Universiteit Utrecht

Graduate School of Natural Sciences
Faculty of Science

Master's thesis *

Cut and Count and Representative Sets on Branch Decompositions

Author:
W.J.A. Pino

Supervisors:
dr. J.M.M. van Rooij
prof. dr. H.L. Bodlaender

August 2016

ICA-3463524

* A thesis submitted to the faculty in partial fulfilment of the requirements for the degree of Master of Science in the Department of Computing Science in the Graduate School of Natural Sciences.

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 randomized and deterministic algorithms that are single exponential in the treewidth, and respectively polynomial and linear in the number of vertices. In this thesis, these methods are adapted to branch decompositions. This yields algorithms, both randomised and deterministic, that are in many cases faster than when tree decompositions would be used.

In particular, the currently fastest randomised algorithms for several problems on planar graphs are obtained. When the involved weights are $\mathcal{O}(n^{\mathcal{O}(1)})$, this thesis presents 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, faster deterministic algorithms for all four mentioned problems are obtained.

Contents

1 Introduction

$\mathcal{NP}$-complete problems are a source of much trouble for computer scientists. When a problem is in $\mathcal{NP}$ this means that, unless $\mathcal{P} = \mathcal{NP}$, there does not exist a polynomial-time algorithm that will solve the problem. Since almost everyone believes the answer to the million dollar question (literally [9]) 'is $\mathcal{P} = \mathcal{NP}$?' to be negative, it is very likely that only exponential-time algorithms exist for problems in $\mathcal{NP}$. This is problematic because the worst-case time it takes for an exponential-time algorithm to solve a problem increases very fast as the size of the instance increases, often making these algorithms useless for most practical instances. This being said, there are often approximation algorithms that can be useful here.

Because a lot of interesting problems are $\mathcal{NP}$-complete, a solution has to be found for this issue. A possible approach lies in parameterized problems. The idea is that although a problem of size n can only be solved in time $\mathcal{O}(f(n))$ where f is an exponential function, it might be that the problem becomes tractable when it is parameterized. This means that for a parameter k the running time becomes $\mathcal{O}(f(k)poly(n))$, where f again is an exponential function and *poly* is a polynomial function. This means that, although the whole algorithm is still exponential, the algorithm has polynomial running time when k is fixed. A problem with such an algorithm is called *Fixed Parameter Tractable* (FPT) [11].

For problems involving graphs, useful parameters that often lead to FPT-algorithms are width parameters. An often used width measure is treewidth, described by Robertson and Seymour in their seminal work on graph minors [28]. Treewidth can be seen as the degree to which a graph resembles a tree; a formal definition will be given in Section 2. Treewidth and the corresponding tree decompositions have led to many fast algorithms for a variety of problems [2].

Solving a problem using a width measure consists of two steps. First of all, a decomposition of small or minimum width has to be found. Then this decomposition is used to solve the problem using dynamic programming. The focus of this paper lies on the second step. The intuition in the dynamic programming routine is that a set of partial solutions is maintained for some part of the graph that can be combined with partial solutions on the other parts to find a solution to the complete problem. Which set needs to be maintained and how this set is updated when a new part of the graph is considered are the essential questions when designing an dynamic programming algorithm on decompositions.

Treewidth is a width measure that has been studied extensively. For a good overview see [5]. Finding a tree decomposition is $\mathcal{NP}$-hard [4]. If the treewidth is fixed, then a decomposition can be found in linear time [3], but the constants in the algorithm given are so large that the approach is not practically feasible. For some types of graphs a decomposition can be found fast [4]. However, a notable open problem is whether there is an polynomial time algorithm that finds the treewidth of a planar graph.

Branchwidth is another well studied graph parameter, with strong relations to treewidth. The branchwidth, bw, and treewidth, tw, 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 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. This difference in the base of the exponent is what this thesis focusses on.

There are several other width measures besides treewidth and branchwidth such as cliquewidth, rankwidth and boolean width [10, 26, 8]. However, these are considered to be out of scope for this thesis.

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 linear in k and single exponential in the number of vertices n, e.g. see [30]. 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. [13] with the '*Cut and Count*' method; this approach gives fast randomized algorithms that are single exponential in the treewidth and polynomial in the number of vertices for various problems, e.g. Steiner Tree, Feedback Vertex Set, Hamiltonian Circuit, TSP, Connected Dominating Set. The constants in the base of the exponential factors in these algorithms are optimal [24, 13]. At the cost of a higher constant in the base of the exponential factor, Bodlaender et al. [6] gave deterministic algorithms that are single-exponential in the treewidth and linear in the number of vertices for these connectivity problems. This technique, based on representative sets and Gaussian elimination, is called the *rank-based* approach. This algorithm was experimentally evaluated by Fafianie et al. [17], 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. [19]. Later, Fomin et al. [18] 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.

In this thesis, it is shown that 'Cut and Count' and the rank-based approach can be used directly on branch decompositions. As a result, in several cases, improvements compared to using tree decompositions are obtained. For an overview of our results, see Table 1.

Fast matrix multiplications and subset convolutions are two other techniques to speed up dynamic programming algorithms on tree and branch decompositions. Dorn [14] showed how to use *matrix multiplication* to speed up algorithms on branch decompositions and van Rooij et al. [7, 31] showed how to speed up algorithms on tree, branch and clique decompositions using *(generalised) subset convolutions*. In this thesis these works are build upon, applying these techniques where possible.

For a comparison of the results in this thesis to the current best treewidth algorithms, see Table 2 and Table 3. Here, $\omega < 2.373$ [21] is the matrix multiplication exponent. Our branch decomposition based results improve known treewidth results for parts of

Problem	Randomized	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	$\mathcal{O}(2^{5.036\sqrt{n}})$	$\mathcal{O}(2^{6.570\sqrt{n}})$

Table 1: Our results using the 'Cut and Count' (randomized) and rank-based (deterministic) techniques.

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 2: Comparison of our results with best known results on treewidth [13] for randomized algorithms on problems where the weights are $\mathcal{O}(n^{\mathcal{O}(1)})$.

the range $bw \leq tw+1 \leq \lfloor \frac{3}{2}bw \rfloor$ (note that $\frac{\omega}{2} < \frac{3}{2}$). Since the constants in the base of the treewidth algorithms in Table 4 are optimal, if $\omega = 2$, the constants in the branchwidth algorithms are also optimal. In case of deterministic algorithms for TSP with 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 [20] (the ratcatcher algorithm can also

Problem	Treewidth	Branchwidth
Steiner Tree	$\mathcal{O}(n2^{3.134\ tw})$ [18]	$\mathcal{O}(n2^{3.790\ bw})$
Connected Dominating Set	$\mathcal{O}(n2^{3.628\ tw})$ [6]	$\mathcal{O}(n2^{4.137\ bw})$
Feedback Vertex Set	$\mathcal{O}(n2^{3.134\ tw})$ [18]	$\mathcal{O}(n2^{3.790\ bw})$
Hamilton Cycle / TSP	$\mathcal{O}(n2^{3.257\ tw})$ [6]	$\mathcal{O}(n2^{3.257\ bw})$

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

Problem	Dorn ($n^{\mathcal{O}(1)}$)	Dorn ($\mathbb{R}$)	Randomized	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	$\mathcal{O}(2^{8.15\sqrt{n}})$	$\mathcal{O}(2^{9.86\sqrt{n}})$	$\mathcal{O}(2^{5.036\sqrt{n}})$	$\mathcal{O}(2^{5.63\sqrt{n}})$

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 [14] when weights are $\mathcal{O}(n^{\mathcal{O}(1)})$; the column 'Dorn ($\mathbb{R}$)' states deterministic results by Dorn [15] for arbitrary real weights; the column 'Randomized' 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.
Note that the mentioned results by Dorn [14, 15] have not been adjusted for the recently slightly improved matrix multiplication constant ω [21].

be used, it exactly computes the branchwidth of planar graphs in $\mathcal{O}(n^3)$ time [22, 29]), the algorithms in this thesis can be applied 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 [14, 15]. When considering randomised algorithms, the currently fastest algorithms for all considered problems are improved when weights are bounded by $\mathcal{O}(n^{\mathcal{O}(1)})$. When considering deterministic algorithms, the currently fastest algorithms for all considered problems with arbitrary real weights are improved, and the currently fastest algorithm for Hamilton Cycle and TSP is improved when weights are bounded by $\mathcal{O}(n^{\mathcal{O}(1)})$.

The results in this thesis are to be presented at the International Symposium on Parameterized and Exact Computation (IPEC). This means that there is overlap between this thesis and the paper in the proceedings of the conference [27].

2 Preliminaries

2.1 Graphs, Sets and Partitions

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 $X_1 \cap X_2 = \emptyset$. The number of connected components in a graph G is denoted by $\mathtt{cc}(G)$.

In a rooted tree, node y is a descendant of a node x when x is encountered when traveling from y to the root. A node is also a descendant of itself.

The open neighborhood of a vertex $v \in V$ is defined by $N(v) = \{u \in V | \{u, v\} \in E\}$ and the closed neighborhood of that vertex by $N[v] = N(v) \cup \{v\}$. The closed and open neighborhoods of a set of vertices $X \subseteq V$ are denoted by $N[X] = \bigcup_{v \in X} N[v]$ and

$N(X) = N[X]/X$ respectively. The powerset of a set S is denoted by 2^S. A weight function $w : U \to \mathbb{Z}$ applied to a set X is defined as $w(X) = \sum_{x \in X} w(x)$.

If two numbers a and b are the same modulo 2, i.e. a is odd iff b is odd, this is denoted by $a \equiv b$.

Throughout the paper the Iverson bracket is used. This notation denotes a 1 if the condition between brackets is satisfied and 0 otherwise, e.g. $[1 = 1]42 = 42$ and $[1 = 2]42 = 0$. To be exact:

$$[P] = \begin{cases} 1, & \text{if } P = \text{True} \\ 0, & \text{otherwise} \end{cases}$$

This notation is also used in combination with sets S, then this denotes $[True]S = S$ and $[False]S = \emptyset$.

For a labelling s on a set of nodes X, $s[v \to a]$ denotes a labelling where node $v \in X$ has label a.

Some definitions and notions regarding partitions are also needed. Consider a set U. The set of all partitions of 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. 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 $X \setminus U$. The term $U[X]$ denotes a partition where all elements are singletons except for the elements in X which form one block. 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.

2.2 Width Measures

For each width measure the definition and a framework for constructing dynamic programming algorithms on the decompositions will be given. As mentioned before, the first step of finding a decomposition is considered out of scope and because of this decompositions are treated as given throughout the thesis.

2.2.1 Treewidth

Definition 2.1 (Tree decomposition)**.** *A tree decomposition of a graph $G(V, E)$ is a tree $\mathbb{T}$ where every node $x \in \mathbb{T}$ is associated with a set of vertices $B_x \subseteq V$ (a bag) for which it holds that $\bigcup_{x \in \mathbb{T}} B_x = V$ and with the following properties:*

- *for any edge $(u, v) \in E$ there exists a node $x \in \mathbb{T}$ such that $\{u, v\} \subseteq B_x$.*
- *if $v \in B_x$ and $v \in B_y$ then $v \in B_z$ for every node z on the path from node x to node y in $\mathbb{T}$.*

The width of a tree decomposition is the size of the largest bag minus one. The treewidth of a graph is the minimum width over all tree decompositions. There exist several other equivalent definitions [4].

When writing a dynamic programming algorithm for a tree decomposition it is convenient if the tree decomposition has a certain structure. This is why the notion of a *nice* tree decomposition is introduced. The definition given here is slightly different than the original definition [23] but has been used before [13].

Definition 2.2 (Nice tree decomposition)**.** *A nice tree decomposition is a tree decomposition $\mathbb{T}$ with a root where each of the bags B_x is of one of the following types:*

- ***Leaf bag**: a leaf x of T with $B_x = \emptyset$.*
- ***Introduce vertex bag**: an internal vertex x of T with one child vertex y for which $B_x = B_y \cup \{v\}$ for some vertex $v \notin B_y$. The bag is said to* introduce *vertex v.*
- ***Introduce edge bag**: an internal vertex x of T labeled with an edge $(u, v) \in E$ with one child vertex y for which $u, v \in B_x = B_y$. The bag is said to* introduce *edge (u, v).*
- ***Forget bag**: an internal vertex x of T with one child vertex y for which $B_x = B_y / \{v\}$ for some vertex $v \in B_y$. The bag is said to* forget *vertex v.*
- ***Join bag**: an internal vertex x of T with two child vertices l and r with $B_x = B_l = B_r$.*

Additionally require that every edge is introduced exactly once.

In [6] it is shown that, given a tree decomposition, a nice tree decomposition of the same width can be found in $\mathcal{O}(n\ tw^{\mathcal{O}(1)})$ time. Once a nice tree decomposition is found, the next step is to formulate how to handle each of the five types of bags and show that this leads to the solution at the root node. When working with tree decompositions, it is convenient to define for each node $x \in \mathbb{T}$ the node set $V_x = \bigcup_{y \text{ descendant of } x} B_y$ (remember that x is its own descendant). In a slight abuse of notation, $G[V_x]$ includes only edges that are already introduced.

2.2.2 Branchwidth

While branchwidth is similar to treewidth with respect to algorithmic complexity results, the definition is quite different.

Definition 2.3 (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.

The removal of an edge in a branch decomposition $\mathbb{T}$ results in a cut in G. That is, every edge x in $\mathbb{T}$ 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, only *rooted* branch decompositions are considered. 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, it is possible to 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 $\mathbb{T}$, it is possible to 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. Because there are only two types of edges, i.e. leaf edges and internal edges, only two types of operations have to be specified in a dynamic programming algorithm.

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 2.4 (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. Now define the following partitioning of $B_x \cup B_y \cup B_z$:*

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

This partitioning is illustrated in Figure 1. Note that $I \cup F \cup L \cup R = B_x \cup B_y \cup B_z$ since every node in a middle set is in at least one other middle set.

Lemma 2.5 (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 $\mathbb{T}$:*

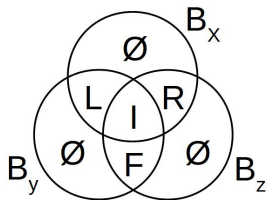

Figure 1: The partitioning of the middle sets

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

Finally, to obtain the results on planar graphs, the following lemma is needed that relates planar graphs to branch decompositions:

Lemma 2.6 (Branch decompositions of planar graphs [16, 20, 22, 29])**.** *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 Algorithmic Techniques

In this section the various techniques relevant for this thesis will be discussed. To facilitate understanding the techniques will be accompanied with examples. 'Cut and Count' and the rank based approach will be explained by means of Steiner Tree. Since they had not been defined on branch decompositions before, they will be explained using tree decompositions. Fast subset convolutions and fast matrix multiplication will be explained using Dominating Set. Because they have both been used to speed up dynamic programming algorithms on branch decompositions and are used only in this capacity in this thesis they shall be defined on branch decompositions.

3.1 Cut and Count

3.1.1 Theory

'Cut and Count' is quite abstract when not applied to a problem so it is advised to read the example when a first reading of the theory is not sufficiently clear.

The 'Cut and Count' technique of Cygan et al. [13] 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. This is done in three steps: first requirements on the set of solutions S are relaxed to find a set of candidate solutions R, then this set R is coupled with consistent cuts to find a set $\mathcal{C}$ of candidate solution-cut pairs, finally it is shown that $|S| \equiv |\mathcal{C}|$. In the count part, $|\mathcal{C}|$ is counted using dynamic programming.

Consider a set of solutions $S \subseteq 2^U$ over a universe U. The question is whether there is a solution, i.e. whether S is non-empty. The first step in the cut part is done by

relaxing the connectivity requirement to obtain a set of candidate solutions $R \supseteq S$. For the second step, define the set $\mathcal{C} = (X, C)$ where $X \in R$ and C is a consistent cut of X. The exact definition of a consistent cut depends on the problem at hand. This set $\mathcal{C}$ is counted in the count part. If the relaxation is done properly then $|\mathcal{C}| \equiv |S|$. For this third step it suffices to ensure that a potential solution $X \in R$ has an odd amount of consistent cuts associated with it if $X \notin S$ and an even amount if $X \in S$. By counting $\mathcal{C}$ it is now possible to determine whether S is non-empty, since solutions $x \in R \setminus S$ cancel because they have an even number of consistent cuts associated to them and only solutions $x \in S$ remain.

The problem is that when $|S|$ is even the solutions cancel each other. To avoid this, introduce a weight function $w(u)$ that assigns a weight to each $u \in U$ as specified in Lemma 3.2. The lemma can now be used to ensure that, if there is a solution, this solution has a unique weight with probability $\geq \frac{1}{2}$. This is done by setting $\mathcal{H} = S$ and $M = 2n$. Since $|\mathcal{C}| \equiv |S|$, for each weight W, $|\{(X, C) \in \mathcal{C} \mid w(X) = W\}| \equiv |\{X \in S \mid w(X) = W\}|$. With these insights it is possible to construct an algorithm [13], which returns **no** if $S = \emptyset$ and **yes** with probability $\geq \frac{1}{2}$ otherwise. `Count` is a subroutine that counts $\mathcal{C} \mod 2$.

Definition 3.1. *A function* $w : U \to \mathbb{Z}$ isolates *a set family* $\mathcal{H} \subseteq 2^U$ *if there is a unique* $H' \in \mathcal{H}$ *with* $w(H') = \min_{H \in \mathcal{H}} w(H)$.

Lemma 3.2 (Isolation Lemma [25])**.** *Let* $\mathcal{H} \subseteq 2^U$ *be a set family over a universe* U *with* $|\mathcal{H}| > 0$. *For each* $u \in U$, *choose a weight* $w(u) \in \{1, 2, \ldots, M\}$ *uniformly at random. Then* $\Pr[w$ *isolates* $\mathcal{H}] \geq 1 - \frac{|U|}{M}$.

Algorithm 1: CUTANDCOUNT(U,$\mathbb{T}$,COUNT)

Input : Set U; tree decomposition $\mathbb{T}$; Procedure COUNT with arguments $w : U \to \mathbb{Z}$, $W \in \mathbb{Z}$, and $\mathbb{T}$.
Output: *no* if no solution, *yes* with $\Pr \frac{1}{2}$ otherwise

```
foreach v ∈ U do
    Choose w(v) ∈ {1, 2, ..., 2|U|} uniformly at random;
foreach 0 ≤ W ≤ 2|U|^2 do
    if CountC(w, W, T) ≡ 1 then return yes;
return no
```

The runtime of the algorithm is dominated by the `Count` routine.

3.1.2 Example

STEINER TREE
Input: An undirected graph $G(V, E)$, a set of terminals $T \subseteq V$ and an integer k.
Question: Does there exist a set $X \subseteq V$ such that $T \subseteq X$, $|X| \leq k$, and $G[X]$ is connected?

For a subset $X \subseteq V$, Cygan et al. [13] 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 $v \in X_2$. This means that a consistent cut of $G[X]$ does not go through any connected component of $G[X]$. Since the unweighted version of Steiner Tree is considered, it is possible to 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 $G[X]$.

The relaxation is quite straightforward: a solution X is in R if it satisfies all requirements mentioned in the problem except for the requirement that $G[X]$ is connected. Define $R_W \subseteq R$ to be set of solutions for the relaxed problem with weight W. Define S_W and $\mathcal{C}_W$ in the same manner.

The set $\mathcal{C} = (X, (X_1, X_2))$ consists of candidate solutions $X \in R$ and consistent cuts (X_1, X_2) of $G[X]$ with an arbitrarily chosen vertex $v_0 \in V$ always in X_1. Since this vertex is arbitrarily chosen it is possible to require that $v_0 = t_0 \in T$. It is necessary to show that $|\mathcal{C}_W| \equiv |S_W|$ for each W.

Lemma 3.3. *Given a graph $G(V, E)$ let X be a subset of vertices such that $t_0 \in X \subseteq V$. The amount of consistently cut subgraphs $(X, (X_1, X_2))$ such that $t_0 \in X_1$ is $2^{\mathrm{CC}(G[X])-1}$.*

Proof. From the definition of a consistently cut subgraph it is known that a connected component of $G[X]$ is either in X_1 or in X_2. For the connected component with t_0 in it there is only one choice; it is in X_1. All other connected components can be on either side of the cut. This gives $2^{\mathrm{CC}(G[X])-1}$ options leading to as many entries in $\mathcal{C}$ for every X. □

Now it is possible to prove the original assertion:

Lemma 3.4. *Let G, w, $\mathcal{C}_W$, and S_W as defined above. Then for every W, $|\mathcal{C}_W| \equiv |S_W|$*

Proof. By the previous lemma it is known that $|\mathcal{C}_W| \equiv \sum_{X \in R_W} 2^{\mathrm{CC}(G[X])-1}$. Thus $|\mathcal{C}_W| \equiv |\{X \in R_W | \mathrm{CC}(G[X]) = 1\}| = |S_W|$. □

The theory from the previous section combined with the lemmas above result in the following:

Lemma 3.5. *Suppose an algorithm* `Count` *is given 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 $W = \mathcal{O}(n)$.*

Note that the `Count` algorithm described in the lemma does exactly what is required: it counts $\mathcal{C}_W$ for each W. For simplicity, the modulus is omitted in the description of our counting algorithms and is taken afterwards. Doing all computations modulo two requires slightly less time and space.

Theorem 3.6. *There exist a Monte-Carlo algorithm that, given a graph G and a tree decomposition $\mathbb{T}$ of G of width tw, solves* Steiner Tree *in time $\mathcal{O}(3^{tw}n^{\mathcal{O}(1)})$.*

Proof. The result follows from Lemma 3.5 if an algorithm is given that computes the required values $A(i, w)$ in $\mathcal{O}(3^{tw}n^{\mathcal{O}(1)})$ time. The algorithm is a dynamic programming algorithm on a tree decomposition. Since this algorithm is discussed in detail in [13] some steps that are nonessential to understanding the gist of the algorithm are skipped here.

The algorithm computes $A(i, w)$ by bottom-up dynamic programming on the tree decomposition $\mathbb{T}$. For each node x of $\mathbb{T}$, the algorithm counts partial-solution-cut pairs $(X, (X_1, X_2))$, where X is a partial solution in $G[V_x]$ if all terminals in $G[V_x]$ are in X, and where the cut (X_1, X_2) is a consistent cut of $G[V_x]$ with additionally that if $t_0 \in X$ then $t_0 \in X_1$. To count these pairs, define a labelling using labels 0, 1_1 and 1_2 on the vertices in the bag 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, associate to each node x of $\mathbb{T}$ a table $A_x(i, w, s)$ with entries for all $0 \leq i \leq k$, $0 \leq w \leq kW$, 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 $w(X) = w$. It suffices to compute the entries $A_r(k, W, \emptyset)$ for the root bag r for all W.

What remains is to specify for each type of bag how to construct the dynamic programming table associated with it. The vertex or edge under consideration is denoted by v or (u, v). The bag itself is x and the left and right child of a bag, if applicable, are y and z.

- **Leaf bag:**

  $$A_x(0, 0, \emptyset) = 1$$

  All other entries are zero.

- **Introduce vertex bag:**

  $$\begin{aligned} A_x(i, w, s[v \to 0]) &= [v \notin T]A_y(i, w, s) \\ A_x(i, w, s[v \to 1_1]) &= A_y(i - 1, w - w(v), s) \\ A_x(i, w, s[v \to 1_2]) &= [v \neq t_0]A_y(i - 1, w - w(v), s) \end{aligned}$$

  The first line holds since all terminals must be in the solution. The third option is only possible if $v \neq t_0$ since t_0 can only be on side 1 of the cut.

- **Introduce edge bag:**

$$A_x(i, w, s) = [s(u) = 0 \vee s(v) = 0 \vee s(u) = s(v)]A_y(i, w, s)$$

If u and v are on different sides of the cut then the entry becomes invalid.

- **Forget vertex bag:**

$$A_x(i, w, s) = \sum_{\alpha \in \{0,1_1,1_2\}} A_y(i, w, s[v \to \alpha])$$

This step sums over all possible states v can have in the child bag.

- **Join bag:**

$$A_x(i_x, w_x, s) = \sum_{i_x = i_y + i_z - |s^{-1}(\{1_1,1_2\})|} \sum_{w_x = w_y + w_z - w(s^{-1}(\{1_1,1_2\}))} A_y(i_y, w_y, s) \cdot A_z(i_z, w_z, s)$$

The labeling has to be identical for both children. The vertices in the bag and their weights are counted twice if they are summed, so they have to be deducted to obtain i_x and w_x.

The bag which takes most time to compute is the join bag. Each entry of the A_x has to be evaluated. There are only a limited number of entries from each of the child nodes that could result in an entry of A_x. When s is fixed this also fixes the labelling for y and z and the amount of i_y and i_z or w_y and w_z that could result in a given i_x or w_x is proportional to n. The main component of the computation time is therefore dependent on the size of A_x. This size is determined by the different options for i_x, w_x and s. The amount of options for i_x and w_x are of order $n^{\mathcal{O}(1)}$ but the amount of labellings s is of order 3^{tw} since there are three different labels and the maximum size of a bag is equal to the treewidth. This results in an overall running time of $\mathcal{O}(3^{tw}n^{\mathcal{O}(1)})$. □

3.2 Rank Based Approach

3.2.1 Theory

The following operators from [6] are used 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 the dominated partitions.

- **Insert:** For $X \cup U = \emptyset$, define $\texttt{ins}(X, \mathcal{F}) = \{(p_{\uparrow U \cup \mathcal{F}}, w) \mid (p, w) \in \mathcal{F}\}$. This operator adds all elements in X as singletons to each partition.

- **Shift:** For $w' \in \mathbb{N}$, define $\texttt{shft}(w', \mathcal{F}) = \{(p, w + w') \mid (p, w) \in \mathcal{F}\}$. This operator increases the weight of each partition in $\mathcal{F}$ by w'.

- **Glue:** For u, v, let $\hat{U} = U \cup \{u, v\}$ and define $\texttt{glue}(\{u, v\}, \mathcal{F}) \subseteq \Pi(\hat{U}) \times \mathbb{N}$ as

$$\texttt{glue}(\{u, v\}, \mathcal{F}) = \texttt{rmc}(\{(\hat{U}[\{u, v\}] \sqcup p_{\uparrow \hat{U}}, w) \mid (p, w) \in \mathcal{F}\})$$

  This operator adds u and v to the base set if needed and combines all blocks with u and v into one block in each partition.

- **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

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

  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.

- **Join:** For a set U' and $\mathcal{G} \subseteq \Pi(U')$, let $\hat{U} = U \cup U'$ and define $\texttt{join}(\mathcal{F}, \mathcal{G}) \subseteq (\Pi(\hat{U}) \times \mathbb{N})$ as

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

  This operator extends all partitions to the same base set $\hat{U}$. It then combines each pair of partitions by means of the $\sqcup$ operator and assigns the sum of the weights as a new weight.

It is straightforward to find the running times of these operations [6].

Corollary 3.7. *Each of the operations union, shift, glue, and project can be performed in time $S \cdot |U|^{\mathcal{O}(1)}$, where S is the size of the input of the operation. Given sets of weighted partitions F and G, $\texttt{join}(F, G)$ can be computed in time $|F| \cdot |G| \cdot |U|^{\mathcal{O}(1)}$.*

The basic idea behind the rank based approach is to limit the sizes of the tables used in a dynamic programming routine. A naive algorithm is taken as starting point. In between the steps of the algorithm, e.g. after processing each bag, a Reduce routine is used to reduce the size of the table. The crux of the Reduce routine is that it does not look at individual entries of the table but looks at sets of entries. The results of this section are discussed in more detail in the original paper [6].

In the naive algorithm, as in the rank based algorithm, the entries in a table in a dynamic programming algorithm are specified by means of a labelling. This labelling specifies the role of the vertices in the bag in the solution, e.g. in or not in the solution. Each entry $A_x(s)$ is filled with weighted partitions. $A_x(s)$ represents all partial solutions on $G[V_x]$ consistent with the labelling s in the following way: the weight of the partition corresponds to the weight of the partial-solution; and vertices are in the same block of the partition that represents that solution, if and only if, the vertices are in the same connected component. The operators specified above are used to handle these weighted partitions.

When reducing a table it is important that the smaller table is representative of the larger table, i.e. if a partial solution in the larger set of partial solutions can be extended to an optimal solution, then there should always be a partial solution in the smaller set that can also be extended to this optimal solution. The formal definition is given below.

Definition 3.8 (Representation)**.** *For sets of weighted partitions $\mathcal{F}, \mathcal{F}' \subseteq (\Pi(U) \times \mathbb{N})$ and a partition $q \in \Pi(U)$, define:*

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

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

The essence of the rank-based approach lies in the `Reduce` procedure from [6]. This procedure reduces the size of the tables used in the dynamic program without loss of representation.

Theorem 3.9. *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 where ω is the matrix multiplication exponent.*

The matrix multiplication exponent will be explained in detail in Section 3.4. For the moment it suffices to know that $\omega < 2.373$. The proof is omitted here in lieu of a sketch of the routine.

For the moment the weights will be disregarded and the focus will be on the partitions. Extending the method to include weights is trivial. The aim of the routine is to find a subset of partitions that is representative. To reiterate, this means that there is a partition in this set that can be extended to the unit partition by combining it with another partition, if there was a partition in the original set for which this also holds. It turns out that it is possible to construct a matrix in which finding a basis plays the same role as selecting a subset of partitions that is representative. This matrix is denoted by $M \in \mathbb{Z}_2^{\Pi(U) \times \Pi(U)}$. The matrix shows which partitions combine to form the unit partition. Formally it can defined as follows:

$$M[p, q] = \begin{cases} 1, & \text{if } p \sqcup q = \{U\} \\ 0, & \text{otherwise} \end{cases}$$

One can see that a basis in M corresponds to a set of partitions for which the mentioned criteria hold and which is therefore representative. The next question is how large this basis, and thus the set, can be. Obtaining an upper bound on the size of the basis amounts to obtaining an upper bound on the rank of the matrix. To find this upper bound it is shown that M can be constructed by taking the product of two cutmatrices C. Define $\texttt{cuts}(t) = \{(V_1, V_2) \mid V_1 \cup V_2 = U \wedge 1 \in V_1\}$ where 1 is an arbitrary but fixed element of U. A matrix $C \in \mathbb{Z}_2^{\Pi(U) \times \texttt{cuts}(t)}$ is a cutmatrix if $C[p, (V_1, V_2)] = [(V_1, V_2)$ is consistent with $p]$. The matrix shows which partitions and

cuts are consistent, i.e. which partitions are a refinement of a cut. It can now be seen that $M \equiv CC^T$. The entry $CC^T[p, q]$ counts the number of cuts that is consistent both p and q. This is 1 iff $p \sqcup q = \{U\}$. To see this consider the blocks in $p \sqcup q$. The block containing 1 must be in V_1. The other blocks can either be in V_1 or V_2. The total amount of cuts consistent with $p \sqcup q$ is thus equal to $2^{|p \sqcup q|-1}$.

The next step is to find a row basis in C because the associated partitions will be representative of the whole set of partitions. If the weights of the partitions are included this should be a minimum weight basis. Since there are only $2^{|U|-1}$ columns in C, this is a bound on the rank of C. For the output, this means that $\mathcal{F} \leq 2^{|U|-1}$.

3.2.2 Example

Weighted Steiner Tree
Input: An undirected graph $G(V, E)$, a set of terminals $T \subseteq V$, a weight function $w : E \to \mathbb{N}$ and an integer k.
Question: Does there exist a set $X \subseteq E$ such that $T \subseteq V(G[X])$, $|X| \leq k$, and $G[X]$ is connected?

First a naive algorithm for weighted Steiner Tree on tree decompositions will be given. Thereafter, it will be showed how to use representative sets and Gaussian elimination to improve the time complexity. Note that, different from Section 3.1, (partial) solutions are now sets of *edges* connecting the terminals T. Note also that the `Reduce` algorithm can be used as a black box when discussing specific problems. It reduces tables irrespective of the problem in which it is used. This means that no more theoretical discussion of the workings of the algorithm is needed throughout the rest of the thesis.

In a bottom-up fashion, the naive algorithm computes a table A_x for each node x of the tree decomposition $\mathbb{T}$. This table keeps track of all possible partial solutions $Y \subseteq E_x$ on $G[V_x]$ that can be extended to a minimal weight solution on G. These partial solutions are subsets $Y \subseteq E(G[V_x])$ such that all terminals in $G[V_x]$ are incident to an edge in Y, and all connected components in $G[Y]$ either contain a vertex in 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 $G[V_x]$ 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 each bag x with children y and z, if applicable, the tables are constructed as below. The vertex or edge under consideration is denoted by v or (u, v).

- **Leaf bag:**
$$A_x(\emptyset) = \{(\emptyset, 0)\}$$

- **Introduce vertex bag:**
$$A_x(s) = \begin{cases} \texttt{ins}(\{v\}, A_y(s_y)), & \text{if } s(v) = 1 \\ A_y(s_y), & \text{if } s(v) = 0 \wedge v \notin T \\ \emptyset, & \text{if } s(v) = 0 \wedge v \in T \end{cases}$$
If v is in the solution then it is inserted as a singleton into each partition. Since no edges to v are introduced, it cannot be connected to another part of the solution. If v is not in the solution then the partitions do not change. Of course if v is a terminal then it has to be in the solution.

- **Introduce edge bag:**
$$A_x(s) = \begin{cases} \texttt{shft}(w(uv), A_y(s) \uplus \texttt{glue}(\{u, v\}, A_y(s))), & \text{if } s(v) = s(u) = 1 \\ A_y(s), & \text{otherwise} \end{cases}$$
If v and u are both in the solution, the blocks with u and v can be merged. If not, the partitions remain the same.

- **Forget vertex bag:**
$$A_x(s) = A_y(s[v \to 0]) \uplus \texttt{proj}(v, A_y(s[v \to 1]))$$
The partial solutions where v was in the solution are combined with those where v was not. If v was in the solution it is removed from the partition and partial solutions where v was a singleton are removed. The latter step ensures that all solutions are connected.

- **Join bag:**
$$A_x(s) = \texttt{shft}(w(s^{-1}(1)), \texttt{join}(A_z(s), A_y(s)))$$
Each partial solution in $A_y(s)$ can be combined with each partial solution from $A_z(s)$. The partitions of each solution need to be combined and the weights have to be added and then adjusted for double nodes.

It is now possible to apply the `Reduce` routine from Theorem 3.9 at each step of the naive algorithm for Steiner Tree and carefully analyse the resulting running time to obtain the following result.

Theorem 3.10. *There exists an algorithm that, given a graph G and a tree decomposition $\mathbb{T}$ of G of width tw, solves* Steiner Tree *in time $\mathcal{O}(n(1 + 2^{\omega+1})^{tw} tw^{\mathcal{O}(1)})$.*

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 node x in the tree 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. This means that $A_z(s) \leq 2^{|s^{-1}(1)|}$ and $A_y(s) \leq 2^{|s^{-1}(1)|}$. Because the naive algorithm is correct and the `Reduce` procedure maintains representation (Theorem 3.9), it is possible to conclude that the new algorithm is correct also.

Processing the bags can be done relatively fast, see Corollary 3.7. The time complexity is dominated by the `Reduce` algorithm which takes longest for a join bag since the input is largest there. It is known that after a join operation $A_x(s) \leq 2^{|s^{-1}(1)|} * 2^{|s^{-1}(1)|} = 2^{2|s^{-1}(1)|}$. The total time needed for REDUCE on all labelings can be seen to be:

$$\mathcal{O}(\sum_{i=0}^{|B_x|} \binom{|B_x|}{i} 2^{2i} 2^{(\omega-1)i} tw^{\mathcal{O}(1)}) = \quad \mathcal{O}(\sum_{i=0}^{|B_x|} \binom{|B_x|}{i} 1^{|B_x|-i} 2^{(\omega+1)i} tw^{\mathcal{O}(1)}) =$$
$$\mathcal{O}((1+2^{\omega+1})^{|B_x|} tw^{\mathcal{O}(1)}) \leq \quad \mathcal{O}((1+2^{\omega+1})^{tw} tw^{\mathcal{O}(1)})$$

In the first formula, the first part, $\sum_{i=0}^{|B_x|} \binom{|B_x|}{i}$, generates all possible labellings. The variable i stands for the number of nodes assigned a 1 in the labeling. The second part, $2^{2i} 2^{(\omega-1)i} tw^{\mathcal{O}(1)}$, gives the time to reduce the table for that labeling. The term 2^{2i} in this part is the maximum size of the table for each labeling. The binomial theorem is used to go from the second step to the third step. Since this time is needed for each bag, the total running time is $\mathcal{O}(n(1+2^{\omega+1})^{tw} tw^{\mathcal{O}(1)})$. □

3.3 Fast Subset Convolutions

3.3.1 Theory

The fast subset convolution technique that is used in this paper is a generalization of fast subset convolution [1] and has been used on tree and branch decompositions [31, 7]. It makes use of transformations that use multiple states.

When constructing a table A_x from tables A_y and A_z in a dynamic programming routine on a branch decomposition, it is often the case that the label of a node in A_x can be caused be several combinations of labels of that node in A_y and A_z (Steiner tree is an exception and therefore a relatively easy example). This is problematic because when composing A_x, in order to determine the value of an entry, a relatively high number of entries from A_y and A_z must be examined. For example, if a node is labeled in A_x to indicate it has a neighbor in the solution it could be that this is a result from either the node being labeled as such in A_y, or being labeled as such in A_z, or being labeled as such in both tables. This increases the running time of the algorithm. The idea behind fast subset convolution is to choose a different set of labels, such that a label in A_x can only be the result of a more limited set of combinations of labels in A_y and A_z. The key here is that, in order to be able to guarantee the requirements of the problem, it must be possible to convert between different sets of labels. To do this the algorithm does

not only keep track of the size of a (partial) solution but also of the number of (partial) solutions of that size. How to convert the labels is different for each set of labels used, but follows the same reasoning as presented in the example.

3.3.2 Example

DOMINATING SET
Input: An undirected graph $G(V, E)$ and an integer k.
Question: Does there exist a set $X \subseteq V$ such that $|X| \leq k$, and $N[X] = V$?

Before introducing the subset convolutions the folklore algorithm will be presented. To make the transition to the algorithm using fast subset convolutions more fluent this algorithm already counts the number of solutions.

Theorem 3.11. *There exists an algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw solves* DOMINATING SET *in time $\mathcal{O}(6^{bw}n^{\mathcal{O}(1)})$.*

Proof. In a bottom-up fashion, the 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 $X \subseteq V_x$ on $G[E_x]$ that can be extended to a solution on G. These partial solutions are subsets $X \subseteq V_x$ such that all vertices in $G[E_x] \setminus B_x$ are either in X or dominated and all connected components in $G[X]$ contain a vertex in B_x.

Each entry $A_x(s, k)$ in the table is indexed by a labelling $s \in \{1, 0_1, 0_0\}^{B_x}$ on the vertices in B_x and a value $0 \leq k \leq n$ which denotes $|X|$, i.e. the amount of nodes in the solution. The labels 1, 0_1 and 0_0 respectively mean that a vertex is in the solution, is not in the solution but is dominated by a vertex in the solution, or is not in the solution and is not dominated by a vertex in the solution either. A vertex v is dominated if $N(v) \cap X \neq \emptyset$. An entry $A_x(s, k)$ has a value that corresponds to the number of partial solutions for which the vertices in B_x have the labeling s and $k = |X|$. The root node r will have one entry for each value of k (since $B_r = \emptyset$) with a value that corresponds to the amount of solutions that consist of k vertices.

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

$$A_x(0_0\ 0_0, 0) = 1 \qquad A_x(0_1\ 1, 1) = 1 \qquad A_x(1\ 0_1, 1) = 1 \qquad A_x(1\ 1, 2) = 1$$

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

$$A_x(s_x, k_x) = \sum_{s_x, s_y, s_z \text{match}} \sum_{k_y + k_z - k^{I,F} = k_x} A_y(s_y, k_y) \cdot A_z(s_z, k_z)$$

Here the term $k^{I,F}$ stands for the amount of nodes in F and I that have label 1. Of particular concern is whether labellings are matching. Labeling s_x, s_y and s_z are said

to be matching if s_y and s_z can combine to form s_x. For each of the partitions of the middle sets this means something different.

The labels on a vertex v in L match if $s_y(v) = s_x(v) \in \{1, 0_1, 0_0\}$. Roughly the same holds for a vertex v in R, the labels match if $s_z(v) = s_x(v) \in \{1, 0_1, 0_0\}$. For both there are 3 combinations of labels that produce a match. For a vertex v in F the labels are matching if either $s_y(v) = s_z(v) = 1$, or $s_y(v), s_z(v) \in \{0_0, 0_1\}$ while not $s_y(v) = s_z(v) = 0_0$. So 4 combinations of labels produce a match. Lastly, for a vertex v in I labels produce a match if either $s_y(v) = s_z(v) = s_x(v) \in \{1, 0_0\}$, or $s_x(v) = 0_1$ while $s_y(v), s_z(v) \in \{0_0, 0_1\}$ and not $s_y(v) = s_z(v) = 0_0$. Here 5 combinations produce valid matchings.

The bottleneck in this algorithm is computing the tables for the internal edges. For each matching set of states the table has to be constructed for each value of k and in the sum there are $\mathcal{O}(n)$ terms for each matching set of states. Since each vertex in L and R has 3 possible matching states, each vertex in F has 4 matching states, and each vertex in I has 5 matching states, the total amount of possible matching sets of states is $3^{|L|+|R|}4^{|F|}5^{|I|}$. This means each table A_x can be computed in $\mathcal{O}(n^2 3^{|L|+|R|}4^{|F|}5^{|I|})$ time.

Under the constraints in Lemma 2.5, this is maximal if $|I| = 0$ and $|R| = |L| = |F| = \frac{1}{2}bw$. Since there are $\mathcal{O}(m)$ internal edges, this leads to a running time of $\mathcal{O}(mn^2 4^{\frac{1}{2}bw}3^{bw}) = \mathcal{O}(6^{bw}n^{\mathcal{O}(1)})$. □

In the complexity analysis it becomes clear that it is mainly the number of valid matching states that influences the running time. This is what the introduction of fast subset convolutions takes advantage of.

To do this an extra label is introduced, namely $0_?$. This label means that a node is not in the solution but it is not specified whether it is dominated or not. In terms of the existing labels, a node with label $0_?$ could have been assigned either 0_1 or 0_0. In the algorithm a different set of labels is used for each vertex. Which labels are used depends on whether a node is in L, R, F or I. Furthermore, this is done asymmetrical, i.e. for an internal edge x with children y and z, it could be that a node in F in the table A_y is labeled using different states as the same node (still in F) in table A_z. This is done to allow for easy recombination later in the algorithm. Firstly, however, the sets of states and how they interact will be formalized. This formalization is similar to that in previous papers [31].

Lemma 3.12. *Let x be an edge of a branch decomposition $\mathbb{T}$ with corresponding middle set B_x and let A_x be a table with entries $A_x(s, k)$ representing the number of partial solutions of* Dominating Set *in $G(E_x)$ of each size k, with E_x being the edges assigned to the leaves of T_x. Here s corresponds to all labelings of the middle set B_x with states such that for every individual node in B_x one of the following fixed sets of states is used:*

$$\{1, 0_1, 0_0\} \qquad \{1, 0_1, 0_?\} \qquad \{1, 0_?, 0_0\}$$

The information represented in the table A_x does not depend on the choice of the set of states from the options given above. Moreover, there exist transformations between

tables using representations with different sets of states on each vertex using $\mathcal{O}(|A_x||B_x|)$ operations.

Proof. To show that this holds a set of formulæ will be provided that can be used to transform the table such that it represents the same information with different sets of states. Since these formulæ can also be used to reverse the transformation it follows that the information is preserved.

The transformation works in $|B_x|$ steps. The invariant is that, at any step i, the first $i-1$ labels in the labeling use the sets of states that are introduced and the last $|B_x|-i$ labels use the original sets of states. The set of states used by the i-th label are changed in step i. Note that when changing from one set of states to another, it is always the case that one label is introduced. Depending on which label this is one of the formulæ below is used.

$$A_x(c_1 \times \{0_?\} \times c_2, k) = A_x(c_1 \times \{0_1\} \times c_2, k) + A_x(c_1 \times \{0_0\} \times c_2, k)$$
$$A_x(c_1 \times \{0_1\} \times c_2, k) = A_x(c_1 \times \{0_?\} \times c_2, k) - A_x(c_1 \times \{0_0\} \times c_2, k)$$
$$A_x(c_1 \times \{0_0\} \times c_2, k) = A_x(c_1 \times \{0_?\} \times c_2, k) - A_x(c_1 \times \{0_1\} \times c_2, k)$$

Here c_1 is a sub-labeling of the first $i-1$ nodes that uses the new sets of states and c_2 is a sub-labeling of the last $|B_x|-i$ nodes that uses the original sets of states. The entries that have a label on the i-th node that is also in the new set of states stay the same. After computing all new values the entries with the removed label on node i are removed. For example, the first formula counts the number of partial solutions that do not contain the i-th vertex v in their solution (it is not specified whether this vertex is dominated) by adding the partial solutions that do not include v but where v is dominated and the partial solutions that do not contain v but where v is not dominated. Note that although the set of states used may differ per vertex and per table, node i within one table will always use the same set of states for its labeling.

For each of the $|A_x|$ entries, $|B_x|$ subtractions or additions are needed so a transformation will use $\mathcal{O}(|A_x||B_x|)$ operations. □

It is now possible to use Lemma 3.12 to improve on Theorem 3.11.

Theorem 3.13. *There exists an algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw solves* Dominating Set *in time $\mathcal{O}(3^{\frac{3}{2}bw}n^{\mathcal{O}(1)})$.*

Proof. Except for how the tables for each internal edge are computed the algorithm is the same as the algorithm of Theorem 3.11. When computing the table A_x for an edge x with children y and z, the algorithm starts by transforming the tables A_y and A_z so that they use the required sets of states for labels. This means that nodes in I are labeled with $\{1, 0_?, 0_0\}$, nodes in L and R are labeled with $\{1, 0_1, 0_0\}$ and nodes in F are labeled with either $\{1, 0_1, 0_0\}$ if they are in table A_y or with $\{1, 0_1, 0_?\}$ if they are in table A_z.

With these new sets of states it is possible to reevaluate when labellings are matching. For any node v the labels match if:

- $v \in I : s_x(v) = s_y(v) = s_z(v) \in \{1, 0_?, 0_0\}$

- $v \in F$: either $s_y(v) = s_z(v) = 1$ or $s_y(v) = 0_0$ and $s_z(v) = 0_1$ or $s_y(v) = 0_1$ and $s_z(v) = 0_?$
- $v \in L : s_x(v) = s_y(v) \in \{1, 0_1, 0_0\}$
- $v \in R : s_x(v) = s_z(v) \in \{1, 0_1, 0_0\}$

Note that there are 3 possibilities for each part of the middle sets. For the nodes in L and R the same is done as in the previous algorithm. For nodes in I the same holds if a node has label one. If a node has label 0_0 in A_x it must be the case that it has label 0_0 in both the tables of the children. This is true since if a node is not in the solution and not dominated, it cannot have been dominated before. The most complicated case is for the nodes in F. If a node is forgotten it must be either in the solution, or it must be dominated. The case where the node is in the solution is covered by the first instance, like in the previous algorithm. However, the algorithm must make sure that each of the other cases is counted only once. To see that this is the case look at the combinations that resulted in valid matchings when the set of states used was $\{1, 0_1, 0_0\}$. The case where $s_y(v) = 0_0$ and $s_z(v) = 0_1$ is covered by the same combination in this algorithm. The case where $s_y(v) = 0_1$ and $s_z(v) = 0_1$ and the case where $s_y(v) = 0_1$ and $s_z(v) = 0_0$ are both counted in the case where $s_y(v) = 0_1$ and $s_z(v) = 0_?$ in this algorithm.

The time analysis of this algorithm is like the time analysis in Theorem 3.11. The only difference is that there are now a smaller number of valid matchings. This means that the factor $3^{|L|+|R|}4^{|F|}5^{|I|}$ is replaced by $3^{|L|+|R|+|F|+|I|}$. The running time under the constraints on the different parts of the middle sets is still maximal if $|I| = 0$ and $|R| = |L| = |F| = \frac{1}{2}bw$. So the algorithm runs in $\mathcal{O}(mn^2 3^{\frac{3}{2}bw}) = \mathcal{O}(3^{\frac{3}{2}bw} n^{\mathcal{O}(1)})$. □

3.4 Fast Matrix Multiplication

3.4.1 Theory

The time it takes to multiply two $(n \times n)$-matrices can be written as $\mathcal{O}(n^\omega)$. The ω, as has been mentioned in Section 3.2, is called the matrix multiplication exponent. Since the time this takes when choosing the naive approach is $\mathcal{O}(n^3)$, it is known that $\omega \leq 3$. The best current value for ω is 2.373 [21].

If, instead of square matrices, a $(n \times p)$-matrix and a $(p \times n)$-matrix are considered, then the running times differ. Two cases can be distinguished here, the case where $p \leq n$ and the case where $p > n$. For the first case there is a $\mathcal{O}(n^{1.85} \cdot p^{0.54})$ algorithm (with $\omega = 2.373$). For the second case there exists an $\mathcal{O}(\frac{p}{n} \cdot n^\omega + \frac{p}{n} \cdot n^2)$ algorithm. This is the result of splitting each matrix into $\frac{p}{n}$ many $(n \times n)$-matrices and multiplying them to find another $\frac{p}{n}$ $(n \times n)$-matrices. Sum over each entry of the resulting matrices to find the result of the multiplication. Suppose that the multiplication is $B \cdot C = A$, then splitting gives matrices B_l and C_l with $1 \leq l \leq \frac{p}{n}$. Multiplying these gives matrices A_l. Now sum over elements A^l_{ij} for each l to find the values A_{ij}.

Fast matrix multiplication was first used in the context of branch decomposition by Dorn [14]. The intuition behind using fast matrix multiplication in dynamic program-

ming algorithms on branch decompositions is that the tables that have to be combined can be seen as matrices. Combining these matrices can be done through matrix multiplication. The most straightforward way to do this is to fix all variables, including the labelling on I, except for the labelling of the nodes in F, L and R. Assume that the labeling has v different values. It is now possible to construct two matrices B and C (for some instance of the fixed variables) of size $(v^{|L|} \times v^{|F|})$ and $(v^{|F|} \times v^{|R|})$ respectively. 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 $BC = A$ 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). Note that it is imperative that column i of B and row i of C correspond to the same labelling on F. The entries in each of the matrices now correspond to a value from each of the respective tables. This is ensured because a single table entry is specified for each matrix entry by the fixed variables and the complete labelling resulting from the fixed labelling on I and the labellings associated with the rows and columns. This means all entries for A_x can be found in the A matrices.

To clarify this strategy it is helpful to see how it can be used to improve the result on Dominating Set of Theorem 3.13.

3.4.2 Example

Theorem 3.14 ([7])**.** *There exists an algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw solves* Dominating Set *in time $\mathcal{O}(3^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$ where ω is the matrix multiplication exponent.*

Proof. As mentioned above, the first step is to fix several variables. In this case k_x, k_y and a labeling on I are fixed for each matrix multiplication. As shown later, it is not necessary to fix k_z since it is well-defined by k_x, k_y and the labeling. Now two matrices B and C are created of size $(3^{|L|} \times 3^{|F|})$ and $(3^{|F|} \times 3^{|R|})$ respectively. The rows and columns are labeled as described in the theory section above. An entry of matrix B will have value $A_y(s_y, k_y)$. Because the labeling on I is fixed and each row and column are assigned a labeling on a part of the middle set, s_y is well-defined. This is true for the whole expression since k_y was fixed already. The entries of C are the values $A_z(s_z, k_x - k_y - k^{I,F})$. As before, the term $k^{I,F}$ stands for the amount of nodes in F and I that have label 1, this term is defined by the labeling s_z. As a result, all variables in the expression are fixed for each entry. Note that, when constructing B and C, the labeling assigned to column i in B and row i in C cannot be equal, different sets of states are used in labeling the parts. However, it is possible to arrange the matrices such that column i in B and row i in C have labellings that match. This is possible because there is a bijection between the valid labellings on F.

If B and C are multiplied, the result is a matrix A. This is an $(3^{|L|} \times 3^{|R|})$ matrix with a labeling on L and R assigned to each row and column. The values of each entry

can be seen to correspond to the result of the sum that was previously used to compute $A_x(s_x, k_x)$. When looking at how each term is constructed it becomes clear that each entry of A equals the sum over all possible labellings of F of the product of compatible terms such that $k_x = k_y + k_z - k^{I,F}$. In other words, the outcome is exactly equal to the outcome of the formula used to compute the internal edge table in the algorithm from Theorem 3.13:

$$A_x(s_x, k_x) = \sum_{s_x, s_y, s_z \text{match}} \sum_{k_y + k_z - k^{I,F} = k_x} A_y(s_y, k_y) \cdot A_z(s_z, k_z)$$

Because of the values that are fixed for each set of matrices, this multiplication needs to be done $n^2 3^{|I|}$ times for each table. For symmetry reasons $|L| = |R|$ in the worst case. The time it takes to multiply B and C follows from the theory and is $\mathcal{O}(\frac{3^{|F|}}{3^{|L|}} 3^{\omega|L|})$. The total time complexity is therefore $\mathcal{O}(mn^2 3^{|I|} \frac{3^{|F|}}{3^{|L|}} 3^{\omega|L|})$. The worst-case under the constraints in Lemma 2.5 arises if $|I| = 0$ and $|R| = |L| = |F| = \frac{1}{2}bw$. The algorithm therefore has a running time of $\mathcal{O}(mn^2 3^{\frac{\omega}{2}bw}) = \mathcal{O}(3^{\frac{\omega}{2}bw} n^{\mathcal{O}(1)})$. □

4 'Cut and Count' on branch decompositions

In this section several examples will be used to show that the 'Cut and Count' technique from [13] can also be used in algorithms on branch decompositions. First of all it is important to note, as will become clear in this section, that only the `Count` procedure has to be changed. This is the only part that uses the tree or branch decomposition. Everything else can remain identical whether a tree of branch decomposition is used. For clarity some context on the 'Cut and Count' framework for each problem is given, but for a detailed description see the original paper [13]

The problems discussed in this section are Steiner Tree, Connected Dominating Set, Feedback Vertex Set and Traveling Salesman. These problems are chosen because they are representative of connectivity problems and because collectively they deal with all peculiarities encountered in [13], such as markers and fast subset convolutions. All other problems discussed in the mentioned paper can be solved using branch decompositions in a similar manner.

4.1 Steiner Tree

Since Steiner Tree was an example in Section 3.1, the general approach is known. The only part that needs to be revised is the `Count` procedure where the dynamic programming on tree decompositions was used because Lemma 3.5 still holds. The `Count` procedure will now be defined on a branch decomposition.

For easier exposition, first the following theorem will be proven. Next, this will be improved this using fast matrix multiplication in Theorem 4.2.

Theorem 4.1. *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 3.5 if an algorithm can be given that computes the required values $A(i, w)$ in $\mathcal{O}(3^{\frac{3}{2}bw}n^{\mathcal{O}(1)})$ time. This algorithm is given below.

The algorithm computes $A(i, w)$ by bottom-up dynamic programming on the branch decomposition $\mathbb{T}$. For each edge x of $\mathbb{T}$, count partial-solution-cut pairs $(X, (X_1, X_2))$, where X is 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, 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, associate to each edge x of $\mathbb{T}$ a table $A_x(i, w, s)$ with entries for all $0 \leq i \leq k$, $0 \leq w \leq kW$, 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 $\mathbb{T}$, $B_x = \{u, v\}$ 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, it is enforced 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, fill the table A_x by combining the counted number of partial-solution-cut pairs from the tables for y and z. For this, it is defined 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 s_x^L denotes the labelling s_x restricted to middle set partition L; for the middle set partitions see Definition 2.4).

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 any middle set partition):

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

This counts the total number of partial-solution-cut pairs $(X, (X_1, X_2))$ that satisfy the constraints. The summations combine all compatible entries from A_y and A_z and the multiplication combines the individual counts. Labellings are compatible if $s_x^L = s_y^L \wedge s_x^R = s_z^R \wedge s_y^F = s_z^F \wedge s_x^I = s_y^I = s_z^I$. To see that exactly these entries are valid, 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, it is possible to 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 at most $3^{|F|}k^2W$ combinations of entries from A_y and A_z have to be inspected, 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 2.5. □

This result can be improved by using the fast matrix multiplications introduced in Section 3.4.

Theorem 4.2. *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 4.1, however, the formula for the table $A_i(i_x, w_x, s_x)$ associated to an internal edge of the branch decomposition is evaluated in a more efficient way.

This is achieved by fast matrix multiplications. The usage is very similar to that in Section 3.4.2. A labelling on I is fixed and i_x, i_y, w_x and w_y are fixed. Then A_x is computed by means of matrix multiplications.

Since there are three labels, $3^{|I|}k^2W^2$ matrix multiplications are performed 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 [14, 21]), where it is used that it is possible to assume $|L| = |R|$ in a worst-case analysis for symmetry reasons. Under the constraints of Lemma 2.5, 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 4.3. *There exists 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 4.2 with Lemma 2.6 and use $\omega < 2.373$ [21]. □

4.2 Connected Dominating Set

Connected Dominating Set
Input: An undirected graph $G(V, E)$ and an integer k.
Question: Does there exist a set $X \subseteq V$ such that $|X| \leq k$, $N[X] = V$, and $G[X]$ is connected?

The 'Cut and Count' routine works in similar ways as for Steiner tree. A vertex v_0 is fixed that needs to be in the solution. For a general algorithm, iterate over all nodes in the neighborhood of a node. A random weight function $w : V \to [0, ..., W]$ is selected. The set R is the set of all dominating sets. The connectivity requirement is again relaxed. The set $\mathcal{C}$ is defined as in the previous section. Proving that these requirements ensure a correct solution is similar to the proof in the previous section. The following lemma now holds:

Lemma 4.4. *Suppose an algorithm* `Count` *is given that, given a graph $G(V, E)$, some fixed vertex $v_0 \in V$, 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 $0 \leq w \leq kW$:*

$$A(i,w) = \left| \left\{ (X, (X_1, X_2)) \;\middle|\; \begin{array}{l} (X_1, X_2) \text{ a consistent cut of } G[X], \\ N[X] = V, v_0 \in X_1, |X| = i, w(X) = w \end{array} \right\} \right| \pmod 2$$

Then, there exists a Monte-Carlo algorithm that solves Connected Dominating Set *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 using $W = \mathcal{O}(n)$.*

Theorem 4.5. *There exist a Monte-Carlo algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw, solves* Connected Dominating Set *in time $\mathcal{O}(4^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$, where ω is the fast matrix multiplication exponent.*

Proof. The result follows from Lemma 4.4 if an algorithm can be given that computes the required values $A(i, w)$ in $\mathcal{O}(4^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$ time.

Compute $A(i, w)$ by bottom-up dynamic programming on the branch decomposition $\mathbb{T}$. For each edge x of $\mathbb{T}$, count partial-solution-cut pairs $(X, (X_1, X_2))$, where X is a partial solution in $G[E_x]$ if $V(G[E_x]) \setminus B_x \subseteq N[X]$, 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 $v_0 \in X$ then $v_0 \in X_1$.

To speed up the dynamic programming algorithm the subset convolutions from Section 3.3 are used. To count partial-solution-cut pairs, a labelling is defined using a combination of labels 1_1, 1_2, 0_1, $0_?$ and 0_0 on vertices in middle set B_x associated to an edge x of $\mathbb{T}$. The first three labels are always used and two out of three of the latter are used; each set of labels consists of four different labels. These labels identify the situation of the vertex in a partial-solution-cut pair $(X, (X_1, X_2))$: 1_1 and 1_2 mean in X_1 and X_2 respectively, 0_0 means not in X but in $N[X]$, $0_?$ means not in X but perhaps in $N[X]$ and 0_0 means not in X and not in $N[X]$.

In a bottom-up fashion, associate to each edge x of $\mathbb{T}$ a table $A_x(i, w, s)$ with entries for all $0 \leq i \leq k$, $0 \leq w \leq kW$, and $s \in \{1_1, 1_2, 0_1, 0_?, 0_0\}^{B_x}$. 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 $w(X) = w$.

Which sets of labels are used for which vertices, which labellings are compatible and how the sets of labels can be transformed is almost identical to the algorithm in Theorem 3.13. The difference here is that instead of a label 1, there are two labels 1_1 and 1_2. This results in not three but four possible combinations of labels at each vertex when computing internal edges. Note that a label on a vertex in one middle set determines the labels on the vertices in the other middle sets.

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

$$\begin{aligned}
&A_x(0, 0, 0_0\ 0_0) && = 1[u \neq v_0 \wedge v \neq v_0] \\
&A_x(1, w(u), 1_1\ 0_1) && = 1[v \neq v_0] \\
&A_x(1, w(v), 0_1\ 1_1) && = 1[u \neq v_0] \\
&A_x(1, w(u), 1_2\ 0_1) && = 1[u \neq v_0 \wedge v \neq v_0] \\
&A_x(1, w(v), 0_1\ 1_2) && = 1[u \neq v_0 \wedge v \neq v_0] \\
&A_x(2, w(u) + w(v), 1_1\ 1_1) && = 1 \\
&A_x(2, w(u) + w(v), 1_2\ 1_2) && = 1[u \neq v_0 \wedge v \neq v_0]
\end{aligned}$$

Here, it is enforced that the cut is consistent, that v_0 is on the correct side of the cut ($v_0 \in X_1$), and that $|X| = i$ and $w(X) = w$.

For an internal edge x of the branch decomposition $\mathbb{T}$ with children y and z, fill the table A_x by combining the counted number of partial-solution-cut pairs from the tables for y and z. The subset convolutions technique dictates which labellings are compatible.

The table A_x is filled by means of the following formula, where i^Z and w^Z denote the number of vertices and the sum of the weights of the vertices with state 1_1 or 1_2 in Z, where Z is any middle set partition:

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

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 multiplications combine the individual counts. 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, it is possible to find the required values $A(i, w)$ at the root edge r of $\mathbb{T}$ where $B_r = \emptyset$.

It is again possible to speed up computation of the tables using fast matrix multiplications. This is very similar to the previous section, the main difference is that there

are now four different labels in each labelling. Fix i_y, i_z, w_y, w_z, and the labelling on I. Now construct the tables B and C. In this case the labelling associated with a column of B is no longer the same as the labelling associated with a row of C. However, since there is a 1-1 correspondence between compatible labellings this is not a problem. Since there are now 4 possible labels on each vertex, the sizes of B and C are $4^{|L|} \times 4^{|F|}$ and $4^{|F|} \times 4^{|R|}$ respectively. The time analysis is, with the exception of the table sizes, equal to the analysis already performed and so the algorithm has a running time of $\mathcal{O}(4^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$. □

Corollary 4.6. *There exist a Monte-Carlo algorithm that, given a planar graph G, solves* Planar Connected Dominating Set *in time $\mathcal{O}(2^{5.036\sqrt{n}})$.*

Proof. Combine Theorem 4.5 with Lemma 2.6 and use $\omega < 2.373$ [21]. □

4.3 Feedback Vertex Set

Feedback Vertex Set
Input: An undirected graph $G(V, E)$ and an integer k.
Question: Does there exist a set $Y \subseteq V$ such that $|Y| \leq k$, and $G[V \setminus Y]$ is a forest?

The 'Cut and Count' algorithm for Feedback Vertex Set is more intricate than thus far seen. In the algorithm, instead of selecting the feedback vertex set itself, the set of vertices not in the feedback vertex set, those forming the forest, is selected. To ascertain whether this set of nodes is a forest the rule is used that a graph G is a forest iff $\mathtt{cc}(G) = n - m$.

To ensure this, a bound on the amount of connected components in the solution is needed. To this end a set of markers $M \subseteq X \subseteq V$ is introduced which is a subset of the solution. The set $\mathcal{C}$ consists of pairs $((X.M), (X_1, X_2))$ where M is a set of markers and, just as before, X is a set of nodes in the solution and (X_1, X_2) is a consistent cut of X. There are two weight functions w_X and w_M which give every vertex two weights. In the algorithm it is required that $M \subseteq X_1$. This means that each solution with a connected component without a marker has an even amount of consistent cuts associated with it, because a connected component without a marker can be on both sides of the cut. These solutions therefore cancel modulo 2. A solution is thus required to have a marker in each connected component, which creates an upper bound on the amount of connected components, and needs to be a forest, which is checkable because of this upper bound.

The dynamic programming algorithm keeps track of the number of vertices in the solution $|X| = i$, the number of edges in the solution j, the number of markers $|M| = h$ and the sum of the weights of both the vertices in the solution and the markers $w_X(X) + w_M(M)$.

With this information it is possible to check if a solution is found. If there is a table entry at the root node (which is not 0) for which it holds that $i = |X| > |V| - k$, and

$h = |M| = |X| - j$ this means that this represents a forest with the set of vertices needed to be removed to create this is $|Y| \leq k$. This means all criteria are met. The following lemma captures this idea.

Lemma 4.7. *Suppose an algorithm* `Count` *is given that, given a graph $G(V, E)$, a weight function $w_X : V \to [0, ..., W]$, and a weight function $w_M : V \to [0, ..., W]$, computes the values $A(i, j, h, w)$ defined below for all $0 \leq i \leq n$, $0 \leq j \leq n$, $0 \leq h \leq n$ and $0 \leq w \leq 2nW$:*

$$A(i,j,h,w) \equiv_2 \left| \left\{ ((X,M),(X_1,X_2)) \;\middle|\; \begin{array}{l} (X_1,X_2) \text{ a consistent cut of } G[X], M \subseteq X_1, |X| = i, \\ |E(G[X])| = j, |M| = h, w_X(X) + w_M(M) = w \end{array} \right\} \right|$$

Then, there exists a Monte-Carlo algorithm that solves Feedback Vertex Set *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 using $W = \mathcal{O}(n)$.*

Theorem 4.8. *There exist a Monte-Carlo algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw, solves* Feedback Vertex Set *in time $\mathcal{O}(3^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$, where ω is the fast matrix multiplication exponent.*

Proof. The result follows from Lemma 4.7 if an algorithm exists that computes the required values $A(i, w)$ in $\mathcal{O}(3^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$ time.

$A(i, w)$ is computed by bottom-up dynamic programming on the branch decomposition $\mathbb{T}$. For each edge x of $\mathbb{T}$, count partial-solution-cut pairs $((X, M), (X_1, X_2))$, where (X, M) is a partial solution in $G[E_x]$ if $M \subseteq X \subseteq V(G[E_x])$ 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 $M \in (X_1 \setminus B_x)$.

To count these pairs, define a labelling using a combination of labels 1_1, 1_2, and 0 on vertices in 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 $((X, M), (X_1, X_2))$: label 0 means not in X, and labels 1_1 and 1_2 mean in X_1 and in X_2 respectively. The labels do not designate whether a vertex is in M, this is only determined when a vertex is no longer in B_x.

In a bottom-up fashion, associate to each edge x of $\mathbb{T}$ a table $A_x(i, j, h, w, s)$ with entries for all $0 \leq i \leq n$, $0 \leq j \leq n$, $0 \leq h \leq n$, $0 \leq w \leq 2nW$, and $s \in \{1_1, 1_2, 0\}^{B_x}$. Such an entry $A_x(i, j, h, w, s)$ counts the number of partial-solution-cut pairs $((X, M), (X_1, X_2))$, as defined above, that satisfy the constrains imposed by the states s on B_x and that satisfy $|X| = i$, $|E(G[X])| = j$, $|M| = h$ and $w_X(X \setminus B_x) + w_M(M) = w$. Note that the weights of vertices that are in the solution *and* in the middle set are not counted. The weights of vertices in the solution are only added when they are no longer in the middle set. This happens automatically for markers since it is chosen which nodes are markers only when they are no longer in the middle set.

To remedy not keeping track of which vertices are in M, some preprocessing is needed. This is the reason weights are only added when vertices are no longer in the middle set. Before constructing the table for an internal edge x of $\mathbb{T}$ with children y and z, the table

A_y is modified. For every entry of A_y, increase the weight by $w_X(X \cap F)$ (this term is different for each entry). By adding this term the weights of vertices in the solution are counted. After this, iteratively cycle through all the vertices in F. For all entries where v has label 1_1 the possibility needs to be considered that v is a marker. Add the amount of solutions corresponding to that original entry to the entry that is the same except for the marker counter h which is $h = h_{orig} + 1$ and the weight value which is $w = w_{orig} + w_M(v)$. In this way it is considered for each vertex in F with label 1_1 that it could be a marker while still maintaining the same bound on the size of the table.

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

$$A_x(0,0,0,0,0\ 0) = 1 \qquad A_x(1,0,0,0,1_1\ 0) = 1$$
$$A_x(1,0,0,0,0\ 1_1) = 1 \qquad A_x(1,0,0,0,1_2\ 0) = 1$$
$$A_x(1,0,0,0,0\ 1_2) = 1 \qquad A_x(2,1,0,0,1_1\ 1_1) = 1$$
$$A_x(2,1,0,0,1_2\ 1_2) = 1$$

Here, it is enforce that the cut is consistent, that $|X| = i$, and that $|E(G[X])| = j$.

For an internal edge x of the branch decomposition with children y and z, fill the table A_x by combining the counted number of partial-solution-cut pairs from the tables for y and z. For this, it is defined that labellings s_x of B_x, s_y of B_y, and s_z of B_z are compatible iff $s_x^L = s_y^L \wedge s_x^R = s_z^R \wedge s_y^F = s_z^F \wedge s_x^I = s_y^I = s_z^I$. Fill A_x by means of the following formula:

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

This counts the total number of partial-solution-cut pairs $((X, M), (X_1, X_2))$ that satisfy the constraints as the summations combine all compatible entries from A_y and A_z and the multiplication combines the individual counts. The preprocessing ensures that weights are only counted once.

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

It is possible to use fast matrix multiplication to speed up the computations. Although there are more values that need to be fixed, these are only dependent on n so they do not affect the resulting running time. Since the amount of labels is equal, the matrices are the same size as in Theorem 4.2 and the running time is also equal. □

Corollary 4.9. *There exist a Monte-Carlo algorithm that, given a planar graph G, solves* Planar Feedback Vertex Set *in time $\mathcal{O}(2^{3.991\sqrt{n}})$.*

Proof. Combine Theorem 4.8 with Lemma 2.6 and use $\omega < 2.373$ [21]. □

4.4 Traveling Salesman

TRAVELING SALESMAN
Input: An undirected graph $G(V, E)$, a weight function $w : E \to \mathbb{N}$ and an integer k.
Question: Does there exist a set $Y \subseteq E$ such that $w(Y) \leq k$, and $G[Y]$ is a Hamiltonian cycle?

The 'Cut and Count' algorithm for TRAVELING SALESMAN resembles that of Section 4.1 and Section 4.2. The main difference is that here the solution consists of edges instead of vertices. So $\mathcal{C}$ consist of pairs $(Y, (X_1, X_2))$ where (X_1, X_2) is a consistent cut of $G[Y]$. The selected random weight function w_2 is also on edges instead of vertices. In the algorithm it is enforced that every node removed from the middle set has degree 2. This ensures that the solution consist of a set of cycles. Since a node v_0 is chosen which is required to be in X_1, solutions with more than one cycle cancel modulo 2.

Lemma 4.10 (based on [13])**.** *Suppose an algorithm* `Count` *is given that, given a graph $G(V, E)$, some fixed vertex $v_0 \in V$, a weight function $w_1 : E \to [0, ..., W]$, and a weight function $w_2 : E \to [0, ..., W]$, computes the values $A(i, w)$ defined below for all $0 \leq i \leq k$ and $0 \leq w \leq nW$:*

$$A(i,w) = \left| \left\{ (Y, (X_1, X_2)) \;\middle|\; \begin{array}{l} (X_1, X_2) \text{ a consistent cut of } G[Y], v_0 \in X_1, \\ w_1(Y) = i, w_2(Y) = w, V = V(G[Y]) \end{array} \right\} \right| \pmod 2$$

Then, there exists a Monte-Carlo algorithm that solves TRAVELLING SALESMAN PROBLEM *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 using $W = \mathcal{O}(n)$.*

The weight function w_1 is intrinsic to the problem and gives the weights of the edges in the problem. The second weight function is added by the 'Cut and Count' technique.

Theorem 4.11. *There exist a Monte-Carlo algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw, solves* TRAVELING SALESMAN *in time $\mathcal{O}(4^{\frac{\omega}{2}bw} n^{\mathcal{O}(1)})$, where ω is the fast matrix multiplication exponent.*

Proof. The result follows from Lemma 4.10 if an algorithm exists that computes the required values $A(i, w)$ in $\mathcal{O}(4^{\frac{\omega}{2}bw} n^{\mathcal{O}(1)})$ time. This algorithm is given below.

Compute $A(i, w)$ by bottom-up dynamic programming on the branch decomposition $\mathbb{T}$. For each edge x of $\mathbb{T}$, count partial-solution-cut pairs $(Y, (X_1, X_2))$, where Y is a partial solution in $G[E_x]$ if $V(G[Y]) = V(G[E_x]) \setminus B_x$, and where the cut (X_1, X_2) is a consistent cut of the subgraph of $G[E_x]$ induced by Y (i.e., a cut in $(G[E_x])[Y]$) with additionally that if $v_0 \in X$ then $v_0 \in X_1$. To count these pairs, define a labelling using labels 0, 1_1, 1_2 and 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

$(Y, (X_1, X_2))$: label 0 means not in Y, and labels 1_1 and 1_2 mean in Y with one adjacent edge in Y and on side X_1 and X_2 of the cut, respectively. Label 2 means in Y and with 2 adjacent edges in Y.

In a bottom-up fashion, associate to each edge x of $\mathbb{T}$ a table $A_x(i, w, s)$ with entries for all $0 \leq i \leq k$, $0 \leq w \leq nW$, and $s \in \{0, 1_1, 1_2, 2\}^{B_x}$. Such an entry $A_x(i, w, s)$ counts the number of partial-solution-cut pairs $(Y, (X_1, X_2))$ as defined above that satisfy the constrains imposed by the states s on B_x and that satisfy $w_1(Y) = i$ and $w_2(Y) = w$.

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

$$\begin{aligned} A_x(0, 0, 0\ 0) &= 1 \\ A_x(w_1(uv), w_2(uv), 1_1\ 1_1) &= 1 \\ A_x(w_1(uv), w_2(uv), 1_2\ 1_2) &= 1[u \neq v_0 \wedge v \neq v_0] \end{aligned}$$

Here, it is enforced that the cut is consistent, that v_0 is on the correct side of the cut ($v_0 \in X_1$), and that $w_1(Y) = i$ and $w_2(Y) = w$.

For an internal edge x of the branch decomposition $\mathbb{T}$ with children y and z, fill the table A_x by combining the counted number of partial-solution-cut pairs from the tables for y and z. For this, it is defined that labellings are compatible if the labels on nodes in L and R are equal for all middle sets and the labels on nodes in F satisfy $s_y(v) = 2 \wedge s_z(v) = 0 \vee s_y(v) = 0 \wedge s_z(v) = 2 \vee s_y(v) = 1_1 \wedge s_z(v) = 1_1 \vee s_y(v) = 1_2 \wedge s_z(v) = 1_2$ for any node $v \in F$. To limit the amount of possible compatible labels per node in I to 4 options a slight modification of generalized fast subset convolution is used [31].

Fill A_x by means of the following formula:

$$A_x(i_x, w_x, s_x) = \sum_{\substack{s_x, s_y, s_z \\ \text{compatable} \\ \text{labellings}}} \sum_{i_x = i_y + i_z} \sum_{w_x = w_y + w_z} A_y(i_y, w_y, s_y) \cdot A_z(i_z, w_z, s_z)$$

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

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

Since the length of the tables is identical to that in Theorem 4.5 and a labelling on a part of a middle set also implies the labelling on that part of the other middle set(s), the time analysis is the same as in Theorem 4.5. □

Corollary 4.12. *There exist a Monte-Carlo algorithm that, given a planar graph G, solves* Planar Travelling Salesman Problem *in time $\mathcal{O}(2^{5.036\sqrt{n}})$.*

Proof. Combine Theorem 4.11 with Lemma 2.6 and use $\omega < 2.373$ [21]. □

5 Rank based approach on branch decompositions

In this section the same representative four problems are chosen as in the previous section. The structure of each section is similar. First a naive algorithm is presented. Afterwards, it is indicated where the `Reduce` routine is used and the complexity analysis shows what effect this has on the worst case running time.

5.1 Steiner Tree

First a naive algorithm for weighted Steiner Tree on branch decompositions will be given. Thereafter, it is shown how to use representative sets and Gaussian elimination to improve the time complexity. The tables of this naive algorithm are the same as those in Section 3.2, the difference lies in how they are constructed.

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

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

Here, it is ensured that terminal vertices in T correspond to 1 labels, and that vertices incident to an edge in the partial solution correspond to 1 labels. It is also needed to ensure 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, 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 every entry of $A_y(s_x^L s_x^I s_F)$ with every entry of $A_z(s_x^R s_x^I s_F)$. The resulting entries in which vertices in F are in separate blocks should be discarded, since this indicates a partial solution with a connected component no longer connected to B_x, which is therefore invalid. Projecting over all vertices in F removes these entries and also makes sure the vertices in F are no longer included in partitions in A_x.

Partitions in the computed set $A_x(s)$ again correspond to the connected components of the partial solution, by definition of the `join` and `proj` operations. Also, the weights of the partitions correspond to the weights of the partial solutions, as edges are chosen 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, it is possible to 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.

The `Reduce` routine from Theorem 3.9 is applied at each step of the naive algorithm for Steiner Tree to obtain the following result.

Theorem 5.1. *There exists an algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw, solves* Steiner Tree *in time $\mathcal{O}(n((1+2^\omega)\sqrt{5})^{bw} bw^{\mathcal{O}(1)})$, where ω is the matrix multiplication exponent.*

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 3.9), 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 from Corollary 3.7. 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, i.e. $j = |s_x^{-1}(1)|$. 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, notation is slightly abused and any labelling with j vertices with label 1 is denoted as s_j. Also, $s_{i,l,f}$ denotes 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.

It is now possible to 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:

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

$$\mathcal{O}(\sum_{i=0}^{|I|}\sum_{r=0}^{|R|}\sum_{l=0}^{|L|}\binom{|I|}{i}\binom{|R|}{r}\binom{|L|}{l}2^{(\omega-1)(i+r+l)}|A_x(s_{i,r,l})|(i+r+l)^{\mathcal{O}(1)}) \leq$$

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

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

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

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

If this is maximized under the constraints in Lemma 2.5, then a worst-case running time is found 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, the time-complexity from the theorem is found. □

Corollary 5.2. *There exists an algorithm that, given a planar graph G, solves* Planar Steiner Tree *in time $\mathcal{O}(2^{8.039\sqrt{n}})$.*

Proof. Combine Theorem 5.1 with Lemma 2.6 and use $\omega < 2.373$ [21]. □

5.2 Connected Dominating Set

Theorem 5.3. *There exists an algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw, solves* Connected Dominating Set *in time $\mathcal{O}(n((2+2^{\omega})\sqrt{6})^{bw}bw^{\mathcal{O}(1)})$, where ω is the matrix multiplication exponent.*

Proof. First the naive algorithm will be discussed. Next Reduce is applied in order to achieve the required running time.

For each edge x there is a table A_x. Each entry $A_x(s)$ consists of a set of weighted partitions. These weighted partitions represent all the partial solutions for a labelling

s. A labelling $s \in \{1, 0_1, 0_?\}^{B_x}$ describes the state of the vertices associated with the edge. A label 1 means the vertex is in the solution, a label 0_1 means it is not in the solution but is dominated, a label $0_?$ means that it is not in the solution but perhaps dominated. $A_x(s)$ 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 X; and vertices are in the same block of the partition p that represents that solution X, if and only if, the vertices are in the same connected component in $G[X]$. Note that the partitions for each entry are partitions of all vertices with label 1.

For a leaf edge x of the branch decomposition $\mathbb{T}$, it holds 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):

$$\begin{aligned} A_x(0_?\ 0_?) &= \{(\emptyset, 0)\} & A_x(1\ 0_?) &= \{(\{u\}, w(u))\} \\ A_x(1\ 0_1) &= \{(\{u\}, w(u))\} & A_x(0_?\ 1) &= \{(\{v\}, w(v))\} \\ A_x(0_1\ 1) &= \{(\{v\}, w(v))\} & A_x(1\ 1) &= \{(\{uv\}, w(u) + w(v))\} \end{aligned}$$

For an internal edge x of the branch decomposition with children y and z, the table A_x is filled by combining the partial-solutions from the tables for y and z. This is done by means of the following formula:

$$A_x(s_x) = \biguplus_{\substack{s_x, s_y, s_z \\ \text{compatable}}} \texttt{proj}(F, \texttt{join}(A_y(s_y), A_z(s_z)))$$

There are different criteria that determine whether s_y and s_z are compatible with s_z for different parts of the middle sets. For a vertex v in L the labellings are compatible if $s_y(v) = s_z(v)$ and similarly for a vertex in R. For a vertex v in I the labellings are compatible if $s_y(v) = s_z(v) = s_x(v) = 1 \vee s_y(v) = s_z(v) = s_x(v) = 0_? \vee s_y(v) = s_x(v) = 0_1 \wedge s_z(v) = 0_? \vee s_z(v) = s_x(v) = 0_1 \wedge s_y(v) = 0_?$. Of course the labellings would also be compatible if $s_y(v) = s_z(v) = s_x(v) = 0_1$ but this case is covered already by the two latter cases (and will never have a better weight) so it is possible to disregard this option. For a vertex v in F the labellings are compatible if $s_y(v) = s_z(v) = 1 \vee s_y(v) = 0_1 \wedge s_z(v) = 0_? \vee s_y(v) = 0_? \wedge s_z(v) = 0_1$.

The Reduce routine can be used after computing each edge. This will be the most expensive step. Note that s_x is an arbitrary labelling having the amount of labels 1, 0_1 and $0_?$ prescribed by j_0, j_1 and j_2 respectively. The amount of vertices having each label is the only information needed in the time analysis, the actual labelling is irrelevant here, but this is a convenient notation. This is a similar abuse of notation as in Section 5.1, however, the s_{j_0,j_1,j_2} notation is not used to avoid terms like $s_{i_0,i_1,i_2,r_0,r_1,r_2,l_0,l_1,l_2}$.

$$\mathcal{O}\Big(\sum_{j_0+j_1+j_2=|I\cup R\cup L|} \binom{|I\cup R\cup L|}{j_0\ j_1\ j_2} 2^{\frac{(\omega-1)}{2} j_0} |A_x(s_x)| j_0^{\mathcal{O}(1)}\Big) =$$

$$\mathcal{O}\Big(\sum_{i_0+i_1+i_2=|I|} \sum_{r_0+r_1+r_2=|R|} \sum_{l_0+l_1+l_2=|L|} \binom{|I|}{i_0\ i_1\ i_2} \binom{|R|}{r_0\ r_1\ r_2} \binom{|L|}{l_0\ l_1\ l_2}$$
$$2^{(\omega-1)(i_0+r_0+l_0)} |A_x(s_x)| (i_0+r_0+l_0)^{\mathcal{O}(1)}\Big) =$$

$$\mathcal{O}\Big(\sum_{i_0+i_1+i_2=|I|} \sum_{r_0+r_1+r_2=|R|} \sum_{l_0+l_1+l_2=|L|} \binom{|I|}{i_0\ i_1\ i_2} \binom{|R|}{r_0\ r_1\ r_2} \binom{|L|}{l_0\ l_1\ l_2}$$
$$2^{(\omega-1)(i_0+r_0+l_0)} \sum_{\substack{s_x,\, s_y,\, s_z \\ \text{compatable} \\ \text{labellings}}} 2^{|s_y^{-1}(1)|} 2^{|s_z^{-1}(1)|} (i_0+r_0+l_0)^{\mathcal{O}(1)}\Big) =$$

The exact labellings s_y and s_z are also irrelevant, as shown below. The sum that is new in the last equation above stands for the size of a table with a labelling that meets the criteria specified by the dummy variables. These criteria are that it should contain an amount of vertices within a part of the labelling corresponding to the value of the dummy variable. For example, if $l_1 = a$ this means that labelling s_x should have a vertices in part L that have label 1. To determine the size of a table that meets these requirements it is necessary to look at the amount of options per vertex. Each part of the labelling is treated separately. This results in several product terms. When labellings are compatible has been specified above for each part of the middle sets.

$$\sum_{\substack{s_x,\, s_y,\, s_z \\ \text{compatable} \\ \text{labellings}}} 2^{|s_y^{-1}(1)|} 2^{|s_z^{-1}(1)|} =$$

$$\prod_{i=1}^{|I|} \sum_{\substack{s_y(i),\, s_z(i) \\ \text{compatable} \\ \text{with } s_x(i)}} 2^{[s_y(i)=1]} 2^{[s_z(i)=1]} \prod_{l=1}^{|L|} \sum_{s_y(l)=s_x(l)} 2^{[s_y(l)=1]}$$

$$\prod_{r=1}^{|R|} \sum_{s_z(r)=s_x(r)} 2^{[s_z(r)=1]} \prod_{f=1}^{|F|} \sum_{\substack{s_y(i),\, s_z(i) \\ \text{compatable}}} 2^{[s_y(f)=1]} 2^{[s_z(f)=1]} =$$

$$2^{2i_0} 2^{i_1} 1^{i_2}\ 2^{l_0} 1^{l_1} 1^{l_2}\ 2^{r_0} 1^{r_1} 1^{r_2}\ 6^{|F|}$$

Put this expression back into the time equation and simplify, to find:

$$\mathcal{O}(\sum_{i_0+i_1+i_2=|I|}\binom{|I|}{i_0\ i_1\ i_2}2^{(\omega+1)i_0}2^{i_0}1^{i_2}\sum_{r_0+r_1+r_2=|R|}\binom{|R|}{r_0\ r_1\ r_2}2^{\omega r_0}1^{r_1}1^{r_2}$$
$$\sum_{l_0+l_1+l_2=|L|}\binom{|L|}{l_0\ l_1\ l_2}2^{\omega l_0}1^{l_1}1^{l_2}\ 6^{|F|}\ (i_0+r_0+l_0)^{\mathcal{O}(1)})\leq$$
$$\mathcal{O}((3+2^{\omega+1})^{|I|}(2+2^{\omega})^{|L\cup R|}6^{|F|}bw^{\mathcal{O}(1)})$$

Maximize under the constraints in Lemma 2.5 to find that the worst-case occurs when $|I|=0$ and $|F|=|L|=|R|=\frac{1}{2}bw$. This gives the result from the theorem. □

Corollary 5.4. *There exists an algorithm that, given a planar graph G, solves* Planar Connected Dominating Set *in time $\mathcal{O}(2^{8.778\sqrt{n}})$.*

Proof. Combine Theorem 5.3 with Lemma 2.6 and use $\omega < 2.373$ [21]. □

5.3 Feedback Vertex Set

Theorem 5.5. *There exists an algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw, solves* Feedback Vertex Set *in time $\mathcal{O}(n((1+2^{\omega})\sqrt{5})^{bw}bw^{\mathcal{O}(1)})$, where ω is the matrix multiplication exponent.*

Proof. First the naive algorithm will be given. Next Reduce is applied in order to achieve the required running time.

To solve the problem it is necessary to redefine the problem. The aim is to find an induced subgraph that is a tree on at least $i=|V|-k$ edges. This is similar to the approach in the proof of Theorem 4.7. A universal vertex v_0 is added with edges E_0 to all other vertices. It is necessary to find a pair (Y,Y_0) such that $Y\subseteq V\setminus\{v_0\}, |Y|\leq k, Y_0\subseteq E_0$ and the graph $(V\setminus Y, E[V\setminus Y]\cup Y_0)$ is a tree. This is equivalent to the original formulation. Note that the tree must contain all edges between vertices in the solution and any edges from E_0.

The tables that need to be filled for each edge x consist of entries $A_x(s,i,j)$. Each entry contains a set of partitions. A labelling $s\in\{1,0\}^{B_x}$ describes the state of the vertices associated with the edge. A label 1 means the vertex is in the solution, a label 0 means it is not. The number of vertices in the solution are denoted by i and the number of edges in the solution are denoted by j. $A_x(s)$ 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 X; and vertices are in the same block of the partition p that represents that solution X, if and only if, the vertices are in the same connected component in $G[X]$. Note that the partitions for each entry are partitions of all vertices with label 1.

If there is an entry at the root which has $i\geq|V|-k$ vertices and $j=i-1$ edges then there is a solution. This holds because a graph with $n=m+1$ vertices and edges must be a tree.

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

$$
\begin{aligned}
A_x(0\ 0, 0, 0) &= \{\{\emptyset\}\} \\
A_x(1\ 0, 1, 0) &= \begin{cases} \emptyset & \text{if } v = v_0 \\ \{\{u\}\} & \text{otherwise} \end{cases} \\
A_x(0\ 1, 1, 0) &= \begin{cases} \emptyset & \text{if } u = v_0 \\ \{\{v\}\} & \text{otherwise} \end{cases} \\
A_x(1\ 1, 2, 0) &= \begin{cases} \{\{u/v\}\} & \text{if } u = v_0 \vee v = v_0 \\ \emptyset & \text{otherwise} \end{cases} \\
A_x(1\ 1, 2, 1) &= \{\{u\ v\}\}
\end{aligned}
$$

For an internal edge x of the branch decomposition with children y and z, fill the table A_x by combining the partial-solutions from the tables for y and z. This is done by means of the following formula:

$$
A_x(s_x, i_x, j_x) = \biguplus_{\forall s_F} \biguplus_{j_y + j_z = j_x} \biguplus_{\substack{i_y + i_z \\ -|(I \cup F) \cap s_y^{-1}(1)| \\ = i_x}} \texttt{proj}(F, \texttt{join}(A_y(s_y, i_y, j_y), A_z(s_z, i_z, j_z)))
$$

The term s_F is a labelling on the vertices in F. To ensure that no vertices are counted double the term $|(I \cup F) \cap s_y^{-1}(1)|$, the amount of vertices in $I \cup F$ with label 1, is subtracted.

It is possible to use the Reduce routine after computing each edge. This will be the most expensive step. Since the table sizes before the reduce step have the same bound as in Theorem 5.1, the time analysis and the result are also the same. □

Corollary 5.6. *There exists an algorithm that, given a planar graph G, solves* Planar Feedback Vertex Set *in time $\mathcal{O}(2^{8.039\sqrt{n}})$.*

Proof. Combine Theorem 5.5 with Lemma 2.6 and use $\omega < 2.373$ [21]. □

5.4 Traveling Salesman

Theorem 5.7. *There exists an algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw, solves* Traveling Salesman *in time $\mathcal{O}(n(5+2^{\frac{\omega+2}{2}})^{bw} bw^{\mathcal{O}(1)})$, where ω is the matrix multiplication exponent.*

Proof. First the naive algorithm will be given. Next Reduce is applied in order to achieve the required running time.

For each edge x there is a table A_x. Each entry $A_x(s)$ consists of a set of weighted partitions. These weighted partitions represent all the partial solutions for a labelling s. A labelling $s \in \{2, 1, 0\}^{B_x}$ describes the state of the vertices associated with the edge.

A label 0 means the vertex is not in the solution, a label 1 means it is in the solution and is the end of a path (it has degree 1), a label 2 means that it is in the solution and is the middle of a path (it has degree 2). $A_x(s)$ 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 $G[Y]$. Note that the partitions for each entry are partitions of all vertices with label 1. This means that there are two vertices in each block of the partition, since every path can have at most two endpoints.

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

$$A_x(0\ 0) = \{(\{\emptyset\}, 0)\} \qquad A_x(1\ 1) = \{(\{uv\}, w(u\ v))\}$$

For an internal edge x of the branch decomposition with children y and z, the table A_x is filled by combining the partial-solutions from the tables for y and z. This is done by means of the following formula:

$$A_x(s_x) = \biguplus_{s_x, s_y, s_z \text{ match}} \texttt{proj}(F \cup (s_x^{-1}(2) \cap (s_y^{-1}(1) \cup s_z^{-1}(1))), \texttt{join}(A_y(s_y), A_z(s_z)))$$

Labellings match if the labels of the vertices in L and R are identical in s_x and s_y or s_z respectively, for all vertices $v \in F$ it holds that $s_y(v) + s_z(v) = 2$ and for all vertices $v \in I$ it holds that $s_y(v) + s_z(v) = s_x(v) \leq 2$. Besides projecting over the vertices in F it is also necessary to project over all vertices with label 2 in s_x that did not have label 2 in either s_y or s_z. This is to avoid cycles in the solution.

The partitions of the vertices with label 1 are all perfect matchings. This allows the use of a different Reduce subroutine.

Theorem 5.8 ([12]). *There exists an algorithm* ReduceMatchings *that, given a set of weighted partitions $\mathcal{F} \subseteq \Pi(U) \times \mathbb{N}$, outputs in time $\mathcal{O}(|\mathcal{F}| 2^{\frac{(\omega-1)}{2}|U|} |U|^{\mathcal{O}(1)})$ a set of weighted partitions $\mathcal{F}' \subseteq \mathcal{F}$, such that $\mathcal{F}'$ represents $\mathcal{F}$ and $|\mathcal{F}'| \leq 2^{|U|/2}$, where ω denotes the matrix multiplication exponent.*

The complexity analysis is similar to that seen in Theorem 5.3. Note that s_x is an arbitrary labelling having the amount of labels 0, 1 and 2 prescribed by j_0, j_1 and j_2 respectively. The amount of vertices having each label is the only information needed in the time analysis, the actual labelling is irrelevant here, but this is a convenient notation.

This notation is also used in Section 5.2.

$$\mathcal{O}\Big(\sum_{j_0+j_1+j_2=|I\cup R\cup L|}\binom{|I\cup R\cup L|}{j_0\ j_1\ j_2}2^{\frac{(\omega-1)}{2}j_1}|A_x(s_x)|j_1^{\mathcal{O}(1)}\Big)=$$

$$\mathcal{O}\Big(\sum_{i_0+i_1+i_2=|I|}\ \sum_{r_0+r_1+r_2=|R|}\ \sum_{l_0+l_1+l_2=|L|}\binom{|I|}{i_0\ i_1\ i_2}\binom{|R|}{r_0\ r_1\ r_2}\binom{|L|}{l_0\ l_1\ l_2}$$
$$2^{\frac{(\omega-1)}{2}(i_1+r_1+l_1)}|A_x(s_x)|(i_1+r_1+l_1)^{\mathcal{O}(1)}\Big)=$$

$$\mathcal{O}\Big(\sum_{i_0+i_1+i_2=|I|}\ \sum_{r_0+r_1+r_2=|R|}\ \sum_{l_0+l_1+l_2=|L|}\binom{|I|}{i_0\ i_1\ i_2}\binom{|R|}{r_0\ r_1\ r_2}\binom{|L|}{l_0\ l_1\ l_2}$$
$$2^{\frac{(\omega-1)}{2}(i_1+r_1+l_1)}\sum_{\substack{s_x,\,s_y,\,s_z\\ \text{compatable}\\ \text{labellings}}}2^{\frac{|s_y^{-1}(1)|}{2}}2^{\frac{|s_z^{-1}(1)|}{2}}(i_1+r_1+l_1)^{\mathcal{O}(1)}\Big)=$$

The exact labellings s_y and s_z are also irrelevant, as shown below. The sum that is new in the last equation above signifies the size of a table with a labelling that meets the criteria specified by the dummy variables. These criteria are that it should contain an amount of vertices within a part of the labelling corresponding to the value of the dummy variable. Each part of the labelling will be dealt with separately. This results in several product terms.

$$\sum_{\substack{s_x,\,s_y,\,s_z\\ \text{compatable}\\ \text{labellings}}}2^{\frac{|s_y^{-1}(1)|}{2}}2^{\frac{|s_z^{-1}(1)|}{2}}=$$

$$\prod_{i=1}^{|I|}\sum_{s_y(i)+s_z(i)=s_x(i)}2^{[s_y(i)=1]/2}2^{[s_z(i)=1]/2}\prod_{l=1}^{|L|}\sum_{s_y(l)=s_x(l)}2^{[s_y(l)=1]/2}$$
$$\prod_{r=1}^{|R|}\sum_{s_z(r)=s_x(r)}2^{[s_z(r)=1]/2}\prod_{f=1}^{|F|}\sum_{s_y(f)+s_z(f)=2}2^{[s_y(f)=1]/2}2^{[s_z(f)=1]/2}=$$
$$1^{i_0}2^{\frac{3}{2}i_1}4^{i_2}\ 1^{l_0}2^{\frac{1}{2}l_1}1^{l_2}\ 1^{r_0}2^{\frac{1}{2}r_1}1^{r_2}\ 4^{|F|}$$

For each vertex v it is needed to look at options for $s_y(v)$ and/or $s_z(v)$ if $s_x(v)$ is set. The last equation in the time analysis is obtained by using the values computed below. For a vertex in I:

$$s_x(v)=0: 2^{0/2}2^{0/2}=1$$
$$s_x(v)=1: 2^{1/2}2^{0/2}+2^{0/2}2^{1/2}=2^{3/2}$$
$$s_x(v)=2: 2^{1/2}2^{1/2}+2^{0/2}2^{0/2}+2^{0/2}2^{0/2}=4$$

For a vertex in L (and idem for a vertex in R):

$$s_x(v) = 0 : 2^{0/2}$$
$$s_x(v) = 1 : 2^{1/2}$$
$$s_x(v) = 2 : 2^{0/2}$$

For a vertex in F:

$$2^{1/2}2^{1/2} + 2^{0/2}2^{0/2} + 2^{0/2}2^{0/2} = 4$$

If this expression for the size of a table back is put into the time equation and the equation is simplified, the result is:

$$\mathcal{O}(\sum_{i_0+i_1+i_2=|I|} \binom{|I|}{i_0\ i_1\ i_2} 1^{i_0} 2^{\frac{\omega+2}{2} i_1} 4^{i_2} \sum_{r_0+r_1+r_2=|R|} \binom{|R|}{r_0\ r_1\ r_2} 1^{r_0} 2^{\frac{\omega}{2} r_1} 1^{r_2}$$
$$\sum_{l_0+l_1+l_2=|L|} \binom{|L|}{l_0\ l_1\ l_2} 1^{l_0} 2^{\frac{\omega}{2} l_1} 1^{l_2}\ 4^{|F|}\ (i_1 + r_1 + l_1)^{\mathcal{O}(1)}) \leq$$
$$\mathcal{O}((5 + 2^{\frac{\omega+2}{2}})^{|I|}(2 + 2^{\frac{\omega}{2}})^{|L\cup R|} 4^{|F|} bw^{\mathcal{O}(1)})$$

When maximizing under the constraints in Lemma 2.5, it is possible to conclude that the expression is maximized when $|I| = bw$ and all other sets are empty. This gives the result from the theorem. □

Corollary 5.9. *There exists an algorithm that, given a planar graph G, solves* Planar Travelling Salesman *in time $\mathcal{O}(2^{6.570\sqrt{n}})$.*

Proof. Combine Theorem 5.5 with Lemma 2.6 and use $\omega < 2.373$ [21]. Because $|I|$ is small the expression determining the running time is now maximized when $|F| = |L| = |R| = 1/2bw$. □

6 Conclusion

In this thesis, two things are shown. First of all, it is 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. This could be a possibility for further research.

The thesis also presents 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.

Further research could also focus on the possibility of improving the deterministic algorithms. It might be possible to apply the techniques by Fomin et al. [19, 18], using

representative sets and matroids, to branch decompositions. When this is possible, chances are that the algorithms will be faster than the deterministic algorithms from this thesis, since their counterparts on tree decompositions are faster then the rank based algorithms from Bodlaender et al. [6].

Acknowledgements

I would like to thank my two supervisors Johan en Hans for their faith in me, their time, their patience, and their insights. All of this culminated in writing a paper together for the IPEC 2016 conference, a result that I am proud of.

I owe my friends for all the times they pretended to understand or care when listening to my ramblings about graphs and complexity. My family also earned my gratitude for supporting me and not making me feel like ten months was an exorbitantly long time to write a thesis.

Finally, I am indebted to my muse for all the inspiration she gave me.

References

[1] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: Fast subset convolution. In *Proceedings of the 39th Annual Symposium on Theory of Computing, STOC 2007*, pages 67–74, 2007.

[2] Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–23, 1993.

[3] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[4] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998.

[5] 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.

[6] 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.

[7] 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.

[8] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Boolean-width of graphs. *Theoretical Computer Science*, 412:5187–5204, 2011.

[9] James A Carlson, Arthur Jaffe, and Andrew Wiles. *The millennium prize problems*. American Mathematical Society, 2006.

[10] Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218–270, 1993.

[11] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[12] Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast Hamiltonicity checking via bases of perfect matchings. In *Symposium on Theory of Computing Conference, STOC 2013*, pages 301–310, 2013.

[13] 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.

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

[15] Frederic Dorn. How to use planarity efficiently: New tree-decomposition based algorithms. In Andreas Brandstädt, Dieter Kratsch, and Haiko Müller, editors, *Proceedings of the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2007*, pages 280–291. Springer Verlag, Lecture Notes in Computer Science, vol. 4769, 2007.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

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

[22] Qian-Ping Gu and Hisao Tamaki. Optimal branch-decomposition of planar graphs in $O(n^3)$ time. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming, ICALP 2005*, pages 373–384. Springer Verlag, Lecture Notes in Computer Science, vol. 3580, 2005.

[23] Ton Kloks. *Treewidth. Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, 1994.

[24] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011*, pages 777–789, 2011.

[25] Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 345–354. ACM, 1987.

[26] Sang-il Oum. Rank-width and vertex-minors. *Journal of Combinatorial Theory, Series B*, 95(1):79–100, 2005.

[27] Willem J.A. Pino, Johan M. M. van Rooij, and Hans L. Bodlaender. Cut and count and representative sets on branch decompositions. In *Proceedings of the 9th International Symposium on Parameterized and Exact Computation, IPEC 2016*, 2016. Accepted.

[28] Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36:49–64, 1984.

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

[30] Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM Journal of Discete Mathematics*, 10:529 – 550, 1997.

[31] 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*, pages 566–577. Springer Verlag, Lecture Notes in Computer Science, vol. 5757, 2009.