Parameterized Approximation Algorithms for Optimization Problems on Graphs

Dissertation (monograph) approved by the Doctoral Degree Committee of Hamburg University of Technology

in pursuit of the academic degree of

Doktor der Naturwissenschaften (Dr. rer. nat.)

written by
Matthias Kaul

from
Ostfildern

2025

Examiners:

Prof. Dr. Anusch Taraz
Prof. Dr. Matthias Mnich
Prof. Dr. Hadas Shachnai
Prof. Dr. Joachim Spoerhase

Defended December 10th 2024

Matthias Kaul, 2025 https://doi.org/10.15480/882.16206

Contents

Summary

In this thesis we study *Parameterized Approximation Algorithms* for network optimization problems. *Approximation* here means that we seek algorithms that are not guaranteed to compute optimal solutions, but merely solutions which are quantifiably good in the sense of being close to optimal; *Parameterized* expresses the fact that we do not wish to solve all instances of a given problem equally well. Rather we come up with some parameter measuring the complexity of an instance and then only require that the algorithm be fast for instances where this parameter is small.

Specifically, we treat three problems:

The Multi-Depot Traveling Salesperson Problem In the traditional Traveling Salesperson Problem (TSP) we are given a set of cities V with a metric distance function $d : V \times V \to \mathbb{R}_+$ and required to compute a cyclical ordering $v_1, \ldots, v_n, v_{n+1} = v_n$ of the cities (a *tour*) such that visiting them in that order minimizes the travel time $\sum_1^n d(v_i, v_{i+1})$. For the Multi-Depot version (MDTSP) we instead compute a collection of tours such that every city is visited, and every individual tour starts at one of a set of specified depot cities $D \subseteq V$.

For this problem we can give for every $\varepsilon > 0$ an algorithm that computes a solution that is at most a factor $(3/2+\varepsilon)$ longer than an optimal solution in time at most $(1/\varepsilon)^{\mathcal{O}(|D| \log |D|)}|V|^{\mathcal{O}(1)}$. This continues and improves on a line of work initiated by Xu and Rodriguez [1] to re-obtain the polynomial time $(3/2)$-approximation for TSP first given by Christofides and Serdyukov, and it is the first better-than-2 approximation algorithm in FPT time.

All-Norm Tree Cover As a variation on the MDTSP we also consider the setting where the tours should have similar sizes, although for simplicity we do not consider the tours to be cycles but trees. We measure the similarity of the sizes by taking the vector of tree sizes and considering its ℓ_p norm. Under this objective we can give a polynomial-time approximation algorithm achieving constant approximation ratio with respect to *all* p-norms simultaneously, and in fact all monotone symmetric norms.

We supplement this result by also giving a constant factor all-norm approximation algorithm for the setting without depots, as well as proving APX-hardness for Tree Cover with Depots with respect to any fixed ℓ_p norm. Thus our results are optimal up to the exact constant of approximation achievable.

The Sparsest Cut Problem In the Sparsest Cut Problem we are given a vertex set V with two edge sets E_C, $E_D \subseteq V \times V$ and a corresponding pair of weights $c : E_C \to \mathbb{R}_+$, $d : E_C \to \mathbb{R}_+$. The goal is to partition the vertices into two parts $X, V \setminus X$ such that few edges from E_C are crossing between them, but many from E_D. Formally we seek to minimize

$$\sum_{e \in E_C \cap \delta(X)} c(e) \frac{\sum_{e \in E_C \cap \delta(X)} c(e)}{\sum_{e \in E_D \cap \delta(X)} d(e)}.$$

The problem is known to be hard to even approximate well in polynomial time in general graphs. However we are able to give a variety of trade-offs between runtime and approximation quality for the case where the graph (V, E_C) has *treewidth* at most some constant k:

- A $\mathcal{O}(k^2)$-approximation in time $2^{\mathcal{O}(k)} \cdot |V|^{\mathcal{O}(1)}$.
- A $\mathcal{O}(1)$-approximation in time $2^{\mathcal{O}(k)} \cdot |V|^{\mathcal{O}(1)}$.
- For any $\varepsilon \in (0, 1]$, a $\mathcal{O}(1/\varepsilon^2)$-approximation in time $2^{\mathcal{O}(\frac{k^{1+\varepsilon}}{\varepsilon})} \cdot |V|^{\mathcal{O}(1)}$.

In particular we thus obtain a constant-factor approximation in time single exponential in the treewidth, resolving a major open question of Chlamtáč et al. who initiated this line of research [2].

Chapter 1

Introduction

A lot of the problems computer science is interested in have been shown to be NP-hard, even quite early in the field's development [3, 4]. For these problems it is believed that no "efficient" algorithms exist, meaning that any procedure solving such problems in general will need an amount of time that scales superpolynomially in the size of the problem instance. This would make it effectively impossible to solve these problems at large scales.

In reality this is not a satisfying resolution of the issue. If we are to believe $\mathsf{P} \neq \mathsf{NP}$ some problems can not be solved quickly; Nonetheless they need to be solved in practice. We thus have to figure out where our model of complexity fails to conform to real world interests in the hope that changing that model to more accurately reflect reality will also render some problems tractable again.

Indeed the demands on an algorithm solving a typical NP-hard problem in polynomial time are very strict. We are asked to come up with *one* algorithm that computes the *best possible* solution to *all* instances of the problem. But real-world engineering rarely works that way. Solutions are designed for concrete instances of problems, factoring in everything known about the given instance. Engineers also rarely pretend to have come up with the absolute best solution. If the demands of a customer are met, or if the given solution is empirically better than all other existing solutions this is more than enough.

To observe this in the wild it suffices to consider the many commercial solvers for Integer Linear Programming (ILP) which is nominally NP-hard. The existing theoretical algorithms for general ILP would suggest that instances with more than 100 variables should not be solvable in reasonable time frames. However good solvers are regularly able to solve instances with tens of thousands of variables in mere minutes. But this is not true for all instances. It is possible to come up with fairly small instances of ILP that will occupy even a very good solver for days. However thanks to (sadly proprietary) engineering magic these hard instances are not, apparently, the typical instances such a solver encounters.

Thus it is apparently possible to give solutions to hard problems in a reasonable time frame, just not always. So perhaps it is precisely the rigorous adherence to *optimal* solutions to very *general* problems which has given rise to the hardness of so many relevant problems, and that consequently

some of our assumptions have to give.

On the one hand, we may choose not to solve such difficult problems exactly; very often it is only hard to provide a solution that is optimal, but if we are willing to settle for a solution whose quality is merely close to optimal the problem may become tractable again. For such problems we can then once again provide efficient *approximation algorithms* which promise to compute a solution whose quality adheres to some guarantee, i.e. does not deviate too much from optimality.

On the other hand, if we wish to compute optimal solutions, we may choose to restrict the instances of our problem that we want to be able to handle. For many NP-hard problems special cases can be solved efficiently, and it is only the most general problem that is difficult. Thus characterizing which properties of an instance render a problem "easy" allows us to devise *parameterized algorithms* that exploit these properties to compute optimal solutions more quickly than would otherwise be possible by restricting the undesirable (super-polynomial) growth of the run time to some parameter.

It is the intersection of these two approaches that we mostly concern ourselves with in this thesis. For many problems even such relaxed views are not enough on their own to render them tractable, and there is a rich theory of computational hardness in both fields underpinning this. For such problems we may need to combine both approaches giving rise to *parameterized approximation algorithms*, a unified framework of discussion for which problems allow us to compute approximately optimal solutions in polynomial time on instances where some parameter has bounded size.

Chapter 2

Preliminaries

We will take a brief moment to introduce formally some of the central concepts used throughout this thesis. For a more in-depth exploration of these concepts an interested reader should consult some standard textbooks on the matters, such as Parameterized Algorithms by Cygan et al. [5], Vazirani's Approximation Algorithms [6], or the excellent surveys of Marx and Feldmann et al. [7, 8].

Some familiarity with foundational structures such as graphs and some basic complexity classes will be assumed in the following, but for completeness we will give a few very brief definitions here before moving on to the more central concepts.

Graphs We call a *graph* a pair of a *finite* set of vertices V and a set of edges $E \subseteq \binom{V}{2}$. Unless otherwise specified graphs have no loops, i.e. edges $\{v, v\}$, nor do they contain parallel edges, i.e. E is a set, not a multiset. A graph is said to be weighted if there is a function $w : E \to \mathbb{R}_+$ assigning weights to the edges. These weights are called *metric* if they form a metric, that is if they fulfill the following properties:

- $w(e) > 0$ for all $e \in E$ and
- For any closed walk $P = e_1, e_2, \ldots, e_k$ we have $\sum_{i=1}^{k-1} w(e_i) \geq w(e_k)$.

The second point here is simply the natural extension of the triangle inequality noting that G might not have proper triangles. In general any such metrically weighted graph G can be reinterpreted as a metric space (V, d) by taking $d(v, w) = dist_G(v, w)$, where $dist_G$ is the shortest path distance in G. This is referred to as the metric closure of G, and we may even perform this transformation if w was not originally metric.

In this way any weighted graph has an associated (finite) metric space, and conversely every finite metric space (V, d) can be represented as a metrically weighted graph by taking $G = (V, \binom{V}{2})$ and $w \equiv d$. This identification is often useful, since it allows us the flexibility of general metric spaces while also maintaining some ability to discuss more graph-like objects such as subgraphs, spanning trees, or connectedness.

2.1 Parameterized Algorithms

A *language* L over some finite alphabet Σ is a subset of all the finite strings that can be written with symbols from Σ, i.e. $L \subseteq \Sigma^*$. Then the *decision problem* associated to such a language is that of deciding whether a given string $x \in \Sigma^*$ is contained in L. We call P the set of languages for which a deterministic Turing Machine running in polynomial time can solve the decision problem, and NP the set of languages decidable in polynomial time by a non-deterministic Turing Machine. For example L might be the set of all strings over $\{0, 1\}$ that correspond to primes when interpreted as binary numbers. The associated decision problem to this language then is to decide whether a given number in binary representation is prime, which rather famously was shown to be in P by Agrawal, Kayal and Saxena [9]. We will usually care about the time and possibly the space it takes for a Turing Machine to decide this, although for convenience we do not work on Turing Machines directly. Instead we refer to these as algorithms and describe them in pseudo code so as to allow for a simple translation of the described procedure into a program on a physical machine.

To allow for *parameterization* in this framework we extend this notion of languages to *parameterized languages*.

Definition 2.1. For some finite alphabet Σ we call L a parameterized language if L is a subset of $\Sigma^* \times \mathbb{N}$. Given a specific word $(x, k) \in L$ we usually refer to x as the *instance* and k as the *parameter*.

This definition can seem fairly abstract, but one may imagine for concreteness that L is the language of words (G, k) where G is (a description of) a graph, and k is the size of a largest independent set in G. In this example we would then see that the words $(G, 1)$ correspond to the complete graphs, the words $(G, 2)$ to those who are not complete, but whose complement is triangle-free, and so on.

In particular this allows us to segment a potentially very hard problem such as independent set into many subproblems which might be easier individually. Deciding whether a word $(G, 1)$ is in L is possible with run time linear in $|G|$; deciding the same for a word $(G, 2)$ is possible in time $\mathcal{O}(n^2)$. Generally, for a word (G, k) we would be able to decide its membership in L by enumerating all subsets of k vertices and checking whether they induce an independent set, as well as checking all subsets of size $k+1$ to see that no larger independent set exists. Doing this is possible in time $\mathcal{O}(\binom{n}{k+1}) \subseteq n^{\mathcal{O}(k)}$.

We can notice that this is a polynomial run time, at least for each value of k individually. Thus for any small value of k we actually have a polynomial-time algorithm for deciding whether a word (G, k) is in L.

This insight gives rise to the first class of efficiency under a parameterized viewpoint, that of having a *slice-wise polynomial-time* algorithm.

Definition 2.2. We denote with XP the class of all parameterized languages L for which there exists a computable function f and an algorithm deciding whether (x, k) is in L running in time $\mathcal{O}(|x|^{f(k)})$. For such algorithms we say that they have slice-wise polynomial run time, or XP run time.

Such XP-algorithms are more efficient than algorithms running in exponential time, however they quickly become unusably slow. Even if the function f grows slowly, such as $f(k) = k$, we can not expect to be able to use them for values of k much larger than 10 if n is appreciably large. We therefore really want a stronger notion of efficiency, one where the run time dependence on the instance size is a uniform polynomial, and merely the coefficients of that polynomial depend on the parameter. We will call algorithms with such run time behaviour *fixed-parameter tractable* (FPT).

Definition 2.3. We denote with FPT the class of all parameterized languages L for which there exist a constant $c \in \mathbb{R}$, a computable function f, and an algorithm deciding whether (x, k) is in L running in time $\mathcal{O}(f(k) \cdot n^c)$. For such algorithms we say that they have fixed-parameter tractable run time, or FPT run time.

This notion of tractability then comes associated with it's own version of polynomial-time reductions, i.e. transformations between parameterized languages that preserve class membership:

Definition 2.4. Let L, L' be two parameterized languages over the same alphabet Σ. We call a function $f : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ a *parameterized reduction* from L to L' if:

- $f((x, k)) \in L'$ if and only if $(x, k) \in L$,
- $f((x, k))$ can be computed in FPT time parameterized on k, and
- there exists some computable function g and constant c such that if $f((x, k)) = (x', k')$ we have $|x'| \in \mathcal{O}(|x|^c)$ and $k' \leq g(k)$.

The definition here is chosen precisely as to allow us to conclude that if L has a parameterized reduction to L' and L' is in FPT, then L is also in FPT.

The folklore example for a parameterized language whose associated decision problem has such an FPT-algorithm is Vertex Cover parameterized with the size of a solution (see for example [5] for an in-depth treatment). Given a graph G and a parameter k the Vertex Cover problem asks us to decide whether there exists a set C of at most k vertices such that every edge of G is incident to at least one vertex from C. To solve this in time $\mathcal{O}(f(k) \cdot n)$ we can run the following well known algorithm:

1. Collect all vertices in G which have degree $> k$, add them to C, and delete them from G to obtain a smaller graph G'.
2. Remove all vertices of degree 0 from G'
3. If G' has more than $k(k-1)$ vertices, assert that G has no VERTEX COVER of size at most k and terminate.
4. In G' enumerate all vertex sets of size at most $k - |C|$ and check if one of them is a VERTEX COVER of G'. If so assert that G has a VERTEX COVER of size at most k, otherwise assert that it does not.

The algorithm works by observing that any VERTEX COVER of size $\leq k$ must contain all high degree vertices, since if a vertex is not contained in the cover all of its neighbours are. After choosing all such vertices to be a part of C we know that in G' every vertex has degree less than k, so every further vertex in C can at most cover edges to $k-1$ neighbours. So if G' has more than $k(k-1)$ vertices at this point, it is impossible to cover all edges with only k vertices and we can terminate the algorithm. Otherwise we know that in the fourth step G' must be very small, i.e. it can have at most $k(k-1)$ vertices. It is then possible to enumerate all subsets of the vertices of G' to find a solution by brute force.

We thus see that the outlined algorithm can be implemented to run in time $\mathcal{O}(n + k^{2k})$, thus achieving the promised property of having FPT run time. In fact it appears to even achieve a stronger property of running in time $\mathcal{O}(f(k) + n^c)$ rather than $\mathcal{O}(f(k) \cdot n^c)$. However it turns out that demanding run times $\mathcal{O}(f(k) + n^c)$ in Definition 2.3 does not actually give rise to a stronger notion of FPT-ness

Observation 2.1. *Any parameterized language L that has an algorithm deciding the membership of words (x, k) in time $\mathcal{O}(f(k) \cdot |x|^c)$ also has an algorithm deciding it in time $\mathcal{O}(g(k) + |x|^{c'})$ for an appropriate constant c' and some computable function g.*

Proof. Suppose a given instance (x, k) has the property that $|x| \geq f(k)$. The promised FPT algorithm runs in time $\mathcal{O}(|x|^{c+1})$ on such instances. On instances with $|x| < f(k)$ it instead runs in time $g(k) = f(k)^{c+1}$. So indeed the same algorithm can be considered to run in time $\mathcal{O}(g(k) + n^{c+1})$. □

This sort of trick playing off the size of the instance against the parameter might not seem too meaningful, but it is a fairly common motif in the analysis of FPT algorithms, and give a good hint towards the qualitative performance difference between FPT and XP run times. Any given FPT algorithm will run in polynomial time even for values of k which scale sufficiently slowly in $|x|$, e.g. for $k \approx f^{-1}(|x|)$; The same is not true for XP algorithms.

It is then often said that in the context of parameterized algorithms the class FPT corresponds to P, whereas XP is the analogue of NP. That is problems in FPT have "efficient" parameterized algorithms, whereas problems in XP\FPT do not.

Parameterized Complexity

To make this more concrete, we will cast a (very) brief look at the fundamentals of parameterized complexity to understand the limits of which problems can have FPT algorithms. Firstly, there is a class of problems which are para-NP-hard, short for parameterized NP-hard. These are the problems that essentially do not benefit at all from the introduction of a given parameter. For instance one may look here at Graph Coloring parameterized on the number of colors. It is well known that already deciding if a graph is 3-colorable is NP-hard [4], so k-coloring parameterized against k must be para-NP-hard. This in means that unless P=NP there can not even be a XP algorithm for this problem, since it would run in $n^{f(k)}$ which for $k = 3$ is a polynomial. These kinds of para-NP-hard problems are in some sense outside the scope of parameterized algorithms. Their choice of parameter confers no additional exploitable information onto a problem.

The "interesting" distinction then is whether a problem admits FPT run times, or only XP run times. To obtain some formal handle on this distinction Downey and Fellows [10] introduced the W-hierarchy (short for weft), a nested sequence of supposedly increasingly hard parameterized languages. The class W[t] is defined to be those parameterized languages whose decision problem is reducible to the problem of deciding whether a Boolean circuit from some bounded-depth family has an accepting input with exactly k variables set to 1, where k is the parameter of the language. The t then further bounds the number of unbounded-weft[1] gates allowed on any input-output path in the circuit class. As we increase t then the class of circuits we can capture grows, thus probably increasing the difficulty of deciding the existence of an accepting input with k ones. The W[t]-complete problems are those that are in W[t] and to which the circuit decision problem can be reduced.

One can immediately observe that all problems in W[t] are at least in XP; we can simply enumerate all choices of k input variables and test if setting some such choice to 1 yields a satisfying assignment. For known inputs the circuits output value can be computed in polynomial time. However it is generally believed that already W[1]-complete problems do not admit FPT algorithms[2]. Specifically it is known that k-Clique (i.e. Clique

[1] Where *weft* is the number of inputs a logical gate has.

[2] One can quickly observe that the problems in W[0] have FPT algorithms, since bounded depth and everywhere bounded weft already imply a bounded number of vari-

parameterized against the size of the clique) is W[1]-complete, for which no FPT algorithm appears to exist. Similarly the inclusions between different levels of the W-hierarchy are believed to be strict, i.e. the conjectured complexity landscape is:

$$\mathsf{FPT} \subsetneq \mathsf{W}[1] \subsetneq \mathsf{W}[2] \subsetneq \dots$$

So far there has been no success in directly connecting to P≠NP, except to the extent that P=NPwould immediately imply the collapse of the entire W-hierarchy to P. However under more quantitative complexity assumptions one can reconnect to non-parameterized complexity theory. Specifically assuming the Exponential Time Hypothesis (ETH) [11] one can show that $\mathsf{FPT} \neq \mathsf{W}[1]$, giving further moral support to this separation [12].

We shall leave the discussion of parameterized complexity here, even if it is very much incomplete. We will in the following only use quite a surface view of the topic, however the interested reader should feel invited to dive deeper into the topic of parameterized complexity, for instance by using the book by Flume and Grohe on the topic [13].

2.2 Approximation Algorithms

To understand what an approximation algorithm is we first need to consider what optimization problems are in general. An optimization problem will again be a language L over some fixed alphabet Σ whose words are the instances of the problem. Additionally there is for each word x in L a set of possible solutions S_x with an associated evaluation function $f_x : S_x \to \mathbb{R}_+$. The goal of such a problem is to compute for a given x a solution $s^* \in S_x$ that is optimal in the sense that $f_x(s^*) = \min\{f_x(s) \mid s \in S_x\}$[3].

For concreteness we can consider L to be the set of all words specifying some weighted undirected graph G, S_G the collection of orderings of the vertices of G, and $f_G(\sigma)$ the length of a shortest tour visiting the vertices of G in the order given by σ. In this example we then recover the Traveling Salesperson Problem (TSP) as our optimization problem.

Optimization problems and decision problems are closely linked. If we again look at the Vertex Cover problem we can either consider it as a decision problem, i.e. deciding whether a Vertex Cover of size at most k exists, or as an optimization problem, i.e. finding the smallest Vertex Cover. If we can solve the optimization problem this also immediately gives a solution to the decision version. But the reverse is also true; if we

ables contributing to the output.

[3]We default here to considering minimization problems rather than maximization. The same kind of questions also exist for maximization problems, but we omit this for brevity.

can solve the associated decision problem for each k we just try all values for k from 1 to $|V(G)|$. The smallest value for which the solution to the decision problem still asserts the existence of a VERTEX COVER of that size must be the optimum.

These kinds of reductions between optimization and decision work for many problems, essentially by taking an optimization problem with language L and subdividing it into languages

$$L_c = \{x \in L \mid \exists s \in S_x : f_x(s) \leq c\}$$

for any $c \in \mathbb{R}_+$. Solving the decision problem for all L_c would then allow us to find optimal solutions to instances of L. As long as there is a sufficiently good bound on the number of possible values of f_x for a given instance x this will be possible with polynomial overhead.

Despite their close relationship there is key benefit of considering optimization problems over decision problems. Decision problems require a binary yes/no answer; for an optimization problem it makes sense to talk about *approximation*, i.e. instead of returning an optimal solution to an instance x we can try to find a $s^* \in S_x$ that is merely close to optimal, meaning that $f_x(s^*) \leq \alpha \min\{f_x(s) \mid s \in S_x\}$ for some choice of $\alpha \geq 1$. This viewpoint allows us to generate an infinite number of new approximate optimization problems, one for each choice of α, some of which might be tractable even if the original optimization problem was not.

With this viewpoint in mind we can now understand what an approximation algorithm is:

Definition 2.5. For an optimization problem with underlying language L and a function $\alpha : L \to \mathbb{R}_+$ we say that an algorithm is an *α-approximation Algorithm* if it computes for each $x \in L$ a $s^* \in S_x$ such that

$$f_x(s^*) \leq \alpha(x) \min\{f_x(s) \mid s \in S_x\}.$$

In most cases we will want to give a constant α here, independent of the instance. However we keep open the option of scaling the approximation ratio with the instance size, since there are approximation problems where this is necessary (under $\mathsf{P}\neq\mathsf{NP}$) if one wants to obtain polynomial run times. For the more usual case of a constant α we refer to it as the *approximation factor* or *approximation ratio* of the algorithm.

This definition conforms broadly to the basic intuition of approximation that one might have, and it is by far the most popular way of considering it, at least in the context of algorithm design. It is however worth noting that there is an alternative approach using the idea of reducing an optimization problem to a collection of decision problems can also be used, and it has been a very fruitful approach in studying the hardness of approximation.

Definition 2.5 (Alternative)**.** For an optimization problem with underlying language L and a function $\alpha : L \to \mathbb{R}_+$ a *α-approximation Algorithm* reports

- for a word $x \in L_c$ that $x \in L_{\alpha(x)\cdot c}$,
- for a word $x \notin L_{\alpha(x)\cdot c}$ that $x \notin L_{\alpha(x)\cdot c}$,
- for a word $x \in L_{\alpha(x)\cdot c} \setminus L_c$ $x \in L_{\alpha(x)\cdot c}$ or $x \notin L_{\alpha(x)\cdot c}$.

This might appear to be a slightly odd definition, in particular the algorithm might make incorrect assertions about membership in $x \in L_{\alpha(x)\cdot c}$. It is useful to imagine that the algorithm is in some way promised that it's input is going to be either in L_c or not even in $x \in L_{\alpha(x)\cdot c}$, and that it only needs to distinguish these two settings. If that promise is broken then, the algorithm is free to report whatever it wants.

This point of view of approximation as distinguishing between being in a language or being "very far" from the language has seen very nice recent advances, for example in the context of Promise Constraint Satisfaction Problems [14, 15, 16, 17]. So while we will rely on the more "traditional" definition, this is a viewpoint worth keeping in mind.

Hardness of Approximation

Let us also briefly touch upon the complexity theory for approximation algorithms. Such a discussion will almost necessarily have to be surface level, lest we be swallowed in an extremely deep field in its own right. However we will occasionally need to make use of some of its results, so we shall at least take a look at the basics.

To understand that some problems can not allow polynomial-time α-approximation algorithms for arbitrarily small values of α it suffices to take a look at, for example, the Bin Packing problem:

Given a set of items $x_1, \ldots, x_n \in [0, 1]$ we want to find the smallest number of bins $B_i \subseteq \{1, \ldots, n\}$ such that every item is in some bin, i.e. we want that $\bigcup B_i = \{1, \ldots, n\}$, and every bin contains items of total volume at most 1, meaning $\sum_{j \in B_i} x_j \leq 1$. The objective function here is the number of bins. In particular this means that the objective function is discrete. However because of this any α-approximation algorithm with $\alpha < \frac{3}{2}$ will be able to distinguish whether a given instance can be packed into two bins, or if at least 3 bins are needed. Since this solves Partition it is (weakly) NP-hard to compute a better-than-(3/2)-approximation for Bin Packing.

However these kind of ad-hoc reductions to NP-complete decision problems are severely limiting in proving hardness of approximation for some problem. Thus much of the theory instead relies on one of the cornerstone results of theoretical computer science, the PCP-theorem [18]. On possible

viewpoint of this result is that it shows NP-hardness for α-approximately maximizing the number of satisfied clauses of some *constraint satisfaction problem* (CSP[4]) for certain values of α.

This beautiful result then is the source for a lot of the known hardness-of-approximation results, and we shall rely on it ourselves to derive some results in this thesis (see section 4.5.3). In spite of this it is perhaps viewed as a bit arcane by many researchers in algorithm design. So if nothing else let this serve as a invitation to read Irit Dinurs new proof of the PCP-theorem which is thoroughly enjoyable [19].

The Algorithm of Christofides-Serdyukov

We now turn to a concrete example of an approximation algorithm which we anyways need again. The Traveling Salesperson Problem asks us to compute a shortest route for some salesperson that wants to visit a collection of cities and finally return to their starting point. More formally we are given a metric space (X, d) consisting of a finite set X representing the cities as well as distances $d : X \times X \to \mathbb{R}_+$ between the cities fulfilling the triangle inequality, i.e. for all $x, y, z \in X$ we have $d(x, z) \leq d(x, y) + d(y, z)$. We are then asked to compute a ordering $x_1, \ldots, x_n, x_{n+1} = x_1$ of the points in X that minimizes the tour length, expressed as

$$\sum_{i=1}^{n} d(x_i, x_{i+1}).$$

The TSP is among the original list of 21 NP-hard problems put forward by Richard Karp in 1972 [4]. It is thus unlikely that there exists an algorithm with polynomial run time that solves arbitrary instances of TSP in time bounded by a polynomial in $|X|$. Under the stronger assumption of the Exponential Time Hypothesis (ETH) it can even be shown that TSP cannot be solved in time $2^{\gamma(|X|)}$ [11]. Even when relaxing the requirements to only computing an α-approximation the problem is known to remain NP-hard for $\alpha \leq 123/122$ due to Karpinski et al. [20].

However the TSP is also host to one of the most well-known approximation algorithms, independently developed by Nicos Christofides and Anatoly Serdyukov in the late seventies [21, 22]. It exploits a way to represent the tours of a metric space (X, d) not as an ordering of the vertices, but as a connected Eulerian multigraph $G = (X, E)$. The visitation order of the vertices when traversing an Eulerian tour of G will give a tour of the metric space, and any tour $\{x_1, \ldots, x_{n+1}\}$ can then be represented as the graph $(X, \{\{x_1, x_2\}, \{x_2, x_3\}, \ldots\})$. This representation as a graph allows

[4]On may for simplicity imagine here that a CSP is a generalized kind of satisfiability problem.

us to separate the TSP into two much easier problems; computing a graph that is connected, and making sure that it is Eulerian.

To ensure connectedness the algorithm of Christofides-Serdyukov starts by computing a spanning tree T of minimum weight for (X, d), where we interpret a metric space as a weighted complete graph on X with edge weights $w(\{v, w\}) := d(v, w)$. Since every optimal TSP-solution G_{OPT} contains a spanning tree of (X, d) after deleting at least one edge we know that $w(T) \leq w(E(G_{OPT}))$.

We can then extend this tree T to be Eulerian by collecting all vertices O of odd degree in T and computing a perfect matching M of minimum weight with respect to d between them using the algorithm of Edmonds [23]. The graph $(X, E(T)\dot{\cup}M)$ is then connected and Eulerian by construction, and the weight of the T is at most that of an optimal tour. To bound the cost of the M we can observe that every TSP-tour contains two disjoint matchings for every even-cardinality set A of vertices by ordering them as $A = \{a_1, \ldots, a_k\}$ in the order that they appear when traversing the tour, and then taking either $\{x_1, x_2\}, \{x_3, x_4\}, \ldots$ or $\{x_2, x_3\}, \{x_4, x_5\}, \ldots \{x_k, x_1\}$ as the matching. Since the optimal solution is at least as long as two disjoint matchings of O the minimum-cost matching M has weight bounded by $\frac{1}{2}w(E(G_{OPT}))$, allowing us to conclude $w(E(T)\dot{\cup}M) \leq 3/2w(E(G_{OPT}))$.

This seemingly simple algorithm has nonetheless been the best polynomial time approximation algorithm for the TSP for 40 years and has only very recently been surpassed by a polynomial-time $(3/2{-}10^{36})$-approximation due to Karlin et al. [24, 25], and even then their algorithm is still a (highly non-trivial) version of the same basic template where we compute separately a spanning tree and a matching. In spite of this slow progress it is widely believed that the TSP should admit a 4/3-approximation, and there has been considerable effort towards proving this [26, 27, 28].

In the meantime the study of such routing problems has branched out considerably with extensions to settings where we want multiple tours to cover the metric space jointly [29, 30, 1], we want a path instead of a cycle [31], or we make some special demands on the metric space such as being euclidean [32] or being the shortest-path metric of an unweighted graph [33]. It is to one of these special settings that we now turn to.

Chapter 3

Approximating Multi-TSP

Based on joint work with Max A. Deppert and Matthias Mnich [34].

The previously described Traveling Salesperson Problem represents an important algorithmic primitive, but it is not usually very adept at modelling the practical concerns of visiting some collection of places in an optimal order. In most practical circumstances where large routing problems have to be solved there will be more than one agent expected to fulfill the routing requirements. Otherwise, at least for physical agents, the instance size will necessarily be limited.

This observation gives rise to so-called Multi-TSP (MTSP). Here we are given a complete graph G on n vertices together with non-negative edge weights $w : E(G) \to \mathbb{R}_{\geq 0}$ and a target number k, and we seek a collection of cycles $C_1, \ldots, C_k$ who jointly cover all vertices of G such that the total sum of edge weights used in those cycles is minimized. There are then many variations on this basic problem framework which either make some additional demands on the nature of the cycles, or adapt the objective function.

A popular direction here is to consider not the sum of all of the edge weights, but rather the length of the longest cycle in the solution. Relatedly we might bound the length of the cycles instead of their number and seek a solution covering all vertices with the fewest cycles. This kinds of problems are then usually referred to as some kind of Vehicle Routing Problem (VRP), and enjoy extensive study in their own right.

However they have a very distinct flavour from MTSP, in that the difficulty for MTSP is more in the sequencing problem that arises for the vertices of each cycle. Meanwhile the difficulty for VRP stems heavily from the need to partition the instance into roughly equal parts. Thus VRPs typically encode Bin Packing problems very easily and become increasingly difficult to solve as the number of allowed subtours rises. The same behaviour is not known for MTSP, and it might well be possible that MTSP is not harder than TSP. Recent work in this direction was done by Traub et al. [35], who were able to give an approximation ratio preserving (up to some ε) reduction to TSP for problems of this type, including MTSP. However the

reduction requires an additional overhead of $|V|^{\mathcal{O}(k)}$, where k is the number of cycles requested. It is not known that this is overhead is necessary.

In the direction of MTSP itself one of the earliest examples of work is by Frieze [29], who studied the case where all salespeople start at some *depot*, i.e. there is a special vertex d in the graph such that every cycle C_i contains d. On its own, this just reverts to the normal TSP by noting that the union of all the C_i gives on single tour through the graph (after possibly shortcutting). Meanwhile any solution to TSP can be padded with empty cycles to give a solution to this variant. To create a distinct problem then it is additionally necessary to demand that every C_i has length at least 3. For this variant Frieze was able to give a (3/2)-approximation in polynomial time, i.e. he could essentially maintain the algorithm of Christofides-Serdyukov for this additional constraint.

Another natural extension of this question is to consider a set $D \subseteq V(G)$ of depots such that every cycle must contain at least one of them, a variant known as Multi-Depot Multi-TSP (MDMTSP). In this way we can for example represent the fact that a company might have multiple fulfillment centers from which it wants to service its customers. This problem formulation is the backbone for many practical applications in the routing of personnel or vehicles to job sites [36, 37]. It is on this problem that we will focus our attention.

Formally we will consider that the MDMTSP takes as input a complete graph G on n vertices together with metric edge weights $w : E(G) \to \mathbb{R}_{\geq 0}$, as well as a set $D \subseteq V(G)$ of $d = |D|$ depots and an integer $m \geq 1$ denoting the number of salespersons available. Now again we are seeking a set of m pairwise edge-disjoint cycles $C_1, \ldots, C_m$ in G whose union covers all vertices of the graph and such that the sum of the weights of the cycles is minimized, but in addition each cycle must contain some depot from D. Such set of cycles is an optimal solution for the MDMTSP instance, and we will denote the value of some optimal solution by $\mathrm{OPT}(G, D, w)$ (or simply OPT if the instance is clear from the context).

For our purposes we will assume that $m = d$, i.e. that there is one agent per depot, and we allow the C_i to be empty. This assumption is made for the following two reasons: on the one hand, the case $m > d$ is negligible as the objective function (the total weight of all tours) is invariant for multiple tours starting from a single depot (if weights satisfy the triangle inequality, it is easy to show that there is always an optimal solution in which at most one route will start and end at each depot). On the other hand, in the case $m \leq d$ we can try each selection of $d' = m$ depots by paying a multiplicative factor of $\binom{d}{m}$ in the run time only. Since we will work in a fixed-parameter tractability regime with respect to d, this additional run-time cost is negligible. Thus, any instance of metric MDMTSP is specified by a triple (G, D, w), where G is a complete graph on n vertices, $D \subseteq V(G)$

is the set of depots, and $w : E(G) \to \mathbb{R}_{\geq 0}$ is a metric.

The polynomial-time approximability of metric MDMTSP is not currently well understood. Clearly it is at least as hard as TSP, but it is not clear how much harder it is, if at all. The added complexity arises from the fact that we not only have to give a good order in which to visit vertices but we also have to partition the vertices appropriately.

In particular there exists a variant of the Christofides-Serdyukov algorithm [21, 22] for MTSP and MDMTSP, but it no longer yields a (3/2) approximation in this setting. This extended algorithm initially computes not a minimum spanning tree, but a k-component forest F of minimum weight, possibly such that each component contains at least one depot. Such a structure is called a constrained spanning forest (CSF). This spanner F is then extended by a matching J between all its odd-degree vertices, thus yielding an Eulerian graph where every connected component contains a depot.

The original analysis of Christofides' and Serdyukov's algorithms here relies on all odd-degree vertices of the spanner F lying on the same tour, so a parity-correcting edge set J can be computed that weighs at most $\frac{1}{2}$ OPT. This argument fails in the setting of multiple subtours, so in polynomial time we can currently only get an approximation guarantee by using the spanner F for J also, thus reverting essentially to the tree-doubling heuristic. This only achieves a tight approximation ratio of $2 - \frac{1}{d}$ for the multi-depot setting, as shown by Xu et al. [1] (see Figure 3.1 for a version of their lower-bound example), because the matching can have weight $\frac{d-1}{d}$ OPT.

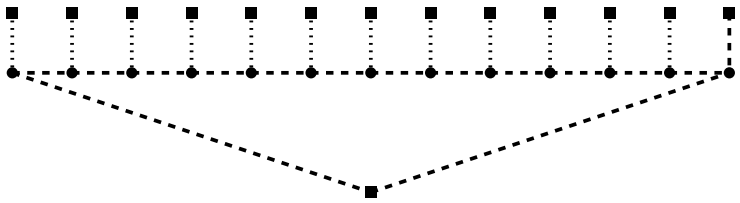


Figure 3.1: An instance on which Algorithm 1 achieves approximation ratio arbitrarily close to 2. Square vertices are depots and all edges have unit weight. Dashed edges indicate a tour of length d, dotted edges a minimum CSF with weight $d - 1$ that requires a join of weight $d - 1$ to complete.

Similarly, the algorithmic approaches to metric TSP based on solving a linear program (LP) are also unlikely to give α-approximation algorithms with $\alpha < 2$ for metric MDMTSP. To this end, consider the following multi-

depot version of the subtour-elimination LP, MDMTSP-LP:

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E(G)} w_e x_e & & \\ \text{subject to} \quad & \sum_{e \in \delta(v)} x_e = 2, & \forall\, v \in V(G) \setminus D & \\ & \sum_{e \in \delta(U)} x_e \geq 2, & \forall\, U \subseteq V(G) \setminus D & \\ & x_e \in [0,2], & \forall e \in E(G) & \end{aligned} \tag{MDMTSP-LP}$$

The following construction in Figure 3.2 shows that MDMTSP-LP has integrality gap 2:

Figure 3.2: An instance on which the multi-depot subtour-elimination LP has integrality gap arbitrarily close to 2. The square vertices are the depots and all edges have unit weight. The dashed edges are assigned $x_e = 0.5$ by the LP, the other edges $x_e = 1$. The LP then has optimum value at most $\ell + 6$, whereas the MDMTSP has optimum value $2(\ell + 4)$.

To improve the approximation guarantee beyond 2 it is then perhaps necessary to relax the requirement of polynomial run time. Xu and Rodrigues [1] show how to obtain a (3/2)-approximation, at the cost of needing time $n^{\Theta(d)}$, which is polynomial only if the number $d = |D|$ of depots is constant. Another (3/2)-approximation for MDMTSP with run time $n^{\Theta(d)}$ follows from the recent work of Traub, Vygen and Zenklusen [35]. They in fact show the much stronger result that any λ-approximation algorithm for metric TSP also gives a $(\lambda + \varepsilon)$-approximation algorithm for metric MDMTSP with an additional run time factor of $n^{\mathcal{O}(d/\varepsilon)}$. Using the recent improvements in the approximability of TSP in polynomial time due to Karlin et al. [24] this does in fact give an $(3/2 - \varepsilon)$-approximation in XP time for some very small ε. Beyond these algorithms the state-of-the-art for MDMTSP is that no α-approximation is known for MDMTSP for any absolute constant $\alpha < 2$ which runs in time $n^{\wr(d)}$.

3.1 Overview of Results

Our main result is a better-than-2 approximation algorithm for metric Multiple TSP on d depots with fixed-parameter tractable run time.

Theorem 3.1. *There is an algorithm that for any given $\varepsilon > 0$ computes in time $(1/\varepsilon)^{\mathcal{O}(d \log d)} \cdot n^{\mathcal{O}(1)}$ a tour T for any set of n cities with metric distances and d depots. The algorithm is randomized, and with constant probability the length of the tour T is at most $(3/2 + \varepsilon) \cdot \mathrm{OPT}$.*

This result significantly improves on the previously best run time $n^{\Theta(d)}$ by Xu and Rodrigues [1] at the cost of some small additive ε in the approximation factor.

To break through the barrier of 2 on the approximation ratio, we need to rework the initial spanner F to be "correctly aligned" with the optimal solution so that each subtour contains an even number of odd-degree vertices, as initially proposed by Xu and Rodrigues [1]. We show that an approximate reworking can be done in time $f(d, \varepsilon) \cdot n^{\mathcal{O}(1)}$ for some suitable function f, resulting in a $(3/2+\varepsilon)$-approximation. To this end, firstly, we give a reduction of metric MDMTSP to a related routing problem which is known as the Rural Postperson Problem (RPP). In the RPP, we are given an edge-weighted graph G and a set R of required edges, and are asked to compute a minimum-weight edge set F such that $R \dot{\cup} F$ is connected and Eulerian. Our reduction reveals an approximation algorithm with run time $\mathcal{O}(n^3 + t)$ to compute solutions no worse than $\frac{3}{2}\,\mathrm{OPT} + \frac{1}{2}w(T)$, where $w(T)$ is the weight of a single-person TSP tour T through the depots and t denotes the time to compute T. Then we use a randomized algorithm of Gutin et al. [38] for the RPP, and an approximate weight reduction scheme of van Bevern et al. [39], to construct a $(1 + \varepsilon)$-approximation algorithm for a variant of RPP with depots.

We are then in a position to speed up the reworking of the initial spanner due to two key insights. Firstly, we allow for some misalignment to remain, as long as it is only due to the presence of some *light* edges, limiting the number of edges we have to consider for removal from F. Secondly, we employ the constructed approximation algorithm for the RPP to complete our now disconnected spanner to a tour. Doing this provides a large speedup over the algorithm of Xu and Rodrigues, who first need to guess a set of edges to reconnect their spanner, and then employ a matching algorithm to obtain a tour. Using the RPP allows us to do both of these steps simultaneously, and considerably faster.

An important special case of metric TSP is when the metric w is induced by the shortest paths in a graph. This version is also known as Graphic TSP, and has been studied extensively from the perspective of approximation algorithms [40, 33]. For MDMTSP on graphic metrics, we obtain a *deterministic algorithm* with slightly better approximation factor and a reduced run time.

Theorem 3.2. *There is an algorithm that, given any graph G on n vertices and set $D \subseteq V(G)$ of d depots, in time $2^d \cdot n^{\mathcal{O}(1)}$ computes a tour T of length*

at most $\frac{3}{2} \cdot \mathrm{OPT}$.

Finally, we will also be able to transfer these techniques back to the original setting of MTSP, i.e. we can drop the requirement of fixing every tour to some depot, instead covering the graph with k arbitrary cycles.

3.2 Formal Problem Statement

Let $\mathcal{U}$ be a finite universe. For a weight function $w : \mathcal{U} \to \mathbb{R}$ and a multiset $U \subseteq \mathcal{U}$ we write $w(U)$ to mean $\sum_{u \in U} w(u)$, where the sum has an additional summand for each copy of an element in U, i.e. it considers multiplicities. The disjoint union $\bigcup_i A_i$ of some sets $\{A_i\}_i$ is considered to be the multiset of all items in the collection. For brevity we often write $2A$ to mean $A \dot\cup A$.

Definition 3.1. An instance (G, D, w) of metric MDMTSP consists of a complete graph G, a set $D \subseteq V(G)$ of d *depots*, and a metric $w : E(G) \to \mathbb{R}_+$ on $V(G)$. A multiset of edges $T \subseteq E(G)$ is called a *tour* of (G, D, w) if

(P1) the multigraph $(V(G), T)$ has even degree at every node in $V(G)$,

(P2) and each connected component of $(V(G), T)$ contains at least one node from D.

We denote by $\mathrm{OPT}(G, D, w)$ the minimum weight of any tour of (G, D, w). If the instance is clear from the context, we may only say OPT.

Edge sets are generally allowed to be multisets, and graphs can have parallel edges.

Imitating the general framework of Christofides-Serdyukov [21, 22], we first compute an edge set F, called a CSF, that ensures the connectivity property (P2). We then compute an additional set of edges J such that $F \dot\cup J$ has property (P1).

Definition 3.2. Let G be a graph and let $D \subseteq V(G)$. A *Constrained Spanning Forest* in (G, D) is a set $F \subseteq E(G)$ of edges such that the graph $(V(G), F)$ is acyclic and every connected component of $(V(G), F)$ contains at least one node from D.

We will make use of the following result.

Theorem 3.3 (Rathinam et al. [36])**.** *Given any graph G on n vertices and m edges with weights $w : E(G) \to \mathbb{R}$ and a set $D \subseteq V(G)$, a minimum-weight CSF of (G, D, w) can be computed in time $\mathcal{O}((n + m) \log n)$.*

Proof. The computation of a minimum-weight CSF for (G, D, w) can be reduced to computing a minimum-weight spanning tree in a graph G' with edge weights w'. The graph G' is obtained from G by adding a single root node r connected to every depot by an edge of some weight less than the weight all other edges in G. Kruskal's algorithm is guaranteed to choose these edges for the minimum-weight spanning tree in (G', w'), and after removing r we are left with a minimum-weight CSF for (G, D, w). □

Proof. For an alternative proof one can also see that the collection of all edge sets which contain neither a cycle nor a path between two depots forms a matroid over the universe $E(G)$. The cycles of this matroid are then precisely the cycles and the inter-depot paths in G.

It is well known [41] that a collection $\mathcal{C}$ of subsets of some universe is the cycle set of a matroid if and only if for any two cycles $C_1, C_2 \in \mathcal{C}$ and any element $x \in C_1 \cap C_2$ we find some $C \in \mathcal{C}$ such that $C \subseteq C_1 \cup C_2 \setminus \{x\}$. For this case of cycles and inter-depot paths this property can be quickly verified, thus witnessing that the given structure is a matroid. Indeed to check this property it is enough to consider that, after contracting all depots, the cycles of this matroid all become proper cycles of the resulting graph. So the cycle set here is just the cycle set of the graphic matroid of G/D. □

The second proof here might look a bit more complicated than is necessary, but it is useful to keep in mind, since this allows the application of other more advanced techniques such as optimization over the intersection of matroids. This is in some way what the algorithm of Frieze [29] does to find k non-empty tours, so if one wanted to recover his result for a multi-depot setting, this second proof is in fact more useful.

To obtain property (P1) one traditionally computes a minimum-weight matching on the odd-degree vertices of the CSF, as in Algorithm 1. However,

Algorithm 1: Multi-Depot Christofides-Serdyukov

Input: A metric MDMTSP instance (G, D, w).
Output: A tour T with $w(T) \leq 2\,\mathrm{OPT}$.
1 Compute a minimum-weight CSF F for (G, D, w);
2 Let U be the set of vertices with odd degree in F;
3 Compute a minimum-weight perfect matching M in $G[U]$;
Result: $T := F \dot\cup M$

this algorithm only achieves a tight approximation ratio of $2 - \frac{1}{d}$ for the multi-depot setting, as shown by Xu et al. [1] (see Figure 3.1), because the matching can have weight $\frac{d-1}{d}\,\mathrm{OPT}$. To avoid such an expensive matching, the constrained spanning forest needs to be rearranged such that there is again a matching of weight $\frac{1}{2}\,\mathrm{OPT}$, as in the work of Xu and Rodrigues [1].

3.3 Reducing MDMTSP to RPP

In this section we show a reduction from the metric MDMTSP to the RPP. Recall that in the RPP there is a required set R of edges that a tour should traverse, rather than a set of vertices.

Definition 3.3 (Rural Postperson Problem)**.** An instance (G, R, w) of RPP consists of a graph G, a set $R \subseteq E(G)$ of required edges, and a metric weight function $w : E(G) \to \mathbb{R}_{\geq 0}$. A *solution* is multiset $J \subseteq E(G)$ for which $(V, R \dot\cup J)$ is Eulerian, and which has only one non-singleton connected component.[1] The weight of a solution J is $w(J) = \sum_{e \in J} w(e)$. The goal of RPP is to compute an *optimal* solution, which is a solution of minimum weight $\mathrm{OPT}(G, R, w)$.

There is a polynomial-time approximation algorithm for RPP [42, 43] which computes a solution $J \subseteq E(G)$ such that $w(R \dot\cup J) \leq \frac{3}{2} w(R \dot\cup J^*)$, that is we get $w(J) \leq \frac{3}{2}\,\mathrm{OPT} + \frac{1}{2} w(R)$, where J^* is some optimal solution. Due to the first inequality and the unavoidable weight of R, the algorithm is often referred to as a (3/2)-approximation for RPP. That is, we consider the weight of OPT to be the whole tour, rather than just the part computed as a solution to the problem. In that way there is an additional cost of $\frac{1}{2} w(R)$ that can be paid.

A similar approximation can also be obtained for metric MDMTSP, where we can provide a (3/2)-approximation if we allow for some additional additive term. This observation motivates the following reduction from metric MDMTSP to RPP.

Observation 3.1. *For each instance* (G, D, w) *of* MDMTSP *there is an instance* (G', R, w') *of* RPP *such that any solution to the* RPP *instance can be transformed in polynomial time into a solution to the* MDMTSP *instance of the same weight.*

Proof. First, compute any TSP tour S on the depots in G, that is, on $G[D]$. Then, for each node $v \in V \setminus D$, introduce a second node v', as well as an edge $e_v = \{v, v'\}$, and set its weight to $w'(e_v) = 0$. For each edge $e \in E(G)$ set $w'(e) = w(e)$, and set R to be the union of S and two copies of each e_v. The any solution to the constructed instance of RPP corresponds to an MDMTSP tour for (G, D, w) of the same weight. □

Notice that this reduction, together with the (3/2)-approximation for RPP, allows us to compute a solution to MDMTSP whose total weight is

[1] This means that vertices not incident to any edge from R do not need to be visited by the computed tour. For metric weight functions, however, one can always reduce to the case where R spans G.

bounded by $\frac{3}{2}\,\mathrm{OPT} + \frac{1}{2}w(S)$. In particular, if all depots are pairwise close to each other this is already a better-than-2 approximation.

In Section 3.5 we will in some sense show a stronger result that there is also a (Turing) reduction from MDMTSP to the special case of RPP where the graph (V, R) has few connected components, which has been shown by Gutin et al. to be tractable [38]:

Proposition 3.1 (Gutin et al. [38])**.** *There is a randomized algorithm for RPP that for any instance (G, R, w), where $(V(G), R)$ has k connected components and w takes only integer values, in time $2^{\mathcal{O}(k)}(n+\mathrm{OPT}(G,R,w))^{\mathcal{O}(1)}$ produces a solution. With constant probability, the computed solution is optimal.*

However, as our reduction is only $(1+\varepsilon)$-approximate with respect to the solution qualities, and needs time exponential in d and ε, we need to remove the polynomial dependence on $\mathrm{OPT}(G, R, w)$ in Proposition 3.1. To this end, we will adapt an approximate weight reduction scheme by van Bevern et al. [39]:

Lemma 3.1 (van Bevern et al. [39, Lemma 2.12])**.** *Let (G, R, w) be an instance of RPP with integral weighs, fix some $\varepsilon > 0$, and define the maximum weight as $\beta = \max\{w(e) \mid e \in E(G)\}$. Then in polynomial time we can compute a weight function $w' : E(G) \to \mathbb{N}_{\geq 0}$ such that*

- $\max\{w'(e) \mid e \in E(G)\} \leq 2|E(G)|/\varepsilon$,
- *and for all $\alpha \geq 1$, any J with $w'(J) \leq \alpha\,\mathrm{OPT}(G, R, w')$ also fulfills $w(J) \leq \alpha\,\mathrm{OPT}(G, R, w) + \varepsilon\beta$, as long as J contains at most two copies of each edge.*

Proof. The rounding scheme simply sets $w'(e) := \lfloor w(e) \cdot \frac{2|E(G)|}{\varepsilon\cdot\beta} \rfloor$ for each edge e of G. This yields the first condition, by definition. For the second condition, observe that for any J we have

$$\frac{\varepsilon\cdot\beta}{2|E(G)|}w'(J) \leq w(J) \leq \frac{\varepsilon\cdot\beta}{2|E(G)|}w'(J) + \frac{\varepsilon\cdot\beta}{2|E(G)|}|J| \leq \frac{\varepsilon\cdot\beta}{2|E(G)|}w'(J) + \varepsilon\beta\ .$$

Hence, the two weight functions are equivalent up to scaling by a constant and the addition of at most $\varepsilon\beta$. □

Notice that the restriction on J having at most two copies of each edge is never a problem: whenever a solution to the RPP has three or more copies of one edge, we can delete two of them to obtain a cheaper solution.

We will combine Proposition 3.1 and Lemma 3.1 to obtain an approximation for k-component RPP whose run time does not depend on OPT.

Corollary 3.1. *There is a randomized algorithm that, for any $\varepsilon > 0$, in time $2^{\mathcal{O}(k)}(n + \frac{1}{\varepsilon})^{\mathcal{O}(1)}$ computes a solution J for any instance (G, R, w) of* RPP *where $(V(G), R)$ has k connected components. The computed solution J has the property that, with constant probability,*

$$w(J) \leq \mathrm{OPT}(G, R, w) + \varepsilon \max\{w(e) \mid e \in J^* \dot{\cup} R\},$$

where J^* is some optimal solution to the instance.

Proof. We first guess the weight β of the heaviest edge in $J^* \dot{\cup} R$, where J^* is some optimal solution. There are only $|E(G)|$ options, so the guessing generates only polynomial overhead. All edges that are more expensive than β can be removed from the instance to get some graph G'. Now we apply Lemma 3.1 to get an instance with weights w' bounded by $2|E(G)|/\varepsilon$ and use the exact algorithm from Proposition 3.1 to get a solution J to (G', R, w') in time $2^{\mathcal{O}(k)}(n + \frac{1}{\varepsilon})^{\mathcal{O}(1)}$. From Lemma 3.1 with $\alpha = 1$ we know that

$$w(J) \leq \mathrm{OPT}(G, R, w) + \varepsilon\beta \leq \mathrm{OPT}(G, R, w) + \varepsilon \max\{w(e) \mid e \in J^* \dot{\cup} R\},$$

which proves the claim. □

We will be using this algorithm to complete partial solutions to instances of MDMTSP. We will need only a slight modification that allows for the presence of depots as follows.

Definition 3.4. An instance (G, D, R, w) of the Depot Rural Postperson Problem (DRPP) consists of an RPP instance (G, R, w) and some depots $D \subseteq V(G)$. A *solution* to an instance of DRPP is a multiset $J \subseteq E(G)$ such that $(V, R \dot{\cup} J)$ is Eulerian and each non-singleton connected component of $(V, R \dot{\cup} J)$ contains at least one depot. The weight of a solution J is $w(J) = \sum_{e \in J} w(e)$. The goal is to compute an *optimal* solution, which is a solution of minimum weight $\mathrm{OPT}(G, D, R, w)$.

The depot version DRPP can be reduced to the RPP quite easily.

Corollary 3.2. *There is a randomized algorithm that computes for any instance (G, D, R, w) of* DRPP *where $(V(G), R)$ has k connected components and any $\varepsilon > 0$, in time $2^{\mathcal{O}(k \log k)}(n + \frac{1}{\varepsilon})^{\mathcal{O}(1)}$ a solution J such that, with constant probability, $w(J) \leq (1 + \varepsilon)\,\mathrm{OPT}(G, D, R, w) + \varepsilon w(R)$.*

Proof. Note first that each connected component of $(V(G), R)$ can be assumed to contain at most one depot, so $|D| \leq k$. Some optimum solution J^* induces a partition of the connected components of $(V(G), R)$ where each partition class corresponds to those components connected to some specific depot. There are at most $|D|^k \in 2^{\mathcal{O}(k \log k)}$ possible partitions, so we can try each partition, solve the regular RPP instance on each of the $|D|$ classes of the partition using the algorithm from Corollary 3.1, and return the best solution we found. □

3.4 Intuition for the Algorithm

The algorithm of Xu and Rodrigues [1] executes, at a very high level, the following steps:

1. Compute a minimum-weight CSF F for (G, D, w).
2. Guess a set X of at most $|D| - 1$ edges such that they are in F but not in some fixed optimal tour T.
3. Discard the guessed edges X from F. This leaves at most $2|D|$ connected components in $(V, F \setminus X)$. If we have guessed correctly, every subtour of T now contains an even number of odd-degree vertices. There must exist some edges A from T such that $(F \setminus X) \cup A$ is a CSF for (G, D) with $w((F \setminus X) \cup A) \leq w(T)$. The value $|A|$ is at most $|D|$, so we also guess A.
4. Since A contains only edges from T, every subtour of T still contains an even number of odd-degree vertices with respect to $(V, (F \setminus X) \cup A)$. If we compute an odd-join J for $(F \setminus X) \cup A$, we have $w(J) \leq \frac{1}{2}\,\mathrm{OPT}$, so return $((F \setminus X) \cup A) \dot\cup J$.

Since the algorithm needs to guess $2|D|$ edges in total (in step 2), it can be implemented in time $n^{\mathcal{O}(d)}$. We modify this guessing step by considering for discarding (in step 3) only very heavy edges, and by sidestepping the guessing of A; instead of computing first a connected structure and then a join we do this simultaneously, using the algorithm for RPP. Specifically:

- In step 2, we only consider edges that are very expensive relative to the total weight of the forest F. If the targeted edge e is not in this collection, we do not delete it but instead use it as part of the augmenting set A, doubling the edge. This will also fix parity, but requires us to relax the original $w((F \setminus X) \dot\cup A) \leq w(T)$ to $w((F \setminus X) \dot\cup A) \leq (1+\varepsilon)w(T)$. The ε can be controlled by how expensive relative to F we allow these non-deleted edges to be.
- In step 3, we do not actually guess A, we merely use its existence. We instead solve an instance of DRPP with at most $2d$ connected components for which $A \dot\cup J$ is a solution. Using the algorithm from Corollary 3.2, we can compute a $(1+\varepsilon)$-approximation for the DRPP in time $f(d, \varepsilon) \cdot n^{\mathcal{O}(1)}$. We use the solution J' as a replacement for $A \dot\cup J$ knowing $w(J') \leq (1+\varepsilon)w(A \dot\cup J)$. Combining inequalities for J and F gives:

$$w((F \setminus X) \dot\cup J') \leq (1+\varepsilon)w(((F \setminus X) \dot\cup A) \dot\cup J) \leq (1+\varepsilon)\frac{3}{2}\,\mathrm{OPT} \quad .$$

An illustration of the augmentation scheme can be found in Figure 3.3.

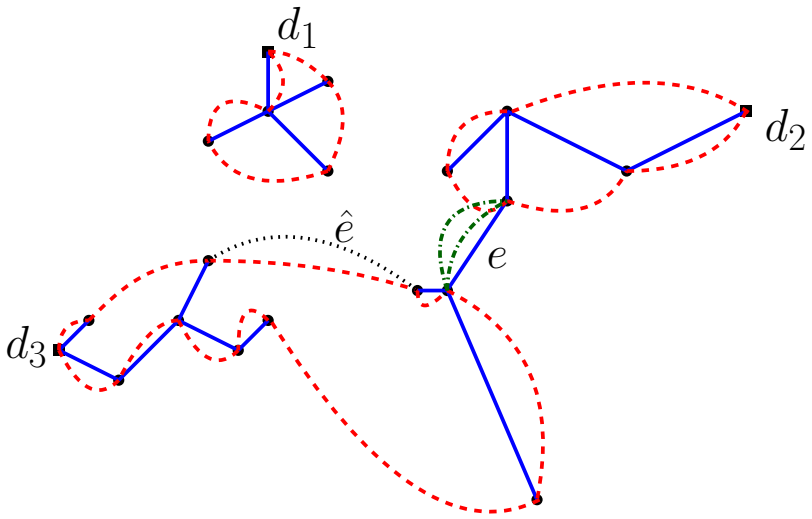


Figure 3.3: Illustration of the augmenting edges explored by our algorithm. Blue solid edges represent a CSF, dashed red edges an optimal tour. Note that the tours of d_2 and d_3 have an odd number of vertices with odd degree in the CSF. Our algorithm considers two options to remedy this. We either add two copies (green, dotted) of the edge e to the optimal tour, if e is considered to be light enough. This joins the tours of d_2 and d_3 to a single tour with an even number of odd-degree vertices. If e is considered too heavy for this, we remove e from the CSF and replace it with the edge $\hat{e}$ (black, dotted). As $\hat{e}$ comes from the optimal tour, this keeps the weight of the CSF below OPT and it fixes the parities.

3.5 Towards Faster Parity Correction

In this section, we give a formal version of the algorithm described in the previous section, which we state as Algorithm 2. We prove this algorithm to be a $(3/2 + \mathcal{O}(\varepsilon))$-approximation for metric MDMTSP in Theorem 3.4. We will restate and reprove some of the results of Xu and Rodrigues [1] to ensure completeness of the presentation and to integrate properly our changes to their algorithm.

To this end we will now fix some notation throughout this section. Let (G, D, w) be a metric MDMTSP instance with $D = \{d_1, \ldots, d_d\}$, an optimal tour T, and the minimum-weight CSF F for (G, D, w) that was computed Algorithm 1. We denote by T_i be the connected component of T containing d_i, by F_i the subtree of F containing d_i, and U_i the set of vertices in T_i that have odd degree with respect to the edges in F. We take $U = \bigcup_i U_i$.

By minimality of F, we already know that $w(F) \leq w(T) = \mathrm{OPT}$. To extend F to an MDMTSP tour, we try to compute a minimum-weight matching between the vertices in U. It is a standard argument from the analysis of the Christofides-Serdyukov algorithm that if $|U_i|$ is even, T_i contains two disjoint matchings for the vertices in U_i. So if every U_i has even cardinality, then any minimum-weight matching has weight at most $\frac{1}{2}\,\mathrm{OPT}$. But this is not the case, since a tree F_i might contain vertices from many different tours, so the odd-degree vertices are distributed arbitrarily. To record this "misalignment" between the trees and subtours we introduce the concept of an *alignment graph*.

Definition 3.5 (Alignment Graph)**.** The alignment graph H for (G, F, T) is constructed as $V(H) = D$, and

$$E(H) = \{\{d_i, d_j\} \mid \exists e \in F \text{ s.t. } |e \cap V(T_i)| = |e \cap V(T_j)| = 1\} \ .$$

We also define a weight function $w_H : E(H) \to \mathbb{R}_+$ as

$$w_H((d_i, d_j)) := \min\{w(e) | e \in F,\ |V(T_i) \cap e| = |V(T_j) \cap e| = 1\} \ .$$

In the following, we assume that H is connected, otherwise the analysis holds independently for each connected component.

Now we take D_{odd} to be the collection of depots d_i for which $|U_i|$ is odd, and A_H to be any D_{odd}-join in H. The join A_H can be used to augment the original tour T to be connected. To do this we transfer the join to the original graph to ensure that it contains a "cheap" matching.

For every edge $e = \{d_i, d_j\} \in E(A_H)$, pick an edge in $\hat{e} \in E(F)$ with $w(\hat{e}) = w_H(e)$ and $|\hat{e} \cap V(T_i)| = |\hat{e} \cap V(T_j)| = 1$. Denote by A the collection of these $\hat{e}$. Observe that every node in $T \dot{\cup} 2A$ has even degree, and

every connected component of the graph contains an even number of vertices from U. Hence, there exists a U-join J in G with $w(J) \leq \frac{1}{2}w(T \dot\cup 2A)$.

Now notice that, if the edges in A have weight at most $\varepsilon w(F)$, this inequality yields that Algorithm 1 already achieves a good approximation ratio, specifically

$$w(F) + w(M) \leq w(F) + w(J) \leq (1+\varepsilon)w(F) + \frac{1}{2}w(T) \leq (\frac{3}{2}+\varepsilon)w(T) \ .$$

Based on this observation, we are willing to augment T with low-weight edges from F to find a low-weight matching. Therefore, we need to distinguish between *heavy* and *light* edges.

Definition 3.6. Let $\varepsilon > 0$. An edge $e \in F$ is *ε-light* if $w(e) \leq \frac{\varepsilon}{d} \cdot w(F)$; else, it is *$\varepsilon$-heavy*.

With this distinction in place, we try to replace the ε-heavy edges in A with some other edges from T.

Lemma 3.2 (compare [1, Section 2])**.** *Let $X \subseteq A$. Then there exist a set of edges $\hat{A} \subseteq E(T)$ such that $(F \setminus X) \cup \hat{A}$ is a CSF for (G, D) and we have $w((F \setminus X) \cup \hat{A}) \leq w(T)$.*

Proof. Consider the forest F' obtained from T by removing exactly one edge from each subtour. F' contains only edges from T, so it is disjoint from X. By a standard matroid exchange argument, for each $e \in X$ there is a $\hat{e} \in F'$ such that $F - e + \hat{e}$ is a CSF and $w(e) \leq w(\hat{e})$. This process can then be iterated to remove all of X. The collection of these $\hat{e}$ is $\hat{A}$, giving

$$w((F \setminus X) \cup \hat{A}) \leq w(F') \leq w(T).$$

□

This process of replacing augmenting edges from A with edges out of T also fulfills the key goal of putting an even number of odd-degree vertices into every connected component of some augmented MDMTSP solution. Consider the following lemma, which is in substance a version of a statement by Xu and Rodrigues [1, Theorem 2].

Lemma 3.3. *Let $A, X, \hat{A}$ be as in Lemma 3.2. Then every connected component of $T \cup (A \setminus X)$ contains an even number of vertices that have odd degree in $(F \setminus X) \cup \hat{A}$.*

Proof. Notice first that the connected components of $T \cup (A \setminus X)$ are the union of some of the subtours of T. Since the edges in $\hat{A}$ belong to some tour, adding them to $F \setminus X$ flips the parity of the degrees of two vertices on the same tour, so the total parity of odd-degree vertices on that tour

Algorithm 2: Extended Multi-Depot Christofides-Serdyukov

Input: A metric MDMTSP instance (G, D, w) and a parameter $\varepsilon > 0$.

Output: A tour T such that $w(T) \leq (3/2 + \varepsilon)\,\mathrm{OPT}$ with constant probability.

```
1  Compute a minimum-weight CSF F for (G, D, w);
2  Let T be the currently best MDMTSP solution, initially 2F;
3  Let Y be the set of edges in F which are ε-heavy;
4  foreach X ⊆ Y, |X| ≤ |D| do
5  |   F' := F \ X;
6  |   If (V, F') contains singleton vertices, attach to each a new
   |    vertex with two parallel edges of length 0 ;
7  |   Compute a solution M for the DRPP instance (G, D, F', w)
   |    using Corollary 3.2 ;
8  |   if w(M ∪̇ F') < w(T) then
9  |   |   set T = M ∪̇ F';
10 |   end
11 end
```

Result: T

does not change. We can therefore restrict ourselves to considering the odd-degree vertices with respect to $F \setminus X$.

Recall that originally A was constructed from a D_{odd}-join A_H in the alignment graph H. So X corresponds to some edge set $X_H \subseteq A_H$, and we know that $A_H \setminus X_H$ constitutes an $(D_{\text{odd}} \Delta \dot{\bigcup}_{e \in X_H} e)$-join, where Δ denotes the symmetric difference. For multisets, the symmetric difference of some sets contains an item if and only if it is contained an odd number of times in their disjoint union. At the same time, removing an edge $e \in X$ from F with corresponding $\{d_i, d_j\} \in X_H$ changes the degree of one node in $V(T_i)$ and one node on $V(T_j)$. So, the depots whose tours contain an odd number of odd-degree vertices with respect to $F \setminus X$ are precisely $(D_{\text{odd}} \Delta \dot{\bigcup}_{e \in X_H} e)$, so $A_H \setminus X_H$ joins them correctly. □

We are now ready to prove that Algorithm 2 returns a $(3/2 + \mathcal{O}(\varepsilon))$-approximation, with constant probability.

Theorem 3.4. *The tour returned by Algorithm 2 has a weight of at most* $(3/2 + \mathcal{O}(\varepsilon))\,\mathrm{OPT}$*, with constant probability.*

Proof. Set T' to be the tour returned by the algorithm. Notice that this is indeed a tour, since there are no singleton vertices due to line 6, i.e. every connected component will contain a depot. The added vertices and

edges in line 6 also do not change the solution value, so we may ignore them for the rest of the analysis. Now let A be the augmenting edge set for some optimal tour as before and X the set of ε-heavy edges in A. We look at the iteration of the algorithm where that X is considered for removal. From Lemma 3.2 we know that there exists some edge set $\hat{A}$ with $w(F \setminus X \cup \hat{A}) \leq \mathrm{OPT}$ and Lemma 3.3 implies that there is an edge set J such that $F \setminus X \dot\cup \hat{A} \dot\cup J$ is Eulerian, contains a depot in each connected component, and $w(J) \leq \frac{1}{2}w(T \cup (A \setminus X))$. Therefore, $\hat{A} \dot\cup J$ is a solution to the DRPP instance (G, D, F', w). Hence, the M computed in the algorithm fulfills $w(M) \leq (1+\varepsilon)w(\hat{A} \dot\cup J) + \varepsilon w(F')$. Putting all these inequalities together yields

$$
\begin{aligned}
&w(T') = w(M) + w(F \setminus X) \\
&\leq (w(F) - w(X) + w(\hat{A})) + (\frac{1}{2}(w(T) + w(A \setminus X)) + \varepsilon(w(\hat{A}) + w(J) + w(F)) \\
&\leq w(T) + \frac{1}{2}(w(T) + d \cdot \frac{\varepsilon}{d} w(F)) + \varepsilon(w(\hat{A}) + w(J) + w(F)) \\
&\leq \frac{3}{2} w(T) + 4\varepsilon w(T),
\end{aligned}
$$

where we use that $A \setminus X$ contains at most d edges, and all of them are ε-light. □

Notice that this algorithm will give a $(3/2 + \varepsilon)$-approximation when called with $\varepsilon/4$ as the parameter of approximation. The additional run time cost will vanish in the $\mathcal{O}$-notation. The probability of success for this algorithm is the same as that for the algorithm in Proposition 3.1. Notice that while that algorithm is called many times, we only need it to succeed for one specific choice of X. If it fails in one of the other attempts, we do not care.

It remains to analyze the run time of this algorithm. We see that Y, the set of ε-heavy edges, has size at most $\frac{d}{\varepsilon}$, so there are only $(\frac{d}{\varepsilon})^d$ possible values for X to be tried. Note also that each loop iteration requires the approximate solution of a DRPP instance with $\mathcal{O}(d)$ components which can be done in time $2^{\mathcal{O}(d \log d)}(n + \frac{1}{\varepsilon})^{\mathcal{O}(1)}$. The total run time of the algorithm then is $(1/\varepsilon)^{\mathcal{O}(d \log d)} \cdot n^{\mathcal{O}(1)}$, showing Theorem 3.1.

3.6 Extension to the Depot-Free Case

It should be noted that all previous analysis can also be used in the case that no depots are present, i.e. in the case of MTSP.

Definition 3.7. An instance (G, k, w) of MTSP consists of a complete graph G, a natural number $k \in \mathbb{N}$, and a metric $w : E(G) \to \mathbb{R}_+$ on $V(G)$.

A multiset of edges $T \subseteq E(G)$ is called a tour of (G, k, w) if

(P1) the multigraph $(V(G), T)$ has even degree at every node in $V(G)$,

(P2) and $(V(G), T)$ has at most k connected components.

Notice that this is essentially the same as Definition 3.1, we merely replace the requirement that every connected component of a tour contains a depot with the one that there are at most k components. One may imagine that in this setting we do not yet know where to construct the depots. Thus we arrange some tours first, and then build a depot on each tour later. If one is given XP time (parameterized on k), this setting reduces trivially to MDMTSP by enumerating all possible choices of k depots. However we want to remain in FPT time, so we can not resort to this. Instead we briefly see that Algorithm 1 adapts to the setting of MTSP.

The only change we really need to make is to initially compute a k-component spanning forest F of minimum weight, rather than a CSF. We then again isolate all edges which are heavy with respect to some parameter ε, and try removing all subsets of $\leq k$ heavy edges from F. Finally we recombine the at most $2k$ components into at most k Eulerian components using the algorithm sketched out in Corollary 3.2. The only difference here is that instead of partitioning the connected components into classes such that every class contains a depot we simply partition into at most k classes. The analysis in Theorem 3.4 then follows as before.

Corollary 3.3. *The tour that is returned by Algorithm 3 has weight at most* $(3/2 + \mathcal{O}(\varepsilon))\,\mathrm{OPT}$*, with constant probability.*

3.7 Approximating Graphic MDMTSP

In the case where the metric is not arbitrary, but rather the induced shortest-path metric of an unweighted graph, we can even eliminate the need for some ε in the approximation factor. Here we can provide a deterministic (3/2)-approximation for MDMTSP with a run time of $2^d \cdot n^{\mathcal{O}(1)}$.

Let (G, D) be an instance of graphic MDMTSP, where G this time is the unweighted graph inducing the shortest-path metric. Note that we can assume G to be connected. The fact that the metric is induced by G allows us to construct TSP tours that are not much more expensive than optimal solutions to MDMTSP, which effectively re-enables the original analysis of the Christofides-Serdyukov Algorithm.

For a given optimal MDMTSP tour T, we can extend it to a TSP tour by introducing at most $2(d-1)$ edges. To do this, contract the subtours of

Algorithm 3: Algorithm MTSP Christofides-Serdyukov

Input: A metric MTSP instance (G, k, w) and a parameter $\varepsilon > 0$.
Output: A tour T such that $w(T) \leq (3/2 + \varepsilon)\,\mathrm{OPT}$ with constant probability.

```
Compute a minimum-weight k-component forest F for (G, w);
Let T be the currently best MDMTSP solution, initially 2F;
Let Y be the set of edges in F which are ε-heavy;
foreach X ⊆ Y, |X| ≤ |D| do
    F' := F \ X;
    Compute a solution M for the RPP instance (G, k, F', w) using
     Corollary 3.2 ;
    if w(M ∪̇ F') < w(T) then
        set T = M ∪̇ F';
    end
end
```

Result: T

T, find a spanning tree in the contracted graph, and double all the edges of that tree. We then see that the solution $F \,\dot\cup\, M$ returned by Algorithm 1 fulfills

$$w(F \,\dot\cup\, M) \leq w(T) + \frac{1}{2}(w(T) + 2(d-1)) = \frac{3}{2}w(T) + d - 1 \ .$$

Notice that the additive term $d-1$ is likely to be very small, since we know that $w(T) \geq n - d$. A similar argument can also be made for metrics which are *continuous* in the sense that the space cannot be partitioned into two very distant parts.

Observation 3.2. *Let (G, D, w) be an* integer-weighted *instance of MDMTSP for which there exists a constant L such that, for all $U \subseteq V(G)$, it holds $\min\{w(u, v) \mid u \in U, v \notin U\} \leq L$. Then Algorithm 1 returns a solution T for (G, D, w) with $w(T) \leq \frac{3}{2}\,\mathrm{OPT}(G, D, w) + L(d-1)$.*

Since we know $w(T)$ to be in $\Omega(n - d)$ also in this case, Algorithm 1 gives an *asymptotic* (3/2)-approximation for any constant d and L, i.e. the constant of approximation is $3/2 + o(1)$. We can even get rid of the additive term in the graphic case (i.e. $L = 1$) with some additional run time.

Observation 3.3. *There is a deterministic $(3/2)$-approximation algorithm for graphic MDMTSP with run time $2^d \cdot n^{\mathcal{O}(1)}$.*

Proof. Let T be some fixed optimal tour. We start by guessing the subset $D' \subseteq D$ of depots whose subtours in T contain at least one edge, generating on overhead of 2^d. Then we know that T contains a CSF F' for (G, D')

with weight $|E(T)| - |D'|$. As before, we connect together all subtours of the depots in D' with $|D'| - 1$ edges, and double these edges. We then see that the tour $F \dot\cup M$ returned by Algorithm 1 fulfills

$$|E(F \dot\cup M)| \leq |E(T)| - |D'| + \frac{1}{2}(|E(T)| + 2(|D'| - 1)) = \frac{3}{2}|E(T)| - 1,$$

and that proves the claim. □

For the special case where we require each depot to have a non-empty tour, we do not even have to guess the correct subset of depots in Observation 3.3, yielding a (3/2)-approximation in truly polynomial time.

3.8 Conclusion & Open Problems

We thus see that both metric MDMTSP and MTSP admit a randomized $(3/2+\varepsilon)$-approximation algorithm in time $(1/\varepsilon)^{\mathcal{O}(d \log d)} \cdot n^{\mathcal{O}(1)}$, filling in the gap between the previously best-known polynomial approximation factor 2, and the (3/2)-approximation of Xu and Rodrigues in time $n^{\Theta(d)}$. However, there remain a number of natural questions regarding the approximability of MTSP-type problems.

- Is it possible to derandomize Algorithm 1? Since we need to solve low-component-count instances of RPP, we are reliant on the Schwartz-Zippel Lemma [44, 45] by way of the algorithm of Gutin et al. [38]. The derandomization of the Schwartz-Zippel Lemma is of course a tempting prospect, but it has resisted such efforts throughout the last 40 years. To find a deterministic algorithm then, both for MDMTSP and RPP, it is probably necessary to resort to a completely different approach. However there does not appear to be any reason for these problems to inherently require randomization, so such an improvement may well be possible.

- Can the approximation factor be improved from $3/2+\varepsilon$ to (3/2)? We lose some approximation quality both when determining which edges to delete from the CSF, and when solving RPP. Improving the first point would require a further refinement of the tree-rearrangement technique introduced by Xu and Rodrigues [1]. For the second point, the RPP algorithm of Gutin et al. would need to be sped up to run in strongly polynomial time. Again, their algorithm relies on algebraic techniques for which derandomization appears difficult, so a major technical innovation for k-component RPP is maybe necessary.

- Does there exist some polynomial-time α-approximation algorithm for MDMTSP with $\alpha < 2$? We know from Traub et al. [35] that any α-approximation algorithm for single-salesperson TSP implies a $(\alpha + \varepsilon)$-approximation for MDMTSP for any constant number of depots, i.e. in time $n^{\Theta(d)}$. For instances with many depots however, the problem remains intractable. It is of particular interest that two major technical tools for the classical TSP, Christofides' Algorithm and the Subtour-Elimination LP, fail to achieve better-than-2-approximations in the multi-depot regime (see Figure 3.1 and Figure 3.2). It appears that to make progress on a polynomial-time algorithm some novel structural insights would be required.

Chapter 4

All-ℓ_p-Norm Tree Cover

Based on joint work with Kelin Luo, Matthias Mnich, and Heiko Röglin [46].

The analysis for MDMTSP in the previous chapter has specifically concerned itself with the question of minimizing the *total* length of a collection of cycles covering some metric space (V, w). This is a good first model of some problems, but it leaves many practical concerns unaddressed. If we return to our very basic motivating example of a set of salespeople trying to visit some customers an optimal solution to MDMTSP will ensure that the total time spent is minimized. It may however assign all customers to just one of the salespeople. We can quite safely assume that this is not a desirable solution in many cases, and that it is usually preferable to have a roughly equal workload between the salespeople.

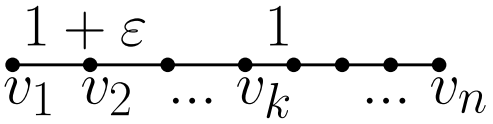
One possible direction to pursue here is that of capacitated VRP [47, 48, 49], where the capacity of each individual tour is bounded and there is then some additional global objective to achieve (fewest number of tours, total weight for some fixed number of tours, etc.). Bounded capacity in this context means that every tour can only visit some bounded number of vertices, possibly weighted. However this still may leave our salespeople a bit dissatisfied; If one of them visits only customers very close to their starting point while another visits the same number of customers that are all very far from each other the second salesperson still ends up with a lot more work.

We therefore may instead ask for a collection of cycles $\{C_i\}_i$ -possibly based at some depots- covering every point of V while minimizing the length of the longest such cycle $\max w(C_i)$. This has been studied as Min-Max Cycle Cover [50], and more generally as Min-Max {Star, Tree, Path}-Cover [51, 52, 53] where instead of cycles we are looking to get a collection of stars, trees, paths, or other connected subgraphs. For ease of presentation we will here focus on the case where we are looking to obtain a cover using trees rather than cycles, but notice that once such a cover has been found we can use the tree-doubling heuristic to transform this into a cycle cover instead loosing a factor 2 in the approximation ratio.

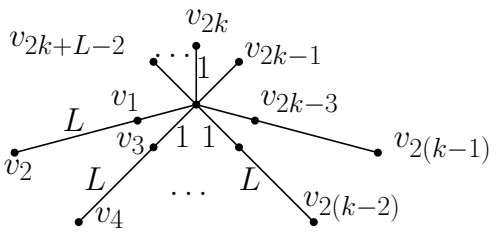
This problem formulation now becomes less of a routing problem and rather takes on the characteristics of typical clustering problems where we

are interested in assigning the points of the space to k groups such that the group sizes are optimal according to some measure; here the size of a group would be the size of a minimum spanning tree for the cluster.

Historically such clustering problems have been studied for two different objectives. On the one hand it is interesting to minimize the size of the largest cluster as in Min-Max Tree Cover; one may consider this as the ℓ_∞ norm of the vector of tree sizes. On the other hand it is still relevant that the total weight of the clustering (i.e. the ℓ_1 norm of the vector of tree sizes) is not too high. To see this consider that an optimal solution with respect to one of the objectives might be very bad with respect to the other one, and that this may indeed indicate a poor clustering from an intuitive viewpoint (see Figure 4.1).



(a) The path forms a metric where every two neighboring vertices in the set $\{v_1, \ldots, v_k\}$ have a distance of $1 + \varepsilon$, while the distance of all other edges is 1. In this case, the optimal solution under the ℓ_1 norm involves removing all the edges with a distance of $1 + \varepsilon$, resulting in an ℓ_∞ norm objective weight of $n - k$. However, the optimal solution for the ℓ_∞ norm has weight less than $n(1 + \varepsilon)/k$.



(b) A graph on $2k + L - 1$ vertices where all edges connecting a vertex v_{2i-3} (where $i \in [k]$) and v_i (where $k \leq i \leq 2k + L - 2$) to the center have a length of 1, and all edges $\{v_{2i-1}, v_{2i}\}_{i \in [k-1]}$ have some large length L. In this case, the optimal solution under the ℓ_∞ norm involves removing all the edges connected to the vertices $\{v_{2i-1}\}_{i \in [k-1]}$, resulting in an ℓ_1 norm objective weight of kL. However, the optimal solution for the ℓ_1 norm has a weight of $k + L - 1$.

Figure 4.1: Two instances of the All-ℓ_p-Norm Tree Cover problem. Figure 4.1a is an instance where an optimal ℓ_1 norm solution does not return a good approximation in the ℓ_∞ norm; Figure 4.1b is an instance where an optimal ℓ_∞ norm solution does not return a good approximation in the ℓ_1 norm.

This phenomenon of there being some trade-off between minimizing the ℓ_1 norm and the ℓ_∞ norm of some size vector of a clustering leads us to the question whether both can be minimized simultaneously, at least up to some constant factor. I.e. is it possible to find clusterings that are both cheap globally (in the ℓ_1 norm) but also cheap locally for each cluster individually

(in the ℓ_∞ norm). Extending this question we may even ask if we can compute clusterings that achieve some uniform approximation guarantee with respect to *any* ℓ_p norm.

Finding such a solution that is approximately optimal with respect to any way of measuring it gives rise to the following problem primitive:

Definition 4.1 (All-ℓ_p-norm clustering problem)**.** An *all-ℓ_p-norm clustering problem* takes as input a point set V, a weight function $w : \mathcal{P}(V) \to \mathbb{R}_{\geq 0}$, and a non-negative integer k. The goal is to find clusters $V_1, \ldots, V_k$ such that $\bigcup_i V_i = V$ while minimizing

$$\alpha = \max_{p \in \mathbb{R}_{\geq 1} \cup \{\infty\}} \frac{\left(\sum_i \left(w(V_i)\right)^p\right)^{1/p}}{w_p^*},$$

where w_p^* is an optimal solution for k clustering problem under the ℓ_p norm objective. An algorithm guaranteeing a constant α not depending on p in this context is deemed to be a constant-factor approximation algorithm.

A solution to such an all-norm optimization problem can in some way be considered to be robust and fair, i.e. it ensures an equitable distribution of the weights between the clusters without blowing up the total weight or the maximum weight too much. For example if the points of the space represent energy prosumers that we want to connect together into some fixed number of networks it is desirable to have similar sizes of the subnetworks to ensure stability of each; however building very long and thus expensive connections only to gain some more balance might be overly expensive, so solutions to such problems do not need to be optimal with respect to any particular norm, but rather "good" with respect to all norms. Because of this viewpoint of a fair distribution of some total load such all-ℓ_p-norm approximation problems have also been considered in the context of scheduling [54, 55, 56, 57], clustering [58, 59], or facility location [60].

One may even extend this formulation beyond just the ℓ_p norms and consider all norms. However there is the risk that such problems may not have reasonable solution, or rather that instance classes exist where α can not be finite depending on the choice of weight function and the set of norms considered. Indeed for the set of all norms attaining constant α is not possible. Consider that there are in particular (semi-)norms that only measure the absolute value of some particular index of the input vector. For our Tree Cover problem it is possible to find solutions where any fixed index is 0, as long as the graph is connected. Approximating all of these simultaneously to a constant factor then requires a solution whose trees are all empty, which is impossible in general.

Thus we focus ourselves here to considering the all-ℓ_p-norm clustering problem where concretely the weight $w(V_i)$ of each cluster is the weight of

a minimum spanning tree on V_i with respect to some metric d on V. We call this problem the All-ℓ_p-Norm Tree Cover problem, and denote its instances by (V, d, w, k). Note that k is part of the input, i.e. we do not parameterize on the number of clusters.

The choice of trees here is somewhat arbitrary, as a constant factor approximation with respect to trees also extends to cycles, paths, etc.. We simply employ trees as a basic connectivity primitive in graphs.

The results we will be able to show do in fact cover the more general setting of *monotone symmetric norms*, including not just ℓ_p norms but also Top-ℓ-norms and positive linear combinations thereof. A norm $f : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ is *symmetric* if it is invariant under permutation of its input coordinates. It is *monotone* if increasing a coordinate's value does not decrease the norm, i.e. if $|a| < |b|$ implies that

$$f(x_1, \ldots, a, \ldots, x_d) \leq f(x_1, \ldots, b, \ldots, x_d).$$

For an in-depth characterisation of such norms refer to Chakrabarty and Swamy [57], for our purposes it suffices to note that in particular the ℓ_p norms are monotone and symmetric, as are for example Top-ℓ-norms.

To mirror the previous work on MDMTSP we also consider a second setting where every tree should be connected to some specified depot and refer to this as the All-ℓ_p-Norm Tree Cover with Depots problem whose instances we denote by (V, d, w, D) where $D \subseteq V$ is the set of depots. In this context it is assumed that every depot is allowed to be the root of one tree in the solution, i.e. the number of depots and agents coincide.

Our contributions

Computing optimal tree covers is NP-hard to approximate to within a factor better than 3/2 even with respect to only the ℓ_∞ norm [61], but this setting does still admit a 3-approximation in polynomial time [52]. Meanwhile for the ℓ_1 norm it is of course possible to compute optimal tree covers exactly in polynomial time using for example the algorithm of Borůvka [62].

Our goal here is to design constant-factor approximations for All-ℓ_p-Norm Tree Cover, that is, for *all* ℓ_p norms simultaneously. Since an exact solution is excluded unless P=NP this is the best we can hope for up to the exact constant.

Building on the examples in Figure 4.1 it is clear that simply computing a constant-factor approximation for one norm objective does not guarantee a constant approximation for another norm. Indeed the algorithms of Even et al. [51] and Khani and Salavatipour [52] fail to return a constant-factor approximation even when considering only the ℓ_1 norm (for an example, see Figure 4.2 in Section 4.2).

Our first main contribution is a polynomial-time constant-factor approximation algorithm for the All-ℓ_p-Norm Tree Cover problem. The core of our algorithm amalgamates the strategies of suitable algorithms for both the ℓ_1 norm and the ℓ_∞ norm. It is well-understood that an optimal solution for the ℓ_1 norm objective can be obtained successively eliminating the largest edges until precisely k components remain. This is equivalent to the algorithm of Kruskal [63], but run in reverse. On the other hand the algorithm of Even et al. works by guessing a optimum solution value R and decomposing the graph into at most k trees of size $\mathcal{O}(R)$ as long as the guessed value was above the true OPT[51]. We follow essentially the outline of this algorithm, but we also make sure that (approximately) the most expensive edges of the graph have been deleted so that we obtain exactly k trees.

In that sense we refine the algorithm proposed by Even et al. [51] to solve other norm problems. Notably, our algorithm produces a feasible solution that is at most double the optimal solution for the ℓ_1 norm and four times the optimal solution for the ℓ_∞ norm simultaneously.

Theorem 4.1. *There exists a polynomial-time $\mathcal{O}(1)$-approximation algorithm for the* All-ℓ_p-Norm Tree Cover *problem.*

We extend this result to the problem with depots. The Min-Max Tree Cover problem with depots was also considered by Even et al. [51], who gave a 4-approximation algorithm. In this formulation, each tree in a tree cover solution is rooted at a specific depot. However also here the algorithm does not guarantee any constant approximation ratio for the ℓ_1 norm objective. We are able to extend the $\mathcal{O}(1)$-approximation algorithm for the tree cover problem without depots to the depot setting.

Theorem 4.2. *There exists a polynomial-time $\mathcal{O}(1)$-approximation algorithm for the* All-ℓ_p-Norm Tree Cover with Depots *problem.*

Finally we complement the algorithmic results by showing that we can not expect approximation schemes, even in the presumably easier setting ℓ_p-Tree Cover with Depots where we only want to approximate the optimum with respect to one specific ℓ_p norm.

Theorem 4.3. *For every $p \in (1, \infty]$ there exists a constant c such that ℓ_p-Tree Cover with Depots is* NP*-hard to approximate within a factor c. The* NP*-hardness holds under randomized reductions.*

4.1 Preliminaries

Let us now set up some of the formal definitions. As before we will identify metric spaces and metrically weighted graphs to allow for an easier treatment of connectedness, trees, etc.. For any such graph G we always keep

in mind the underlying metric space induced by the shortest-path metric on G.

We here concern ourselves only with tree covers, which we take to be defined as follows:

Definition 4.2 (Tree cover)**.** For a graph G, a *tree cover* is a collection $T_1, \ldots, T_k$ of trees for which $\bigcup_i V(T_i) = V(G)$. We call k the *size* of such a tree cover.

Definition 4.3 (Tree cover with depots)**.** For a graph G and a depot set $D \subseteq V(G)$, a *tree cover with depots* is a tree cover $T_1, \ldots, T_{|D|}$ of G if each T_i contains a unique depot from D.

Notice in particular that there is no expectation of disjointness for the trees, so this definition does not quite coincide with the notion of a CSF. However this more relaxed view will simplify the presentation. If disjointness is required we may assign every vertex to exactly one of the trees currently containing it and recompute minimum spanning trees for each of the resulting clusters. This increases the cluster weights by at most a factor of 2 due to the gap between Steiner trees and minimum spanning trees (see Lemma 4.1).

For any connected subgraph H of a graph G, we will use $w(H)$ to denote the weight of a minimum spanning tree (MST) in the subgraph H. It is important to note that this weight can be different from the sum of the weight of the edges of the subgraph H. We then use the ℓ_p norm ($p \geq 1$) as a measure of the quality of a tree cover. Specifically, the weight of a tree cover $\{T_1, T_2, \ldots\}$ is defined as the ℓ_p norm of the corresponding tree weights, that is, $w_p = \left(\sum_i (w(T_i))^p\right)^{1/p}$.

There are two natural optimization questions for tree covers (and for tree covers with depots) which have been considered in the literature: The ℓ_1-Tree Cover Problem, where the aim is to minimize the total weights of the trees, and the ℓ_∞-Tree Cover Problem, in which the objective is to minimize the weight of the largest tree within the cover. The ℓ_1-Tree Cover Problem admits a polynomial-time algorithm, regardless of the presence of depots; the ℓ_∞ tree cover problem is APX-hard, again regardless of whether depots are present or not.

We consider the interpolation between these two problems where we allow arbitrary ℓ_p norms:

Definition 4.4 (ℓ_p-Tree Cover (with Depots))**.** For a given (complete) graph $G = (V, d)$, some integer k (resp. depots $D \subseteq V$), and a $p \in [1, \infty)$, find a tree cover $T_1, \ldots, T_k$ (resp. with depots) which minimizes

$$\left(\sum_{i=1}^{k} (w(T_i))^p\right)^{1/p}.$$

We will denote the minimum value of this expression for some instance as $OPT_{p,k}$ just $w^*_{p,k}$, where we usually omit k if it is clear from context. All of the ℓ_p-variants (except $p = 1$) are NP-hard, because the proof of hardness for the ℓ_∞-case works for all values of $p > 1$ [61].

Definition 4.5 (All-p-Norm Tree Cover Problem (resp. All-p-Norm Tree Cover Problem with Depots))**.** Given a graph $G = (V, d)$ and some integer k (resp. depots $D \subseteq V$), find a tree cover $T_1, \ldots, T_k$ (resp. with depots) which minimizes

$$\max_{p\in[1,\infty)} \frac{\left(\sum_{i=1}^{k}(w(T_i))^p\right)^{1/p}}{w^*_p} \ .$$

To distinguish the versions for a fixed p from those for all p we use (ℓ_p, k) (or (ℓ_p, D) for problems involving depots) to denote the tree cover problem for a specific value of p. Meanwhile we use $(\ell_{p\in[1,\infty)}, k)$ (or $(\ell_{p\in[1,\infty)}, D)$ when depots are involved) to represent the *All-ℓ_p-Norm tree cover problem* that encompasses all values of $p \in [1, \infty)$. We omit naming the underlying metric space since it will always be (V, d), unless explicitly stated otherwise.

We will now briefly touch on the gap between Steiner trees and minimum spanning trees since this argument will be repeatedly exploited:

Lemma 4.1 (Kou et al. [64])**.** *Let (V, d) be some metric space with a terminal set $F \subseteq V$, $\hat{T}$ a minimum weight (Steiner) tree containing all vertices from F, and T a minimum spanning tree of $(F, d\big|_F)$. Then we have that $w(T) \leq (2 - 2/|F|) \cdot w(\hat{T})$.*

Proof. The easiest way to obtain the result is to take $\hat{T}$ and use the tree-doubling heuristic to compute a TSP-tour of F in (V, d) such that its weight is bounded by $2 \cdot w(\hat{T})$. Removing the most expensive edge of that tour yields a (perhaps not yet minimal) spanning tree T' in $(F, d\big|_F)$ of weight at most $(2 - 2/|F|)w(\hat{T})$, allowing us to conclude

$$w(T) \leq w(T') \leq (2 - 2/|F|)w(\hat{T}). \qquad \square$$

Lemma 4.1 will prove useful as we occasionally want to forget about some vertices of our instance. This may actually increase overall weights, since certain Steiner trees become unavailable. However the lemma ensures that the increase in weight is bounded.

4.2 All-ℓ_p-Norm Tree Cover Problem

Algorithm for the no-depot setting: We show that the Tree Cover algorithm of Even et al. [51] gives a constant-factor approximation to the

ℓ_p-tree cover problem without depots if it returns k trees. The original algorithm works as follows:

1. Guess an upper bound R on the optimum value of the ℓ_∞ norm tree cover problem.
2. Remove all edges with weight greater than R and compute a minimum spanning tree T_i for each connected component of the resulting graph.
3. Check that $\sum \lfloor \frac{w(T_i)+2R}{2R} \rfloor = k$, otherwise reject R.
4. Call a spanning tree T_i *small* if $w(T_i) < 2R$.
5. Call a spanning tree T_i *large* if $w(T_i) \geq 2R$. Decompose each such tree into edge-disjoint subtrees $\tilde{T}_j$ such that $2R \leq w(\tilde{T}_j) \leq 4R$ for all but one residual $\tilde{T}_j$ in each component[1].
6. Output the collection of all small spanning trees, as well as all subtrees $\tilde{T}_j$ created in Step 5.

Even at al. show that this procedure will output *at most* k trees if $OPT_\infty \leq R$, although they relax the condition in Step 3 to $\sum \lfloor \frac{w(T_i)+2R}{2R} \rfloor \leq k$. For technical reasons however, we need the equality in this spot. Since every tree has size most $4R$, the algorithm evidently gives a 4-approximation in the ℓ_∞ norm if the correct $R = OPT_{\infty,k}$ is guessed. Otherwise one can use binary search to find in polynomial time the smallest R for which one gets at most k trees.

We notice that this algorithm will sometimes compute fewer trees than k, causing it to not return a good approximation for the other ℓ_p-objectives, not even for ℓ_1 (see Figure 4.2).

For this reason we need the strengthened property in Step 3. Notice however that such R does not necessarily exist, for example for the instance in Figure 4.2. We will describe later how to circumvent this, for now we assume that such R exists and can be computed in polynomial time.

So suppose that the algorithm has returned k trees $T_1, \ldots, T_k$, where we simply remove some edges should the algorithm return fewer than k trees. This only improves the objective value, so we can assume without loss of generality that the algorithm already returned k trees. As a warm-up, we will start by considering only the case that $p = 1$, i.e., we show that this algorithm computes a good approximation for (ℓ_1, k) tree cover.

Lemma 4.2. *Let $T_1, \ldots, T_k$ be the set of trees returned by the algorithm for some R. Then we have*

$$\sum w(T_i) \leq 2OPT_{1,k} \ .$$

[1]This can be done with a simple greedy procedure, cutting of subtrees of this size. For the details of this algorithm, er to Even et al. [51].

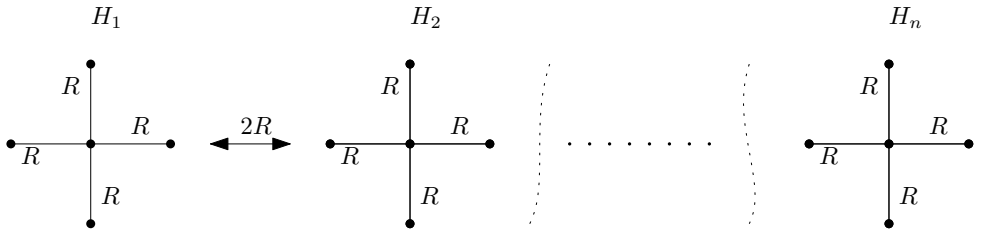


Figure 4.2: Example instance where the algorithm of Even et al. does not return a good approximation in the ℓ_1-objective. The instance consists of n identical copies H_i of the 4-star where all edges have length R and the copies are pairwise at distance $2R$. For $k = 5n - 1$, the instance has $OPT_\infty = OPT_1 = R$, but the algorithm will return $2n$ trees, each with a size of $2R$.

Proof. We observe first that an optimal solution for (ℓ_1, k) tree cover can be computed by removing iteratively the most expensive edge as long as this does not cause the graph to decompose into more than k components, as this is equivalent to running Kruskal's algorithm. So the optimal solution contains no edges of weight greater than R, thus we consider the graph $G' := G - \{e \mid d(e) > R\}$. Let k_s be the number of small spanning trees S_i computed by the algorithm, and let k_ℓ be the number of large spanning trees L_i. Then we certainly have $\sum w(T_i) \leq \sum w(S_i) + \sum w(L_i)$, since the trees computed by the algorithm are edge-disjoint subtrees of the initial spanning trees. Further, we have

$$OPT_{1,k} \geq \sum w(S_i) + \sum w(L_i) - (k - k_s - k_\ell)R,$$

because the optimal solution will remove $k - k_s - k_\ell$ edges from within the spanning trees computed by the algorithm, each of weight at most R. But notice that we can use Step 3 to obtain

$$k = \sum \left\lfloor \frac{w(S_i)}{2R} + 1 \right\rfloor + \sum \left\lfloor \frac{w(L_i)}{2R} + 1 \right\rfloor \leq k_s + k_l + \sum \frac{w(L_i)}{2R},$$

which implies that $(k - k_s - k_\ell)2R \leq \sum w(L_i)$. This inequality allows us to conclude that

$$\begin{aligned} OPT_{1,k} &\geq \sum w(S_i) + \sum w(L_i) - \frac{1}{2}\sum w(L_i) \\ &\geq \frac{1}{2}(\sum w(S_i) + \sum w(L_i)) \\ &\geq \frac{1}{2}\sum w(T_i) \ . \end{aligned}$$

□

Notice that the strengthened version of Step 3 is indeed necessary here.

We can use a similar technique to show that the solution of the algorithm actually approximates (ℓ_p, k) tree cover for any choice of p, and with a constant of approximation independent of p. The key will be to show that an optimal solution to (ℓ_p, k) tree cover must, as in Lemma 4.2, have a total weight comparable to that of the solution computed by the algorithm. In a second step, one can then show that the algorithm distributes the total weight of its trees fairly evenly because the large trees all have sizes between $2R$ and $4R$.

For the small trees meanwhile we can demonstrate that choosing them differently from the algorithm will incur some weight of at least R. Thus, either the algorithms solution has correctly chosen the partition on these small trees, or the optimal solution actually has an expensive tree not in the algorithms solution, which we can use to pay for one of the large trees that the algorithm has, but the optimal solution does not. Notice that, if the algorithm achieves an equal distribution of some total weight that is comparable to the total weight of an optimal ℓ_p solution, it also achieves a good approximation of OPT_p due to convexity of the ℓ_p norms.

To start with, we will only give a very rough analysis of the quality of the solution returned by the algorithm, although it already shows a constant factor of approximation. Thereafter, in Section 4.4, we provide a more precise analysis of the algorithm to argue that it achieves a smaller constant approximation factor, specifically a 9-approximation.

Let us fix some $p \in [1, \infty)$ and any (ℓ_p, k) tree cover solution $\hat{T}_1, \ldots, \hat{T}_k$ (imagine that it is optimal if you wish). Then let k_s be the number of connected components computed by the algorithm that had a small spanning tree, and call those components $S_1, \ldots, S_{k_s}$. Similarly for the large spanning trees, let there be k_ℓ components $L_1, \ldots, L_{k_\ell}$. Now we denote by E_ℓ the set of edges that are both contained in some $\hat{T}_i$ and in the cut induced by some S_j or L_j. In particular all these edges have weight at least R. We count separately the small T_i incident to some edge from E_ℓ, say there are $k_{s,1}$ of them. We will also denote by $k_{s,2}$ the number of S_i for which one of the $\hat{T}_j$ is a minimum spanning tree, and denote the set of these $\hat{T}_j$ as $T^=$. Note that any small component not counted by $k_{s,1}$ or $k_{s,2}$ is split into at least two components by the $\hat{T}_i$. For an illustration of this setting, refer to Figure 4.3.

We can now start to measure the total sum of edge weights in the $\hat{T}_j$, i.e., $\sum_j w(\hat{T}_j)$. We begin by relating it to the small trees of the algorithm's solution that do not agree with the optimum solution.

Observation 4.1. *It holds that* $\sum_j w(\hat{T}_j) - \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) \geq k_{s,1} \frac{R}{2}$.

Proof. We have $k_{s,1}$ disjoint vertex sets in G, each incident to at least one edge of weight at least R present in E_ℓ, so we get $|E_\ell| \geq k_{s,1}/2$ from the

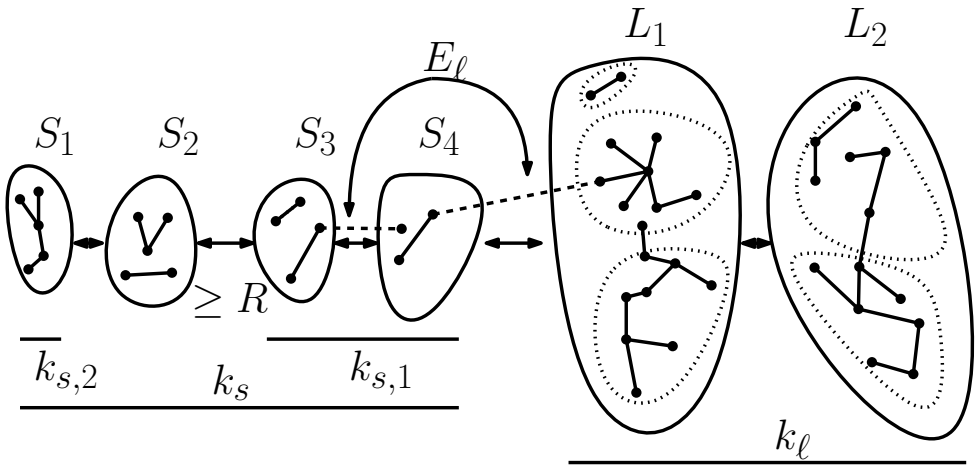


Figure 4.3: Illustration of how the algorithm's solution and some optimal solution can be aligned against each other. The algorithm's partition of the graph into connected component is indicated by the solid zones, the further subdivision of the large components by the dotted zones. The trees of the optimal solution are drawn, and all edges crossing the boundary of a connected component are marked as dashed.

handshake lemma and thus

$$\sum_j w(\hat{T}_j) - \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) \geq |E_\ell| R \geq \frac{k_{s,1}}{2} R,$$

noting that the trees in $T^=$ do not contain any of the edges from E_ℓ. □

To compare the total weight against the number of large trees we can use a similar argument as in Lemma 4.2. We again note that the initial spanning trees have weight at least $(k-k_\ell-k_s)2R$, and any tree cover cannot remove too many edges from them. The second part of this argument is no longer true though since it is now possible for a solution to have many edges between the components of $G - \{e \mid d(e) \geq R\}$, allowing it to remove many edges within the components However, careful analysis will show that this case does not pose a problem, since such inter-component edges are themselves expensive.

Observation 4.2. *We have* $\sum_j w(\hat{T}_j) \geq (k-k_\ell-k_{s,1}-k_{s,2})R + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)$.

Proof. Suppose we delete from the $\hat{T}_j$ all edges from E_ℓ. The resulting graph will contain $k + |E_\ell|$ trees. We will count only those trees that lie in a large component L_i, of which there are at most $k + |E_\ell| - 2k_s + k_{s,1} + k_{s,2}$. This is because every small component contains at least 2 of the $\hat{T}_j$, except for $k_{s,1} + k_{s,2}$ many. Now observe that to get a $(k + |E_\ell| - 2k_s + k_{s,1} + k_{s,2})$-component spanning forest of *minimum* weight for the L_i, we start with the

initial minimum spanning trees in each component (which have weight at least $(k - k_s - k_\ell)2R$) and remove at most $(k + |E_\ell| - k_\ell - 2k_s + k_{s,1} + k_{s,2})$ edges, each of weight at most R. The total remaining weight then is

$$\begin{aligned}&(k - k_s - k_\ell)2R - (k + |E_\ell| - k_\ell - 2k_s + k_{s,1} + k_{s,2})\\ &= (k - k_\ell - k_{s,1} - k_{s,2} - |E_\ell|)R.\end{aligned}$$

Thus, we can measure the total weight of edges which are part of some $\hat{T}_i$ and lie in a large component as being at least the total weight the spanning trees of the L_i, minus the edges that may have been removed, and obtain

$$\begin{aligned}&\sum_j w(\hat{T}_j) - \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) - |E_\ell|R && \geq (k - k_\ell - k_{s,1} - k_{s,2} - |E_\ell|)R.\\ \implies &\sum_j w(\hat{T}_j) - \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) && \geq (k - k_\ell - k_{s,1} - k_{s,2})R. \qquad \square\end{aligned}$$

Notice that with these two observations, we can almost show some result along the lines of $OPT_{1,k} \geq c{\cdot}k{\cdot}R$ for some $c \in \mathbb{R}$, since either there are many large trees, in which case Observation 4.2 gives us the desired result, or there are many small trees in which case we can use Observation 4.1, unless $k_{s,2}$ is large. However, the case that $k_{s,2}$ is large, i.e., we have computed many of the trees that are present in the optimum solution, should also be beneficial, so we maintain a separate record of $k_{s,2}$ in the analysis to obtain:

Lemma 4.3. *It holds that* $\sum_j w(\hat{T}_j) \geq \frac{R}{6}(k - k_{s,2}) + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)$.

Proof. We can combine Observation 4.2 and Observation 4.1 convexly with coefficients $1/3, 2/3$ to obtain

$$\begin{aligned}&\sum_j w(\hat{T}_j) \geq \frac{1}{3}\left[(k - k_\ell - k_{s,1} - k_{s,2})R\right] + \frac{2}{3}\left[k_{s,1}\frac{R}{2}\right] + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)\\ \iff &\sum_j w(\hat{T}_j) \geq \frac{R}{3}(k - k_\ell - k_{s,2}) + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)\\ \implies &\sum_j w(\hat{T}_j) \geq \frac{R}{6}(k - k_{s,2}) + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i),\end{aligned}$$

where the final inequality follows from the following reasoning:

$$k = \sum \left\lfloor \frac{w(S_i) + 2R}{2R} \right\rfloor + \sum \left\lfloor \frac{w(L_i) + 2R}{2R} \right\rfloor \geq k_s + 2k_\ell,$$

and thus

$$k - k_\ell - k_{s,2} \geq k - \frac{k - k_s}{2} - k_{s,2} = \frac{k}{2} + \frac{k_s}{2} - k_{s,2} \geq \frac{k - k_{s,2}}{2} \quad . \qquad \square$$

The upshot of Lemma 4.3 is then that any tree cover using k trees with some $k_{s,2}$ trees in common with the algorithm's solution will have the property that the other $k - k_{s,2}$ trees have an average weight of $\Omega(R)$.

Theorem 4.4. *If the algorithm of Even et al. returns k trees $T_1, \ldots, T_k$, we have*

$$\left[\sum_{i=1}^{k}(w(T_i)^p)\right]^{1/p} \leq 24 \left[\sum_{i=1}^{k}(w(\hat{T}_i)^p)\right]^{1/p}$$

for any p and for any tree cover $\hat{T}_i$ of G with k trees.

Proof. From Lemma 4.3 we obtain

$$\sum_{i=1}^{k}(w(\hat{T}_i)^p) \geq (k - k_{s,2}) \left(\frac{R}{6}\right)^p + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)^p$$

using convexity of $|x|^p$ for any $p \geq 1$. At the same time, from the algorithm it is immediately clear that

$$\sum_{i=1}^{k}(w(T_i)^p) \leq (k - k_{s,2})(4R)^p + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)^p,$$

because every tree is either identical to one of the $\hat{T}_j$, or has weight at most $4R$. Putting these inequalities together yields the statement of the theorem:

$$\begin{aligned}
\frac{\sum_{i=1}^{k}(w(T_i)^p)}{\sum_{i=1}^{k}(w(\hat{T}_i)^p)} &\leq \frac{(k - k_{s,2})(4R)^p + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)^p}{(k - k_{s,2})\left(\frac{R}{6}\right)^p + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)^p} \\
&\leq \frac{(k - k_{s,2})(4R)^p}{(k - k_{s,2})\left(\frac{R}{6}\right)^p} \\
&\leq 24^p \ .
\end{aligned}$$

□

The whole proof, in particular the result of Lemma 4.3 can be reinterpreted to give an even stronger result; Namely that the tree cover returned by the algorithm is approximately *strongly optimal*, an concept introduced by Alon et al. [65] for analysing all-norm scheduling algorithms. The point is to show a strong version of lexicographic minimality of the constructed solution. Formally let $T_1, \ldots, T_k$ be the algorithms solution and $\hat{T}_1, \ldots, \hat{T}_k$ any other tree cover. Further let the trees be sorted by size as $w(T_1) \geq w(T_2) \geq \ldots$ and $w(\hat{T}_1) \geq w(\hat{T}_2) \geq \ldots$. Then $\{T_i\}$ is strongly optimal if for any j we have

$$\sum_{i=1}^{j} w(T_i) \leq \sum_{i=1}^{j} w(\hat{T}_i).$$

Similarly one can speak of approximate strong optimality if for some $c \in \mathbb{R}_{\geq 1}$ we have

$$\sum_{i=1}^{j} w(T_i) \leq c \sum_{i=1}^{j} w(\hat{T}_i).$$

This property was also considered as global c-balance by Goel and Meyerson [66]. It is sufficient for our purposes since a solution that is c-approximately strongly optimal is also a c-approximation with respect to any ℓ_p norm; in fact we get a stronger result here, since c-approximate strong optimality implies a c approximation with respect to *any* convex symmetric function, including all monotone symmetric norms. This fact is the backbone of many all-norm approximation algorithms, for example [55]. Thus we will get a 24-approximation from this analysis not only for all ℓ_p norms, but all monotone symmetric norms.

Lemma 4.4. *Let $T_1, \ldots, T_k$ be the algorithms solution and $\hat{T}_1, \ldots, \hat{T}_k$ any other tree cover where $w(T_1) \geq w(T_2) \geq \ldots$ and $w(\hat{T}_1) \geq w(\hat{T}_2) \geq \ldots$. Then we have*

$$\sum_{i=1}^{j} w(T_i) \leq 24 \sum_{i=1}^{j} w(\hat{T}_i) \; \forall j \in 1, \ldots, k.$$

Proof. We upper bound the values $\sum_{i=1}^{j} w(T_i)$ by assuming that all trees which are not accounted for by the $k_{s,2}$ small component trees shared between the T_i and the $\hat{T}_i$ have size exactly $4R$. This will ensure that the shared trees are $T_{k-k_{s,2}+1}, \ldots, T_k$. Similarly we lower bound $\sum_{i=1}^{j} w(\hat{T}_i)$ by allowing a non-descending permutation of the $\hat{T}_i$. That is we reorder the $\hat{T}_i$ such that the first $k - k_{s,2}$ trees are not the shared small component trees with the T_i.

It then follows directly that

$$\sum_{i=1}^{j} w(T_i) \leq j \cdot 4R = 24(j \cdot \frac{R}{6}) \leq 24 \sum_{i=1}^{j} w(\hat{T}_i) \text{ for } j \leq k - k_{s,2},$$

as well as

$$\sum_{i=1}^{k-k_{s,2}} w(T_i) + \sum_{i=1+k-k_{s,2}}^{j} w(T_i) \leq 24 \sum_{i=1}^{k-k_{s,2}} w(\hat{T}_i) + \sum_{i=1+k-k_{s,2}}^{j} w(\hat{T}_i)$$

$$\implies \sum_{i=1}^{j} w(T_i) \leq 24 \sum_{i=1}^{j} w(\hat{T}_i) \text{ for } j > k - k_{s,2}. \qquad \square$$

Ensuring k trees Recall that the previous analysis relies on the algorithm being able to find some R such that Step 3 holds, which is not generally true as per Figure 4.2. We now demonstrate how to modify the algorithm in such a way that this is avoided. Consider a list $e_1, \dots, e_m$ of the edges of G, such that $d(e_i) \leq d(e_j)$ if $i \leq j$, i.e., they are sorted by length with ties broken arbitrarily. Then we consider separately the graphs $G_j := (V, \{e_i \mid i \leq j\})$ for all $j \in 0, \dots, m$ and try to find for each G_j an $R \in [d(e_{j-1}), d(e_j)]$ that is accepted by the algorithm, where we set $d(e_0) := 0$, and allow the larger interval $[d(e_m), \sum_i d(e_i)]$ for G_m. Note that the intervals can potentially only contain a single point, but they are not empty.

Now observe that for G_0 with $R = 0$, the algorithm would only accept this choice of R if $k = |V|$. Similarly, for G_m and $R = \sum_i d(e_i)$, the algorithm would require $k = 1$. We can then show that the k changes by at most one as we change R or keep R and move from G_j to G_{j+1}. Letting $k(G_j, R)$ denote the value of k that would be accepted by the algorithm we have:

Observation 4.3. *It holds that $k(G_{j-1}, d(e_j)) - k(G_j, d(e_j)) \leq 1$.*

Proof. Between G_{j-1} and G_j only the presence of e_j changes. If this does not change the connected components, we have $k(G_{j-1}, d(e_j)) = k(G_j, d(e_j))$. Otherwise there is exactly one connected component C in G_j that is split into two parts C_1, C_2 by removing e_j. We then see that

$$\begin{aligned}\left\lfloor \frac{w(C_1) + 2d(e_j)}{2d(e_j)} \right\rfloor + \left\lfloor \frac{w(C_2) + 2d(e_j)}{2d(e_j)} \right\rfloor &\leq 2 + \left\lfloor \frac{w(C_1)}{2d(e_j)} + \frac{w(C_2)}{2d(e_j)} \right\rfloor \\ &\leq 1 + \left\lfloor \frac{w(C) + 2d(e_j)}{2d(e_j)} \right\rfloor \quad . \qquad \square\end{aligned}$$

To see that changing R by a sufficiently small amount changes $k(G_j, R)$ by at most one, consider that there are only finitely many critical points where $k(G_j, R)$ changes at all, and they can be computed in polynomial time. They are all of the form $R = w(C_i)/2\ell$ for some connected component C_i of G_j and $\ell \in 1, \dots, n$. At these points, $w(C_i)/2R$ will be an integer, and so we get $\lfloor w(C_i)/2R \rfloor = 1 + \lfloor (w(C_i) - \varepsilon)/2R \rfloor$.

If all critical points are pairwise different, we can just iterate over them to find some G_j and R with $k(G_j, R) = k$.

If on the other hand a point is critical for multiple different components, assign to each component a distinct weight which can be taken to be arbitrarily small, ensuring that all critical points are now different. In effect, this is equivalent to taking all components where $\frac{w(C_i)+2R}{2R}$ is an integer and allowing the expression $\lfloor \frac{w(T_i)+2R}{2R} \rfloor$ to also take the value $\frac{w(T_i)}{2R}$. One may check that this does not impact the analysis, since in the analysis we only need that each component has a minimum spanning tree whose weight is at least $\lfloor \frac{w(T_i)}{2R} \rfloor \cdot 2R$. However, for legibility reasons we will suppress this and

generally assume that some R can be found with $k(G_j, R) = k$. Combining with Theorem 4.4, this completes the proof of Theorem 4.1.

4.3 All-ℓ_p-Norm Tree Cover Problem with Depots

We now introduce depots to our problem and try to maintain an All-ℓ_p norm approximation. In fact we will also here show c-approximate strong optimality of a solution, so this result too can be extended to all monotone symmetric norms. The basic idea is quite similar to the non-depot case; We will start by partitioning the metric space into trees of roughly equal size R, or, if that is not possible, some smaller trees that are however at distance R to all other trees.

However since now we need to connect to the depots this is no longer so straightforward. Vertices which are far from all depots will inherently cause a high weight, no matter where we assign them. To then keep the balance between tree size and distance between trees we need to categorize the vertices according to their distance to the depots. Specifically we partition V into *layers* $L_0, L_1, \ldots$ where layer L_i contains all $v \in V$ for which $d(v, D) \in [2^i, 2^{i+1})$. We then partition each layer individually with the algorithm of Even et al. [51], but using increasing values of $R_i = 2^i$ in layer L_i.

To simplify things we adapt the algorithm slightly. Layer i will be partitioned according to the following scheme:

1. Set $R_i = 2^i$.

2. Remove all edges with weight greater than R_i and compute a minimum spanning tree T_C for each connected component C of the resulting graph.

3. Call a spanning tree T_C *small* if $w(T_C) < 2R_i$.

4. Call a spanning tree T_C *large* if $w(T_C) \geq 2R_i$. Decompose each such tree into edge-disjoint subtrees $\tilde{T}_j$ such that $2R_i \leq w(\tilde{T}_j) \leq 6R_i$.

5. Output the collection of all small spanning trees, as well as all subtrees $\tilde{T}_j$ created in Step 4.

Here the decomposition in Step 4 avoids the creation of small trees from large component by adding such a small tree to a neighbouring large tree if it is necessary. This worsens the constant we can get, but it simplifies analysis considerably.

By running this decomposition for each layer separately we get a tree cover $\mathcal{T}_i$ for each layer i. However notice that one critical property of the no-depot setting is missing here; small component trees may not be far from all other vertices, they are only far from vertices in their layer. We sidestep this by separating the instance into two parts, the layers of even and odd indices. For $v \in L_i$ and $w \in L_{i+2}$ we are guaranteed from the triangle inequality that $d(v,w) \geq d(w,D) - d(v,D) \geq 2^{i+2} - 2 \cdot 2^i = 2^{i+1}$.

We thus treat the odd and even layer vertices as two separate instances. This incurs a factor of 2 in the approximation ratio since we now combine two unrelated instances, and an additional factor 2 because we lose some points of the metric space so we are constricted to finding spanning trees on the restricted space rather than Steiner trees in the whole space, see Lemma 4.1. So for all of the following we assume that all vertices are in even layers, where the analysis for all vertices being in even layers is fully analogue (see Figure 4.4 for illustration).

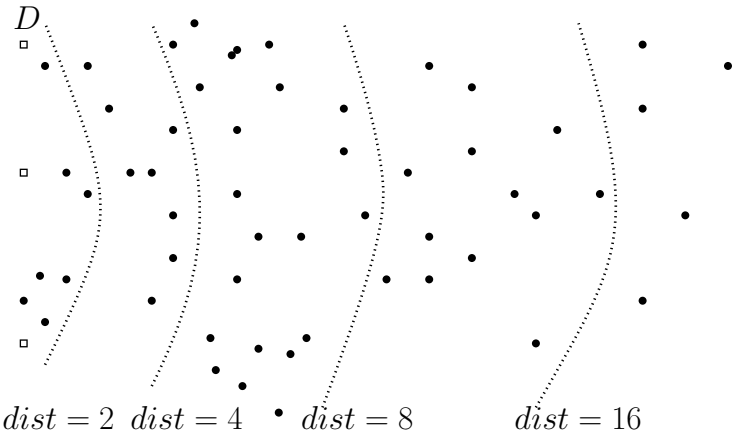
For this partitioning of the space into the trees from $\{\mathcal{T}_i\}$ we can show that any depot-based tree cover $\{\hat{T}_o\}_{o\in D}$ can be restructured as $\{\hat{T}'_o\}_{o\in D}$ so as to agree with the partitions, i.e. such that for any i and any $T \in \mathcal{T}_i$ there is a $o \in D$ for which $V(T) \subseteq V(\hat{T}'_o)$ and $w(\hat{T}_o) \leq c \cdot w(\hat{T}'_o)$.

Lemma 4.5. *Let $\{\hat{T}_o\}_{o\in D}$ be any tree cover with depots and $\{\mathcal{T}_i\}_i$ the trees computed in layer i by the partitioning algorithm. There exists a tree cover with depots $\{\hat{T}'_o\}_{o\in D}$ such that for any i and any $T \in \mathcal{T}_i$ there is a $o \in D$ for which $V(T) \subseteq V(\hat{T}'_o)$ and $w(\hat{T}_o) \leq c \cdot w(\hat{T}'_o)$, where $c = 48$.*

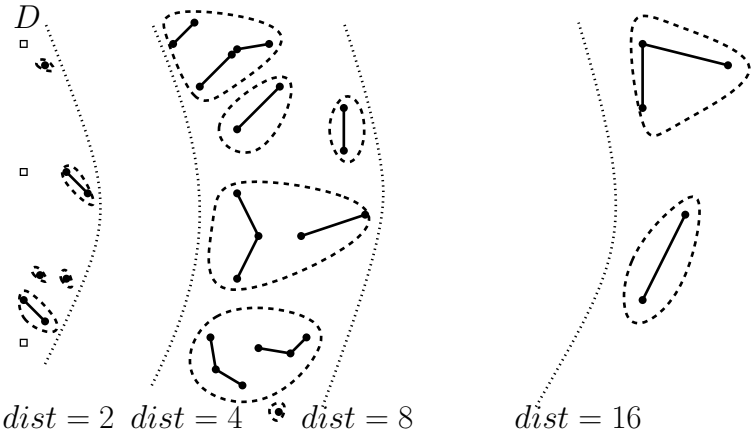
Proof. We will assign every T_i to some $\hat{T}_o$ in a way that does not increase the weight of any tree too much. Consider first a T_i generated from some small component C in layer L_j, and take $\hat{T}_o$ to be some tree containing a vertex from T_i. Then $w(T_i) \leq 2 \cdot 2^j$, and $\hat{T}_d$ contains an edge entering C. All those edges have weight at least 2^j if they are contained in layer j or coming from layer $j+2$, and 2^{j-1} if they are coming from layer $j-2$. Thus we can assign T_i to $\hat{T}_d$ and charge the extra weight against that edge, loosing a factor 8 at most.

For the large components we need a more careful argument. Let C be such a component, $\{F_i\}_i$ the collection of subtrees cut from C, and $\{\tilde{T}_o\}_o$ the trees $\hat{T}_o$ containing a vertex from C. For each $\tilde{T}_o$ we look at its edges $X_o := \{e \in E(\tilde{T}_o \mid e \cap C \neq \emptyset\}$ which are incident to C. Then we define a capacity of such a tree as $c(\tilde{T}_o) = \lfloor \lambda d(X_o)/2^j \rfloor$, where we will choose λ in a moment. This capacity represents how many of the F_i can be absorbed by $\tilde{T}_o$. Then for any constant choice of λ assigning at most $c(\tilde{T}_o)$ of the F_i to $\tilde{T}_o$ will only increase the weight of $\tilde{T}_o$ by a constant factor $6 \cdot \lambda$, charged against X_o.

To show that such an assignment exists we employ Hall's theorem. We allow any F_i to be assigned to a $\tilde{T}_o$ with which it shares a vertex, so we



(a) Initial instance, with the depots indicated as boxes and the vertices as dots. Vertices are precategorised by their distance to the depots.



(b) In the first step we discard every second layer and cluster each remaining layer independently. Connected components after removing edges of length $> R_i$ are indicated by dashed lines, the trees extracted within them by solid lines .

Figure 4.4: Illustration of the initial partitioning step of our algorithm.

construct the bipartite graph of this incidence system and verify Hall's criterion. Let $\{F_i\}_{i\in A}$ be some subset of the F_i, and $\{\tilde{T}_o\}_{o\in B}$ the set of $\tilde{T}_o$ incident to them. We must then show that $\sum_{o\in B} c(\tilde{T}_o) \geq |A|$. To do this define V_A as $\bigcup_{i\in A} V(F_i)$, and consider the number of connected components we obtain after removing from $\{\tilde{T}_o\}_{o\in B}$ all edges entering C, and counting only those components intersecting V_A. Call this number k. Then for each such connected component $\{\tilde{T}_o\}_{o\in B}$ contains a Steiner Tree within C.

Again here we exploit Lemma 4.1, as well as the fact that for each of the k components there is an edge of weight $\geq 2^{j-1}$ entering C. This allows us to conclude

$$\sum_{o\in B} d(X_o) \geq k2^{j-1} + \frac{1}{2}\max\{0, \sum_{i\in A} w(F_i) - k2^j\},$$

where we also note that if $k \geq |A|$ we could get an assignment with $\lambda = 2$. So we have $k < |A|$, and $w(F_i) \geq 2 \cdot 2^j$, thus obtaining:

$$\sum_{o \in B} d(X_o) \geq k2^{j-1} + |A|2^{j-1} = (|A| + k)2^{j-1}.$$

This gives Hall's criterion for $\lambda = 4$, concluding our argument.

Thus we can match any trees T in the $\{\mathcal{T}_i\}_i$ to some $\hat{T}_o$ while only increasing the weight by a factor at most 24, charged against some edges of the $\hat{T}_o$ incident to T. Since every edge gets charged at most twice by this the total increase in weight for each tree is bounded by a factor of 48. □

Notice that Lemma 4.5 allows us to assume that tree covers are aligned with the partition we compute while only increasing the size of each tree by a constant factor. This in particular means that we obtain a constant factor approximation for Tree Cover with Depots for any norm for which we can get a constant factor approximation after assuming such alignment.

Based on the guarantee of Lemma 4.5 we are now ready to develop an algorithm for assigning the trees in the $\mathcal{T}_i$ to depots. To do so we will need some additional notation. We partition every $\mathcal{T}_i$ into sets $\{\mathcal{T}_{i,o}\}_{o \in D}$ by assigning every $T \in \mathcal{T}_i$ to a $\mathcal{T}_{i,o}$ such that $d(T, o)$ is minimal over the choice of o. There is another simplification we can now make. For every tree T in some $\mathcal{T}_{i,o}$ we can add an edge $\{v, o\}$ to T such that $v \in V(T)$ and $d(v, o) = d(T, o)$. By similar arguments as before this increases the weight of our solutions by at most a factor of 4 since that additional edge has weight bounded by either $w(T)$, or the weight of any edge in $\delta(V(T))$. This causes all trees in $T_{i,o}$ to have pairwise distance 0, allowing us to recombine them without additional weights. Further, it also ensures that *all* trees in $\mathcal{T}_{i,o}$ have the same weight, up to constant. Specifically any such tree has weight at least 2^i from the edge to o and at most $8 \cdot 2^i$ if it was a large tree of size $6 \cdot 2^i$ and the edge to o had weight 2^{i+1}. Giving up on those constants in the approximation factor we shall therefore also assume that they all have size exactly 2^i.

To recap, after reductions we have a partitioning of the vertices into trees, each of which belongs to some collection $\mathcal{T}_{i,o}$ where any tree in $\mathcal{T}_{i,o}$ has size 2^i and is at distance 0 to o, but possibly at a much larger distance to the other depots. We further assume that any solution treats these trees as immutable, so in a fixed solution $\{\hat{T}_o\}_{o \in D}$ there is for every $T \in \bigcup_{i,o} \mathcal{T}_{i,o}$ a $\hat{T}_o$ containing T.

We will now try to iteratively match the trees in $\mathcal{T}_i$ to depots. To do so we set a current size R_i, initially 1. We then collect all trees of size $\Theta(R_i)$, and compute a matching between those trees and the depots at distance $\Theta(R_i)$ to them.

Any leftover trees after this step are paired up and treated as a new tree of size $2R_i$. We then update $R_{i+1} = 2R_i$ and repeat the process. There are some additional details to take care of, but in substance this will be enough to obtain a relatively even distribution of the trees to the depots. For the full description, refer to Algorithm 4; For an illustration see Figure 4.5.

Algorithm 4: Assignment algorithm

Input: A metric (V, d), depots D, trees $\{\mathcal{T}_{i,o}\}_{i,o}$ from the partitioning algorithm.

Output: An assignment of trees to depots

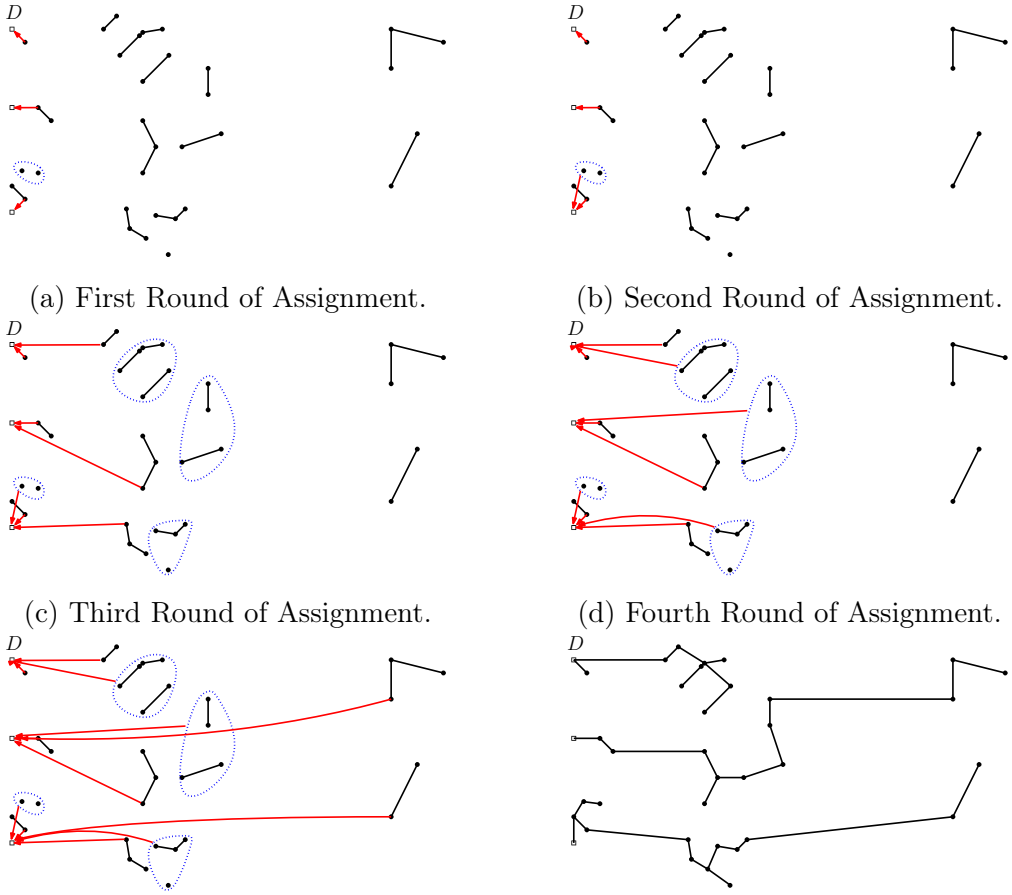
1 . $i \leftarrow 0, R_i \leftarrow 1$;
2 $\mathcal{F}_{i,o} \leftarrow \mathcal{T}_{i,o}$;
3 **while** $\{\mathcal{F}_o\}_o \neq \emptyset$ *for any* $o \in D$ **do**
4 Construct a bipartite matching graph H_i between the trees in $\mathcal{F}_{i,o}$ and the depots;
5 $V(H_i) := \bigcup_{o \in D} \mathcal{F}_o \dot{\cup} D$;
6 $E(H_i) := \{\{F, o\} \mid d(F, o) \leq R_i\}$;
7 Compute a maximal matching M_i in H_i and assign every matched tree to the depot it was matched to;
8 Remove all matched trees from the $\mathcal{F}_{i,o}$;
9 For every $\mathcal{F}_{i,o}$ with odd cardinality, assign one tree from $\mathcal{F}_{i,o}$ to o;
10 For every non-empty $\mathcal{F}_{i,o}$ pair up the trees in $\mathcal{F}_{i,o}$; for every pair F_1, F_2 set $\mathcal{F}_{i,o} := \mathcal{F}_{i,o} \setminus \{F_1, F_2\} \cup \{F_1 \cup F_2\}$;
11 $\mathcal{F}_{i+1,o} := \mathcal{F}_{i,o} \cup \mathcal{T}_{i+1,o}$;
12 $R_{i+1} \leftarrow 2R_i, i \leftarrow i + 1$;
13 **end**

Some notes on the algorithm to simplify analysis:

- In line 9, the possible additional assignment of a tree from F_o happens only when o was already assigned another tree in line 7. Thus the additional weight due to this step is an increase of the tree sizes by at most a factor 2. Loosing this factor we may thus assume that line 7 is never applied.

- The recombination of trees in line 10 is free due to the inclusion of edges to o for every tree from $\mathcal{T}_{i,o}$.

- The load of any depot at the termination of the algorithm depends only on the last iteration i in which it was assigned some tree, i.e. the load will be at most $2 \cdot 2^i$ by a geometric sum argument.



(a) First Round of Assignment.

(b) Second Round of Assignment.

(c) Third Round of Assignment.

(d) Fourth Round of Assignment.

(e) Fifth Round of Assignment.

(f) Final output, with minimum spanning trees computed over the assigned trees for every depot.

Figure 4.5: Illustration of the iterative assignment of trees to depots in the algorithm. Red arrows indicate when trees are assigned to depots; Blue dotted potatoes indicate merged trees.

To compare the output of the algorithm to any other depot-based tree cover $\{\hat{T}_o\}_{o\in D}$ we will employ the following scheme: For each size R_i we will cut up the trees $\hat{T}_o$ into parts of size R_i. If enough such parts exist we will then be able to assign every tree of size R_i from the algorithms solution to one such part, thus successfully demonstrating that the algorithms solution provides a "good" tree cover. So fix an arbitrary depot-based tree cover $\hat{\mathcal{T}} := \{\hat{T}_o\}_{o\in D}$ and denote by Λ_i the number of such parts we could get when extracting fragments of size R_i, i.e. $\Lambda_i := \sum_{o\in D}\lfloor \hat{T}_o/R_i \rfloor$. The critical lemma we need then is this:

Lemma 4.6.

$$\Lambda_i \geq \frac{1}{2} \sum_{j=i+1}^{\infty} |M_j| 2^{j-i},$$

where M_j are the matchings computed by the algorithm.

Proof. We split the analysis into two parts, depending on whether the size of $\sum_{j=i+1}^{\infty} |M_j| 2^{j-i}$ is largely caused by trees of size $\geq R_i$ being assigned, or by many trees of size $< R_i$. This will ultimately cause us to lose the additional factor $\frac{1}{2}$.

First consider some tree T of size R_j, $j \geq i$. We are assuming that such a tree is entirely assigned to some $\hat{T}_o$ where it contributes 2^{j-i} to Λ_i. This also remains the case if the tree has been recombined and matched at some iteration later than j. Thus if we only consider such large trees we do in fact obtain $\Lambda_i \geq \sum_{j=i+1}^{\infty} |M_j| 2^{j-i}$. So we can assume that such trees do not exists and focus instead on the trees which are smaller than R_i but are not yet assigned in iteration i.

For these small trees the lemma promises some concentration property. In principle they are small enough to not contribute to Λ_i. However we will now show that there are too many such trees, or rather too few depots to accept them. Thus some depot must either contain many of them, or they must be assigned to depots further away. Either way they make the necessary contribution to Λ_i.

Let D_i be the set of depots o for which $\mathcal{F}_{i,o}$ was not empty at the end of iteration i. In particular these depots have been assigned some tree during the iteration. Now in H_i let $\hat{D}$ be the set of depots reachable from D_i using an alternating path with respect to M_i (i.e. alternatingly using an edge from M_i and not from M_i). By maximality of M_i all depots in $\hat{D}$ were assigned some tree during this iteration; further we know that for any tree T in $\mathcal{F}_{i,o}$ that is at distance at most 2^i to a depot of $\hat{D}$ we have $d(T, D \setminus \hat{D}) > R_i$. If not we can find an alternating path to some empty depot and augment the matching.

So consider $\mathcal{T}$ to be the set of all trees which are unassigned after iteration i and have size $\leq R_i$, as well as the set of trees assigned to some depot in $\hat{D}$ during iteration i. Now take $\{\hat{T}_o\}_{o \in X}$ to be the set of trees from $\hat{\mathcal{T}}$ containing any tree from $\mathcal{T}$. We can observe the following:

$$\sum_{o \in A} w(\hat{T}_o) \geq \sum_{T \in \mathcal{T}} w(T) + R_i \cdot |A \setminus \hat{D}|,$$

where the additional term $R_i \cdot |A \setminus \hat{D}|$ originates from the fact that any depot outside of $\hat{D}$ has distance at least R_i to the trees in $\mathcal{T}$. Now consider that $\sum_{T \in \mathcal{T}} w(T) = |\hat{D}| R_i + \sum_{j=i+1}^{\infty} |M_j| 2^{j-i}$ (again, we assumed that no trees of

size greater R_i are present in the original partition). We thus obtain

$$\sum_{o\in A}\lfloor w(\hat{T}_o)/2^i\rfloor \geq \big(|\hat{D}|R_i + \sum_{j=i+1}^{\infty}|M_j|2^{j-i} + R_i\cdot|A\setminus\hat{D}|\big)/R_i - |A|$$
$$= \sum_{j=i+1}^{\infty}|M_j|2^{j-i}$$

Notice that this argument induces some double counting on the trees of size $> R_i$ since they are both counted for there size but also potentially in the distances between small trees and far away depots. Thus we need to give up a factor 2 in the Λ_i to obtain the final statement of the lemma. □

This lemma now guarantees the existence of sufficiently many trees of some size R_i in $\hat{\mathcal{T}}$ to account for the trees of size R_i in the algorithms solution. The only issue here is the factor of $\frac{1}{2}$ in the lemma statement. However notice from definition that $\Lambda_{i-1} \geq 2\Lambda_i$, so we shall work instead with a reformulation of Lemma 4.6 as:

$$\Lambda_{i-1} \geq \sum_{j=i+1}^{\infty}|M_j|2^{j-i}.$$

Again we will show approximate strong optimality for the algorithms solution, so we obtain a constant factor approximation for all monotone symmetric norms.

Lemma 4.7. *For every $A \subseteq D$ we can find set $B \subseteq D$ with $|B| \leq |A|$ such that*

$$\sum_{o\in A} w(\mathcal{T}_o) \leq 4\sum_{o\in B} w(\hat{T}_o),$$

where $\{\mathcal{T}_o\}_{o\in A}$ is the set of trees computed by the assignment algorithm for the depots in A.

Proof. Fix some $A \subseteq D$ and for every $o \in A$ set r_o to be the iteration of the algorithm in which $\mathcal{T}_o$ was last assigned a tree. Hence $w(\mathcal{T}_o) \leq 2\cdot 2^{r_o}$.

Now we shall demonstrate how to find some comparatively sized trees among the $\hat{T}_o$. For every $\mathcal{T}_o$ we want to be able to match it to one of the fragments of size 2^{r_o-1} accounted for by Λ_{r_o-1}. However the fragments are not disjoint between the different Λ_i, i.e. a tree $\hat{T}_o$ of size 2^i will contribute 1 to Λ_i, 2 to Λ_{i-1}, etc. but we can not use that contribution repeatedly. To avoid overcounting we thus execute a little trick: Assume without loss of generality that $\Lambda_{i-1} = \sum_{j=i+1}^{\infty}|M_j|2^{j-i}$. To do this decrease the sizes of the $\hat{T}_o$ until the equalities hold. This only worsens the constant c we can

get for this lemma. Observe now that

$$\Lambda_{i-1} - 2 \cdot \Lambda_i = \sum_{j=i}^{\infty} |M_j| 2^{j-i} - \sum_{j=i+1}^{\infty} |M_j| 2^{j-i} = |M_i|.$$

This means we can partition our $\hat{T}_o$ as follows: For the final iteration i^* of the algorithm we can extract $\Lambda_{i^*-1} = |M_{i^*}|$ many fragments of size 2^{i^*-1}. For i^*-1 we can then partition each of these into two fragments of size 2^{i^*-2}. However we can extract an additional $\Lambda_{i^*-2} - 2 \cdot \Lambda_{i^*-1} = |M_{i^*-1}|$ fragments, disjoint from the ones we already have. We continue this argument downwards. In this way we partition the $\hat{T}_o$ into M_i disjoint fragments of size 2^{i-1}.

Now from construction the number of $o \in D$ for which $r_o = i$ can be at most $|M_i|$. Thus we assign each $\mathcal{T}_o$ with $o \in A$ uniquely to one of the fragments of size 2^{r_o-1} we are guaranteed. We then take B to be the set of trees $\hat{T}_o$ containing one of the chosen fragments. This clearly gives $|B| \leq |A|$, but it also guarantees:

$$\sum_{o \in B} w(\hat{T}_o) \geq \sum_{o \in A} 2^{r_o-1} \geq \frac{1}{4} \sum_{o \in A} w(\mathcal{T}_o).$$

□

The proof of Theorem 4.2 now is a direct consequence of Lemma 4.7, where the computation of the exact constant we can for this algorithm get is left open. The present analysis suggests some very large number, but to maintain some presentability it is also highly inefficient. Many of the reductions made could in principle be omitted at the weight of a much more detailed analysis.

4.4 Improved Analysis for the No-Depot Setting

Speaking of constants of approximation, we *will* give an improved analysis for the approximation ratio attained by the modified algorithm of Even et al. [51] with respect to our all-ℓ_p norm objective in the non-depot setting since we can in fact get a reasonably small constant for this setting. This involves centrally one observation, namely that the previous analysis pays a considerable portion of its constant of approximation to the fact that there can be up to $\frac{k}{2}$ large components, requiring it to give up a factor of 2 in Lemma 4.3. However notice that this is only the case if all large components have a spanning tree of size less than $4R$. This can be avoided by pretending that such components are in fact small. Notice that we only use that all trees produced by the algorithm have size at most $4R$, so this does not

degrade the analysis, but it will allow us to assume that there are at most $\frac{k}{3}$ large components. Plugging this in, as well as rearranging the arithmetic a bit will yield a much smaller factor of approximation.

Theorem 4.5. *There is an algorithm that, given any edge-weighted graph G and integer $k \in \mathbb{N}$, in polynomial time computes a set $\{T_1, \dots, T_t\}$ of at most k trees covering all vertices of G such that, for any $p \in [1, \infty)$ and tree cover $\hat{T}_i, \dots, \hat{T}_k$ we have*

$$(\sum w(T_i)^p)^{1/p} \leq 14(\sum w(\hat{T}_i)^p)^{1/p} \ .$$

Proof. Suppose that the algorithm decomposes the graph into s small components of size at most $4R$, and ℓ large components of size greater than $4R$, and fix some tree cover $\hat{T}_1, \dots, \hat{T}_k$.

Denote by s_1 the number of small components for which some $\hat{T}_i$ is a spanning tree, by s_2 the number of small components which intersect multiple $\hat{T}_i$, and by s_3 the number of other small components, which are in particular incident to some edge e of a $\hat{T}_i$ with $d(e) \geq R$.

We can make the following observations, analogously to the old analysis:

1. $\sum w(\hat{T}_i) \geq \frac{s_3}{2} R + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)$,
2. $\sum w(\hat{T}_i) \geq (k - s - \ell)2R - (k - s - \ell - s_2)R + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)$,
3. and $\ell \leq \frac{k-s}{3}$,

where in the final point we now get a factor 1/3 rather than 1/2. We will now also apply the third inequality immediately, allowing us to get:

$$\begin{aligned}
\sum w(\hat{T}_i) &\geq (k - s - \ell)2R - (k - s - \ell - s_2)R + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) \\
&= (k - s_1 - s_3 - \ell)R + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) \\
&\geq (\frac{2}{3}k + \frac{1}{3}s - (s_1 + s_3))R + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) \\
&\geq \frac{2}{3}(k - s_1 - s_3)R + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) \ .
\end{aligned}$$

To remove the s_3, we take a convex combination of this inequality with $\sum w(\hat{T}_i) \geq \frac{s_3}{2} R + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)$ and obtain:

$$\begin{aligned}
\sum w(\hat{T}_i) &\geq \frac{3}{7} \cdot \frac{2}{3}(k - s_1 - s_3)R + \frac{4}{7} \cdot \frac{s_3}{2} R + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) \\
&= \frac{2}{7}(k - s_1)R + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) \ .
\end{aligned}$$

This gives

$$\begin{aligned}\left[\sum w(T_i)^p\right]^{1/p} &\leq \left[(k-s_1)(4R)^p + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)^p\right]^{1/p} \\ &\leq 14\left[(k-s_1)\left(\frac{2}{7}R\right)^p + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)^p\right]^{1/p} \\ &\leq 14\left[\sum w(\hat{T}_i)^p\right]^{1/p} \quad . \qquad \square\end{aligned}$$

Notice again that the result is actually a lower bound on the average tree size in any tree cover, excepting those trees that already agree with the solution computed by the algorithm. So this analysis does indeed give c-approximate strong optimality.

We can obtain an even stronger bound by exploiting that the number s_3 of small components witnessing some edge e with $d(e) \geq R$ must be small:

Lemma 4.8. *Using the same notation as in the proof of Theorem 4.5, if $s_3 \geq \frac{1}{3}(k-s_1)$, the algorithm computes a solution which is an 8-approximation.*

Proof. Let $\hat{T}_i$ be some tree from an optimal solution to (ℓ_p, k) tree cover that contains fully covers a small component, as well as some edge e with $d(e) \geq R$. Then take $T_1, \ldots, T_j$ to be the spanning trees of those small components fully covered by $\hat{T}_i$. We can notice that $\hat{T}_i$ contains at least $j/2$ edges of weight greater than R. Hence, we obtain $w(\hat{T}_i)^p \geq \sum_j w(T_j)^p + (j/2)R^p$.

That is, in the algorithm all trees in the components counted by s_3 are being paid for by the optimal solution, and the optimal solution pays an additional R^p for half of them, which we can use to cover the weight of some of the algorithms large trees.

So let T_3 be the set of minimum spanning trees of the components counted by s_3. We obtain the following bounds:

$$\sum_i w(\hat{T}_i)^p \geq \sum_{T \in T^=} w(T)^p + \sum_{T \in T_3} w(T)^p + \frac{s_3}{2}R^p; \tag{4.1}$$

$$\sum_i w(T_i)^p \leq \sum_{T \in T^=} w(T)^p + \sum_{T \in T_3} w(T)^p + \frac{k-s_1-s_3}{2}(4R)^p \quad . \tag{4.2}$$

Here, Equation (4.2) follows from the fact that the trees of the algorithm have average weight $\leq 2R$ because of Step 3, and so the worst possible arrangement with respect to any ℓ_p objective is to have $k/2$ trees of size $4R$. To obtain Equation (4.2), we then additionally exclude the trees counted by s_1 and s_3 from that computation.

Together, Equations (4.1) and (4.2) yield

$$\frac{\sum_i w(T_i)^p}{\sum_i w(\hat{T}_i)^p} \leq \frac{k - s_1 - s_3}{s_3} 4^p \leq 2 \cdot 4^p \leq 8^p \ .$$

This proves the lemma.

Interestingly, note that for larger p we can get much better bounds here. Namely, for $s_3 \geq \lambda(k - s_1)$ we would get

$$\left(\frac{\sum_i w(T_i)^p}{\sum_i w(\hat{T}_i)^p}\right)^{1/p} \leq \left(\frac{k - s_1 - s_3}{s_3}\right)^{1/p} 4 \leq \left(\frac{1-\lambda}{\lambda}\right)^{1/p} \cdot 4 \ .$$

Here, we can choose λ quite small as p grows, for example for $p = 2$ we could already take $\lambda = \frac{1}{5}$, obtaining better constants for specific, large values of p. □

Theorem 4.6. *There is an algorithm that, given any edge-weighted graph G and integer $k \in \mathbb{N}$, in polynomial time computes a set $\{T_1, \ldots, T_t\}$ of at most k trees covering all vertices of G such that, for any $p \in [1, \infty)$ and tree cover $\hat{T}_i, \ldots, \hat{T}_k$ we have*

$$\left(\sum w(T_i)^p\right)^{1/p} \leq 9 \left(\sum w(\hat{T}_i)^p\right)^{1/p} \ .$$

Proof. We follow the proof of Theorem 4.5 to obtain

$$\sum w(\hat{T}_i) \geq \frac{2}{3}(k - s_1 - s_3)R + \sum_{T_i \in T^=} w(T_i) \ .$$

Then by Lemma 4.8 we can assume $s_3 \leq (k - s_1)/3$, otherwise we already have the statement of the theorem. So we obtain

$$\sum w(\hat{T}_i) \geq \frac{4}{9}(k - s_1)R + \sum_{T_i \in T^=} w(T_i),$$

and thus

$$\sum w(\hat{T}_i)^p \geq (k - s_1)(\frac{4}{9}R)^p + \sum_{T_i \in T^=} w(T_i)^p \ .$$

Together with

$$\sum w(T_i)^p \leq (k - s_1)(4R)^p + \sum_{T_i \in T^=} w(T_i)^p,$$

this implies the theorem. □

4.5 Computational Hardness

To complement the algorithmic results, we also show some lower bounds and give a short discussion on what kinds of algorithmic improvements are still possible under these hardness constraints. Specifically, we show:

1. For any $p \in (1, \infty]$, problem ℓ_p-Tree Cover with Depots is weakly NP-hard, even with only 2 trees.
2. For any $p \in (1, \infty]$, problem ℓ_p-Tree Cover with Depots with k trees is (strongly) W[1]-hard.
3. For any $p \in (1, \infty]$ there exists some $\varepsilon > 0$ such that ℓ_p-Tree Cover with Depots is NP-hard to approximate within a factor $< 1 + \varepsilon$ under randomized reductions.

Results 1 and 2 were already essentially presented by Even et al. [51], but we restate them here for completeness. Note that, given these results, numerous otherwise desirable algorithmic outcomes become unattainable. For instance, there cannot be polynomial-time approximation schemes for ℓ_p-Tree Cover with Depots, nor can we expect exact algorithms when parameterizing on the number of depots unless established hardness hypotheses (P≠NP, FPT≠W[1]) fail, Further, the reduction for Item 1 produces bipartite graphs of tree-depth 3, so parameterization by the structure of the input graph also appears to be out of reach.

4.5.1 Weak NP-hardness

We will show NP-hardness for ℓ_p-Tree Cover with Depots by reduction from Partition. In the Partition problem, we are given a collection of integers $x_1, \ldots, x_n$; the task is to compute a *solution* in form of a subcollection $X \subseteq \{x_1, \ldots, x_n\}$ such that $2\sum_{x \in X} x = \sum_{i=1}^{n} x_i$, or to decide that no solution exists. It is known that Partition is (weakly) NP-complete [4].

We transform an instance $\{x_1, \ldots, x_n\}$ of Partition to an equivalent instance of ℓ_p-Tree Cover with Depots by constructing a graph G where $V(G) = \{d_1, d_2\} \cup \{v_1, \ldots, v_n\}$ and $E(G) = \{d_1, d_2\} \times \{v_1, \ldots, v_n\}$, i.e. G is the complete bipartite graph $K_{2,n}$. We then set $D = \{d_1, d_2\}$ and $w(d_i, v_j) = x_j$.

One can briefly observe that, for any given subset of the vertices X, there is a minimum Steiner tree for X that consists of a star centered on one of the depots, plus possibly one additional edge to connect the other depot in case $\{d_1, d_2\} \subseteq X$. In particular this means that in an optimal ℓ_p-Tree Cover with Depots every tree will contain exactly one depot, so the optimal solutions correspond to partitions $X_1 \dot{\cup} X_2$ of $\{v_1, \ldots, v_n\}$ and

have solution value $[(\sum_{v_i \in X_1} x_i)^p + (\sum_{v_i \in X_2} x_i)^p]^{1/p}$. Since the ℓ_p norms are strongly convex for $p > 1$, the instance (G, D, w, p) then has optimal solution value $[2(\frac{1}{2}\sum_{i=1}^{n} x_i)^p]^{1/p}$ if and only if the original instance of Partition admits a solution.

4.5.2 W[1]-hardness for ℓ_p-Tree Cover with Depots parameterized on $|D|$

The previous construction can be expanded by allowing more than 2 depots, say $k > 2$. The resulting instance will again have optimal solution value $[k(\frac{1}{k}\sum_{i=1}^{n} x_i)^p]^{1/p}$ if and only if there exists a partition of X into k bins $X_1, \ldots, X_k$ such that the items in each bucket sum up to the same weight. Using this construction we can then solve Unary Bin Packing with k bins which was shown by Jansen et al. to be W[1]-hard when parameterized by k [67]. Note here that in the case of unary bin packing we may assume without loss of generality that all bins must be completely filled by providing additional items of unit size.

4.5.3 APX-hardness for ℓ_p-Tree Cover with Depots

Although the earlier constructions are relatively straightforward, determining a lower bound for the approximability of ℓ_p-Tree Cover with Depots requires more work. We proceed by reducing from a variant of maximum satisfiablitity of linear equations over $\mathbb{F}_2$ of arity 3, MAX-E3LIN2 for short. In this problem, we are given a set of variables $x_1, \ldots, x_n$, as well as a set of m ternary equations $x \oplus y \oplus z = b$ with $b \in \{0, 1\}$. Here, $\oplus$ signifies addition modulo 2. The goal is to compute an assignment of $\{0, 1\}$ values to the variables which satisfies a maximum number of equations.

In addressing this problem, Håstad [68], in his seminal work, demonstrated that for any $\varepsilon > 0$, it is NP-hard to distinguish those instances where $(1-\varepsilon)m$ equations can be satisfied from those where at most $(\frac{1}{2}+\varepsilon)m$ can be satisfied. For our purposes we need to also bound the variable degree. A modification of MAX-E3LIN2 achieving bounded variable degree was used by Karpinski et al. [20] to show improved lower bounds for the approximability of the TSP. They proved that MAX-E3LIN2 remains hard to approximate even if every variable occurs in exactly three equations. In return they need to allow some equations of arity 2, as well as a degradation in the approximability lower bound. Specifically they prove the following theorem.

Theorem 4.7 ([20], Theorem 3)**.** *For every $\varepsilon > 0$ there exists a collection of systems of linear equations with $31m$ equations and $21m$ variables over $\mathbb{F}_2$ such that:*

- *each variable occurs in exactly three equations;*
- *$30m$ of the equations contain 2 variables;*
- *m of the equations are of the form $x \oplus y \oplus z = 0$;*
- *and it is NP-hard to decide whether some assignment to the variables satisfies $(31 - \varepsilon)m$ equations, or if every assignment satisfies at most $(30.5 + \varepsilon)m$ equations.*

This regularisation is achieved by taking an instance of MAX-E3LIN2 and replacing every occurrence of a variable with a new, dedicated variable, and then linking these new variables together by additional equations to force them to be equal if they correspond to the same original variable. This increases the size of the system, hence there are $31m$ instead of m equations. We will call this problem 3R{2,3}L2, for 3-regular linear equations of arity 2 or 3 over the binary domain. Its instances are denoted $(X, \mathcal{C})$ where X is the set of variables and $\mathcal{C}$ the set of clauses.

Note: The hardness of approximation for 3R{2,3}L2 presented by Karpinski et al. [20] relies on reducing the variable degree of an instance of MAX-E3LIN2 by means of a certain type of amplifier graph. They show how to construct such amplifiers probabilistically in polynomial time, but a deterministic construction is not known. Thus, the reduction only works probabilistically. If one desires deterministic guarantees, we may settle for a weaker statement by noting that for an instance with m clauses their transformation requires at most one such graph for each $i \in 1, \dots m$, and the graph for each i can be chosen independently of the instance and has size linear in i. Thus their results also hold in a deterministic setting if we allow for polynomial runtime *and* a polynomial advice string, i.e. assuming $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ rather than $\mathsf{P} \neq \mathsf{NP}$.

For an instance of 3R{2,3}L2 we may construct an instance of ℓ_p-Tree Cover with Depots by first introducing some gadgets for the variables and clauses:

- For every variable x, introduce three vertices $\hat{x}, x_0$, and x_1, as well as edges $\hat{x}, x_i$ for $i = 0, 1$ with weight 3. Add the x_i's as depots.
- For every ternary clause C, introduce vertices $\hat{C}, C_{000}, C_{110}, C_{101}$, and C_{011} with edges $\{\hat{C}, C_i\}$ of weight 3. Add the C_i's as depots.

- For every binary clause $C = x \oplus y = 0$, introduce vertices $\hat{C}, C_{00}$, and C_{11} with edges $\{\hat{C}, C_i\}$ of weight 2. Add the C_i's as depots. Further add vertices $\hat{C_{00}}$ and $\hat{C_{11}}$ where $\hat{C}_i$ is connected only to C_i by an edge of weight 1.

- For every binary clause $C = x \oplus y = 1$, introduce vertices $\hat{C}, C_{01}$, and C_{10} with edges $\{\hat{C}, C_i\}$ of weight 2. Add the C_i's as depots. Further add vertices $\hat{C_{01}}$ and $\hat{C_{10}}$ where $\hat{C}_i$ is connected only to C_i by an edge of weight 1.

We connect the gadgets together as follows:

- For every clause $C = x \oplus y \oplus z = 0$ we connect $C_{b_1 b_2 b_3}$ to x_{b_1}, y_{b_2}, and z_{b_3} with a path of length 2 where both edges have weight 1.

- For every clause $C = x \oplus y = b$ we connect $C_{b_1 b_2}$ to x_{b_1}, y_{b_2} with a path of length 2 where both edges have weight 1.

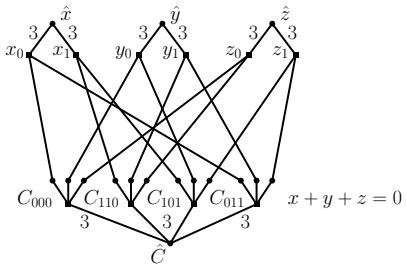
An illustration of this construction is given in Figure 4.6. Notice that there are $106m$ depots.

The goal of this construction is to represent the original system of linear equations in the following sense: For every variable x, the vertex $\hat{x}$ should be assigned to either x_0 or x_1. As this is relatively expensive, we would like to not assign any further vertices to that depot. The other depot however is still free, and should correspond to the variable assignment we want to choose.

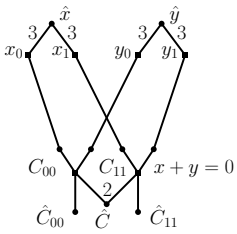
Similarly for every clause $C = x \oplus y \oplus z = 0$ we need to assign $\hat{C}$ to one of the depots in the clause gadget. These correspond directly to the way in which we would like the clause to be satisfied. That is, assigning $\hat{C}$ to C_{011} means that we have set $x = 0$, $y = 1$, and $z = 1$ to satisfy the clause.

Now we may note that, if those choices are consistent between a clause and its incident variables, we will be able to also assign the vertices on the paths that we introduced between the gadgets. Specifically every such vertex will be adjacent to a depot that has not yet been assigned any vertex. We try to assign all of them to the neighbouring C_i if it is empty. The remainder we then assign to the adjacent x_i. Since every variable appears in most 3 clauses, this will leave every tree with size at most 3, if we had a satisfying assignment to begin with. In fact every tree will have size exactly 3 in this setting, since the variable degrees are exactly 3.

Conversely, every clause that is not satisfied will cause one tree somewhere to have weight at least 3. For every ℓ_p norm other than ℓ_1 this imbalance in the tree sizes will then be measureable in the value of an optimum solution.



(a) The gadget for a ternary clause $x \oplus y \oplus z = 0$, along with the gadgets for x, y, z and the respective connections.



(b) The gadget for a binary clause $x \oplus y = 0$, along with the gadgets for x, y and the respective connections.

Figure 4.6: Illustration of clause and variable gadgets for 3 and 2-ary constraints

Completeness

We begin by describing how we can transform solutions to the $(31-\varepsilon)m$-satisfiable instances of 3R{2,3}L2 to good solutions to the constructed instances of ℓ_p-Tree Cover.

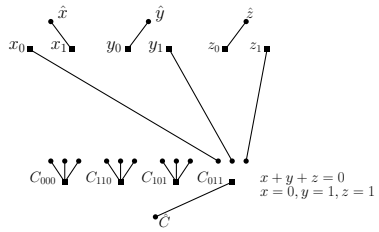
Lemma 4.9. *Let $(X, \mathcal{C})$ be an instance of* 3R{2,3}L2 *in which $(31-\varepsilon)m$ clauses can be simultaneously satisfied. Let (G, D, w) be the instance of ℓ_p-*Tree Cover *obtained from $(X, \mathcal{C})$. Then*

$$OPT_p \leq [(106 - 2\varepsilon)m3^p + \varepsilon(2^p + 4^p)]^{1/p} \ .$$

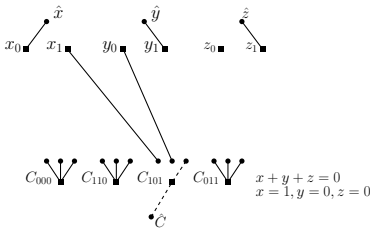
Proof. Let (G, D, w) be the instance obtained from transforming $(X, \mathcal{C})$ and take $f : X \to \{0, 1\}$ to be an assignment to the variables satisfying $(31-\varepsilon)m|\mathcal{C}|$ clauses.

To construct a tree cover of G we do the following:

1. For every $x \in X$, assign $\hat{x}$ to $x_{1-f(x)}$.
2. For every satisfied clause $C = x \oplus y \oplus z = 0$, assign $\hat{C}$ to $C_{f(x)f(y)f(z)}$.
3. For every satisfied clause $C = x \oplus y = b$ with $b \in \{0, 1\}$, assign $\hat{C}$ to $C_{f(x)f(y)}$, as well as $\hat{C}_{b_x b_y}$ to $C_{b_x b_y}$ for any $b_x, b_y \in \{0, 1\}$.
4. For every unsatisfied clause $C = x \oplus y \oplus z = 0$, assign $\hat{C}$ to $C_{f(x)f(y)1-f(z)}$.
5. For every unsatisfied clause $C = x \oplus y = b$ with $b \in \{0, 1\}$, assign $\hat{C}$ to $C_{f(x)1-f(y)}$, as well as $\hat{C}_{b_x b_y}$ to $C_{b_x b_y}$ for any $b_x, b_y \in \{0, 1\}$.



(a) The gadget for a satisfied ternary clause under variable assignment $x = 0$, $y = 1$, $z = 1$ with the selected solution. All trees have size 3, or belong to the depot corresponding to a variables assigned value.



(b) The gadget for an unsatisfied ternary clause under variable assignment $x = 1$, $y = 0$, $z = 0$ with the selected solution. The tree with the dashed edges will have weight 4, witnessing the fact that the clause was unsatisfied.

Figure 4.7: Illustrations of the appearance of Tree Cover solutions on satisfied and unsatisfied clauses.

6. Finally, assign every of yet unassigned vertex by the following rules:
   (a) If it has a neighbouring depot C_I for some $C \in \mathcal{C}, I \in \{0,1\}^*$ such that C_I has not been assigned $\hat{C}$, assign the vertex to this depot.
   (b) Otherwise if it has a neighbour $x_f(x)$ for some $x \in X$, assign it to this depot.
   (c) Finally if it is not assigned by the previous rules, it is incident to either $C_{f(x)f(y)1-f(z)}$ or $C_{f(x)1-f(y)}$ for some clause C. Assign it to this depot.

We can observe that because every variable appears in at most three clauses the application of this rule will cause every depot to be assigned a total weight of at most 3, except those corresponding to the C_I that was assigned $\hat{C}$ for some unsatisfied clause C; These depots are assigned a total weigth of 4, consisting of $\hat{C}$, possibly $\hat{C}_I$, and one additional vertex. However there are only εm such depots.

Noting that the total weight of all trees is $3 \cdot 106 \cdot m$, we obtain that the value of this solution with respect to ℓ_p is

$$[(106 - 2\varepsilon)m3^p + \varepsilon(2^p + 4^p)]^{1/p} \quad . \qquad \square$$

Soundness

We will for clarity conduct the soundness analysis in two stages. First, we will look at solutions to the ℓ_p-Tree Cover which are *honest* in the sense

that they assign every vertex to a neighbouring depot. These rather closely correspond to solutions of the original instance of 3R{2,3}L2 so we will be able to essentially show that if the instance was at most $(30.5+\varepsilon)m$-satisfiable, there are at least $(\frac{1}{2}-\varepsilon)m$ trees of size at least 4 in any honest solution.

In a second step we then show that one can not gain much by using a *dishonest* solution instead. Assigning a vertex to a depot that is not neighbouring incurs some weight with respect to ℓ_1, which will drive up the weights in all other ℓ_p norms as well.

Lemma 4.10. *Any honest solution $\{T_d\}_{d\in D}$ to some instance (G, D, w) of ℓ_p-Tree Cover derived from a $(31-c)m$-satisfiable instance $(X, \mathcal{C})$ of* 3R{2,3}L2 *will have*

$$\left[\sum_{d\in D} w(T_d)^p\right]^{1/p} \geq \left[(106-2c)m3^p + cm(2^p+4^p)\right]^{1/p} .$$

Proof. Suppose we are given an honest solution to the ℓ_p-Tree Cover instance constructed from an instance of 3R{2,3}L2. We obtain an assignment f to the variables by setting $f(x)=1$ if $\hat{x}$ is assigned to x_0, and vice versa. Now take any clause C not satisfied by f and let C_I be the depot which was assigned $\hat{C}$. Then there exists at least one vertex in $N(C_I)\setminus\{\hat{C}\}$ such that it is adjacent to two depots, both of which are already have weight at least 3 assigned to them. We call the set of all such vertices V_{excess}.

These vertices must be assigned to one of their neighbouring depots causing it to have size ≥ 4. We formally detect this by estimating the following:

$$\sum_{d\in D} \max\{w(T_d)-3, 0\} \geq |V_{excess}| \geq cm.$$

Noting again that we have $\sum_{d\in|D|} w(T_d) \geq 318m$, and that a lower bound on the ℓ_p value can be obtained by distributing the total weight as evenly as possible, under the constraint that $\sum_{d\in D}\max\{w(T_d)-3,0\} \geq cm$, we then see that

$$\left[\sum_{d\in D} w(T_d)^p\right]^{1/p} \geq \left[(106-2c)m3^p + cm(2^p+4^p)\right]^{1/p} . \qquad \square$$

Lemma 4.11. *Any solution $\{T_d\}_{d\in D}$ to an instance (G, D, w) of ℓ_p-Tree Cover derived from a $(31-c)m$-satisfiable instance $(X, \mathcal{C})$ of* 3R{2,3}L2 *will have*

$$\left[\sum_{d\in D} w(T_d)^p\right]^{1/p} \geq \left[(106-\frac{c}{2})3^p + \frac{c}{4}(2^p+4^p)\right]^{1/p}.$$

Proof. We now also have to consider the dishonest solutions, i.e., solutions where some vertices were not assigned to a neighbouring depot. For each such vertex there is some increase in the ℓ_1 value of the solution, so there can not be too many of them. But then most of the instance looks like it is honest, since for every variable on which the solution cheats it can satisfy at most 3 additional clauses above what an optimal solution to 3R{2,3}L2 could.

So let $V_{\text{dishonest}} \subseteq V(G)$ be the set of vertices not assigned to a neighbouring depot. Then we see that $\sum_{d\in D} w(T_d) \geq 318m + |V_{\text{dishonest}}|$ and it follows that $\sum_{d\in D} \max\{w(T_d) - 3, 0\} \geq |V_{\text{dishonest}}|$.

Now suppose we take the set of variables $X_{\text{dishonest}} \subseteq X$ whose corresponding gadget contains a vertex from $V_{\text{dishonest}}$, and similarly $\mathcal{C}_{\text{dishonest}}$ for the clauses. We now delete from $(X, \mathcal{C})$ all variables in $X_{\text{dishonest}}$, all clauses in $\mathcal{C}_{\text{dishonest}}$, and finally all clauses containing a variable from $X_{\text{dishonest}}$. Call the resulting instance of 3R{2,3}L2 $(X', \mathcal{C}')$. We notice that $\mathcal{C}'$ still contains at least $31m - 3|V_{\text{dishonest}}|$ clauses. Also, any assignment to X' satisfying a clause $C \in \mathcal{C}'$ will also satisfy that clause in $(X, \mathcal{C})$, irrespective of how we extend it to $X \setminus X'$. Thus any assignment to X' will leave at least $cm - 3|V_{\text{dishonest}}|$ clauses unsatisfied. Thus we have an honest assignment leaving some clauses unsatisfied, so Lemma 4.10 applies. We see that we have $\sum_{d\in D} \max\{w(T_d) - 3, 0\} \geq cm - 3|V_{\text{dishonest}}|$, and thus

$$\sum_{d\in D} \max\{w(T_d) - 3, 0\} \geq \max\{cm - 3|V_{\text{dishonest}}|, |V_{\text{dishonest}}|\} \geq \frac{c}{4}m \ .$$

This gives the desired relation

$$\left[\sum_{d\in D} w(T_d)^p\right]^{1/p} \geq \left[(106 - \frac{c}{2})m3^p + \frac{c}{4}m(2^p + 4^p)\right]^{1/p} \ . \qquad \square$$

Hardness of Approximation

Lemmas 4.9 and 4.11 jointly show that our reduction can, up to a factor of 4, detect the number of clauses that are satisfiable in an instance of 3R{2,3}L2. Therefore, the gap shown by Karpinski et al. [20] now transfers also to our setting:

Theorem 4.8. *For every $p \in (1, \infty)$ there exists a constant c such that ℓ_p-*Tree Cover with Depots *is* NP*-hard to approximate within a factor c. The* NP*-hardness holds under randomized reductions.*

Proof. Fix some p and ε and take an instance of 3R{2,3}L2 from an instance family for which it is NP-hard to distinguish whether the instance is $(31 - \varepsilon)m$-satisfiable or at most $(30.5 + \varepsilon)m$. Karpinski et al. [20] show

that such a family exists under randomized reductions. Now suppose, for sake of contradiction, we had a polynomial-time c-approximation algorithm for ℓ_p-Tree Cover. Apply this algorithm to an instance generated by our reduction and denote by ALG the value of the computed solution. If the initial instance of 3R{2,3}L2 was $(31-\varepsilon)m$-satisfiable, then we obtain

$$ALG \leq c \cdot OPT_p \leq c[(106-2\varepsilon)m3^p + \varepsilon m(2^p+4^p)]^{1/p}$$

from Lemma 4.9. Otherwise, we have

$$ALG \geq OPT_p \geq [(106-\frac{1}{4}+\frac{\varepsilon}{2})m3^p + (\frac{1}{8}-\frac{\varepsilon}{4})m(2^p+4^p)]^{1/p} \ .$$

So unless P=NP, we must have

$$c[(106-2\varepsilon)m3^p+\varepsilon m(2^p+4^p)]^{1/p} \geq [(106-\frac{1}{4}+\frac{\varepsilon}{2})m3^p+(\frac{1}{8}-\frac{\varepsilon}{4})m(2^p+4^p)]^{1/p} \ .$$

Brief computation yields

$$\begin{aligned} c &\geq \frac{[(106-\frac{1}{4}+\frac{\varepsilon}{2})m3^p + (\frac{1}{8}-\frac{\varepsilon}{4})m(2^p+4^p)]^{1/p}}{[(106-2\varepsilon)m3^p+\varepsilon m(2^p+4^p)]^{1/p}} \\ &\geq \left[\frac{(106-\frac{1}{4}+\frac{\varepsilon}{2})3^p + (\frac{1}{8}-\frac{\varepsilon}{4})(2^p+4^p)}{(106-2\varepsilon)3^p+\varepsilon(2^p+4^p)}\right]^{1/p} > 1, \end{aligned}$$

where the final inequality follows immediately from the relation $2^p+4^p > 2\cdot 3^p$, which holds for all $p \in (1,\infty)$. □

Notice that this computation does not hold at $p=1$, since $2^1+4^1 = 2\cdot 3^1$. Indeed ℓ_1-Tree Cover is polynomial-time solvable. Further, the computation requires p to be a real number, so the above reduction does not yield hardness for ℓ_∞-Tree Cover. This should be unsurprising, since by Lemma 4.9 there is always a solution whose value with respect to ℓ_∞ is 4, unless the original instance of 3R{2,3}L2 was satisfiable (recall that satisfibility of a system of linear equations over $\mathbb{F}_2$ can of course be checked in polynomial time.).

APX-Hardness for ℓ_∞-Tree Cover with Depots

Obtaining some hardness of approximation for ℓ_∞-Tree Cover with Depots is nonetheless possible and not too complicated. This is already known due to Xu and Wen [61], however we restate the result here to demonstrate a reduction that is similar to the one used to prove Theorem 4.8. We reduce from 3-Occurrence SAT, i.e. SAT where every variable occurs in exactly 3 clauses, which is known to be NP-hard for example due to Berman et al. [69]. We denote instances of 3-Occurrence SAT as $(X,\mathcal{C})$ where X is the set of variables and $\mathcal{C}$ the set of clauses. For any such instance we construct an equivalent instance of ℓ_∞-Tree Cover with Depots in similar fashion to before:

1. For each $x \in X$, introduce vertices $x_0, x_1, \hat{x}$ with edges $\{x_i, \hat{x}\}$ of weight 2. Take the x_i to be depots.

2. For each clause $C = x^1 \vee x^2 \vee \ldots x^s \vee \overline{y^1} \vee \overline{y^2} \cdots \vee \overline{y^t}$ introduce a vertex C and edges $\{C, x^i_1\}$, $\{C, y^i_0\}$ with weight 1.

Any solution to this instance with value 2 with respect to ℓ_∞ will correspond exactly to a satisfying assignment f of the instance $(X, \mathcal{C})$ by taking $f(x) = i$ where i is such that $\hat{x}$ was assigned to x_{1-i}. Note that we do not have to worry here about the solution "cheating", since every $\hat{x}$ must be assigned to one of the neighboring X_i's, otherwise the solution value is at least 3. Every clause is then satisfied by the variable to whose depot it has been assigned.

Conversely, any satisfying assignment f can be transformed into a solution of the ℓ_∞-Tree Cover instance with value 2. We assign $\hat{x}$ to $x_{1-f(x)}$ for each variable $x \in X$, and C to $x_{f(i)}$, where x is a variable satisfying the clause C with respect to f. This achieves value 2, since—without loss of generality—every variable satisfies at most two clauses (otherwise it occurs only negated/non-negated and can be removed).

Thus, we cannot distinguish instances of ℓ_∞-Tree Cover with Depots with solution value 2 from those of solution value 3 in polynomial time unless P=NP, so ℓ_∞-Tree Cover with Depots is NP-hard to approximate to within a factor $\frac{3}{2} - \varepsilon$ for every $\varepsilon > 0$.

4.6 Conclusion & Open Problems

The present analysis raises some interesting directions for further research; In particular it would be interesting to know if the given constants can be improved. Especially the analysis for the case with depots appears to have a considerable amount of slack. Finding a simpler analysis of the algorithm might go a long way to obtain a better constant of approximation.

On the other side it would also be interesting to find good (unconditional) lower bounds, since our regime allows for these (see for example fig. 4.1).

Beyond a refined understanding of our concrete setting it would also be good to extend these kinds of multi-objective optimization problems to more domains. The presence of unconditional lower bounds in our setting will prevent this for many kinds of problems, however some weakenings of our all-norm objective might allow us to step around these lower bounds. For example in unrelated machine scheduling it is not possible to find an assignment that simultaneously minimizes all symmetric monotone norms of the machine load vector up to some universal constant. However this does not exclude some more relaxed solutions, for example on might be able to give a small portfolio of solutions such that for each norm at least

one solution from the portfolio is approximately minimal. Even weaker, we could try to find a distribution over the solutions such that every norm is approximately minimal with probability bounded away from 0.

Such relaxed notions of being optimal with respect to many different objectives could help to provide a theoretical framework for avoiding "overfitting" in optimization problems, where the strictly optimizing for one single objective may produce unnatural solutions which perform poorly on other metrics.

Ideas of this sort have seen enthusiastic adoption in Machine Learning in the form of *regularization* where the goal is explicitly to produce a solution which is perhaps not optimal, but approximately optimal with respect to a variety of objectives. In particular in Machine Learning it is understood that models with very large weights have probably been overfit, and so one typically minimizes both the loss of the model *and* some norm of the model weights. Doing this gives much better results in practice, as the resulting models are less likely to be over-specialized to their training data.

By the same token then it will be interesting to understand whether such a phenomenon can also be found in classical optimization problems by computing solutions which are good in many different ways rather than optimal in only one.

Chapter 5

Approximating Sparsest Cut in Bounded Treewidth Graphs

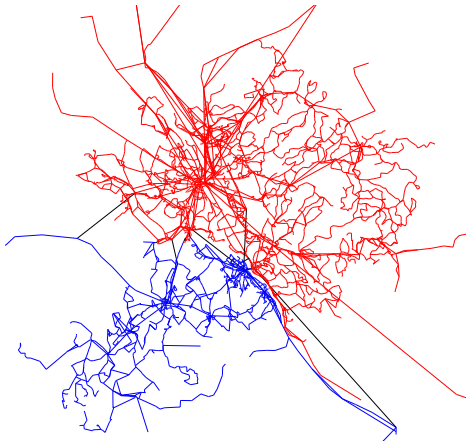
Based on joint work with Parinya Chalermsook, Matthias Mnich, Joachim Spoerhase, Sumedha Uniyal, and Daniel Vaz [70].

We now turn away from algorithms for the designing of networks and instead look at their natural dual formulations, cut problems. The best known dual pair of network design/cut problem here is probably the min cut/max flow relationship discovered by Ford and Fulkerson [71] for the single-commodity flow problem, i.e. the question of routing as many edge-disjoint paths as possible between two fixed endpoints in a capacitated graph.

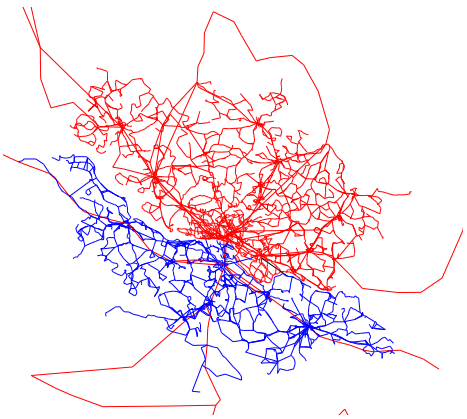
Notwithstanding its importance this particular problem is in many instances a rather poor model of the real-world demands in network analysis. Suppose you are trying to analyse the public transit map of some city –say Hamburg– to identify weak points in the network that should be strengthened. You might be tempted to compute a minimum cut of the network, i.e. two sets of stations between there are only very few connections. This will not, however give you a good understanding of where the true weak points are. It will instead return the information that there is only one single connection to Stade, a small town about 40km outside of Hamburg proper. And while the people of Stade would certainly appreciate an additional connection, relatively few people would profit from it in total, and the overall structure of the network would remain essentially unchanged.

The issue of course is that such multi-agent environments are not well-modelled by single-commodity flows. Instead we need to take into account that there are different levels of demand for transit between different endpoints of the network, and that a weak point then does not arise merely from a low capacity between two regions of the network, but from the combination of a low capacity *and* a high demand.

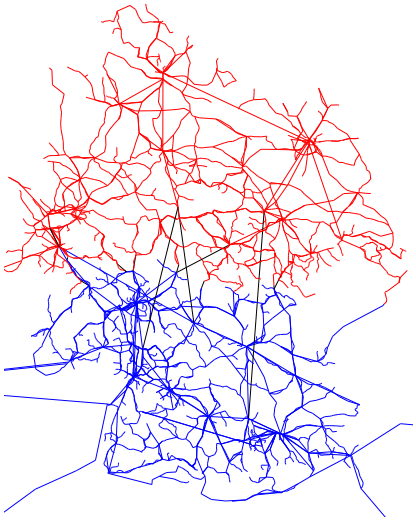
In order to take these effects into account we therefore consider the Sparsest Cut problem, where are given a graph together with capacities and demands on the edges, and our goal is to find a cut that minimizes the ratio between the capacities and demands across the cut. For some



(a) The map of public transit connections in the Cologne/Bonn area. The cut runs along the Rhine river and then swerves west to separate Bonn from Cologne. Personal experience suggests that getting from Bonn to Cologne is indeed difficult at times. The indicated graph has 5696 vertices, but a treewidth of at most 19.



(b) The map of public transit connections in Hamburg. The cut runs precisely along the Elbe river, with the exception of some (grade-separated) rail lines extending across it. The indicated graph has 7077 vertices, but a treewidth of at most 18.



(c) A sparsest cut in the regional rail network of Germany, highlighting a very clean North/South divide. The indicated graph has 5515 vertices, but a treewidth of at most 15.

Figure 5.1: Illustration of uniform sparsest cuts in various real-world transport networks. The edges are colored in red/blue if they are fully contained on one side of the cut, and black if they cross the cut. Also indicated is some heuristic upper bound on the treewidth of these networks.

empirical evidence that this is a credible see Figure 5.1. Formally, we define it as follows:

Problem Definition In the Sparsest Cut problem the input is a graph $G = (V, E_G)$ with positive edge capacities $\{\text{cap}_e\}_{e \in E_G}$ and a demand graph $D = (V, E_D)$ (on the same set of vertices) with positive demand values $\{\text{dem}_e\}_{e \in E_D}$. The aim is to compute the values

$$\Phi_{G,D} := \min_{S \subseteq V} \Phi_{G,D}(S), \qquad \Phi_{G,D}(S) := \frac{\sum_{e \in E_G(S, V-S)} \text{cap}_e}{\sum_{e \in E_D(S, V-S)} \text{dem}_e}.$$

The value $\Phi_{G,D}(S)$ is called the *sparsity* of the cut S. To avoid dividing by zero here we shall only consider cuts S for which $\sum_{e \in E_D(S, V-S)} \text{dem}_e$ is non-zero and treat all other cuts as having essentially infinite sparsity, i.e. they will never be the cut of minimum sparsity.

Beyond the the ability of Sparsest Cut to model many real world problems in network analysis [72] it is also of great theoretical interest in both computer science and pure math, in particular in the theory of metric embeddings. Specifically it can be shown that the integrality gap of the standard linear/semidefinite programming relaxation of the Sparsest Cut problems corresponds exactly to the minimum distortion with which arbitrary finite metrics/negative-type[1] metrics can be embedded into ℓ_1.

As the Sparsest Cut problem is NP-hard [73], the focus has largely been to study approximation algorithms, specifically through the lens of creating rounding algorithms for LP/SDP relaxations so as to exploit the close relationship of their integrality gaps to the theory of metric embeddings. Over the past four decades several great advances have been made in this direction. With respect to the integrality gap of the LP Linial, London, and Rabinovich as well as Aumann and Rabani [74, 75, 76] were able to show a gap of $\mathcal{O}(\log n)$ using Bourgain's Embedding Theorem [77]. Exploiting the smaller gap between negative-type metrics and ℓ_1 a line of work due to Arora et al. [78, 79] has yielded a $\mathcal{O}(\sqrt{\log n})$ approximation based on semi-definite programming.

On the lower-bound side it is known that some dependence on n in the approximation ratios is unavoidable unless Khot's Unique Games Conjecture [80] fails [81, 82, 83].

In light of the relevance of sparsest cuts significant effort has been invested into understanding when Sparsest Cut instances are "easy". Of particular interest here are structural bounds on the capacity graph of the underlying instance, whereas the demand graph is generally unrestricted.

[1] A metric space (X, d) has negative type if $(X, \sqrt{d})$ embeds isometrically into ℓ_2. Alternatively one can check that the matrix $(d(i,j))_{i,j \in X}$ is positive semi-definite, hence the relationship to SDP.

In trees, optimal sparsest cuts can be found in polynomial time (see e.g., Gupta et al. [84]). For many other well-known graph classes however finding optimal sparsest cuts is already NP-hard, and thus there are instead attempts to find constant-factor approximation algorithms in polynomial time. This has been successful for example for planar, ℓ-outerplanar, bounded-pathwidth and bounded-treewidth graphs [85, 86, 2, 87, 88].

A more general viewpoint of such a treatment of special cases, the conjecture of Gupta, Newman, Rabinovich, and Sinclair [89], postulates that any minor-free graph metric embeds into ℓ_1 with constant distortion, which would imply that all such graphs admit a constant approximation for the Sparsest Cut problem essentially by rounding a solution to the LP relaxation. The conjecture has been verified in various graph classes but remains open even for bounded-treewidth graph families. Thus exploring further developments in the direction of Sparsest Cut approximability for bounded treewidth graphs [2, 85] is of particular interest.

Beyond this there is also a deep connection between treewidth and the power of hierarchies of increasingly tight convex relaxations of combinatorial optimization problems (see, for instance, the work by Laurent [90]), which we will touch on in more depth in Section 5.4.

In the bounded-treewidth setting, a (problem-independent!) LP rounding algorithm performs surprisingly well for many combinatorial optimization problems: not only does it achieve optimal solutions for various fundamental problems [91, 92], but it has also led to tight approximation factors for problems such as Group Steiner Tree [93, 94, 95]. In this way, for these aforementioned problems, such a problem-oblivious LP rounding algorithm provides a natural framework to generalize an optimal algorithm on trees to nearly-optimal ones on low (perhaps super-constant) treewidth graphs.

In 2010, Chlamtáč, Krauthgamer, and Raghavendra [2] initiated the study of sparsest cuts in this low-treewidth regime, where the treewidth of the capacitated subgraph G is restricted to some integer k (no restrictions are placed on the demand graph D). Chlamtáč et al. devised a factor-2^{2^k} approximation algorithm (CKR) that runs in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$, with k being the treewidth of the capacitated subgraph G . Later, Gupta, Talwar and Witmer [85] showed how to obtain a factor-2 approximation (GTW) with a blown-up run time of $n^{\mathcal{O}(k)}$; they further showed that there is no $(2-\varepsilon)$-approximation for any $\varepsilon > 0$ on constant-treewidth graphs, assuming the Unique Games Conjecture. Cohen-Addad, Mömke, and Verdugo [96] recently matched this lower bound with a factor-2 approximation algorithm (CMV) with run time $2^{2^{\mathcal{O}(k)}} \cdot n^{\mathcal{O}(1)}$, which removes the dependency on k in the exponent of n, but suffers from a doubly-exponential dependence on k.

Perhaps it is worth noting here that the restriction to bounded treewidth for the capacity graph is not only of theoretical interest due to its connection

to metric embeddings of bounded-width metrics, but it also makes some sense in application. In practical terms a logistics network will usually need to be able to route demand between any of its nodes, leading to an unrestricted demand graph; However it is expensive to install such networks, and especially to include redundant connections. This should lead to real world logistics networks being rather sparse. Indeed the treewidth estimates given in Figure 5.1 seem to support this basic intuition.

It remains an intriguing open question whether one can simultaneously achieve the best run time and approximation factor. In particular we will here address the following question:

> Does Sparsest Cut admit a factor-2 approximation algorithm with run time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$?

Our Results

We present several results that may be seen as an intermediate step towards the optimal result. Our main technical results are summarized in the following theorem.

Theorem 5.1. *For the following choices of functions t and α, there are algorithms that run in time $t(k) \cdot n^{\mathcal{O}(1)}$ and achieve approximation factors $\alpha(k)$ for the* Sparsest Cut *problem on graphs of treewidth k:*

- $t(k) = 2^{\mathcal{O}(k)}$ *and* $\alpha(k) = \mathcal{O}(k^2)$.
- $t(k) = 2^{\mathcal{O}(k^2)}$ *and* $\alpha(k) = \mathcal{O}(1)$.
- *For any* $\varepsilon \in (0, 1]$, $t(k) = \exp(O\big(\frac{k^{1+\varepsilon}}{\varepsilon}\big))$ *and* $\alpha(k) = \mathcal{O}(1/\varepsilon^2)$.

For the proof, we refer to Sections 5.3.1 to 5.3.3 for each of the respective constructions.

Our first result directly improves the approximation factor of 2^{2^k} by Chlamtáč et al., while keeping the run time single-exponential in k. Our second result shows that, with only slightly larger run time, one can achieve a constant approximation factor. Compared to Gupta et al. our result has a constant blowup in the approximation factor (independent of k), but has a much better run time ($2^{\mathcal{O}(k^2)}$ instead of $n^{\mathcal{O}(k)}$); compared to Chlamtáč et al., our result has a much better approximation factor ($\mathcal{O}(1)$ instead of 2^{2^k}), while maintaining nearly the same asymptotic run time.

Finally, our third result gives us a range of different results trading off the size of the approximation factor against the run time of the algorithm. We remark that, by plugging in $\varepsilon = \Omega(1/\log k)$, we obtain a factor-$\mathcal{O}(\log^2 k)$ approximation in time $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

Overview of Techniques

In this section, we sketch the main ideas used in deriving our results. We assume certain familiarity with the notions of treewidth and tree decomposition, however for completeness the formal definition of tree decompositions is restated in Section 5.1. Let $(\mathcal{T}, \mathcal{B})$ be a tree decomposition of a graph G, with $\mathcal{B}$ being a collection of bags $B_t \subseteq V(G)$ for all $t \in V(\mathcal{T})$. The width of $(\mathcal{T}, \mathcal{B})$ is $w(\mathcal{T}, \mathcal{B}) = \max_{i \in V(\mathcal{T})} |B_i| - 1$, and the *treewidth* of G is the minimum width over all tree decompositions of G.

The run times of algorithms that deal with the treewidth parameter generally depend on $w(\mathcal{T}, \mathcal{B})$, so when designing an algorithm in low-treewidth graphs G one usually starts with a near-optimal tree decomposition $(\mathcal{T}, \mathcal{B})$, satisfying $w(\mathcal{T}, \mathcal{B}) = \mathcal{O}(k)$, where k is the treewidth of G. As an example, the CKR algorithm [2][2] for Sparsest Cut runs in time $2^{\mathcal{O}(w(\mathcal{T},\mathcal{B}))} \cdot n^{\mathcal{O}(1)}$ and gives approximation factor $2^{2^{w(\mathcal{T})}}$. In this setting, it is always desirable to use a tree decomposition of lowest possible width, even though an increase in the width would still allow us to obtain the desired run time of $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$. In particular, with width $w(\mathcal{T}, \mathcal{B}) = \mathcal{O}(\log n) + \beta(k)$ the CKR algorithm runs in time $2^{\mathcal{O}(\beta(k))} \cdot n^{\mathcal{O}(1)}$, but it achieves a poor approximation ratio.

We will exploit this flexibility in the run time component by employing a different analysis of the CKR algorithm so that its approximation factor depends on the diameter d of $\mathcal{T}$. We can then increase the width of a given decomposition while decreasing its diameter to obtain trade-offs between the runtime and approximation factor. In hindsight, this analysis is already present in the original work of Chlamtáč et al. and gives an approximation factor of $\mathcal{O}(d^2)$, although this is not useful when used naïvely, since d will be $\Omega(\log n)$. However, we can use it by modifying the tree decomposition to obtain a trade-off between runtime and approximation guarantee.

For a simplified example of how to obtain such a trade-off, consider a path decomposition $(\mathcal{P}, \mathcal{B})$. If we combine pairs of bags along the path we will obtain a new path decomposition whose width is twice that of $(\mathcal{P}, \mathcal{B})$, but its diameter has been halved. Taking this further, if $\mathcal{P}$ had diameter $\mathcal{O}(\log n)$ we could combine subpaths of $\log n / w(\mathcal{P}, \mathcal{B})$ many bags to obtain a modified decomposition of width $\log n + w(\mathcal{P}, \mathcal{B})$ and of diameter $\mathcal{O}(w(\mathcal{P}, \mathcal{B}))$. We would then still be able to run the CKR algorithm in time $2^{\mathcal{O}(w(\mathcal{P},\mathcal{B}))} \operatorname{poly}(n)$, but achieving an approximation ratio of $\mathcal{O}(w(\mathcal{P}, \mathcal{B})^2)$.

Let us now go from this simple example to the general setting, where instead of path decompositions we deal with tree decompositions. Initially, we obtain a tree decomposition $(\mathcal{T}, \mathcal{B})$ of G of width $\mathcal{O}(k)$ and diameter $\mathcal{O}(\log n)$ by running an algorithm by Korhonen [98] and subsequently applying an algorithm of Bodlander [99], in time $2^{\mathcal{O}(k)} \cdot \mathcal{O}(n)$. Combin-

[2]Going forward we will instead refer to the arXiv version of the paper by Chlamtáč et al. [97], as we need some results that are only present in that version.

ing such a tree decomposition with the aforementioned bag composition argument would already be sufficient to greatly improve the approximation factor of Chlamtáč et al., if it could be extended to tree rather than path decompositions. While it is possible to reduce the analysis of the CKR algorithm to the case of path decompositions, one cannot simply combine sets of neighbouring bags in the case of tree decompositions: since nodes may have many neighbours, this would cause too large an increase in the width.

In fact such a construction has natural limits since low-diameter treewidth decompositions also imply low-depth tree depth decompositions. Thus a treewidth decomposition of width $f(k) + \log n$ and constant diameter does not always exist for tree-width-k graphs. However we will still be able to use a tree decomposition that behaves essentially like this by relaxing our view of a diameter.

We introduce the concept of "combinatorial diameter" of a tree decomposition. Informally, the combinatorial length of a path between u and v in $\mathcal{T}$ measures the number of "non-redundant bags" that lie on the unique path in $\mathcal{T}$ connecting the bags of u and v. We say that the combinatorial diameter $\Delta(\mathcal{T}, \mathcal{B})$ of $(\mathcal{T}, \mathcal{B})$ is the maximum combinatorial length of any path in $\mathcal{T}$. We refer to Section 5.2.1 for formal definitions. The notion of combinatorial diameter allows us to argue that, by purposefully modifying a tree decomposition, many of the nodes do not impact the approximation factor (i.e. are redundant) since their bags contain vertices that occur only briefly during rounding, and thus are in some sense unable to carry information forward that could degrade performance. Formally, Theorem 5.2 shows that the approximation factor of the CKR algorithm can be upper bounded in terms of the combinatorial diameter as $\min\{\mathcal{O}(\Delta(\mathcal{T}, \mathcal{B})^2), 2^{2^{w(\mathcal{T},\mathcal{B})}}\}$. Moreover, in the special case of $\Delta(\mathcal{T}, \mathcal{B}) = 1$, the CKR algorithm gives a 2-approximation, which follows from the arguments of Gupta et al. [85].

With this result at hand it then suffices to construct a tree decomposition $(\mathcal{T}, \mathcal{B})$ with simultaneously low $w(\mathcal{T}, \mathcal{B})$ and low $\Delta(\mathcal{T}, \mathcal{B})$ to obtain a good approximation algorithm, i.e. one that achieves a $\mathcal{O}(\Delta(\mathcal{T}, \mathcal{B})^2)$-approximation in time $2^{w(\mathcal{T},\mathcal{B})} \cdot n^{\mathcal{O}(1)}$. Constructing such tree decompositions is possible using a generalised version of the width/diameter trade-off sketched for path decompositions, now exploiting the fact that we only need to decrease the combinatorial diameter, while the actual diameter may stay large.

This framework is surprisingly powerful, and allows us to construct a range of different tree decompositions exhibiting different widths and combinatorial diameters, which we summarise in Table 5.1. Furthermore, we can also interpret the existing results by CKR [97], GTW [85], and CMV [96] in our framework as giving constructions of tree decompositions with different trade-offs between width and combinatorial diameter. Under this lens, the existing work on approximation algorithms for the Sparsest Cut

	Lemma 5.9	Lemma 5.10	Lemma 5.11 $\forall q \in \mathbb{N}_1$	GTW [100]	CMV [96]
$\Delta(\mathcal{T}, \mathcal{B})$	$\mathcal{O}(k)$	3	$2q+1$	1	1
$w(\mathcal{T}, \mathcal{B})$	$\mathcal{O}(\log n + k)$	$\mathcal{O}(\log n + k^2)$	$\mathcal{O}(\log n + qk^{1+1/q})$	$\mathcal{O}(k \log n)$	$\mathcal{O}(\log n) + 2^{\mathcal{O}(k)}$

Table 5.1: A summary of achievable trade-offs between width and combinatorial diameter of a tree decomposition for a graph with n vertices and treewidth k. All of these constructions give rise to $\mathcal{O}(\Delta(\mathcal{T}, \mathcal{B})^2)$-approximation algorithms running in time $2^{w(\mathcal{T},\mathcal{B})} \cdot n^{\mathcal{O}(1)}$. For $\Delta(\mathcal{T}, \mathcal{B}) = 1$ the approximation factor can be shown to be 2, by the arguments in GTW [85].

problem in the bounded-treewidth regime can then be understood as being applications of the CKR algorithm to tree decompositions with the right compromise between width and combinatorial diameter.

5.1 Preliminaries

Tree decompositions Let G be a graph. A tree decomposition $(\mathcal{T}, \mathcal{B})$ of G is a tree $\mathcal{T}$ together with a collection $\mathcal{B} = \{B_i\}_{i \in V(\mathcal{T})}$ of *bags*, where the bags $B_i \subseteq V(G)$ satisfy the following properties:

- $V(G) = \bigcup_{i \in V(\mathcal{T})} B_i$.
- For each edge $uv \in E(G)$, there is a bag B_i containing both u and v.
- For each vertex $v \in V(G)$, the collection of bags containing v induces a subtree of $\mathcal{T}$.

The *width* of $(\mathcal{T}, \mathcal{B})$ is given by $w(\mathcal{T}, \mathcal{B}) = \max_{i \in V(\mathcal{T})} |B_i| - 1$, and the *treewidth* of G is the minimum width of any tree decomposition of G.

Korhonen [98] has shown how to compute a tree decomposition of a treewidth-k graph that has width $2k$ in time $2^{\mathcal{O}(k)} \cdot n$, and a procedure due to Bodlaender [99] allows us to transform any such decomposition into one that has diameter $\mathcal{O}(\log n)$ and width at most $6k + 2$.

We will also assume that the number of nodes in the tree $\mathcal{T}$ is bounded by $\mathcal{O}(n)$, by the following simple process: repeatedly delete any bag that has at most one child and is a subset of its parent; after this process ends, there are at most n bags with at most one child (since each must introduce a new vertex), and thus at most $2n$ nodes in total.

We generally use r to denote the root of $\mathcal{T}$, and $p \colon V(\mathcal{T}) \to V(\mathcal{T})$ for the parent of a node with respect to root r, where $p(r) = r$. We sometimes refer to $B_{p(i)}$ as the *parent bag* of B_i. Let $\mathcal{T}_{i \leftrightarrow j}$ be the set of nodes on the unique path in tree $\mathcal{T}$ between nodes $i, j \in V(\mathcal{T})$ (possibly $i = j$). For a

set $X \subseteq V(\mathcal{T})$ of nodes, we use the shorthand $B(X) = \bigcup_{i \in X} B_i$ (the union of bags for nodes in X).

We will treat cuts in a graph as assignments of $\{0, 1\}$ to each vertex, which we formally denote as X-assignments:

Definition 5.1. Let X be some finite set. We call a map $f \colon X \to \{0, 1\}$ an *X-assignment*. We denote by $\mathcal{F}[X]$ the set of all X-assignments. For some probability distribution μ over $\mathcal{F}[X]$ and set $Y \subseteq X$ we define $\mu|_Y$ to be the distribution given by

$$\Pr_{f \sim \mu|_Y}[f = f'] = \Pr_{f \sim \mu}[f|_Y = f'] \quad \forall f' \in \mathcal{F}[Y]\,.$$

5.2 Algorithm and Combinatorial Diameter

Our approach is based on the relation between the algorithm of Chlamtáč et al. [97] and our novel notion of "combinatorial diameter". In Section 5.2.1, we present the definition of combinatorial diameter. The subsequent sections give the description of Chlamtáč et al. and prove its connection to the combinatorial diameter.

5.2.1 Our New Concept: Combinatorial Diameter

Let G be a graph and let $(\mathcal{T}, \mathcal{B})$ be a tree decomposition of G, where we say $\mathcal{B} = \{B_i\}_{i \in V(\mathcal{T})}$.

Definition 5.2 (Redundant bags)**.** Fix $s, t \in V(\mathcal{T})$. Let $v \in V(\mathcal{T}) \setminus \{s, t\}$ be a node with neighbors u and w on the path $\mathcal{T}_{s \leftrightarrow t}$. When $B_v \cap B_w \subseteq B_u$, we say that v is *(s, t)-redundant*.

Intuitively, each redundant node v can be thought of as a subset of u, since the vertices in $B_v \setminus B_u$ occur only in B_v within $\mathcal{T}_{s \leftrightarrow t}$. As a consequence, we can show that they do not affect the rounding behaviour of the CKR algorithm with respect to s and t (therefore "redundant"). It might be surprising that redundancy depends on the direction of the traversal, i.e. a node might be (s, t)-redundant but not (t, s)-redundant. We will be able to circumvent this by observing that the CKR algorithm is not impacted by the choice of direction, so the definition will effectively be symmetric in the sense that if a node is redundant in one of the directions, we may assume that the algorithm is using this direction.

Definition 5.3 (Simplification)**.** Let $(\mathcal{T}, \mathcal{B})$ be a tree decomposition, and let $s, t \in V(\mathcal{T})$. A *simplification* of $\mathcal{T}_{s \leftrightarrow t}$ is a path P which can be generated from $\mathcal{T}_{s \leftrightarrow t}$ by repeatedly applying the following rule:

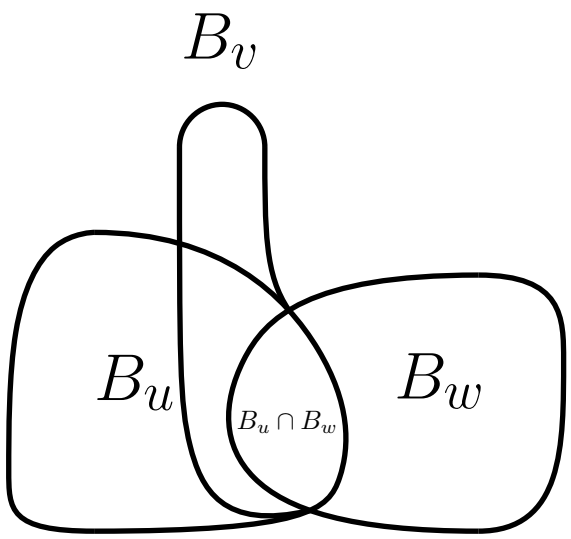


Figure 5.2: Illustration of the bags of a redundant node v and its neighbors. Notice that the vertices in B_v are either in B_u, in which case they are already processed at u, or they are neither in B_u nor in B_w, in which case they have no impact on the rounding algorithm along this path.

> Delete an (s,t)-redundant node v with neighbors u, w on the path $\mathcal{T}_{s \leftrightarrow t}$, and add the edge $\{u, w\}$. We call this operation *bypassing* v.

Definition 5.4 (Combinatorial diameter)**.** Let $(\mathcal{T}, \mathcal{B})$ be a tree decomposition, and let $s, t \in V(\mathcal{T})$. We say $\mathcal{T}_{s \leftrightarrow t}$ has *combinatorial length* at most ℓ if it has a simplification of length at most ℓ. The *combinatorial diameter* of $(\mathcal{T}, \mathcal{B})$ is defined as the minimum δ such that, for all u, v, the path $\mathcal{T}_{u \leftrightarrow v}$ has combinatorial length at most δ.

5.2.2 Algorithm Description and Overview

We will restate here the essential aspects of the algorithm by Chlamtáč et al. [97] since it is central to our approach. The algorithm is initially provided a Sparsest Cut instance $(G, D, \text{cap}, \text{dem})$ alongside a tree decomposition $(\mathcal{T}, \mathcal{B})$ of G. The goal is then to compute a cut in G that has low sparsity.

The algorithm starts by computing, for every vertex set $L = B_i \cup \{s, t\}$ consisting of a bag B_i and a pair of vertices $s, t \in V(G)$, a distribution μ_L over L-assignments. The collection of distributions for all sets L satisfies *consistency* constraints, that is, any two distributions must agree on their joint domains, i.e., $\mu_L|_{L \cap L'} = \mu_{L'}|_{L \cap L'}$ for each pair of sets L, L' with the structure above. Such distributions can be seen as an abstraction of fractional solutions in the context of Sherali-Adams linear programming hierarchies [101].

If we denote $\mathsf{lpcut}(s,t) = \Pr_{f\sim\mu_{B\cup\{s,t\}}}[f(s) \neq f(t)]$ for any $s,t \in V(G)$, and an arbitrary bag B of $\mathcal{B}$, we can compute the collection of distributions that minimizes

$$\alpha := \frac{\sum_{\{s,t\}\in E_G} \mathrm{cap}_{\{s,t\}} \cdot \mathsf{lpcut}(s,t)}{\sum_{\{s,t\}\in E_D} \mathrm{dem}_{\{s,t\}} \cdot \mathsf{lpcut}(s,t)}\,.$$

We say that α is the *cut sparsity predicted by distributions* $\{\mu_L\}_L$. Notice that lpcut is well defined by the consistency requirement, since the choice of B does not impact the distribution over $\{s,t\}$-assignments. For ease of notation, we will refer to the implied distribution over some vertex set $X \subseteq B \cup \{s,t\}$ by μ_X, where formally $\mu_X = \mu_{B\cup\{s,t\}}|_X$.

Such a collection of distributions can be computed in time $2^{\mathcal{O}(w(\mathcal{T}))}\cdot n^{\mathcal{O}(1)}$, using Sherali-Adams linear programming hierarchies restricted to subsets of the form $B_i \cup \{s,t\}$ (see Chlamtáč et al. [97]). This motivates the function name lpcut. Additionally, such a linear program is a relaxation of the Sparsest Cut problem since the local distributions selecting a fixed (optimal, global) cut are feasible solutions and thus α is a lower bound for the minimum sparsity of a cut.

Algorithm 5 obtains a $V(G)$-assignment f by rounding this collection of distributions $\{\mu_L\}$: starting at a bag B_0 (which is considered the root), it chooses an assignment for vertices in B_0 by sampling from the distribution μ_{B_0}; then, for each bag B for which its parent B' is processed, it chooses a B-assignment according to μ_B conditioned on the assignment for $B \cap B'$. As we will see, this rounding approximately preserves the probability of cutting a pair (s,t), thus implying an approximation to the problem.

We now recall a number of useful results about the algorithm and the assignment it computes. Details about the algorithm and the attendant lemmas can be found in the work of Chlamtáč et al. [97].

Denote by $\mathcal{A}$ the distribution over $V(G)$-assignments produced by the algorithm.

Lemma 5.1 ([97, Lemma 3.3])**.** *For every bag B the assignment $f|_B$ computed by Algorithm 5 is distributed according to μ_B, formally meaning that $\Pr_{f\sim\mathcal{A}}[f|_B = f'] = \Pr_{f^*\sim\mu_B}[f^* = f']$ for all $f' \in \mathcal{F}[B]$.*

A direct consequence of this lemma is the fact that any edge $\{s,t\}$ of G is cut by the algorithm with probability $\mathsf{lpcut}(s,t)$. In particular, the expected capacity of the rounded cut is therefore

$$\sum_{\{s,t\}\in E_G} \mathrm{cap}_{\{s,t\}} \cdot \mathsf{lpcut}(s,t)\,,$$

according to the distributions $\{\mu_L\}$. However, the same property does not hold for the demand edges in D, since the Lemma 5.1 only applies to bags

Algorithm 5: Algorithm SC-ROUND

Data: $G, (\mathcal{T}, \{B_i\}_{i \in V(\mathcal{T})}), \{\mu_L\}$

Start at any bag B_0, sample $f|_{B_0}$ from μ_{B_0};

We process the bags in non-decreasing order of distance from B_0 ;

foreach *bag B with a processed parent bag B'* **do**

Let $B^+ = B \cap B'$ the subset of B on which f is fixed. Let $B^- := B \setminus B^+$. Sample $f|_{B^-}$ according to

$$\Pr[f|_{B^-} = f'] = \Pr_{f^* \sim \mu_B}[f^*|_{B^-} = f' \mid f^*|_{B^+} = f|_{B^+}] \quad \forall f' \in \mathcal{F}[B^-]$$

end

Result: f

of $\mathcal{T}$. Specifically, the bags of $\mathcal{T}$ do not necessarily contain all of the edges of D, as we do not assume that D has bounded treewidth.

Denote by $\mathsf{algcut}(s,t)$ the probability that the algorithm separates s and t, that is, $\mathsf{algcut}(s,t) = \Pr_{f \sim \mathcal{A}}[f(s) \neq f(t)]$. We would like to lower bound the value $\mathsf{algcut}(s,t) \geq c\,\mathsf{lpcut}(s,t)$ for all demand edges $\{s,t\}$ and some value $c > 0$. This would imply that the expected demand of the rounded cut is at least $c \sum_{\{s,t\} \in E_D} \mathrm{dem}_{\{s,t\}}\,\mathsf{lpcut}(s,t)$, and having a good expected demand and capacity is sufficient for computing a good solution by the following observation.

Observation 5.1 ([97], Remark 4.3)**.** *Let $\{\mu_L\}_L$ be a collection of distributions and α be the cut sparsity predicted by $\{\mu_L\}_L$.*

Then if we have that $\mathsf{algcut}(s,t) \geq c \cdot \mathsf{lpcut}(s,t)$ for all $\{s,t\} \in E_D$ and $\mathsf{algcut}(s,t) = \mathsf{lpcut}(s,t)$ for all $\{s,t\} \in E_G$, we get

$$\mathbb{E}_{f \sim \mathcal{A}}\left[\sum_{\{s,t\} \in E_G} \mathrm{cap}_{\{s,t\}}\,|f(s) - f(t)| - \frac{\alpha}{c} \sum_{\{s,t\} \in E_D} \mathrm{dem}_{\{s,t\}}\,|f(s) - f(t)|\right] \leq 0\,.$$

Furthermore, an assignment f is c-approximate if the value in the expectation above is non-positive, and such a solution can either be obtained by repeated rounding or by derandomization using the method of conditional expectations, without increasing the asymptotic run time.

This observation implies that the bottleneck to obtaining a good approximation factor is the extent to which our rounding algorithm can approximate the marginal of μ_L on the individual edges of D. Our main result relates this marginal to the combinatorial diameter of $\mathcal{T}$. It can now be stated as follows:

Theorem 5.2. *Let* $(G, D, \text{cap}, \text{dem})$ *be some instance of* Sparsest Cut *and* $(\mathcal{T}, \mathcal{B})$ *a tree decomposition of* G *with width* $w(\mathcal{T}, \mathcal{B})$ *and combinatorial diameter* $\Delta(\mathcal{T}, \mathcal{B})$.

Then SC-Round *satisfies* $\mathsf{algcut}(s,t) \geq \Omega(\frac{1}{\Delta(\mathcal{T},\mathcal{B})^2}) \cdot \mathsf{lpcut}(s,t)$ *for every edge* $\{s,t\} \in E_D$. *Thus, we have a factor-*$\mathcal{O}(\Delta(\mathcal{T},\mathcal{B})^2)$ *approximation algorithm for* Sparsest Cut *with run time* $2^{\mathcal{O}(w(\mathcal{T},\mathcal{B}))} \cdot n^{\mathcal{O}(1)}$.

The rest of this section is devoted to proving this theorem.

5.2.3 Step 1: Reduction to Short Paths

In this section, we show that when the combinatorial diameter of the tree decomposition is $\delta = \Delta(\mathcal{T}, \mathcal{B})$, the analysis can be reduced to the case of a path decomposition of length δ. We employ the following lemma to simplify our analysis of the behavior of the algorithm.

Lemma 5.2 ([97, Lemma 3.4])**.** *The distribution over the assignments* f *is invariant under any connected traversal of* $\mathcal{T}$*, i.e., the order in which bags are processed does not matter, as long as they have a previously processed neighbor. The choice of the first bag* B_0 *also does not impact the distribution.*

Let $\{s,t\} \in E_D$ be a demand edge. If s and t are contained in a common bag, then $\mathsf{algcut}(s,t) = \mathsf{lpcut}(s,t)$ by Lemma 5.1 and we are done; therefore, we assume that there is no bag containing both s and t. We want to estimate the probability that s and t separated by the algorithm, that is, the probability that $f(s) \neq f(t)$.

The lemma above allows us to reduce to the case in which the algorithm first rounds a bag B_1 containing s, then rounds bags $B_2, \ldots, B_{\ell-1}$ along the path to a bag B_ℓ containing t, and finally B_ℓ. At this point the algorithm has already assigned $f(s)$ and $f(t)$, so the remaining bags of $\mathcal{T}$ can be rounded in any connected order without impacting the separation probability. Hence, it is sufficient to characterize the behavior of the rounding algorithm along paths in $\mathcal{T}$.

Let P be the shortest path connecting a bag containing s to a bag containing t; denote such path by $P = v_1 v_2 \ldots v_\ell$ such that $s \in B_{v_1}$ and $t \in B_{v_\ell}$. By Lemma 5.2, we can assume that the algorithm first processes B_{v_1}, and then all other bags $B_{v_2}, \ldots, B_{v_\ell}$, in this order. Let $\mathcal{B}_P = \{B_{v_i}\}_{i=1}^{\ell}$.

Observe that, except for v_1 and v_ℓ, no other bag of P contains s or t. We repeatedly apply the reduction rule from Definition 5.3 until the resulting path has length at most δ. The following lemma asserts that the distribution of the algorithm is preserved under this reduction rule.

We slightly abuse the notation and denote by $\mathcal{A}$ the distribution of our algorithm on path P starting from v_1.

Lemma 5.3. *Let u, v, w be three consecutive internal nodes on P such that $B_v \cap B_w \subseteq B_u$. Let P' be the simplification of P obtained by bypassing v, and let $\mathcal{A}'$ be the distribution obtained by running the algorithm on path P', starting on v_1. Then $\mathcal{A}'$ is exactly the same as $\mathcal{A}$ restricted to $B(P')$.*

Proof. We can assume, without loss of generality, that u, v, w appear on P in the order of rounding; for otherwise, we apply Lemma 5.2 twice: first, to reverse P, and preserve the distribution $\mathcal{A}$; then, to undo the reversing of P' caused by the previous application.

We modify the path decomposition $(P, \mathcal{B}_P)$ into a (tree) decomposition $(\hat{\mathcal{T}}, \hat{\mathcal{B}})$ as follows: remove bag v and add two new bags v', v'' where bag v' is connected to u and w with $B_{v'} = B_u \cap B_v$ and v'' is connected to v' with $B_{v''} = B_v$. This remains a tree decomposition for the vertices in $B(P)$ since vertices in $B_v \setminus B_u$ only occur in the bag $B_{v''}$ (due to our assumption that $B_v \cap B_w \subseteq B_u$).

Running the algorithm SC-Round on $\hat{\mathcal{T}}$ produces exactly the same distribution as $\mathcal{A}$, since we can first round the bags from s to u, then v' and v'', and then the bags from u to t. Since $B(P') = B(P) \setminus (B_{v''} \setminus B_{v'})$, we have that by Lemma 5.2, $\mathcal{A}|_{B(P')}$ is the distribution of SC-Round on the path $\hat{P} = v_1 \dots uv'w \dots v_\ell$, obtained by removing v'' from $\hat{\mathcal{T}}$. Now since $B_{v'} \subseteq B_u$, the rounding algorithm in fact does not do anything at bag v', so it can be removed without affecting the distribution. We obtain path P' as a result, and thus $\mathcal{A}|_{B(P')}$ is the same distribution as $\mathcal{A}'$. □

This result allows us conduct the rounding analysis on simplifications of paths. It remains to show that this is beneficial, that is, that the rounding error can be bounded by the length of the path on which we round. As in the work of Chlamtáč et al. [97], we use Markov flow graphs to analyze that error. However, as the length of their paths is $\Theta(\log n)$, they need to derive a bound depending not on that length, but on the width of the decomposition. Since we can now bound the length, we only need a small part of their argument.

5.2.4 Step 2: Markov Flow Graphs

Let $P = v_1, \dots, v_\ell$ be a path with length ℓ and $s \in B_{v_1}$, $t \in B_{v_\ell}$. We run Algorithm 5 from v_1 to v_ℓ to compute some assignment f. Let $\mathcal{A}$ be the probability distribution of the resulting assignment f. Recall that $\mathsf{algcut}(s,t)$ denotes the probability that the algorithm assigns $f(s) \neq f(t)$, and $\mathsf{lpcut}(s,t)$ is the probability that s and t are separated according to the distributions $\{\mu_L\}_L$, i.e., $\Pr_{f \sim \mu_{B \cup \{s,t\}}}[f(s) \neq f(t)]$. In the second step, we analyze the probability of $\mathsf{algcut}(s,t)$ in terms of $\mathsf{lpcut}(s,t)$. This step is encapsulated in the following lemma.

Lemma 5.4. *There exists a directed graph H, nodes $s_0, s_1, t_0, t_1 \in V(H)$, and a weight function w_H on the edges, satisfying the following properties:*

1. *For $i = 0, 1$, we have that $\Pr_{f\sim\mathcal{A}}[f(s) = i \ \wedge \ f(t) = 1 - i]$ is at least an $\Omega(1/\ell^2)$-fraction of the minimum (s_i, t_{1-i})-cut of H, where ℓ is the length of the path P.*

2. *For $i = 0, 1$, the value of a maximum (s_i, t_{1-i})-flow in H is at least $\Pr_{f\sim\mu}[f(s) = i \ \wedge \ f(t) = 1 - i]$.*

With Lemma 5.4 in hand, we can now complete the proof of Theorem 5.2 as follows.

Proof of Theorem 5.2. We run the algorithm of Chlamtáč et al. to get some $V(G)$-assignment f.

Consider a pair $\{s, t\} \in E_D$. Using Lemma 5.2 and Lemma 5.3, we can reduce the analysis to a path P of length at most $\delta = \Delta(\mathcal{T}, \mathcal{B})$, which is a simplification of a path in $\mathcal{T}$. Now, by Lemma 5.4 applied with $\ell = \delta$ and max-flow-min-cut theorem, we get that

$$\begin{aligned}
\mathsf{algcut}(s,t) &= \Pr_{f\sim\mathcal{A}}[f(s) = 0 \ \wedge \ f(t) = 1] + \Pr_{f\sim\mathcal{A}}[f(s) = 1 \ \wedge \ f(t) = 0] \\
&\geq \Omega(1/\delta^2)(\mathrm{mincut}(s_0, t_1) + \mathrm{mincut}(s_1, t_0)) \\
&= \Omega(1/\delta^2)(\mathrm{maxflow}(s_0, t_1) + \mathrm{maxflow}(s_1, t_0)) \\
&\geq \Omega(1/\delta^2)(\Pr_{f\sim\mu}[f(s) = 0 \ \wedge \ f(t) = 1] + \Pr_{f\sim\mu}[f(s) = 1 \ \wedge \ f(t) = 0]) \\
&= \Omega(1/\delta^2)\,\mathsf{lpcut}(s,t)\,.
\end{aligned}$$

Recall that, by consistency of the distributions, the probability of an $\{s, t\}$-assignment is the same for every distribution μ_L s.t. $\{s, t\} \subseteq L$; for this reason, we slightly abuse notation and write $\Pr_{f\sim\mu}$ to mean the probability according to any distribution containing $\{s, t\}$.

We conclude that f separates each pair $\{s, t\}$ with probability that is a factor of $\mathcal{O}(\delta^2)$ away from $\mathsf{lpcut}(s, t)$ as desired. Applying Observation 5.1 with $c = \Omega(1/\delta^2)$, we can obtain (deterministically) an assignment f^* that is an $\mathcal{O}(\delta^2)$-approximation for the Sparsest Cut instance.

Finally, the run time can be seen to be $2^{\mathcal{O}(w(\mathcal{T},\mathcal{B}))} \cdot n^{\mathcal{O}(1)}$ using the arguments by Chlamtáč et al. [97], which are summarized as follows: To compute the distributions $\{\mu_L\}_L$, we write a linear program containing a variable for every subset $S \subseteq L$ for every considered set $L = B \cup \{s, t\}$. Therefore, we take subsets of at most $\mathcal{O}(n)$ sets, each of size at most $w(\mathcal{T}, \mathcal{B}) + 3$, which totals to $\mathcal{O}(n) \cdot 2^{w(\mathcal{T},\mathcal{B})}$. Both solving the linear program to obtain the distributions and the rounding procedure are polynomial in the number of variables, thus leading to the claimed run time. □

The rest of this section is dedicated to proving Lemma 5.4. The tools needed for this proof are implicit in the work of Chlamtáč et al. [97]. We restate them for the sake of completeness and in order to adjust it to our terminology. To start, we will take care of a small technical detail inherent to the analysis of Chlamtáč et al., requiring the LP solution to be symmetric in 0 and 1.

Definition 5.5. For any set X and X-assignment g we define the *mirror* of g to be $\overleftrightarrow{g} := \sigma \circ g$, where $\sigma(0) = 1$ and $\sigma(1) = 0$.

Notice that the mirror of an assignment represents the same cut; it merely exchanges the labels. The approximation factor analysis of Chlamtáč et al. requires the distributions to be symmetric in their labeling, in particular $\Pr[f(v) = 0] = \Pr[f(v) = 1]$ for all $v \in V(G)$. They resolve this by demanding symmetry via the Sherali-Adams LP which can be shown to not worsen the relaxation. Using the following lemma, we are able to prove that the rounding analysis also holds in the non-symmetric case.

Lemma 5.5. *Let $G, (\mathcal{T}, \{B_i\}_{i \in V(\mathcal{T})}), \{\mu_L\}$ be the input of Algorithm 5. Consider the modified decomposition $(\mathcal{T}, \{B'_i\}_{i \in V(\mathcal{T}')})$, where a dummy vertex e has been added to every bag B'_i, and the collection of distributions $\{\overleftrightarrow{\mu}_L\}$ defined as:*

$$\overleftrightarrow{\mu}_{L \cup \{e\}}(g', e \to 0) = \frac{1}{2}\mu_L(g'), \quad \overleftrightarrow{\mu}_{L \cup \{e\}}(g', e \to 1) = \frac{1}{2}\mu_L(\overleftrightarrow{g'}) \quad \forall g' \in \mathcal{F}[L]\,.$$

Then $\overleftrightarrow{\mu}$ has the following properties:

1. *For all $L' = L \cup \{e\}$ and $g' \in \mathcal{F}[L']$, we have that $\Pr_{g \sim \overleftrightarrow{\mu}_{L'}}[g = g'] = \Pr_{g \sim \overleftrightarrow{\mu}_{L'}}[g = \overleftrightarrow{g'}]$*
2. *If g, g^* are random variables representing the resulting assignments of Algorithm 5 run on $G, (\mathcal{T}, \{B_i\}_i), \{\mu_L\}$, and $G, (\mathcal{T}, \{B'_i\}_i), \{\overleftrightarrow{\mu}_{L'}\}$, respectively, then we have*

$$\Pr[g = g'] + \Pr[g = \overleftrightarrow{g'}] = \Pr[g^*|_{V(G)} = g'] + \Pr[g^*|_{V(G)} = \overleftrightarrow{g'}]$$

for all $g' \in \mathcal{F}[V(G)]$.

The content of the lemma is at its core not very surprising. If we do not care about the permutation of the labels, we do not care about whether the algorithm outputs g or $\overleftrightarrow{g}$. But if that is the case, the distributions also should not need to maintain some distinction between the labels. In fact, one could run the algorithm unmodified, and then permute the labels with probability 1/2. Clearly, this does not change the distribution over cuts, and the choice of labels is now symmetric.

Proof. To make the argument formal, we shall use the value of $g(e)$ to indicate whether or not we are permuting the labels. By Lemma 5.2 we can model the rounding algorithm for $G, (\mathcal{T}, \{B'_i\}_i), \{\vec{\mu}_{L'}\}$ as choosing first a value for $g(e)$, and then proceeding in the same order as the rounding over $G, (\mathcal{T}, \{B_i\}_i), \{\mu_L\}$. With probability $1/2$ we get $g(e) = 0$. Since every bag contains e, e is always conditioned on, so the symmetrized algorithm performs exactly the same computations as the original run would. Meanwhile if $g(e) = 1$, the symmetrized algorithm samples some intermediate assignment g' with exactly the probability that the original algorithm would have sampled $\overleftarrow{g'}$.

As a result,

$$\Pr[g^*|_{V(G)} = g'] = \frac{1}{2}\Pr[g = g'] + \frac{1}{2}\Pr[\overleftarrow{g} = g'] \qquad \forall g' \in \mathcal{F}[V(G)]\,,$$

which implies

$$\Pr[g^*|_{V(G)} = g'] + \Pr[g^*|_{V(G)} = \overleftarrow{g'}] = \Pr[g = g'] + \Pr[g = \overleftarrow{g'}] \quad \forall g' \in \mathcal{F}[V(G)]\,.$$

□

Notice that while we could construct the symmetrized $\overleftrightarrow{\mu}$ efficiently, we do not need to, as μ is distributed identically, and thus satisfies the same bounds, as $\overleftrightarrow{\mu}$. More concretely, we apply the analysis to $\overleftrightarrow{\mu}$, which is possible because it is symmetric. We can then use Lemma 5.5 to conclude that the probabilities of cutting a pair of vertices in μ and $\overleftrightarrow{\mu}$ are the same, since these probabilities are symmetric, in the sense that

$$\Pr[f(s) \neq f(t)] = \Pr[f(s) = 1 \wedge f(t) = 0] + \Pr[f(s) = 0 \wedge f(t) = 1].$$

Therefore, the bounds on the probabilities of cutting a pair in $\overleftrightarrow{\mu}$ apply also to μ, and thus our results apply obtaining a solution directly from μ.

We now proceed as follows: first, we describe the construction of the graph H, and then we proceed to analyze the values of maximum flow and minimum cut. We will only analyze the flow and cut for $i = 0$, that is, (s_0, t_1)-flow and (s_0, t_1)-cut. The other case is analogous.

Construction of Graph H: By Lemma 5.5 we can assume that the distributions $\{\mu_L\}_L$ are symmetric in the labels $\{0, 1\}$. In particular, this gives us that $\Pr[f(v) = 1] = \Pr[f(v) = 0] = 1/2$ for any vertex v.

The rounding can be modeled by a simple Markov process. Denote by $I_0, \ldots, I_\ell$ the sets that are conditioned on in Algorithm 5, $I_i = B_{v_i} \cap B_{v_{i+1}}$ for $i \in \{1, \ldots, \ell - 1\}$; we refer to these sets as *conditioning sets*.

For the initial and final sets of the rounding procedure we are only interested in s and t, so take $I_0 = \{s\}$, $I_\ell = \{t\}$. Now we are ready to describe our graph H:

- **Vertices**: The vertices of H are arranged into disjoint layers $L_0, \dots, L_\ell$ with $L_i = \mathcal{F}[I_i]$. Observe that $|L_i| = 2^{|I_i|}$. The vertices of H represent the intermediate states the algorithm might reach.

- **Edges**: For each i, there is a directed edge from every vertex in L_i to every vertex in L_{i+1}. The weight of an edge (f_i, f_{i+1}) where $f_i \in L_i$, $f_{i+1} \in L_{i+1}$, is equal to the probability of the joint event, that is we set $w_H(f_i, f_{i+1}) = \Pr[f|_{I_i} = f_i \wedge f|_{I_{i+1}} = f_{i+1}]$.

  We remark that the weight is 0 whenever f_i and f_{i+1} are contradictory, and that probabilities are well defined, as $I_i \cup I_{i+1} \subseteq B_{i+1}$.

Observe that the weight of an edge is the probability that both of its endpoints are reached by the algorithm, and hence the probability that the algorithm transitions along that edge.

Observation 5.2. *Let $\mathcal{I} = \bigcup_i I_i$. The distribution $\mathcal{A}|_{\mathcal{I}}$ can be viewed as the following random walk in H: Pick a random vertex in L_0 and start taking a random walk where each edge is taken with probability proportional to its weight. Formally, once a node f_i is reached, choose the next node f_{i+1} with probability $w_H(f_i, f_{i+1}) / \Pr[f|_{I_i} = f_i]$.*

At this point, we rename $\mathcal{A} := \mathcal{A}|_{\mathcal{I}}$. Notice that the layer L_0 contains two vertices corresponding to the assignment $f(s) = 0$ and $f(s) = 1$, respectively. We denote them by $L_0 = \{s_0, s_1\}$. Similarly, $L_\ell = \{t_0, t_1\}$. Notice further that $\Pr_{f\sim\mathcal{A}}[f(s) = 0, f(t) = 1]$ is exactly the probability that the random walk starts at $s_0 \in L_0$ and ends at $t_1 \in L_\ell$.

Maximum (s_0, t_1)-Flow: We are now ready to show that the value of the maximum (s_0, t_1)-flow is at least $\Pr_{f\sim\mu}[f(s) = 0, f(t) = 1]$.

We define the flow $g\colon E(H) \to \mathbb{R}_{\geq 0}$ as

$$g(f_i, f_{i+1}) = \Pr_{f\sim\mu_{B_{v_{i+1}}\cup\{s,t\}}}[f(s) = 0, f(t) = 1, f|_{I_i} = f_i, f|_{I_{i+1}} = f_{i+1}]$$

for $i \in \{1, \dots, \ell - 1\}$, $f_i \in L_i$ and $f_{i+1} \in L_{i+1}$.

We remark that g is an s_0-t_1-flow, that is, it satisfies flow conservation at all vertices in H except s_0, t_1, and the capacities of the graph H are respected, that is, $g(e) \leq w_H(e)$ for all $e \in E(H)$. The value of g is given by:

$$\begin{aligned}\sum_{(s_0,f^*)\in\delta^+(s_0)} g(s_0, f^*) &= \sum_{f^*\in\mathcal{F}[I_1]} \Pr_{f\sim\mu_{B_{v_1}\cup\{s,t\}}}[f(s) = 0, f(t) = 1, f|_{I_1} = f^*] \\ &= \Pr_{f\sim\mu_{B_{v_1}\cup\{s,t\}}}[f(s) = 0, f(t) = 1]\,.\end{aligned}$$

This concludes the proof of Point 2 of Lemma 5.4.

A Potential Function: Before we show a cut with the desired capacity, we need to introduce some notation. For $i = 0, \dots, \ell$, let X_i be a random variable indicating the vertex in L_i visited by the random walk (i.e., picked by the algorithm. We denote by $\mathbf{X} = X_0 X_1 \dots X_\ell$ the path taken in the random walk process. We can interchangeably view distribution $\mathcal{A}$ as either the distribution that samples an assignment $f \colon \mathcal{I} \to \{0,1\}$ or one that samples a (random walk) path $\mathbf{X}$.

We define, for every layer L_i and every vertex $v \in L_i$,

$$A(v) := \Pr_{\mathbf{X} \sim \mathcal{A}}[X_0 = s_0 \mid X_i = v] - \frac{1}{2}\,.$$

Intuitively, this function captures the extent to which v has information about the initial state of the Markov process. On the one hand, if $A(v)$ is equal to 0, v knows essentially nothing about X_0, the choice of v does not imply anything about X_0. On the other hand, if $A(v)$ is far from 0, then we can glean some information about X_0 from v being visited; in particular, if the probability that s and t are cut is low, we must have $A(t_1) \approx -1/2$.

To track how A changes from layer to layer, we use the potential function $\phi \colon \{0, \dots, \ell\} \to \mathbb{R}_{\geq 0}$, which we define as:

$$\phi(i) := \mathrm{Var}_{\mathbf{X} \sim \mathcal{A}}[A(X_i)]\,.$$

The following argument by Chlamtáč et al. bounds the change in potential in terms of the probability that $X_0 = s_0$ and $X_\ell = t_1$.

Corollary 5.1 (From [97, Lemma 5.2])**.** *Without loss of generality, we can assume that* $\phi(0) - \phi(\ell) \leq 2 \Pr_{\boldsymbol{X} \sim \mathcal{A}}[X_0 = s_0 \wedge X_\ell = t_1]$.

Proof. Suppose that $\Pr_{\mathbf{X} \sim \mathcal{A}}[X_0 = s_0 \wedge X_\ell = t_1] > \frac{1}{4}$. Then we already have that $\mathsf{algcut}(s,t) \geq \frac{1}{2}\,\mathsf{lpcut}(s,t)$ and thus do not need any further analysis. Otherwise, we observe that L_0 and L_ℓ only contain two nodes each. We can therefore compute the variances explicitly:

$$\begin{aligned}
\phi(0) - \phi(\ell) &= A(s_0)^2 - A(t_1)^2 \\
&= \frac{1}{4} - A(t_1)^2 \\
&= \frac{1}{4} - (2 \Pr_{\mathbf{X} \sim \mathcal{A}}[X_0 = s_0 \wedge X_\ell = t_1] - \frac{1}{2})^2 \\
&\leq 2 \Pr_{\mathbf{X} \sim \mathcal{A}}[X_0 = s_0 \wedge X_\ell = t_1]\,.
\end{aligned}$$

□

Minimum (s_0, t_1)-cut: We are now ready to analyze the value of minimum (s_0, t_1)-cut in H. It suffices to give a lower bound on $\phi(0) - \phi(\ell)$ by Corollary 5.1. The following lemma from Chlamtáč et al. [97] is necessary to control the change of ϕ when moving between layers.

Lemma 5.6 ([97, Lemma 5.2])**.** *For all $0 < l_1 < l_2 < \ell$ we have*

$$\phi(l_1) - \phi(l_2) = \sum_{u \in L_{l_1}, v \in L_{l_2}} \Pr_{\boldsymbol{X} \sim \mathcal{A}}[X_{l_1} = u \wedge X_{l_2} = v](A(u) - A(v))^2 .$$

With Lemma 5.6 it is possible to prove the following lemma which is shown implicitly by Chlamtáč et al. [97]. We provide a full proof for completeness.

Lemma 5.7 (Analogous to [97, Lemma 5.4])**.** *Let C be the set of edges (f_i, f_{i+1}) in $E(H)$ such that $|A(f_i) - A(f_{i+1})| \geq \rho$, for some threshold $\rho > 0$. Then $\sum_{e \in C} w_H(e) \leq (\phi(0) - \phi(\ell)) \cdot 1/\rho^2$.*

Proof. It holds that

$$\begin{aligned}
\sum_{e \in C} w_H(e) &= \sum_{i \in 0,\dots,\ell-1} \sum_{\substack{f_i \in L_i, f_{i+1} \in L_{i+1} \\ |A(f_i) - A(f_{i+1})| \geq \rho}} w_H(f_i, f_{i+1}) \\
&\leq \frac{1}{\rho^2} \sum_{i \in 0,\dots,\ell-1} \sum_{\substack{f_i \in L_i, f_{i+1} \in L_{i+1} \\ |A(f_i) - A(f_{i+1})| \geq \rho}} w_H(f_i, f_{i+1})(A(f_i) - A(f_{i+1}))^2 \\
&\leq \frac{1}{\rho^2} \sum_{i \in 0,\dots,\ell-1} (\phi(i) - \phi(i+1)) \\
&= \frac{1}{\rho^2}(\phi(0) - \phi(\ell)),
\end{aligned}$$

where the second inequality is an application of Lemma 5.6. □

We can apply Lemma 5.7 in the following fashion. Suppose $A(t_1) \geq 0$. In that case we have $\Pr[X_0 = s_0 \mid X_\ell = t_1] \geq 1/2$, so s and t are cut with probability at least $\frac{1}{2}\,\mathsf{lpcut}(s,t)$. This error is already small enough, so assume $A(t_1) < 0$. Then $A(s_0) - A(t_1) > 1/2$. Since every path from s_0 to t_1 has exactly ℓ edges, any such path must contain an edge (f_i, f_j) with $A(f_i) - A(f_j) > 1/(2\ell)$. Cutting all such edges therefore separates s_0 and t_1. Hence, by applying Lemma 5.7, the minimum s_0-t_1-cut has size at most

$$\begin{aligned}
\mathcal{O}(\ell^2)(\phi(0) - \phi(\ell)) &\leq \mathcal{O}(\ell^2) \Pr[X_0 = s_0 \wedge X_\ell = t_1] \\
&= \mathcal{O}(\ell^2) \Pr[f(s) = 0 \wedge f(t) = 1] .
\end{aligned}$$

This concludes the proof of Point 1 of Lemma 5.4. We see that the cutting probability predicted by the distributions is realised by the rounded solution f, up to a factor $\Omega(1/\ell^2)$.

The analysis of this section gives an alternative to the one given by Chlamtáč et al.. Their constant of approximation depends on the size of the layers (i.e., the width of the bags of the tree decomposition) of H rather

than the number of layers (i.e., the length of the path on which the analysis is performed). While the layer sizes depend only on k, the dependence is exponential. The number of layers meanwhile is a priori $\log(n)$, which would give a worse approximation guarantee. However, we will show how to modify a tree decomposition to ensure that all paths used in the analysis are short so that H has few layers.

5.3 Combinatorially Shallow Tree Decompositions

In this section, we show how to construct tree decompositions with low combinatorial diameter, thus achieving the approximation results stated in Theorem 5.1. We start by restricting our consideration to decompositions that are shallow in the traditional sense. For a given graph G with treewidth k, we consider a tree decomposition $(\mathcal{T}, \mathcal{B})$ with diameter $d = \mathcal{O}(\log n)$ and width $\mathcal{O}(k)$ [99, 98]. Fix some root r in $V(\mathcal{T})$.

Our goal is now to modify $(\mathcal{T}, \mathcal{B})$ such that every node has a combinatorially short path to r. This is a necessary requirement, but perhaps surprisingly it is not sufficient. The combinatorial lengths of paths do not necessarily induce a metric[3] on $V(\mathcal{T})$, and therefore bounding the length to r does not on its own suffice to bound the combinatorial diameter. Instead, for any pair $s, t \in V(\mathcal{T})$, we will bound the combinatorial length from s and t to their lowest common ancestor, which is sufficient to bound the combinatorial length of $\mathcal{T}_{s \leftrightarrow t}$.

We will not show explicitly that the modified structures are in fact tree decompositions. However, the following lemma can be used to verify this fact for all of our constructions.

Lemma 5.8. *Let $(\mathcal{T}, \{B_i\}_{i \in V(\mathcal{T})})$ be a tree decomposition of a graph G, rooted at r. Then $(\mathcal{T}, \{B'_i\}_{i \in V(\mathcal{T})})$ is also a tree decomposition of G if $B_i \subseteq B'_i \subseteq B_i \cup B'_{p(i)}$ for all bags B_i.*

Proof. Fix some $s \in V(G)$. We need to show that $\mathcal{T}'_s := \{i \in V(\mathcal{T}) \mid s \in B'_i\}$ is connected. As $\mathcal{T}_s := \{i \in V(\mathcal{T}) | s \in B_i\}$ is connected and $\mathcal{T}_s \subseteq \mathcal{T}'_s$, it suffices to show that any $i \in \mathcal{T}'_s$ is connected to $\mathcal{T}_s$ in $\mathcal{T}'_s$. We do this by induction over the distance of i to the root.

For $i = r$ we have $B'_r = B_r$, so either $i \notin \mathcal{T}'_s$ or $i \in \mathcal{T}_s$. Otherwise, consider some $i \in \mathcal{T}'_s$, so $s \in B_i \cup B'_{p(i)}$. Then we either have $s \in B_i$, in

[3]Consider bags $\{ab\}, \{abc\}, \{acd\}, \{ade\}, \{aef\}, \{afg\}, \{a\}$ occuring in that order as a path. The whole path can be reduced to just the endpoints. The subpath $\{ab\}, \{abc\}, \{acd\}, \{ade\}, \{aef\}, \{afg\}$ is irreducible. Thus the distance from $\{ab\}$ to $\{afg\}$ is larger than the sum of the distances from $\{ab\}$ to $\{a\}$ and $\{afg\}$ to $\{a\}$.

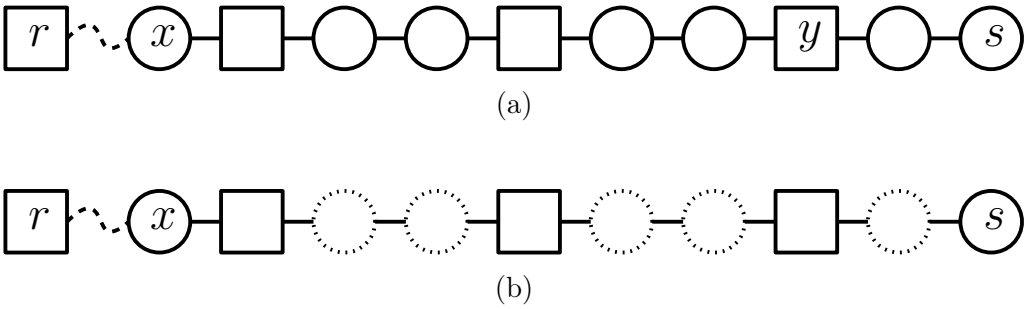


Figure 5.3: Illustration of a path from the root to some node s. The square nodes are the synchronization nodes. The bridge from y to its synchronization ancestor is marked in bold in Figure 5.3a. The dotted nodes in Figure 5.3b mark those nodes which can be removed when simplifying the x-s-path in $\mathcal{T}'$.

which case we are done, or $s \in B'_{p(i)}$. But this gives $p(i) \in \mathcal{T}'_s$, and $p(i)$ is closer to r than i. Thus we can assume that $p(i)$ is connected to $\mathcal{T}_s$, and hence i is also connected to $\mathcal{T}_s$ via $p(i)$. □

We introduce three objects, which we will call *bridges*, *highways*, and *super-highways*, and show that they can be used to prove the three parts of Theorem 5.1.

5.3.1 Bridges

Fix a parameter $\lambda \in \{1, \dots, d\}$. Define $\ell \colon V(\mathcal{T}) \to \mathbb{N}_0$ to be the *level* of a node in $\mathcal{T}$, that is, $\ell(i)$ is the number of edges on $\mathcal{T}_{i \leftrightarrow r}$.

Definition 5.6. We call a node $i \in V(\mathcal{T})$ a *synchronization node* if its level is a multiple of λ. Define also the *synchronization ancestor* $\sigma(v)$ of any node v to be the first node on the path from v to r that is a synchronization node, excluding v itself.

We can construct a tree decomposition $(\mathcal{T}', \mathcal{B}' = \{B'_i\}_{i \in V(\mathcal{T}')})$ by taking $\mathcal{T}' = \mathcal{T}$ and setting $B'_i = B(\mathcal{T}_{i \leftrightarrow \sigma(i)})$ for each node $i \in V(\mathcal{T})$, that is, the new bag is obtained by combining all the bags from v up to its synchronization ancestor. This increases the width of the decomposition by a factor of at most λ. We may view this path connecting v to the synchronization point as a *bridge* crossing over all intermediate nodes in one step.

Lemma 5.9. *It holds that $(\mathcal{T}', \mathcal{B}')$ has combinatorial diameter $\mathcal{O}(d/\lambda)$.*

Proof. Fix any two nodes $s, t \in V(\mathcal{T}')$ and take x to be their lowest common ancestor in $\mathcal{T}'$. Then the combinatorial length of $\mathcal{T}'_{s \leftrightarrow t}$ is at most the sum of the combinatorial lengths of $\mathcal{T}'_{s \leftrightarrow x}$ and $\mathcal{T}'_{x \leftrightarrow t}$.We remark that triangle

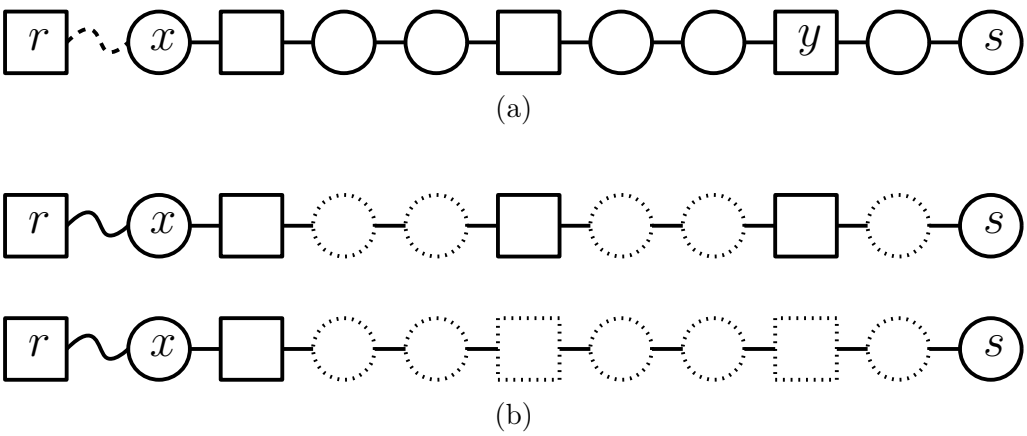


Figure 5.4: The bold nodes in Figure 5.4a mark the bridge and highway from y to r. The dotted nodes in Figure 5.4b illustrate the redundant nodes during the two simplification rounds for the x-s-path, leaving a path of length 2.

inequality holds in this case, because x is on the path from s to t. Thus, it suffices to show that the combinatorial length of $\mathcal{T}'_{s\leftrightarrow x}$ is $\mathcal{O}(d/\lambda)$. The result follows analogously for $\mathcal{T}'_{x\leftrightarrow t}$.

Using the rules of Definition 5.3, we can bypass any node that is neither a synchronization node nor s or x. To do this, iteratively pick any such node v with neighbors u and w, where u is the neighbour closer to s. The order in which these nodes v are bypassed does not matter. Because no synchronization nodes are ever bypassed, u must be somewhere on the subpath from v to the first synchronization node below v. Then we have $\sigma(v) = \sigma(u)$, giving $B'_v \subseteq B'_u$ and in particular $B'_v \cap B'_w \subseteq B'_u$ so that v can be bypassed.

Therefore, the path $\{v \in \mathcal{T}'_{s\leftrightarrow x} | v = s \vee v = x \vee v \text{ is a synchronization node}\}$ is a simplification of $\mathcal{T}'_{s\leftrightarrow x}$. Since there are at most d/λ synchronization nodes on any upward path, the lemma follows. □

This lemma, in conjunction with Theorem 5.2 and the fact that $(\mathcal{T}', \mathcal{B}')$ can be computed in polynomial time from $(\mathcal{T}, \mathcal{B})$, yields:

Corollary 5.2. *For every λ, there is an algorithm that computes an $\mathcal{O}((d/\lambda)^2)$-approximation for* Sparsest Cut *instances where G has treewidth at most k, in time $2^{\mathcal{O}(\lambda k)} \cdot n^{\mathcal{O}(1)}$.*

Setting $\lambda = \lceil d/k \rceil$ results in an $\mathcal{O}(k^2)$-approximation in time $2^k \cdot n^{\mathcal{O}(1)}$, while setting $\lambda = d$ gives an $\mathcal{O}(1)$-approximation in time $n^{\mathcal{O}(k)}$.

5.3.2 Highways

The idea of extending bags towards the root can be exploited further by adding the vertices in a synchronization bag to all of its descendants. We may regard this as giving each node a bridge to the next synchronization node, as well as a *highway* along the synchronization nodes towards the root. This idea leads to the following construction.

Let $(\mathcal{T}', \{B'_i\}_{i \in V(\mathcal{T}')})$ be a modified tree decomposition with $\mathcal{T}' = \mathcal{T}$ as before, and

$$B'_v := B(\{w \in \mathcal{T}_{v \leftrightarrow r} \mid w \in \mathcal{T}_{v \leftrightarrow \sigma(v)} \vee w \text{ is a synchronization node}\}) .$$

for each $v \in V(\mathcal{T})$.

The size of these bags is at most $(k+1)(\lambda + d/\lambda)$, which for $\lambda = d/k$ is in $\mathcal{O}(d + k^2) \subseteq \mathcal{O}(\log n + k^2)$.

Notice that the bag B_r is now contained in any bag B'_i, so we have some hope that the combinatorial diameter of $(\mathcal{T}', \{B'_i\}_{i \in V(\mathcal{T}')})$ is low. Indeed this is true.

Lemma 5.10. *It holds that $\mathcal{T}'$ has combinatorial diameter at most* 3*.*

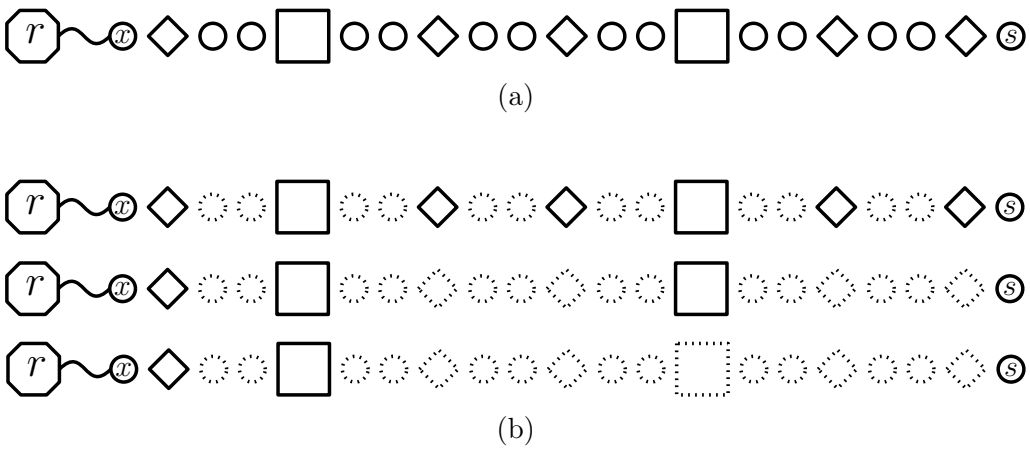
Proof. As before, we split any s-t-path at x, the lowest common ancestor of s and t, and consider only the s-x-path. Every non-synchronization node v on $\mathcal{T}_{s \leftrightarrow x}$ has a node below it which is either a synchronization node or s. The bag of that node is a superset of B'_v, so all non-synchronization nodes except s and x can be bypassed. Call that reduced path P.

Consider the neighbor of s in P, which we denote v, and assume that v is not the neighbor of x in P. Then v must be a synchronization node, and its next node in P is $\sigma(v)$. Now, the intersection $B'_v \cap B'_{\sigma(v)}$ contains exactly all of the bags of synchronization nodes in $\mathcal{T}_{\sigma(v) \leftrightarrow r}$, and thus, $B'_v \cap B'_{\sigma(v)} \subseteq B'_s$. This implies that v can be bypassed, and by repeating this process, we can bypass every synchronization node except for the neighbor of x.

This gives a possible simplification of $\mathcal{T}_{s \leftrightarrow t}$ as the path $(s, \sigma_s, x, \sigma_t, t)$, where the σ_s and σ_t are the synchronization nodes below x on the paths to s and t, respectively. There is a further reduction of the whole path, since B'_x is precisely $B'_{\sigma_s} \cap B'_{\sigma_t}$. This allows us to remove x as well, giving a simplification of length 3. □

Using the fact that $d \in \mathcal{O}(\log n)$, and setting $\lambda = d/k$ gives a fixed-parameter algorithm that yields a constant-factor approximation:

Corollary 5.3. *There exists an algorithm that in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$ computes a factor-$\mathcal{O}(1)$ approximation for* Sparsest Cut *instances where G has treewidth at most k.*



(a)

(b)

Figure 5.5: Illustration of an upward path with nodes of layer -1 as circles, nodes of layer 0 as diamonds, and nodes of layer 1 as squares. The root is at some unspecified maximum layer. The bold nodes inFigure 5.5a mark the super-highway from s to r. Figure 5.5b illustrates the simplification rounds for the x-s-path, removing all nodes of some layer in each round, except x, s, and possibly one node close to x.

5.3.3 Super-Highways

We can think of the previous construction as having two layers, bridges to synchronization nodes and highways along synchronization nodes to the root. The highways need to cover many synchronization nodes, leading to large bags in $\mathcal{T}'$. To improve on this we introduce a network of*super-highways* of different layers, where each layer covers fewer, more spaced-out synchronization nodes on a root-leaf path. When we connect a node to the root we can then move up the tree layer by layer with increasing speed, decreasing the size of bags in $\mathcal{T}'$. Each added layer is paid for by the need for an additional node in path simplifications for moving between layers, giving a trade-off between run time and approximation guarantee.

Let $q \in \mathbb{N}_1$ be a parameter representing the number of layers. For a node $v \in \mathcal{T}$, we define the *layer* $\pi(v)$ of v as

$$\pi(v) := \max\{-1, \max\{j \in \{0, \dots, q-1\} \mid \ell(v) \equiv 0 \mod k^{j/q} d/k\}\}.$$

By this definition, all synchronization nodes are assigned to some layer of non-negative index, and all other nodes are on layer -1. We now get a new tree decomposition $(\mathcal{T}', \{B'_i\}_{i \in V(\mathcal{T}')})$ by constructing bags

$$B'_v = B(\{w \in \mathcal{T}_{p(v) \leftrightarrow r} \mid \pi(w) = \max\{\pi(u) \mid u \in \mathcal{T}_{p(v) \leftrightarrow w}\}\} \cup \{v\})$$

for each $v \in V(\mathcal{T}')$. Informally, we start at some node v and move towards r by first taking all nodes of layer -1 until we hit a node of layer 0, then

taking only nodes of layer 0 until we hit layer 1, and so on. The nodes at higher layers are spaced further apart. Thus this process "speeds up" thereby generating smaller bags. To be precise, there are q layers and at most $k^{1/q}$ nodes of any one layer in a bag, so $(\mathcal{T}', \mathcal{B}')$ has width $\mathcal{O}(d + qk^{1+1/q})$.

We now show that $(\mathcal{T}', \{B'_i\}_{i \in V(\mathcal{T})})$ has combinatorial diameter depending only on q.

Lemma 5.11. *It holds that $(\mathcal{T}', \{B'_i\}_{i \in V(\mathcal{T}')})$ has combinatorial diameter at most $2q + 1$.*

Proof. As before, we only show that any upward path from s to x has combinatorial length at most $q+1$. We need to perform a round of reductions for every layer, with the goal of leaving only s, x, as well as the first node of at least that layer below x. For layer -1, this holds with the same argument as before.

We can now proceed by induction, fixing some layer i and assuming that the s-x-path P has been reduced to contain only s, then nodes of layers $\geq i$, followed by a sequence $(\sigma_{i-1}, \sigma_{i-2}, \ldots, \sigma_0, x)$, where each node σ_j is in layer j. Here, we assume w.l.o.g. that x is at layer -1. Now consider any node v of layer i, except the one closest to x. Because its neighbors also have level at least i (or are s), the intersection of their bags can be represented as the union of bags of $\mathcal{T}$ whose layer is at least i. Let w be the predecessor of v on s-x-path P. The set B'_w is constructed from some upward path starting at w, containing only nodes of non-decreasing layer. This upward path hits layer i between w and v, but not layer $i + 1$ since a node of layer $i + 1$ would be on P between w and v. So then B'_w covers all nodes of layer at least i that B'_v covers, and therefore v can be bypassed, concluding the inductive step.

The simplification of $\mathcal{T}_{s \leftrightarrow x}$ this produces is a path $(s, \sigma_{q-1}, \ldots, \sigma_0, x)$, where $\pi(\sigma_i) = i$. If we add the same simplification for $\mathcal{T}_{t \leftrightarrow x}$ we get a simplification for $\mathcal{T}_{s \leftrightarrow t}$ that takes the form $(s, \sigma_{q-1}, \ldots, \sigma_0, x, \sigma'_0, \ldots, \sigma'_{q-1}, t)$. As before x can be bypassed since its bag is the intersection of the bags of σ_0 and σ'_0. Thus any s-t-path in $\mathcal{T}'$ has combinatorial length at most $2q+1$. □

This implies the existence of the following family of algorithms.

Corollary 5.4. *There exists an algorithm that, for any $q \in \mathbb{N}_1$, computes a $\mathcal{O}(q^2)$-approximation for* Sparsest Cut *on graphs of treewidth k running in time $\mathcal{O}(2^{qk^{1+1/q}}) \cdot n^{\mathcal{O}(1)}$. In particular, taking $q = \log k$ will give us a $\mathcal{O}(\log^2 k)$-approximation in time $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$.*

5.3.4 Recovering the Algorithm of CMV

We will also briefly turn to the results of Cohen-Addad et al. [96] to demonstrate how they may be represented under our view point. Construction of

their paper can be considered as giving tree decompositions of particularly low combinatorial diameter 1.

Theorem 5.3 (adapted from [96])**.** *For every graph G with treewidth k there exists a tree decomposition of width $k2^{\mathcal{O}(k)} + \mathcal{O}(\log n)$ that has combinatorial diameter at most* 1*.*

Proof. Start with a binary tree decomposition $(\mathcal{T}, \mathcal{B})$ of G with width k, set $\lambda = k$, and assign levels and synchronization nodes as in Section 5.3.1. For each synchronization node v we define a new bag B'_v as

$$B'_v := B_v \cup \bigcup_{x \in V(\mathcal{T}), \sigma(x) = v} B_x.$$

That is, we take the content of all bags in the subtree below v, stopping at the next layer of synchronization nodes. These bags B' then have size at most $k2^k$.

Now create a new tree decomposition $\mathcal{T}'$ by taking as $V(\mathcal{T}')$ all synchronisation nodes from $\mathcal{T}$, and use as their bags the corresponding B'. The new decomposition still is rooted at r. Two nodes v, w in $\mathcal{T}'$ are adjacent if and only if $\sigma(v) = w$ or $\sigma(w) = v$. We further expand the bags B' by setting, for each $v \in V(\mathcal{T}')$,

$$B'_v := B'_v \cup \bigcup_{u \in \mathcal{T}'_{v \leftrightarrow r}} B_u.$$

Observe that for some v in $V(\mathcal{T}')$ with parent w, we have $B'_v \cap B'_w = B_v$. So this construction simply adds the content of all bag intersections on the path from v to the root of the tree. Since there are at most $\mathcal{O}(\log n/k)$ such intersections on the path, the bags all have size $\mathcal{O}(k2^k + \log n)$.

As in Section 5.3, we now show that any upward path P from s to x has combinatorial length at most 1, giving $\Delta(\mathcal{T}', \mathcal{B}') \leq 2$. Consider any node $v \in V(P)$, not s or x, with neighbors u and w where u is a descendent of v and w an ancestor. From construction of the B' we get

$$B'_w \cap B'_v \subseteq \bigcup_{x \in \mathcal{T}'_{v \leftrightarrow r}} B_x \subseteq \bigcup_{x \in \mathcal{T}'_{u \leftrightarrow r}} B_x \subseteq B'_u\,.$$

So v can be bypassed, and consequentially the path s, x is a simplification of P. □

The analysis of the approximation error in Lemma 2.4 of [85] can be used to show that SC-Round yields a 2-approximation on tree decomposition of combinatorial diameter 1. With Theorem 5.3 this gives a 2-approximation for Sparsest Cut on graphs of treewidth k in time $2^{k2^{\mathcal{O}(k)}} \cdot n^{\mathcal{O}(1)}$.

5.4 Extension to Graph-Constrained Max-Cut

The entirety of the previous analysis can in fact be extended to capture a much wider class of problems. We can observe that in the analysis of the CKR algorithm the capacity edges only make a very cursory appearance where it is noted that any such edge $\{s, t\}$ is cut with probability *exactly* $\mathsf{lpcut}(s, t)$ (see Lemma 5.1). A particular consequence of this fact is that edges with $\mathsf{lpcut}(s, t) = 1$ are definitely cut, edges with $\mathsf{lpcut}(s, t) = 0$ are definitely not cut.

Since these probabilities are obtained from the variables of some LP we can directly manipulate them, making discrete demands on them. In fact the analysis of Lemma 5.1 does not only extend to the edges of G, but to the distribution of cuts over every bag of the tree decomposition. Because of this we can, informally speaking, enforce any property on the cut that can be checked by only looking at the bags of a tree decomposition individually. For example, the property of being an independent set in some graph G can be checked in such a way, thus this observation would immediately give us a 2-approximation for Max-Cut where one side of the cut is additionally required to be an independent set.

This line of reasoning gives rise to an 2-approximation algorithm of Shen et al. [102] for *Graph-Constrained Max-Cut* (GCMC). In this problem we are given an undirected graph G and costs $c : \binom{V(G)}{2} \to \mathbb{R}_+$, alongside a class $\mathcal{C}_G \subseteq 2^{V(G)}$ of "feasible cuts". The goal is then to compute some cut $C \subseteq \mathcal{C}_G$ such that $\sum_{v \in C, w \notin C} c(v, w)$ is maximal. In this context sparsest cut can be reinterpreted as solving the case where $\mathcal{C}_G$ is the set of all cuts in G that have some fixed capacity, subject to which we then want to maximize the demand we cut.

For these types of problems to make sense $\mathcal{C}_G$ can not just be a list of cuts, otherwise brute force enumeration of $\mathcal{C}_G$ solves the problem in polynomial time. Instead we are interested in the case where $\mathcal{C}_G$ is implicitly defined as some property of C relative to G, for example being an independent set in G. Concretely, Shen et al. give a formalism where $\mathcal{C}_G$ should be described as the accepted solutions of some dynamic program on a tree decomposition of G.[4]

[4]We make some slight changes to their presentation, largely because they treat the treewidth as a true constant. They only desire XP run times, so this is immaterial for them. Since we are interested in achieving
FPT run times here we maintain a more detailed distinction. It is quickly checked that the definition given here agrees with the one given in [102].

So let $(\mathcal{T}, \mathcal{B})$ be a width-k tree decomposition of G with bags $\{B_t\}_t$ where we assume $\mathcal{T}$ to be binary and rooted at r without loss of generality. Then a *Dynamic Program* (DP) on $\mathcal{T}$ is specified as:

- For each node $t \in \mathcal{T}$ a state space Σ_t.
- For each node $t \in \mathcal{T}$ and $\sigma \in \Sigma_t$, a local solution $B_{t,\sigma} \subseteq B_t$.
- For each node $t \in \mathcal{T}$ with children x, y and state $\sigma \in \Sigma_t$ a collection $\mathcal{F}_{t,\sigma} \subseteq \Sigma_x \times \Sigma_y$ of feasible child-state combinations.
- For every pair of feasible child states $(\sigma_x, \sigma_y) \in \mathcal{F}_{t,\sigma_t}$ we are guaranteed consistency of local solutions as $B_{x,\sigma_x} \cap B_t = B_{t,\sigma_t} \cap B_x$ as well as $B_{y,\sigma_y} \cap B_t = B_{t,\sigma_t} \cap B_y$.

The Σ_t here are supposed to represent the entries of the DP-table at a node, corresponding to some local way $B_{t,\sigma}$ of partitioning the bag. The side constraint on this specification is of course that we should be able to resolve the feasibility of such a DP in
FPT time, parameterized on the treewidth. Thus all state spaces should have size $g(k)$ for some computable g, and every state of a bag should correspond uniquely to some way of partitioning that bag. Thus the following additional demand is made:

- There exists a computable function g such that $|\sigma_t| \leq g(k)$ for all $t \in \mathcal{T}$. Notice that this also bounds the size of the $\mathcal{F}_{\sigma,t}$.

A *solution* $\mathcal{S}$ to such a DP is quite simply a choice of state σ_t for each $t \in \mathcal{T}$, such that for a node t with children x, y we have $(\sigma_x, \sigma_y) \subseteq \mathcal{F}_{t,\sigma}$. Such a solution $\mathcal{S}$ then corresponds to a cut C by choosing $C = \bigcup_{\sigma_t \in \mathcal{S}} B_{t,\sigma_t}$ (for brevity we will also call C a solution to the DP. We then mean that there exists a collection of consistent states that would yield this cut.). With this in hand we can then define the class $\mathcal{C}_G$ of feasible cuts for some graph as the set of cuts corresponding to some solution of a fixed DP.

It should be clear that such a DP can be solved naïvely in polynomial time $\mathcal{O}(\max_{t \in \mathcal{T}} |\Sigma_t|^3 \cdot n)$ by computing for each bag a set of possible states that can be extended downwards, starting from the leaves. Here polynomial means polynomial in the size of G and the DP state spaces. The overhead of $|\Sigma_t|^3$ originates from checking the consistency of between parent and child states according to the $\mathcal{F}_{t,\sigma}$.

This specification of a DP over a tree decomposition is probably familiar to most researchers in parameterized algorithms, and programs like it are a common starting point for the study of structurally parameterized algorithms on graphs [5]. This is perhaps not surprising as this formalism is very powerful. Koutecký et al. [103] were able to extend the work of Shen et al.

by proving that one can alternatively assume that $\mathcal{C}_G$ is given by some formula in Monadic Second-Order Logic (MSO) in one free variable since all such $\mathcal{C}_G$ can be captured by a DP as specified. Although the state spaces generated by this reduction are possibly extremely large their size is still some computable function of the treewidth of the original graph as per the famous work of Bruno Courcelle [104] showing (among other things) that every property definable in MSO can be tested in
FPT time on graphs of bounded treewidth, parameterized against the treewidth and the length of the MSO formula. Thus through this line of work one gets 2-approximation algorithm in XP time for GCMC for every constraint that is expressible in MSO [102, 103].

For our purposes we will not engage with the MSO side of this analysis, since it is anyways captured by the dynamic program. However it is useful to keep this mind to understand the expressive power of such DPs. We now show that we can use our techniques to capture the dynamic program itself and thus can obtain our variety of trade-offs between run time and approximation quality. Most of these are not particularly relevant since the run time cost of embedding the DP is so high as to mostly dominate the parameterized component of the run time in any event. However employing the construction of Cohen-Addad et al. [96] we will be able to obtain a 2-approximation in
FPT time, thus answering the open question remaining from Koutecky et al. [103] of whether this is possible.

Theorem 5.4. *For any formula φ in MSO there is a $1/2$-approximation algorithm for GCMC running in time $g(k, \varphi) \cdot \mathrm{poly}(n)$, where the constraint is specified by φ, k is the treewidth of the input graph, and g is some universal computable function.*

Theorem 5.5. *For any graph constraint specified by a fixed dynamic program there exists a $1/2$-approximation algorithm for GCMC running in time $g(k, \max |\Sigma_t|) \cdot \mathrm{poly}(n)$, where k is the treewidth of the input graph and g is some universal computable function.*

Our argument is basically that such a DP can also be solved "in-line" when computing the distributions used by the CKR algorithm, meaning that we can make the additional demand that rounding these distributions always returns a solution to the DP. We will do this by embedding the DP states directly into the tree decomposition itself, at the cost of increasing the width considerably. This is largely for ease of presentation, specific DPs could be represented much more concisely. However since Theorem 5.4 will anyways need DPs essentially derived from Courcelle's theorem [104] which are inherently expensive to solve we will forego a detailed analysis of the run time dependence vis-a-vis the treewidth and content ourselves with obtaining FPT time.

So let $(G, c, \mathcal{C}_G)$ be some instance of GCMC where $\mathcal{C}_G$ is specified by some DP and a width-k tree decomposition $\mathcal{T}$ of G is given. We assume $\mathcal{T}$ to be binary, to have depth $\mathcal{O}(n)$, and to be rooted at r. We now transfer the DP to the tree decomposition in the following way: For every node $t \in \mathcal{T}$ and for every state $\sigma \in \Sigma_t$ we create a new vertex of v_σ in G and add it to B_t, as well as to the parent bag $B_{\pi(t)}$. Call the resulting tree decomposition $(\mathcal{T}', \mathcal{B}')$ and its bags B'_t, where $B'_t = B_t \cup \Sigma_t \cup \Sigma_x \cup \Sigma_y$ for a node t with children x and y. This new tree decomposition has width $g(k)$ for some computable function g depending only on the DP.

We now compute the distributions necessary for the CKR algorithm on this new tree decomposition under the following additional constraint: Let f be a B'_t-assignment. We say that such an assignment is *feasible* with respect to the DP if f assigns 1 to exactly one state $\sigma_t \in \Sigma_t$ as well as to two states σ_x, σ_y for the children of t. Further these states must fulfill $(\sigma_x, \sigma_y) \in \mathcal{F}_{t,\sigma_t}$. Finally the assignment must choose the correct local solution in B_t, so $f^{-1}(1) \cap B_t = B_{t,\sigma_t}$.

This now allows us to collect the set of infeasible assignments $\mathcal{F}_{inf}$ and compute a collection of distributions $\{\mu_L\}$ for the CKR algorithm fulfilling

$$\Pr_{f' \sim \mu}[f' = f] = 0 \ \forall f \in \mathcal{F}_{inf}. \tag{5.1}$$

Since the distributions are computed from some Sherali-Adams LP in which these probabilities are concrete variables this is easily enforced. It is then a direct consequence of Lemma 5.1 that any assignment rounded by the CKR will assign to every bag of the tree decomposition exactly one of its states, that states are consistent between parent and child nodes, and that the assignment to the vertices of a bag agrees with that foreseen by the associated state. Thus any assignment rounded by the algorithm is a solution to the DP.

It is also clear that the analysis of the rounding error on the edges not contained in some bag is untouched by this modification since it does not actually modify anything, so we still get an approximation guarantee of $\Delta(\mathcal{T}')^2$, with a special case of $\Delta(\mathcal{T}', \mathcal{B}') = 1$ where we would get a 2-approximation.

However adding the states to all bags will definitely increase the combinatorial diameter to be the "normal" diameter. Hence we may only apply the restructuring arguments from Section 5.3 after the fact. This is not prohibitive with respect to embedding the DP, as every bag of the original tree decomposition will still be fully contained in some bag of the restructured decomposition. So property 5.1 can still be maintained by direct manipulation of the variable of the LP used for computing μ.

Employing specifically the construction of Cohen-Addad et al. [96] (Theorem 5.3) to restructure the tree decomposition we thus get the following process:

1. Start with a graph G with a binary tree decomposition $\mathcal{T}$ of depth $\mathcal{O}(\log n)$.
2. Add to every bag the associated DP states of the node and its children. This increases the width of the decomposition to $w(\mathcal{T}) + 3\max|\Sigma_t|$.
3. Reconfigure $\mathcal{T}$ as in Theorem 5.3, obtaining a new tree decomposition of width at most $\mathcal{O}((w(\mathcal{T}) + \max|\Sigma_t|)2^{\mathcal{O}(w(\mathcal{T})+\max|\Sigma_t|)} + \log n)$.
4. Compute distributions $\{\mu_L\}$ as for the CKR algorithm, with the additional property 5.1 enforced.
5. Perform the rounding of the CKR algorithm.

This procedure provides the guarantees promised in Theorem 5.5 and thus through the argumentation of Koutecky et al. [103] also those of Theorem 5.4.

5.5 Conclusion & Open Problems

Our work is an attempt to simultaneously obtain the best run time and approximation factor for Sparsest Cut in the low-treewidth regime. We introduce a new measure of tree decomposition called combinatorial diameter and show various constructions with different tradeoffs between $w(\mathcal{T}, \mathcal{B})$ and $\Delta(\mathcal{T}, \mathcal{B})$. We leave the question of getting 2-approximation in $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time as the main open problem. One way to design such an algorithm would be to show the existence of a tree decomposition with $w(\mathcal{T}, \mathcal{B}) = \mathcal{O}(\log n + k)$ and $\Delta(\mathcal{T}, \mathcal{B}) = 1$.

Another interesting question is to focus on polynomial-time algorithms and optimize the approximation factor with respect to treewidth. In particular, is there an $\log^{\mathcal{O}(1)} k$ approximation in polynomial time? This question is open even for the *uniform* Sparsest Cut (unit demand for every vertex pair), for which a fixed-parameter algorithm [105] but no polynomial-time algorithm is known.

A broader direction would be to improve our understanding of LP-rounding algorithms on the lift-and-project convex programs in general. For instance, can we prove a similar trade-off result for other combinatorial optimization problems in this setting? A good candidate is the Group Steiner Tree problem, for which a factor-$\mathcal{O}(\log^2 n)$ approximation in time $n^{\mathcal{O}(k)}$ is known (and the algorithm there is "the same" algorithm as used for finding sparsest cuts). Is it possible here to improve to a factor-$\mathcal{O}(\log^2 n)$ approximation in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$? In particular the question of interest here is to what extent we can extend the analysis of the probability of cutting an *edge* of the demand graph to larger structures such as the terminal sets of the Group Steiner Tree problem.

Bibliography

[1] Zhou Xu and Brian Rodrigues. "A 3/2-Approximation Algorithm for the Multiple TSP with a Fixed Number of Depots". In: *INFORMS Journal on Computing* 27.4 (Nov. 2015), pp. 636–645. DOI: 10.1287/ijoc.2015.0650.

[2] Eden Chlamtac, Robert Krauthgamer, and Prasad Raghavendra. *Approximating Sparsest Cut in Graphs of Bounded Treewidth*. June 23, 2010. arXiv: 1006.3970 [cs]. Pre-published.

[3] Stephen A. Cook. "The Complexity of Theorem-Proving Procedures". In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. STOC '71. New York, NY, USA: Association for Computing Machinery, May 3, 1971, pp. 151–158. DOI: 10.1145/800157.805047.

[4] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Proceedings of a Symposium on the Complexity of Computer Computations*. Complexity of Computer Computations. Ed. by Raymond E. Miller. The IBM Research Symposia Series. Boston, MA: Springer US, 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

[5] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Cham: Springer International Publishing, 2015. DOI: 10.1007/978-3-319-21275-3.

[6] Vijay V. Vazirani. *Approximation Algorithms*. Berlin, Heidelberg: Springer, 2003. DOI: 10.1007/978-3-662-04565-7.

[7] Dániel Marx. "Parameterized Complexity and Approximation Algorithms". In: *The Computer Journal* 51.1 (Jan. 1, 2008), pp. 60–78. DOI: 10.1093/comjnl/bxm048.

[8] Andreas Emil Feldmann, Karthik C. S, Euiwoong Lee, and Pasin Manurangsi. "A Survey on Approximation in Parameterized Complexity: Hardness and Algorithms". In: *Algorithms* 13.6 (6 June 2020), p. 146. DOI: 10.3390/a13060146.

[9] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. "PRIMES Is in P". In: *Annals of Mathematics. Second Series* 160.2 (2004), pp. 781–793. DOI: 10.4007/annals.2004.160.781.

[10] Rod G. Downey and Michael R. Fellows. "Fixed-Parameter Tractability and Completeness I: Basic Results". In: *SIAM Journal on Computing* 24.4 (Aug. 1995), pp. 873–921. DOI: 10.1137/S0097539792228228.

[11] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. "Which Problems Have Strongly Exponential Complexity?" In: *Journal of Computer and System Sciences* 63.4 (Dec. 1, 2001), pp. 512–530. DOI: 10.1006/jcss.2001.1774.

[12] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. "Strong Computational Lower Bounds via Parameterized Complexity". In: *Journal of Computer and System Sciences* 72.8 (Dec. 1, 2006), pp. 1346–1367. DOI: 10.1016/j.jcss.2006.04.007.

[13] Jörg Flume and Martin Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Berlin, Heidelberg: Springer, 2006. DOI: 10.1007/3-540-29953-X.

[14] Libor Barto, Jakub Bulín, Andrei Krokhin, and Jakub Opršal. "Algebraic Approach to Promise Constraint Satisfaction". In: *Journal of the ACM* 68.4 (July 14, 2021), 28:1–28:66. DOI: 10.1145/3457606.

[15] Per Austrin, Venkatesan Guruswami, and Johan Håstad. "$(2 + \varepsilon)$-Sat Is NP-hard". In: *SIAM Journal on Computing* 46.5 (Jan. 2017), pp. 1554–1573. DOI: 10.1137/15M1006507.

[16] Andrei Krokhin and Jakub Opršal. "An Invitation to the Promise Constraint Satisfaction Problem". In: *ACM SIGLOG News* 9.3 (July 2022), pp. 30–59. DOI: 10.1145/3559736.3559740. arXiv: 2208.13538 [cs, math].

[17] Antoine Mottet. "Promise and Infinite-Domain Constraint Satisfaction". In: 32nd EACSL Annual Conference on Computer Science Logic (CSL 2024). Schloss-Dagstuhl - Leibniz Zentrum für Informatik, 2024. DOI: 10.4230/LIPIcs.CSL.2024.41.

[18] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. "Proof Verification and the Hardness of Approximation Problems". In: *Journal of the ACM* 45.3 (May 1, 1998), pp. 501–555. DOI: 10.1145/278298.278306.

[19] Irit Dinur. "The PCP Theorem by Gap Amplification". In: *Journal of the ACM* 54.3 (June 1, 2007), 12–es. DOI: 10.1145/1236457.1236459.

[20] Marek Karpinski, Michael Lampis, and Richard Schmied. "New Inapproximability Bounds for TSP". In: *Journal of Computer and System Sciences* 81.8 (Dec. 1, 2015), pp. 1665–1677. DOI: 10.1016/j.jcss.2015.06.003.

[21] Nicos Christofides. "Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem". In: *Operations Research Forum* 3.1 (Mar. 3, 2022), p. 20. DOI: 10.1007/s43069-021-00101-z.

[22] Анатолий И. Сердюков. “О Некоторых Экстремальных Обходах в Графах”. In: *Управляемые Системы* 17 (1978).

[23] Jack Edmonds. “Maximum Matching and a Polyhedron with 0,1-Vertices”. In: *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics* 69B (1 and 2 Jan. 1965), p. 125. DOI: 10.6028/jres.069B.013.

[24] Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. “A (Slightly) Improved Approximation Algorithm for Metric TSP”. In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2021. New York, NY, USA: Association for Computing Machinery, June 15, 2021, pp. 32–45. DOI: 10.1145/3406325.3451009.

[25] Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. “A Deterministic Better-than-3/2 Approximation Algorithm for Metric TSP”. In: *Integer Programming and Combinatorial Optimization*. Ed. by Alberto Del Pia and Volker Kaibel. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2023, pp. 261–274. DOI: 10.1007/978-3-031-32726-1_19.

[26] Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. “An Improved Approximation Algorithm for TSP in the Half Integral Case”. In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2020. New York, NY, USA: Association for Computing Machinery, June 22, 2020, pp. 28–39. DOI: 10.1145/3357713.3384273.

[27] Anupam Gupta, Euiwoong Lee, Jason Li, Marcin Mucha, Heather Newman, and Sherry Sarkar. “Matroid-Based TSP Rounding for Half-Integral Solutions”. In: *Integer Programming and Combinatorial Optimization*. Ed. by Karen Aardal and Laura Sanità. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 305–318. DOI: 10.1007/978-3-031-06901-7_23.

[28] Billy Jin, Nathan Klein, and David P. Williamson. “A 4/3-Approximation Algorithm for Half-Integral Cycle Cut Instances of the TSP”. In: *Integer Programming and Combinatorial Optimization*. Ed. by Alberto Del Pia and Volker Kaibel. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2023, pp. 217–230. DOI: 10.1007/978-3-031-32726-1_16.

[29] A. M. Frieze. “An Extension of Christofides Heuristic to the K-Person Travelling Salesman Problem”. In: *Discrete Applied Mathematics* 6.1 (May 1, 1983), pp. 79–83. DOI: 10.1016/0166-218X(83)90102-6.

[30] Tolga Bektaş. "The Multiple Traveling Salesman Problem: An Overview of Formulations and Solution Procedures". In: *Omega* 34.3 (June 1, 2006), pp. 209–219. DOI: 10.1016/j.omega.2004.10.004.

[31] Rico Zenklusen. "A 1.5-Approximation for Path TSP". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '19. USA: Society for Industrial and Applied Mathematics, Jan. 6, 2019, pp. 1539–1549. DOI: 10.1137/1.9781611975482.93.

[32] Sanjeev Arora. "Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and Other Geometric Problems". In: *Journal of the ACM* 45.5 (Sept. 1, 1998), pp. 753–782. DOI: 10.1145/290179.290180.

[33] András Sebő and Jens Vygen. "Shorter Tours by Nicer Ears: 7/5-Approximation for the Graph-TSP, 3/2 for the Path Version, and 4/3 for Two-Edge-Connected Subgraphs". In: *Combinatorica* 34.5 (Oct. 1, 2014), pp. 597–629. DOI: 10.1007/s00493-014-2960-3.

[34] Max Deppert, Matthias Kaul, and Matthias Mnich. "A $(3/2+\varepsilon)$-Approximation for Multiple TSP with a Variable Number of Depots". In: 31st Annual European Symposium on Algorithms (ESA 2023). Schloss-Dagstuhl - Leibniz Zentrum für Informatik, 2023. DOI: 10.4230/LIPIcs.ESA.2023.39.

[35] Vera Traub, Jens Vygen, and Rico Zenklusen. "Reducing Path TSP to TSP". In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2020. New York, NY, USA: Association for Computing Machinery, June 22, 2020, pp. 14–27. DOI: 10.1145/3357713.3384256.

[36] Sivakumar Rathinam, Raja Sengupta, and Swaroop Darbha. "A Resource Allocation Algorithm for Multivehicle Systems With Nonholonomic Constraints". In: *IEEE Transactions on Automation Science and Engineering* 4.1 (Jan. 2007), pp. 98–104. DOI: 10.1109/TASE.2006.872110.

[37] Waqar Malik, Sivakumar Rathinam, and Swaroop Darbha. "An Approximation Algorithm for a Symmetric Generalized Multiple Depot, Multiple Travelling Salesman Problem". In: *Operations Research Letters* 35.6 (Nov. 1, 2007), pp. 747–753. DOI: 10.1016/j.orl.2007.02.001.

[38] Gregory Gutin, Magnus Wahlström, and Anders Yeo. "Rural Postman Parameterized by the Number of Components of Required Edges". In: *Journal of Computer and System Sciences* 83.1 (Feb. 1, 2017), pp. 121–131. DOI: 10.1016/j.jcss.2016.06.001.

[39] René van Bevern, Till Fluschnik, and Oxana Yu. Tsidulko. "On Approximate Data Reduction for the Rural Postman Problem: Theory and Experiments". In: *Networks* 76.4 (2020), pp. 485–508. DOI: 10.1002/net.21985.

[40] Tobias Moemke and Ola Svensson. "Approximating Graphic TSP by Matchings". In: *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science. Oct. 2011, pp. 560–569. DOI: 10.1109/FOCS.2011.56.

[41] Bernhard Korte and Jens Vygen. "Matroids". In: *Combinatorial Optimization: Theory and Algorithms*. Ed. by Bernhard Korte and Jens Vygen. Algorithms and Combinatorics. Berlin, Heidelberg: Springer, 2018, pp. 325–357. DOI: 10.1007/978-3-662-56039-6_13.

[42] Greg N. Frederickson. "Approximation Algorithms for Some Postman Problems". In: *Journal of the ACM* 26.3 (July 1, 1979), pp. 538–554. DOI: 10.1145/322139.322150.

[43] Klaus Jansen. "An Approximation Algorithm for the General Routing Problem". In: *Information Processing Letters* 41.6 (Apr. 17, 1992), pp. 333–339. DOI: 10.1016/0020-0190(92)90161-N.

[44] J. T. Schwartz. "Fast Probabilistic Algorithms for Verification of Polynomial Identities". In: *Journal of the ACM* 27.4 (Oct. 1, 1980), pp. 701–717. DOI: 10.1145/322217.322225.

[45] Richard Zippel. "Probabilistic Algorithms for Sparse Polynomials". In: *Symbolic and Algebraic Computation*. Ed. by Edward W. Ng. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1979, pp. 216–226. DOI: 10.1007/3-540-09519-5_73.

[46] Matthias Kaul, Kelin Luo, Matthias Mnich, and Heiko Röglin. "Approximate Minimum Tree Cover in All Symmetric Monotone Norms Simultaneously". In: *42nd International Symposium on Theoretical Aspects of Computer Science, STACS 2025, March 4-7, 2025, Jena, Germany*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025, 57:1–57:18. DOI: 10.4230/LIPICS.STACS.2025.57.

[47] T.K. Ralphs, L. Kopman, W.R. Pulleyblank, and L.E. Trotter. "On the Capacitated Vehicle Routing Problem". In: *Mathematical Programming* 94.2 (Jan. 1, 2003), pp. 343–359. DOI: 10.1007/s10107-002-0323-0.

[48] Fabrizio Grandoni, Claire Mathieu, and Hang Zhou. "Unsplittable Euclidean Capacitated Vehicle Routing: A $(2 + \varepsilon)$-Approximation Algorithm". In: 14th Innovations in Theoretical Computer Science Conference (ITCS 2023). Schloss-Dagstuhl - Leibniz Zentrum für Informatik, 2023. DOI: 10.4230/LIPIcs.ITCS.2023.63.

[49] Jannis Blauth, Vera Traub, and Jens Vygen. "Improving the Approximation Ratio for Capacitated Vehicle Routing". In: *Mathematical Programming* 197.2 (Feb. 1, 2023), pp. 451–497. DOI: 10.1007/s10107-022-01841-4.

[50] Wei Yu and Zhaohui Liu. "Improved Approximation Algorithms for Some Min-Max and Minimum Cycle Cover Problems". In: *Theoretical Computer Science*. Computing and Combinatorics 654 (Nov. 22, 2016), pp. 45–58. DOI: 10.1016/j.tcs.2016.01.041.

[51] G. Even, N. Garg, J. Könemann, R. Ravi, and A. Sinha. "Min–Max Tree Covers of Graphs". In: *Operations Research Letters* 32.4 (July 1, 2004), pp. 309–315. DOI: 10.1016/j.orl.2003.11.010.

[52] M. Reza Khani and Mohammad R. Salavatipour. "Improved Approximation Algorithms for the Min-Max Tree Cover and Bounded Tree Cover Problems". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Ed. by Leslie Ann Goldberg, Klaus Jansen, R. Ravi, and José D. P. Rolim. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 302–314. DOI: 10.1007/978-3-642-22935-0_26.

[53] Esther M. Arkin, Refael Hassin, and Asaf Levin. "Approximations for Minimum and Min-Max Vehicle Routing Problems". In: *Journal of Algorithms* 59.1 (Apr. 1, 2006), pp. 1–18. DOI: 10.1016/j.jalgor.2005.01.007.

[54] Yossi Azar, Leah Epstein, Yossi Richter, and Gerhard J Woeginger. "All-Norm Approximation Algorithms". In: *Journal of Algorithms* 52.2 (Aug. 1, 2004), pp. 120–133. DOI: 10.1016/j.jalgor.2004.02.003.

[55] Daniel Golovin, Anupam Gupta, Amit Kumar, and Kanat Tangwongsan. "All-Norms and All-L_p-Norms Approximation Algorithms". In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (2008). Schloss-Dagstuhl - Leibniz Zentrum für Informatik, 2008. DOI: 10.4230/LIPIcs.FSTTCS.2008.1753.

[56] V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. "A Unified Approach to Scheduling on Unrelated Parallel Machines". In: *Journal of the ACM* 56.5 (Aug. 21, 2009), 28:1–28:31. DOI: 10.1145/1552285.1552289.

[57] Deeparnab Chakrabarty and Chaitanya Swamy. "Approximation Algorithms for Minimum Norm and Ordered Optimization Problems". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2019. New York, NY, USA: Association for Computing Machinery, June 23, 2019, pp. 126–137. DOI: 10.1145/3313276.3316322.

[58] Amit Kumar and Jon Kleinberg. "Fairness Measures for Resource Allocation". In: *SIAM Journal on Computing* 36.3 (Jan. 2006), pp. 657–680. DOI: 10.1137/S0097539703434966.

[59] Eden Chlamtáč, Yury Makarychev, and Ali Vakilian. "Approximating Fair Clustering with Cascaded Norm Objectives". In: *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Proceedings. Society for Industrial and Applied Mathematics, Jan. 2022, pp. 2664–2683. DOI: 10.1137/1.9781611977073.104.

[60] Swati Gupta, Jai Moondra, and Mohit Singh. *Which L_p norm is the fairest? Approximations for fair facility location across all "p"*. May 19, 2023. DOI: 10.48550/arXiv.2211.14873. arXiv: 2211.14873 [cs]. Pre-published.

[61] Zhou Xu and Qi Wen. "Approximation Hardness of Min–Max Tree Covers". In: *Operations Research Letters* 38.3 (May 1, 2010), pp. 169–173. DOI: 10.1016/j.orl.2010.02.004.

[62] Otakar Borůvka. "Über ein Minimalproblem." In: *Práce moravské přírodovědecké společnosti* 3 (1926), pp. 37–58.

[63] Joseph B. Kruskal. "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem". In: *Proceedings of the American Mathematical Society* 7.1 (1956), pp. 48–50. DOI: 10.1090/S0002-9939-1956-0078686-7.

[64] L. Kou, G. Markowsky, and L. Berman. "A Fast Algorithm for Steiner Trees". In: *Acta Informatica* 15.2 (1981), pp. 141–145. DOI: 10.1007/BF00288961.

[65] Noga Alon, Yossi Azar, Gerhard J. Woeginger, and Tal Yadid. "Approximation Schemes for Scheduling". In: *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '97. USA: Society for Industrial and Applied Mathematics, Jan. 5, 1997, pp. 493–500.

[66] Ashish Goel and Adam Meyerson. "Simultaneous Optimization via Approximate Majorization for Concave Profits or Convex Costs". In: *Algorithmica* 44.4 (Apr. 1, 2006), pp. 301–323. DOI: 10.1007/s00453-005-1177-7.

[67] Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. "Bin Packing with Fixed Number of Bins Revisited". In: *Journal of Computer and System Sciences* 79.1 (Feb. 1, 2013), pp. 39–49. DOI: 10.1016/j.jcss.2012.04.004.

[68] Johan Håstad. "Some Optimal Inapproximability Results". In: *Journal of the ACM* 48.4 (July 1, 2001), pp. 798–859. DOI: 10.1145/502090.502098.

[69] Piotr Berman, Marek Karpinski, and Alexander D. Scott. "Computational Complexity of Some Restricted Instances of 3-SAT". In: *Discrete Applied Mathematics* 155.5 (Mar. 15, 2007), pp. 649–653. DOI: 10.1016/j.dam.2006.07.009.

[70] Parinya Chalermsook, Matthias Kaul, Matthias Mnich, Joachim Spoerhase, Sumedha Uniyal, and Daniel Vaz. "Approximating Sparsest Cut in Low-treewidth Graphs via Combinatorial Diameter". In: *ACM Transactions on Algorithms* 20.1 (Jan. 22, 2024), 6:1–6:20. DOI: 10.1145/3632623.

[71] L. R. Ford and D. R. Fulkerson. "Maximal Flow Through a Network". In: *Canadian Journal of Mathematics* 8 (Jan. 1956), pp. 399–404. DOI: 10.4153/CJM-1956-045-5.

[72] David B. Shmoys. "Cut Problems and Their Application to Divide-and-Conquer". In: *Approximation Algorithms for NP-hard Problems*. USA: PWS Publishing Co., Aug. 1, 1996, pp. 192–235.

[73] David W. Matula and Farhad Shahrokhi. "Sparsest Cuts and Bottlenecks in Graphs". In: *Discrete Applied Mathematics* 27.1 (May 1, 1990), pp. 113–123. DOI: 10.1016/0166-218X(90)90133-W.

[74] Nathan Linial, Eran London, and Yuri Rabinovich. "The Geometry of Graphs and Some of Its Algorithmic Applications". In: *Combinatorica* 15.2 (June 1, 1995), pp. 215–245. DOI: 10.1007/BF01200757.

[75] Yonatan Aumann and Yuval Rabani. "An O(Log k) Approximate Min-Cut Max-Flow Theorem and Approximation Algorithm". In: *SIAM Journal on Computing* 27.1 (Jan. 1998), pp. 291–301. DOI: 10.1137/S0097539794285983.

[76] T. Leighton and S. Rao. "An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems with Applications to Approximation Algorithms". In: *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*. 29th Annual Symposium on Foundations of Computer Science. Oct. 1988, pp. 422–431. DOI: 10.1109/SFCS.1988.21958.

[77] J. Bourgain. "On Lipschitz Embedding of Finite Metric Spaces in Hilbert Space". In: *Israel Journal of Mathematics* 52.1 (Mar. 1, 1985), pp. 46–52. DOI: 10.1007/BF02776078.

[78] Sanjeev Arora, James Lee, and Assaf Naor. "Euclidean Distortion and the Sparsest Cut". In: *Journal of the American Mathematical Society* 21.1 (2008), pp. 1–21. DOI: 10.1090/S0894-0347-07-00573-5.

[79] Sanjeev Arora, Satish Rao, and Umesh Vazirani. "Expander Flows, Geometric Embeddings and Graph Partitioning". In: *Journal of the ACM* 56.2 (Apr. 17, 2009), 5:1–5:37. DOI: 10.1145/1502793.1502794.

[80] Subhash Khot. "On the Power of Unique 2-Prover 1-Round Games". In: *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*. STOC '02. New York, NY, USA: Association for Computing Machinery, May 19, 2002, pp. 767–775. DOI: 10.1145/509907.510017.

[81] S. Chawla, R. Krauthgamer, R. Kumar, Y. Rabani, and D. Sivakumar. "On the Hardness of Approximating MULTICUT and SPARSEST-CUT". In: *20th Annual IEEE Conference on Computational Complexity (CCC'05)*. June 2005, pp. 144–153. DOI: 10.1109/CCC.2005.20.

[82] Subhash A. Khot and Nisheeth K. Vishnoi. "The Unique Games Conjecture, Integrality Gap for Cut Problems and Embeddability of Negative-Type Metrics into L1". In: *Journal of the ACM* 62.1 (Mar. 2, 2015), 8:1–8:39. DOI: 10.1145/2629614.

[83] Nikhil R. Devanur, Subhash A. Khot, Rishi Saket, and Nisheeth K. Vishnoi. "Integrality Gaps for Sparsest Cut and Minimum Linear Arrangement Problems". In: *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '06. New York, NY, USA: Association for Computing Machinery, May 21, 2006, pp. 537–546. DOI: 10.1145/1132516.1132594.

[84] Anupam Gupta and Amitabh Basu. *Lecture 19: Sparsest Cut and L1 Embeddings*.

[85] Anupam Gupta, Kunal Talwar, and David Witmer. "Sparsest Cut on Bounded Treewidth Graphs: Algorithms and Hardness Results". May 6, 2013. arXiv: 1305.1347 [cs].

[86] Chandra Chekuri, Anupam Gupta, Ilan Newman, Yuri Rabinovich, and Alistair Sinclair. "Embedding K-Outerplanar Graphs into L1". In: *SIAM Journal on Discrete Mathematics* 20.1 (Jan. 1, 2006), pp. 119–136. DOI: 10.1137/S0895480102417379.

[87] James R. Lee and Anastasios Sidiropoulos. "Pathwidth, Trees, and Random Embeddings". In: *Combinatorica* 33.3 (June 1, 2013), pp. 349–374. DOI: 10.1007/s00493-013-2685-8.

[88] Vincent Cohen-Addad, Anupam Gupta, Philip N. Klein, and Jason Li. "A quasipolynomial $(2+\varepsilon)$-approximation for planar sparsest cut". In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2021. New York, NY, USA: Association for Computing Machinery, June 15, 2021, pp. 1056–1069. DOI: 10.1145/3406325.3451103.

[89] A. Gupta, I. Newman, Y. Rabinovich, and A. Sinclair. "Cuts, trees and $l_1$embeddings of graphs". In: *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*. 40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039). Oct. 1999, pp. 399–408. DOI: 10.1109/SFFCS.1999.814611.

[90] Monique Laurent. "A Comparison of the Sherali-Adams, Lovász-Schrijver, and Lasserre Relaxations for 0-1 Programming". In: *Mathematics of Operations Research* 28.3 (2003), pp. 470–496. JSTOR: 4126981.

[91] Avner Magen and Mohammad Moharrami. "Robust Algorithms for on Minor-Free Graphs Based on the Sherali-Adams Hierarchy". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Ed. by Irit Dinur, Klaus Jansen, Joseph Naor, and José Rolim. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 258–271. DOI: 10.1007/978-3-642-03685-9_20.

[92] Daniel Bienstock and Nuri Ozbay. "Tree-Width and the Sherali-Adams Operator". In: *Discrete Optimization* 1.1 (June 1, 2004), pp. 13–21. DOI: 10.1016/j.disopt.2004.03.002.

[93] Parinya Chalermsook, Syamantak Das, Bundit Laekhanukit, and Daniel Vaz. "Beyond Metric Embedding: Approximating Group Steiner Trees on Bounded Treewidth Graphs". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '17. USA: Society for Industrial and Applied Mathematics, Jan. 16, 2017, pp. 737–751.

[94] Parinya Chalermsook, Syamantak Das, Guy Even, Bundit Laekhanukit, and Daniel Vaz. "Survivable Network Design for Group Connectivity in Low-Treewidth Graphs". In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018). 2018. DOI: 10.4230/LIPIcs.APPROX-RANDOM.2018.8.

[95] Naveen Garg, Goran Konjevod, and R. Ravi. "A Polylogarithmic Approximation Algorithm for the Group Steiner Tree Problem". In: *Journal of Algorithms* 37.1 (Oct. 1, 2000), pp. 66–84. DOI: 10.1006/jagm.2000.1096.

[96] Vincent Cohen-Addad, Tobias Mömke, and Victor Verdugo. "A 2-approximation for the bounded treewidth sparsest cut problem in FPT Time". In: *Mathematical Programming* (Jan. 4, 2024). DOI: 10.1007/s10107-023-02044-1.

[97] Eden Chlamtac, Robert Krauthgamer, and Prasad Raghavendra. "Approximating Sparsest Cut in Graphs of Bounded Treewidth". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Ed. by Maria Serna, Ronen Shaltiel, Klaus Jansen, and José Rolim. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 124–137. DOI: 10.1007/978-3-642-15369-3_10.

[98] Tuukka Korhonen. "A Single-Exponential Time 2-Approximation Algorithm for Treewidth". In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. 2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS). Feb. 2022, pp. 184–192. DOI: 10.1109/FOCS52979.2021.00026.

[99] Hans L. Bodlaender. "NC-algorithms for Graphs with Small Treewidth". In: *Graph-Theoretic Concepts in Computer Science*. Ed. by J. Leeuwen. Red. by G. Goos, J. Hartmanis, D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmüller, J. Stoer, and N. Wirth. Vol. 344. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 1–10. DOI: 10.1007/3-540-50728-0_32.

[100] Anupam Gupta and Jochen Könemann. "Approximation Algorithms for Network Design: A Survey". In: *Surveys in Operations Research and Management Science* 16.1 (Jan. 1, 2011), pp. 3–20. DOI: 10.1016/j.sorms.2010.06.001.

[101] Hanif D. Sherali and Warren P. Adams. "A Hierarchy of Relaxation between the Continuous and Convex Hull Representations". In: *SIAM Journal on Discrete Mathematics* 3.3 (May 1, 1990), pp. 411–430. DOI: 10.1137/0403036.

[102] Xiangkun Shen, Jon Lee, and Viswanath Nagarajan. "Approximating Graph-Constrained Max-Cut". In: *Mathematical Programming* 172.1 (Nov. 1, 2018), pp. 35–58. DOI: 10.1007/s10107-017-1154-3.

[103] Martin Koutecký, Jon Lee, Viswanath Nagarajan, and Xiangkun Shen. "Approximating Max-Cut under Graph-MSO Constraints". In: *Operations Research Letters* 46.6 (Nov. 1, 2018), pp. 592–598. DOI: 10.1016/j.orl.2018.10.004.

[104] Bruno Courcelle. "The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs". In: *Information and Computation* 85.1 (Mar. 1, 1990), pp. 12–75. DOI: 10.1016/0890-5401(90)90043-H.

[105] Paul Bonsma, Hajo Broersma, Viresh Patel, and Artem Pyatkin. "The Complexity of Finding Uniform Sparsest Cuts in Various Graph Classes". In: *Journal of Discrete Algorithms* 14 (July 1, 2012), pp. 136–149. DOI: 10.1016/j.jda.2011.12.008.