the vertices are in the same connected component in $G[Y]$.

For a leaf edge x of the branch decomposition $\mathbb{T}$, we have that $B_x = \{u, v\}$ for an edge (u, v) in E. The table A_x associated to x can be filled as follows:

$$A(0\ 0) = \{(\emptyset, 0)\}[u \notin T \wedge v \notin T]$$
$$A(1\ 0) = \{(\{\{u\}\}, 0)\}[v \notin T]$$
$$A(0\ 1) = \{(\{\{v\}\}, 0)\}[u \notin T]$$
$$A(1\ 1) = \{(\{\{u\}, \{v\}\}, 0), \{\{u\ v\}\}, w((u, v)))\}$$

Here, we make sure that terminal vertices in T correspond to 1 labels, and that vertices incident to an edge in the partial solution correspond to 1 labels. We also make sure that the partition corresponds to the connected components on the vertices with a 1 label, and that the weight of the partition equals the weight of the partial solution.

For an internal edge x of the branch decomposition $\mathbb{T}$ with children y and z, we fill the table A_x by means of the following formula:

$$A_x(s_x) = \biguplus_{s_F \in \{0,1\}^F} \texttt{proj}\big(F, \texttt{join}(A_y(s_x^L s_x^I s_F), A_z(s_x^R s_x^I s_F))\big)\,.$$

Here $s_x^L s_x^I s_F$ stands for the concatenation of the labelling s_x restricted to L, the labelling s_x restricted to I, and the labelling s_F on F (note that this gives a valid labelling on B_y).

For every labelling s_F on F, the above formula combines all entries with compatible weighted partitions from $A_y(s_x^L s_x^I s_F)$ and $A_z(s_x^R s_x^I s_F)$. Partitions in the computed set $A_x(s)$ now correspond to the connected components of the partial solution, by definition of the `join` and `proj` operations. This is because, the resulting entries in which vertices in F are in separate blocks (separate connected components in $G[E_x]$) are discarded by the project operation. Also, we require compatible weighted partitions in the join operation to make sure that no connected components that do not contain vertices in B_x are combined, i.e., these do not result in new non-empty partitions. The weights of the partitions in $A_x(s)$ correspond to the weights of the partial solutions, as we choose edges from G in a partial solution in leaf edges of the branch decomposition and the `join` operation sums up the weights.

By computing A_x for all edges in the branch decomposition $\mathbb{T}$ in the above way, we can find the weight of the minimum weight solution to Steiner Tree at the root edge r of $\mathbb{T}$ where $B_r = \emptyset$ as the weight of the empty partition.

Using Representative Sets

The essence of the rank-based approach lies in the `Reduce` procedure from [2]. This procedure reduces the size of the tables used in the dynamic program without loss of representation.

▶ **Definition 9** (Representation [2]). For sets of weighted partitions $\mathcal{F}, \mathcal{F}' \subseteq (\Pi(U) \times \mathbb{N})$ and a partition $q \in \Pi(U)$, define:

$$\texttt{opt}(q, \mathcal{F}) = \min\{w \mid (p, w) \in \mathcal{F} \wedge p \sqcup q = \{U\}\}.$$

We say that $\mathcal{F}'$ *represents* $\mathcal{F}$, if for all $q \in \Pi(U)$, it is the case that $\texttt{opt}(q, \mathcal{F}') = \texttt{opt}(q, \mathcal{F})$.

▶ **Theorem 10** ([2]). *There exists an algorithm* `Reduce` *that, given a set of weighted partitions $\mathcal{F} \subseteq (\Pi(U) \times \mathbb{N})$, outputs a set of weighted partitions $\mathcal{F}' \subseteq \mathcal{F}$, such that $\mathcal{F}'$ represents $\mathcal{F}$ and $|\mathcal{F}'| \leq 2^{|U|-1}$, in $\mathcal{O}(|\mathcal{F}|2^{(\omega-1)|U|}|U|^{\mathcal{O}(1)})$ time.*

We apply the above theorem at each step of the naive algorithm for Steiner Tree and carefully analyse the resulting running time to obtain the following result.

▶ **Theorem 11.** *There exist an algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw, solves* Steiner Tree *in time $\mathcal{O}(n((1+2^\omega)\sqrt{5})^{bw} bw^{\mathcal{O}(1)})$.*

Proof. The algorithm computes the tables A_x in a bottom-up fashion over the branch decomposition $\mathbb{T}$ according to the formulae in the description of the naive algorithm. Directly after the algorithm finishes computing a table A_x for any edge x in the branch decomposition, the `Reduce` algorithm is applied to each entry $A_x(s_x)$ of the table to control the sizes of the sets of weighted partitions. Because the naive algorithm is correct and the `Reduce` procedure maintains representation (Theorem 10), we conclude that the new algorithm is correct also.

To prove the running time, consider a non-leaf edge x in the branch decomposition $\mathbb{T}$ with left child y and right child z. The operations in the naive algorithm used to compute, for a labelling $s_x \in \{0,1\}^{B_x}$, the set of weighted partitions $A_x(s_x)$ can be implemented in $\mathcal{O}(bw^{\mathcal{O}(1)})$ time times the number of combinations of entries from A_y and A_z involved. This can be done using the straightforward implementations (see also [2]). As each combination of entries from A_y and A_z can lead to an entry in $A_x(s_x)$ before the `Reduce` step is applied, the running time is dominated by the time required by the `Reduce` algorithm.

For a fixed $s_x \in \{0,1\}^{B_x}$, let j be the amount of vertices in s_x with label 1, which we will also denote by $j = |s_x^{-1}(1)|$. For the set of weighted partitions $A_x(s_x)$, `Reduce` takes time:

$$\mathcal{O}\big(|A_x(s_x)|2^{(\omega-1)j} j^{\mathcal{O}(1)}\big).$$

The size of $A_x(s_x)$ is the result of combining, for every labelling $s_F \in \{0,1\}^F$, every entry of $A_y(s_x^L s_x^I s_F)$ with every entry of $A_z(s_x^R s_x^I s_F)$. Using $s_y = s_x^L s_x^I s_F$ and $s_z = s_x^R s_x^I s_F$, the sizes of $A_y(s_y)$ and $A_z(s_z)$ are bounded by $2^{|s_y^{-1}(1)|}$ and $2^{|s_z^{-1}(1)|}$, respectively, since these table were reduced after computing A_y and A_z. Therefore, the total time it takes to reduce the sets of partitions for all entries in $A_x$is:

$$\mathcal{O}\Big(\sum_{j=0}^{|I\cup R\cup L|} \binom{|I \cup R \cup L|}{j} 2^{(\omega-1)j} |A_x(s_j)| j^{\mathcal{O}(1)}\Big).$$

The sum and the binomial coefficient consider all possible labellings using j for the number of 1 labels. This is the only information needed about the labellings. As such, we will slightly abuse notation and denote any labelling with j vertices with label 1 as s_j. Also, we will denote by $s_{i,l,f}$ any labelling with i vertices with label 1 on I, l vertices with label 1 on L, and f vertices with label 1 on F.

We can now expand the sum, differentiating between I, L and R, and use that $A_y(s_y)$ and $A_z(s_z)$ are bounded by $2^{|s_y^{-1}(1)|}$ and $2^{|s_z^{-1}(1)|}$, respectively:

$$\mathcal{O}\Big(\sum_{j=0}^{|I\cup R\cup L|}\binom{|I\cup R\cup L|}{j}2^{(\omega-1)j}|A_x(s_j)|j^{\mathcal{O}(1)}\Big) =$$
$$\mathcal{O}\Big(\sum_{i=0}^{|I|}\sum_{r=0}^{|R|}\sum_{l=0}^{|L|}\binom{|I|}{i}\binom{|R|}{r}\binom{|L|}{l}2^{(\omega-1)(i+r+l)}|A_x(s_{i,r,l})|(i+r+l)^{\mathcal{O}(1)}\Big) =$$
$$\mathcal{O}\Big(\sum_{i=0}^{|I|}\sum_{r=0}^{|R|}\sum_{l=0}^{|L|}\binom{|I|}{i}\binom{|R|}{r}\binom{|L|}{l}2^{(\omega-1)(i+r+l)}\sum_{f=0}^{|F|}\binom{|F|}{f}|A_y(s_{i,l,f})||A_z(s_{i,r,f})|bw^{\mathcal{O}(1)}\Big) \leq$$
$$\mathcal{O}\Big(\sum_{i=0}^{|I|}\sum_{r=0}^{|R|}\sum_{l=0}^{|L|}\binom{|I|}{i}\binom{|R|}{r}\binom{|L|}{l}2^{(\omega-1)(i+r+l)}\sum_{f=0}^{|F|}\binom{|F|}{f}2^{i+l+f}2^{i+r+f}bw^{\mathcal{O}(1)}\Big).$$

Next, we rearrange the terms and repeatedly apply the binomial theorem to obtain a more simple expression:

$$\mathcal{O}\Big(\sum_{i=0}^{|I|}\binom{|I|}{i}2^{(\omega+1)i}\sum_{r=0}^{|R|}\binom{|R|}{r}2^{\omega r}\sum_{l=0}^{|L|}\binom{|L|}{l}2^{\omega l}\sum_{f=0}^{|F|}\binom{|F|}{f}2^{2f}bw\Big)^{\mathcal{O}(1)}\Big) \leq$$
$$\mathcal{O}\big((1+2^{\omega+1})^{|I|}(1+2^{\omega})^{|R|}(1+2^{\omega})^{|L|}5^{|F|}bw^{\mathcal{O}(1)}\big).$$

If we maximize this under the constraints in Lemma 3, then we find a worst-case running time of:

$$\mathcal{O}(((1+2^{\omega})\sqrt{5})^{bw}bw^{\mathcal{O}(1)}).$$

In this case $|R| = |L| = |F| = \frac{1}{2}bw$ and $|I| = 0$. Taking into consideration that this must be done for every edge in the branch decomposition, we find the time-complexity from the statement of theorem. ◀

The other deterministic result in Table 1 follow in a similar fashion but are omitted due to space restrictions. These omitted proofs, use besides (generalised forms of) subset convolution, also the fact that the `Reduce` procedure can be modified to output a set of weighted partitions of size at most $2^{|U|/2}$ in case of the Hamilton Cycle and TSP problems [4].

5 Conclusion

In this paper, we have shown two things. First of all, we have shown that cut and count and the rank-based approach can be used not only on tree decompositions but also on branch decompositions. This means the techniques are more powerful than they were known to be. Perhaps these techniques can also be used in combination with other width measures.

We have also given fast algorithms, especially on planar graphs, for several connectivity problems. These algorithms use branch decompositions and therefore affirm the use of this type of decomposition as a solid foundation for algorithms.

References

1 Hans L. Bodlaender. Treewidth: Structure and algorithms. In *Proceedings of the 14th International Colloquium on Structural Information and Communication Complexity, SIROCCO 2007*, volume 4474 of *Lecture Notes in Computer Science*, pages 11–25. Springer Verlag, 2007.

2 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015.

3 Hans L. Bodlaender, Erik Jan van Leeuwen, Johan M. M. van Rooij, and Martin Vatshelle. Faster algorithms on branch and clique decompositions. In *Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science, MFCS 2010*, volume 6281 of *Lecture Notes in Computer Science*, pages 174–185. Springer Verlag, 2010.

4 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast Hamiltonicity checking via bases of perfect matchings. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 301–310. ACM, 2013.

5 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 *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159, 2011.

6 Frederic Dorn. Dynamic programming and fast matrix multiplication. In *Proceedings of the 14th Annual European Symposium on Algorithms, ESA 2006*, volume 4168 of *Lecture Notes in Computer Science*, pages 280–291. Springer Verlag, 2006.

7 Frederic Dorn. *Designing Subexponential Algorithms: Problems, Techniques & Structures.* PhD thesis, Institutt for informatikk, Universitetet i Bergen, 2007.

8 Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. Efficient exact algorithms on planar graphs: exploiting sphere cut decompositions. *Algorithmica*, 58:790–810, 2010.

9 Stefan Fafianie, Hans L. Bodlaender, and Jesper Nederlof. Speeding up dynamic programming with representative sets: an experimental evaluation of algorithms for steiner tree on tree decompositions. *Algorithmica*, 71(3):636–660, 2015.

10 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Representative sets of product families. In *Algorithms – ESA 2014 – 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 443–454. Springer, 2014.

11 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 142–151, 2014.

12 Fedor V. Fomin and Dimitrios M. Thilikos. New upper bounds on the decomposability of planar graphs. *Journal of Graph Theory*, 51:53–81, 2006.

13 François Le Gall. Faster algorithms for rectangular matrix multiplication. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 514–523. IEEE Computer Society, 2012.

14 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC*, pages 296–303, 2014.

15 Qian-Ping Gu and Hisao Tamaki. Optimal branch-decomposition of planar graphs in $O(n^3)$ time. *ACM Trans. Algorithms*, 4(3), 2008.

16 Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

17 Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Discr. Math.*, 10:529–550, 1997.

18 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Proceedings of the 17th Annual European Symposium on Algorithms, ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer Verlag, 2009.