Universidade Estadual de Campinas
Instituto de Computação

Marcelo Pinheiro Leite Benedito

Approximation and Parameterized Algorithms for Pair Connectivity Problems

Algoritmos de Aproximação e Parametrizados para Problemas de Conectividade de Pares

CAMPINAS
2023

Marcelo Pinheiro Leite Benedito

Approximation and Parameterized Algorithms for Pair Connectivity Problems

Algoritmos de Aproximação e Parametrizados para Problemas de Conectividade de Pares

Tese apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Thesis presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Supervisor/Orientador: Prof. Dr. Lehilton Lelis Chaves Pedrosa

Este exemplar corresponde à versão final da Tese defendida por Marcelo Pinheiro Leite Benedito e orientada pelo Prof. Dr. Lehilton Lelis Chaves Pedrosa.

CAMPINAS

2023

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

B434a

Benedito, Marcelo Pinheiro Leite, 1994-
Approximation and parameterized algorithms for pair connectivity problems / Marcelo Pinheiro Leite Benedito. – Campinas, SP : [s.n.], 2023.

Orientador: Lehilton Lelis Chaves Pedrosa.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Otimização combinatória. 2. Algoritmos de aproximação. 3. Algoritmos parametrizados. 4. Decomposição em árvore (Teoria dos grafos). 5. Largura da árvore (Teoria dos grafos). I. Pedrosa, Lehilton Lelis Chaves, 1985-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações Complementares

Título em outro idioma: Algoritmos de aproximação e parametrizados para problemas de conectividade de pares
Palavras-chave em inglês:
Combinatorial optimization
Approximation algorithms
Parameterized algorithms
Tree decomposition (Graph theory)
Treewidth (Graph theory)
Área de concentração: Ciência da Computação
Titulação: Doutor em Ciência da Computação
Banca examinadora:
Lehilton Lelis Chaves Pedrosa [Orientador]
Uéverton dos Santos Souza
Santiago Valdés Ravelo
Mário César San Felice
Orlando Lee
Data de defesa: 16-08-2023
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-1816-8261
- Currículo Lattes do autor: http://lattes.cnpq.br/0825672800157119

Universidade Estadual de Campinas
Instituto de Computação

Marcelo Pinheiro Leite Benedito

Approximation and Parameterized Algorithms for Pair Connectivity Problems

Algoritmos de Aproximação e Parametrizados para Problemas de Conectividade de Pares

Banca Examinadora:

- Prof. Dr. Lehilton Lelis Chaves Pedrosa
  Universidade Estadual de Campinas
- Prof. Dr. Uéverton dos Santos Souza
  Universidade Federal Fluminense
- Prof. Dr. Santiago Valdés Ravelo
  Universidade Federal do Rio Grande do Sul
- Prof. Dr. Mário César San Felice
  Universidade Federal de São Carlos
- Prof. Dr. Orlando Lee
  Universidade Estadual de Campinas

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 16 de agosto de 2023

Agradecimentos

Ao alcançar o final desta jornada, não posso deixar de refletir sobre sua natureza transformadora. É raro encontrar oportunidades na vida em que se possa dedicar uma parte substancial do tempo a explorar profundamente um tema complexo, daqueles que demoram a revelar seus segredos. Trilhando esse caminho, não desvendamos fatos apenas sobre o nosso objeto de estudo, mas também sobre nós mesmos. Somos influenciados pelas pessoas que nos rodeiam e pelas lições daqueles que percorreram esse caminho antes de nós. Espero que este trabalho sirva de apoio para aqueles que estão em busca de sua própria jornada.

Muitos fizeram parte dessa história. Aos meus pais, Carlos e Mitsi, agradeço por terem me ensinado bons valores e me motivado a buscar meus sonhos. Agradeço a meus irmãos, André e Henrique, que sempre estiveram e sei que sempre estarão lá por mim. Sem o seu amor e incentivo, não chegaria até aqui.

Aos meus professores, tenho gratidão pelo constante estímulo ao conhecimento, independentemente dos desafios que se apresentaram. Em particular, agradeço ao Professor Lehilton pelos estimados anos de trabalho e amizade, cujos ensinamentos levarei para o resto da vida. Além disso, gostaria de reconhecer a importância do Professor André Gustavo, que me acompanhou nos primeiros passos dessa jornada acadêmica.

A todos os amigos que conheci ao longo deste período, expresso meu agradecimento por terem deixado uma marca positiva em minha vida. Também gostaria de agradecer ao Laboratório de Otimização e Combinatória, ao Instituto de Computação e à Universidade Estadual de Campinas pela estrutura de excelência, e a todos os servidores que tornaram possível a dedicação à pesquisa.

Este tempo foi repleto de aprendizados e crescimento pessoal, e me sinto grato por ter compartilhado esses momentos com a Rafaela, cujo apoio e carinho se mostraram essenciais.

Por último, agradeço pelo auxílio financeiro da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processos 2015/11937-9 e 2019/10400-2, e do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), processos 425340/2016-3, 422829/2018-8 e 312186/2020-7.

Resumo

Nesta tese, tratamos de problemas de conectividade de pares em grafos, cuja entrada inclui um conjunto de pares de vértices, chamados de demandas, e uma solução consiste de alguma estrutura conectando as demandas. Estudamos os problemas do ponto de vista de algoritmos de aproximação e parametrizados, que são combinados para superar barreiras advindas do uso de cada estratégia separadamente. Os resultados utilizam tanto técnicas existentes, como programação dinâmica sobre decomposição em árvores, como novas técnicas que podem ser generalizadas para problemas relacionados.

No Star k-Hub Center (SkHC), recebemos um grafo com um vértice central e um inteiro k e a tarefa é selecionar k vértices, chamados de terminais, e atribuir cada vértice a um terminal. O objetivo é minimizar o comprimento do maior caminho que conecta cada par de vértices e passa pelo centro e terminais atribuídos. Iniciamos o estudo do SkHC na área de complexidade parametrizada, derivando os primeiros resultados de dificuldade em parâmetros estruturais e apresentando um algoritmo de aproximação parametrizado pela *treewidth*, isto é, um algoritmo que executa em tempo $\mathcal{O}^*((\mathrm{tw}/\epsilon)^{\mathcal{O}(\mathrm{tw})})$ e produz uma solução com fator $1+\epsilon$ do valor ótimo.

No Multiple Allocation k-Hub Center (MAkHC), recebemos um grafo cujos vértices são clientes ou terminais, um conjunto de demandas composto por pares de clientes e um inteiro k. O objetivo é selecionar k terminais, minimizando o maior caminho que conecta os vértices de uma demanda passando pelo terminal associado. Também damos início ao estudo deste problema sob as lentes de complexidade parametrizada, fornecendo limitantes inferiores de aproximação e uma redução que elimina a possibilidade de existência de algoritmos com outros parâmetros. O resultado principal para o MAkHC é uma $(2+\epsilon)$-aproximação parametrizada pela *treewidth*, usando técnicas que removem a dependência de outros parâmetros e lidam com o conjunto de demandas arbitrário.

No Spanning Tree-Star (STS), temos um grafo com duas funções de custos nas arestas e uma solução é um subgrafo conexo gerador. O objetivo é minimizar a soma dos custos das arestas da solução, onde arestas pendantes e não pendantes são precificadas distintamente. Também consideramos variantes do STS em que as arestas não pendantes induzam um caminho ou um ciclo. Apresentamos algoritmos para o STS e variantes, parametrizados por *treewidth* ou *pathwidth*, que executam em tempo com expoente simples. Estes resultados usam *rank-based approach* e aplicam uma nova modelagem de rótulos.

Abstract

In this thesis, we deal with a series of pair connectivity problems in graphs, whose input includes a set of pairs of vertices, called demands, and a solution consists of some structure connecting the demands. The problems are studied from the point of view of approximation and parameterized algorithms, which are combined to overcome hardness barriers coming from using each approach separately. Our results leverage both existing ideas, such as frameworks for dynamic programming over a tree decomposition, and novel techniques that generalize to related problems.

In the Star k-Hub Center (SkHC), we are given a graph with a special center vertex and an integer k, and the task is to select a set of k vertices, called hubs, so that each vertex is connected to one hub. The goal is to minimize the length of the longest path connecting each pair of vertices through the center and assigned hubs. We initiate the study of parameterized algorithms for SkHC, giving the first hardness results on structural graph parameters and presenting an efficient parameterized approximation scheme with treewidth as parameter, that is, an algorithm that runs in time $\mathcal{O}^*((\mathrm{tw}/\epsilon)^{\mathcal{O}(\mathrm{tw})})$ and produces a solution whose cost is within a factor $1+\epsilon$ of the optimal value.

In the Multiple Allocation k-Hub Center (MAkHC), we are given a graph, whose vertices can be clients or hub locations, a set of demands composed by pairs of clients and an integer k. The task is to select k vertices minimizing the length of the longest path connecting the vertices of a demand through the assigned hub. We also initiate the study of this problem under the lens of parameterized complexity, giving inapproximability lower bounds and a reduction that rules out algorithms for several parameters. Our main result for MAkHC is a $(2+\epsilon)$-approximation parameterized by the treewidth of the graph, using techniques to remove dependency on extra parameters and to deal with arbitrary sets of demands.

In the Spanning Tree-Star (STS), we are given a graph with two edge-cost functions, and a solution is a spanning connected subgraph. The goal is to minimize the sum of the edge costs in a solution, where pendant and non-pendant edges are charged differently. We also consider variants of STS where non-pendant edges of a solution induce a path or a cycle. We give improved algorithms for STS and variants, parameterized by treewidth or pathwidth, running in single-exponential time. These results rely on rank-based techniques and a new flexible label modeling of the problems.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Computing is ubiquitous in our world: it is present in interactions between people, everyday financial transactions, going from one place to another and getting the latest news, all happening instantly. This is the result of the hard work done by our society throughout generations, researching, building and being curious about difficult matters. Algorithms are no different, and our understanding about them and their surrounding problems is always evolving.

The concept of an algorithm has been around for more than two thousand years, describing mathematical procedures such as performing division and finding the greatest common divisor or primes [40]. The last centuries were marked by an effort of building the first computational machines and laying the theoretical foundation of algorithms, that concerned on whether a procedure is correct [60]. As soon as people were able to specify programs and actually run them in early computers, we had algorithms to solve some of the world's problems. When practitioners were content with the elapsed time to obtain an output, the algorithm was deemed *fast*; the others would be classified as *slow*, especially when they demanded a very long time to solve even the simplest instances. For many such problems, eventually a new algorithm that is fast would appear and replace the slow one. However, there are a few *hard* problems for which no fast algorithm was discovered and, up to this day, no one knows how to solve them *efficiently*.

The need to formalize these notions gave rise to the complexity theory, which studies the resources required to solve a given computational problem, mostly time and space. We say that a decision problem belongs to P if there is an efficient (polynomial-time) algorithm to solve it, and problems in NP are those that can be efficiently verified by a non-deterministic Turing machine. A problem P is NP-hard if, for every problem in NP, there is a polynomial-time reduction to P, and, if P is also in NP, then it is called NP-complete [91]. Cook and Levin laid the foundation by showing that the satisfiability problem is NP-complete [52, 125]; Karp followed by giving 21 others [109].

When deciding whether a problem is tractable, there are many adjacent questions. The main underlying question is "what is the fastest algorithm to solve it?". It turns out that such a simple question might be too hard to answer, or might not give sufficient information about the problem in practice. For one, if the considered problem is of the optimization kind, when instances admit distinct solutions that can be ranked, a better question would be "how well can we solve the problem efficiently?". For other, if we

insist in finding an optimal solution, but the considered instances are restricted to some fixed subset, a more suitable question would be "which groups of instances can be solved efficiently?".

In this thesis, we are interested in combinatorial optimization problems, whose sets of feasible solutions are finite, oftentimes exponentially large in the input size. Each solution maps to a real value according to some given function, and the objective is to find an element that minimizes or maximizes this function. Ideally, one desires to find an algorithm that gives the best solution and runs in a reasonable amount of time, being it seconds or weeks, depending on the application. It turns out that the decision versions of many optimization problems are NP-hard, which is evidence that no efficient algorithms to solve them exist [53].

One example is a clustering algorithm that groups objects based on similarity for a given number of wanted clusters, and can be used, for example, as a subroutine of a machine learning system [146]. In such an application, an algorithm could be invoked thousands of times during a regular execution, thus it must run in a very short time frame to produce fairly good solutions. Now, consider the application of establishing emergency service facilities, such as fire stations, that must cover a city so that the response time is acceptable for every neighborhood. A procedure to solve this problem can afford a much higher running time, since it can be the difference of building fewer (prohibitively expensive) stations or better covering the city, reducing the response time.

The previous problems can be modeled as variants of k-Center [62, 130], which is a classical location problem recollected in this thesis. Although both examples share the same underlying basic problem, for the first, one might give up finding optimal solutions, while, for the second, the interest is in optimal solutions only. This highlights the importance of knowing and using distinct algorithm design strategies depending on the circumstance.

1.1 Investigated problems and motivation

Location theory deals with the spatial interaction of various objects, subject to diverse constraints, such as capacities, connectivity requirements, network topologies and robustness in the face of uncertainty [135, 70]. Many of these combinatorial optimization problems are centered on connecting pairs of objects through a network, so their demands are satisfied. For example, in the family of hub location problems, pairs of clients need to be connected through a hub network that routes all communications or products. Given clients and possible hub locations, a solution is a subset of open hubs and an assignment of clients to open hubs, respecting the constraints imposed beforehand. Amongst the modeled scenarios, we have delivery systems [107, 160], public transportation [93] and commercial aviation of civilians and cargo [39, 108].

Various hub location problems have emerged through the years, differing on the solution domain, whether it is discrete or continuous; on the number of hub stops serving each demand; on the number of selected hubs, and so on [4, 73]. Central to this classification is the nature of the objective function: for *median* problems, the objective is to

minimize the total length of the paths serving the demands, while, for *center* problems, the objective is to find a solution whose maximum length is minimum. We are interested in two center problems within this family.

In the Star k-Hub Center (SkHC), we are given a graph with a special center vertex and an integer k, and the task is to select a set of k vertices, called hubs, and assign each vertex to exactly one hub, so that we minimize the length of the longest path that connects a pair of vertices through the center and assigned hubs. This models a two-level hub system, since a path between vertices that are assigned to distinct hubs needs to pass through the center vertex, while a path between vertices assigned to the same hub needs to pass only through the hub itself. Note that every vertex, but the center, can be chosen as a hub, and the set of demands is not dependent on the input, that is, every vertex must be connected to each other.

In the Multiple Allocation k-Hub Center (MAkHC), we are given a graph whose vertices can be clients or hub locations, a set of demands composed by pairs of clients and an integer k, and the task is to select a set of k hubs among the hub locations so that each demand is assigned to a hub. The goal is to minimize the length of the longest path connecting the vertices of a demand through the assigned hub. This problem concerns with the one-stop model [105, 166], in which a demand uses only one hub to be satisfied, and thus the length of a path is the sum of the distances between each client and the selected hub. Note that a client may be assigned to multiple hubs for distinct demands, and the set of demands that must be satisfied is given by the input.

Although both problems belong to the same family, they pose different challenges. The first takes place in a particular type of network, where every vertex is assigned to exactly one hub and the path between two vertices depend on which hubs they are assigned to. The second has no additional constraints on the network and receives an arbitrary set of demands, but each client can be connected to multiple hubs and the path of a demand uses only one hub.

We are also interested in another family of network design problems modeling transportation and telecommunication systems [2, 118, 10]. In such problems, each vertex must either be internal, connected to a high capacity but expensive backbone, or be connected directly to an internal vertex. In the Spanning Tree-Star (STS), the input is a graph and two edge-cost functions, and a solution is a connected subgraph, whose internal vertices induce a tree. The goal is to minimize the sum of edge costs, where the function used to charge an edge depends on whether both endpoints are internal. Other variants consist of finding minimum-cost connected spanning subgraphs in which internal vertices induce a path or a cycle, namely, Spanning Path-Star (SPS) and Spanning Cycle-Star (SCS).

These are some problems that match our framework of pair connectivity in the location theory literature. As we have seen, these families of problems are recurrent in practice, and since they are composed by NP-hard problems, researchers do not hope to find efficient algorithms that solve each and every instance. Then, we turn to alternative techniques to solve them in some scenarios. Specifically, we are interested in approximation and parameterized algorithms. Moreover, we study the recent trend of combining both techniques [69, 134, 1, 111].

Approximation algorithms find (possibly) non-optimal solutions with guaranteed quality in polynomial time, whereas parameterized algorithms solve the problem exactly, but might have non-polynomial factors in their running time, depending on a fixed parameter. A parameterized approximation algorithm finds a solution with guaranteed quality, as an approximation algorithm, and has a bounded running time, as a parameterized algorithm. This approach is used mainly when neither technique applied separately is satisfactory. In this thesis, we investigate these types of algorithms for pair connectivity problems and, alongside obtaining algorithmic results, we also study the intrinsic difficulty of problems, that, above all, help us guide the search for an adequate solution method.

An important topic we discuss is the notion of treewidth, that captures the tree-likeness of a graph by means of a structural decomposition, i.e., a tree decomposition. Given its versatility and expressiveness, treewidth is extensively used in parameterized algorithms as a parameter. A small value indicates the input graph is close to a tree, and thus the algorithm may run faster on these instances. This concept aligns perfectly with dynamic programming algorithms, where a global solution is obtained by considering solutions for smaller problems that are reused multiple times. We apply the technique of dynamic programming over a tree decomposition to all problems we investigate, together with some consolidated approaches and novel ideas.

1.2 Results and thesis organization

In the following, we state our main results for the presented problems. These are thoroughly investigated in individual chapters, each containing the related works and a complete exposition of results and techniques. Chapter 2 presents common basic concepts and definitions that are used in the subsequent chapters, on which we build our results.

Chapter 3 is dedicated to SkHC. We initiate the parameterized study of the problem, showing that it is W[1]-hard when the parameter is the combination of the vertex cover number of the graph and the number of hubs, and that there is no parameterized $(1.25-\epsilon)$-approximation algorithm when the parameter is the number of hubs, for $\epsilon > 0$. The first positive result is an algorithm parameterized by tw and r, the treewidth of the graph and the cost of the solution, that either produces a solution for SkHC of size k and value at most r, or decides that no such solution exists. Actually, this algorithm solves a slightly modified problem with a dynamic programming formulation, which we show to translate to an algorithm for SkHC. Building on this result, we give an efficient parameterized approximation scheme with parameter tw, that is, an algorithm that finds a solution with k hubs and value at most $(1 + \epsilon)r$, or concludes that every solution has value greater than r. The main ingredient to achieve this improvement is storing the distances in an approximate fashion, and showing that the final value has bounded error, when compared to the exact value.

Chapter 4 is dedicated to MAkHC. We also initiate the parameterized study of this problem, showing that, for $\epsilon > 0$, there is no parameterized $(3 - \epsilon)$-approximation when the parameter is the number of hubs, the cost of the solution is bounded by a constant and the graph is unweighted, unless FPT = W[2]. It is also hard to find a good constant-factor

approximation when the graph is planar, as we show it is NP-hard to obtain a $(3-\epsilon)$-approximation for MAkHC, even if the maximum degree is 3. Aiming for a different set of parameters, we show that, unless FPT = W[1], the problem does not admit an algorithm parameterized by the number of hubs, the highway and skeleton dimensions and the pathwidth of the graph, even constrained to a specific type of graph, or when parameterized by the number of hubs and the vertex cover of the graph. Our main result for this problem is a $(2+\epsilon)$-approximation parameterized by the treewidth of the graph, in which we combine ideas from a classical algorithm in location theory, a framework that reduces the size of the dynamic programming table and a technique we devise to deal with the generic set of demands, given as input. An additional algorithm is given for unweighted planar graphs, with the number of hubs and the cost of the solution as parameters, achieving a similar approximation.

Chapter 5 is dedicated to STS and variants. We provide the first single-exponential time algorithm for STS, parameterized by either the pathwidth or the treewidth of the graph, running in $\mathcal{O}^*(4^{1.7\mathrm{pw}})$ and $\mathcal{O}^*(4^{3.5\mathrm{tw}})$, respectively. This result relies on the rank-based approach and our label modeling of the problem, that we believe can be used in other connectivity problems. The algorithm is then a combination of a dynamic programming formulation and a subroutine from the rank-based approach that removes redundant partial solutions from the table in each step, with no damage to the optimal solution. On the hardness side, we prove that there is no algorithm solving STS in time $\mathcal{O}^*((4-\epsilon)^{\mathrm{pw}})$, for $\epsilon > 0$, and that a similar algorithm parameterized by clique-width is unlikely. Analogous results are also given for SPS and SCS, the path and cycle variants.

Chapter 6 brings our final remarks.

Publications The following papers were published as a result of this thesis:

- "An efficient parameterized approximation scheme for the Star k-Hub Center" in the Latin and American Algorithms, Graphs and Optimization Symposium [16].
- "A Parameterized Approximation Algorithm for the Multiple Allocation k-Hub Center" in the Latin American Symposium on Theoretical Informatics [14].

Chapter 2

Preliminaries

A problem in the field of combinatorial optimization is composed of a set of instances, a set of feasible solutions, a function that maps solutions to values and a goal, either minimizing or maximizing this value [117]. The complexity of these problems is usually studied by considering their decision versions, where the question is whether there exists a solution of a given value. We introduce some of the complexity classes of combinatorial optimization.

A decision problem is in P (polynomial time) if there exists an algorithm that solves the problem and runs in polynomial time. The class NP (non-deterministic polynomial time) contains the decision problems that can be solved by a non-deterministic Turing machine in polynomial time or verified by a deterministic Turing machine in polynomial time. For decision problems P and P', a *polynomial-time reduction*[1] from P to P' is a polynomial-time mapping of each instance of P to an instance of P', such that a YES-instance of P is mapped to a YES-instance of P', and a NO-instance of P is mapped to a NO-instance of P'. Then, an algorithm for P' can be used to solve P, and thus the latter is no harder to solve than the former. A problem is NP-hard if every problem from NP can be reduced to it in polynomial time. This implies that, if there exists an efficient algorithm to solve any NP-hard problem, one can use it to solve all the problems in NP in polynomial time. Problems both in NP and NP-hard are called NP-complete. A detailed guide to these classes is given in [91].

The presented complexity classes are the foundation of many fields. One of the most important unsolved problem in Computer Science is determining the relationship between these classes, coined the P versus NP problem. The debate is whether the class of problems that can be efficiently verified is the same as the class of problems that can be efficiently solved, and the prevailing opinion amongst researchers is that $\mathsf{P} \neq \mathsf{NP}$ [98]. Figure 2.1 relates the complexity classes, considering $\mathsf{P} \neq \mathsf{NP}$. Under this assumption, NP-hard problems are, in fact, intractable and thus we need alternative ways[2] to tackle these problems, other than polynomial-time algorithms. We focus on the intersection of approximation and parameterized algorithms, reviewed in the following sections. While not all mentioned concepts are used to devise our results, we included them as educational value for a reader unfamiliar with the subject.

[1] Also known as Karp reduction.

[2] Other alternatives are exact formulations [137] and heuristic methods [157].

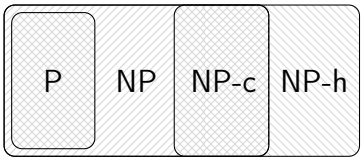


Figure 2.1: Diagram relating complexity classes.

We use standard graph theory notation. For a graph G, let $V(G)$ and $E(G)$ be the sets of vertices and edges of G, respectively. A graph G' is a subgraph of G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. For $X \subseteq E(G)$, $G[X]$ is the subgraph that has the endpoints of X as vertices and X as the set of edges. The *neighborhood* $N(u)$ of a vertex $u \in V(G)$ is the set of vertices adjacent to u; the *closed neighborhood* is defined as $N[u] = N(u) \cup \{u\}$. Let $\mathtt{cc}(G)$ be the number of connected components of G. The set containing non-negative integer numbers is represented as $\mathbb{N}$.

2.1 Approximation algorithms

An approximation algorithm produces solutions with provable quality and runs in polynomial time, thus, it concedes guaranteeing only a fairly good solution, instead of an optimal, but in exchange it does not take long to do so. They can be used when one needs to ensure the quality of the solution and cannot admit the non-polynomial running time of an exact method. Researchers are also interested in determining how close any algorithm can approximate an optimal solution in polynomial time, leading to the study of inapproximability lower bounds. These are mostly based on strong hypotheses, such as the $\mathsf{P} \neq \mathsf{NP}$ conjecture, then there is a high confidence on these barriers.

In the following, consider a minimization problem and let $\alpha > 0$. An algorithm $\mathcal{A}$ is an *α-approximation algorithm* if, for each instance I of the problem, $\mathcal{A}$ runs in time $|I|^{\mathcal{O}(1)}$, i.e., polynomial in the input size, and returns a solution of value $\mathcal{A}(I) \leq \alpha \cdot \mathrm{OPT}(I)$, where $\mathrm{OPT}(I)$ is the value of an optimal solution for I. A family of algorithms $\{\mathcal{A}_\epsilon\}_{\epsilon>0}$ is a *polynomial-time approximation scheme* (PTAS) if, for any $\epsilon > 0$, $\mathcal{A}_\epsilon$ is a $(1+\epsilon)$-approximation running in time $|I|^{f(\epsilon)}$, for a computable function f; if $\mathcal{A}_\epsilon$ runs in time $f(\epsilon) \cdot |I|^{\mathcal{O}(1)}$, then it is an *efficient polynomial-time approximation scheme* (EPTAS). If there is an inapproximability result stating that, unless a strong hypothesis fails, there can be no approximation algorithm with ratio $\alpha - \epsilon$, for $\epsilon > 0$, and we have an algorithm with approximation ratio α, then we say this algorithm is *tight*, i.e., it is the best possible approximation for the problem with respect to the hypothesis.

A popular way of dealing with NP-hard problems is devising *heuristics*, which are procedures based on practical experiences and insights, with no *a priori* commitment to solution quality or running time [133]; then, an empirical study on a wide set of instances is needed to assess its usefulness. Consequently, these are often employed on applications with a well-behaved set of instances, or in situations that the given statistical guarantees are sufficient. Approximation algorithms, on the other hand, are based on mathematical proofs that guarantee the solution quality, ranging over all instances of a problem. Although it may be hard to create an algorithm ensuring close-to-optimal

solutions, many approximations perform well in practice. For example, Jain et al. [106] presented two algorithms for the Facility Location (FL), with approximation ratios 1.861 and 1.61, and conducted an experimental study that revealed their algorithms had an average error of less than 3% and ran in at most 2 seconds.

Williamson and Shmoys [164] listed the main reasons for studying approximation algorithms: (*i*) we need solutions for NP-hard problems that are intrinsically connected to information technology, especially when the vast amount of data results in bigger instances, impractical for exact methods; (*ii*) solving a simpler model of a real-life application approximately can facilitate the creation of a good heuristic, and, overtime, evolve into an approximation solving the original problem; (*iii*) the mathematical rigor on devising an algorithm to solve the problem approximately leads to a deeper understanding of its underlying structure, helping us identify classes of instances in which an algorithm thrives in practice; (*iv*) in association with complexity theory, approximation algorithms and inapproximability lower bounds gives us a sense of the hardness of solving a problem.

Design techniques Approximation algorithms have diverse design techniques, from simple implementations to others using complex data structures or other approaches that carry large constants in their time complexity. We highlight the main techniques.

A *greedy* strategy is one that makes locally optimal choices during each step of its execution, in the hope this leads to a good solution. It is used ubiquitously in the design of algorithms, and, in the field of approximation, often yields the simplest proofs and implementation. A *local search* algorithm holds on to an incumbent solution and searches for improved ones on its neighborhood. This relationship is defined beforehand, providing a subset of similar candidates from the search space. The algorithm iteratively moves to one of them and the process continues, until a stop criteria is reached.

The toolbox of *linear programming* (LP) models the problem in a mathematical language, and is part of several involved techniques [80]. A popular use is designing *rounding* algorithms, that transforms a solution of a relaxed model into a feasible solution by rounding the variables to integer values [155]. In *primal-dual* algorithms, the problem is described as two LP models, which are manipulated in tandem to obtain a feasible solution [13, 164]. In spite of using LP models, this technique does not need a solution to its relaxation, thus, these algorithms are often fast. Other design techniques involve dynamic programming, enumeration in (pseudo) polynomial time, changing metrics, applying cuts, using randomness, semidefinite programming, and more problem-specific approaches [164, 162].

2.2 Parameterized algorithms

A parameterized algorithm solves a problem exactly while maintaining the non-polynomial running time confined to a *parameter* of the input. Then, as long as this parameter is small, the algorithm is fast and is often much faster then usual exact algorithms which depend exponentially on the whole input. The area of parameterized complexity can be viewed as an expansion of the classical complexity theory, since now the tractability of a

problem does not depend only on the number of bits used to describe an instance, but also on some specific part of the input, i.e., the parameter.

This allows a fine-grained analysis that characterizes instances that are easier to solve to optimality, rather than trying to give an algorithm to solve all instances at once. In other words, while an approximation algorithm focuses on efficiently producing a solution with reasonable quality for any given instance, a parameterized algorithm focus on a family of instances and aims at obtaining an exact solution with non-polynomial dependency as small as possible. For example, Vertex Cover (VC) has a simple exhaustive algorithm solving the problem in time $2^k n^{\mathcal{O}(1)}$, where k is the size of the solution, thus, if $k = \mathcal{O}(\log n)$, the problem is polynomially solvable for these instances. This problem has been extensively studied, and the current best algorithm with k as parameter has running time $\mathcal{O}^*(1.25298^k)$ [96]. Note that such algorithms have a deep impact on practical scenarios, considering there are problems, such as Clique, whose best-known algorithm executes in time $\mathcal{O}(n^{\Theta(k)})$, where k is the size of the clique [138].

Formally, a *parameterized problem* is comprised of an instance I of some decision problem[3] and an integer parameter k that depends on I. If, for each instance (I, k) of a parameterized problem, there is an algorithm that decides I in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where f is a computable function, we say that the problem is *fixed-parameter tractable* (FPT) and the procedure is a *parameterized algorithm* [58]. The running time of the algorithm is often presented in the form $\mathcal{O}^*(f(k)) = f(k) \cdot \text{poly}(|I|)$, which omits polynomial factors. A *slice-wise polynomial algorithm* runs in time $f(k) \cdot |I|^{g(k)}$ for an instance I, parameter k and computable functions f and g, and the class containing such problems is XP.

Kernelization One of the contributions of parameterized complexity is formalizing the concept of preprocessing routines, that effectively simplifies an instance of a problem before the main algorithm is invoked. This concept is called kernelization, and is aimed at solving the easy parts of an instance, reducing it to a hard kernel.

Formally, a *kernelization algorithm* maps an instance (I, k) to an equivalent instance (I', k') such that (I, k) is a Yes-instance if, and only if, (I', k') is a Yes-instance. Moreover, it must run in polynomial time, in terms of $|I|$ and k, and $|I'| + k' \leq g(k)$, for a computable function g. If g is polynomial, then the problem admits a *polynomial kernel*. A parameterized problem is FPT if, and only if, it admits a kernelization algorithm [35, 58], thus, giving a kernelization algorithm is an alternative way of defining the fixed-parameter tractability of a problem. A common research question is determining the smallest kernel size, both from an algorithmic and hardness point of views.

As an example, we present a simple kernelization for VC, parameterized by k. Oftentimes, the procedure is defined as a set of reduction rules, which are executed exhaustively until none applies: (i) remove isolated vertices; and (ii) if there is a vertex with degree more than k, then remove it and decrease the parameter in one. Now, one can prove that the (reduced) graph G has maximum degree k, and a set of k vertices can cover at most k^2 edges, then, every Yes-instance has $|V(G)| \leq k^2 + k$ and $|E(G)| \leq k^2$, resulting in a quadratic kernel.

[3]Optimization problems can be posed as decision problems by asking whether there is a solution with some specific value (given as input), since the possible solutions are Yes and No.

Klam value A running time of $2^{2^{2^{2^{2^k}}}} \cdot n^{\mathcal{O}(1)}$ is gladly accepted by the presented definition of parameterized algorithms, but it is clear that such an algorithm would not perform well in practice. Downey and Fellows addressed this problem by introducing the *klam value*, defined as the largest value of k that an algorithm is capable of solving in practice, considering (i) the running time has the form $f(k) \cdot |I|^c$, for $c \leq 3$, so the complexity is not shifted into the polynomial portion; and (ii) $f(k) \leq 10^{20}$, i.e., we bound the maximum number of computational steps with a reasonable absolute value [139, 68].

Table 2.1 compares the efficiency of algorithms for VC, considering only the exponential term of the running time. The algorithms are the naive exhaustive search running in $\mathcal{O}^*(2^k)$, Chandran and Grandoni [42] in time $\mathcal{O}(1.2745^k k^4 + kn)$ and exponential space, Chen et al. [45] in time $\mathcal{O}(1.286^k + kn)$, Chen et al. [46] in time $\mathcal{O}(1.2738^k + kn)$ and Harris and Narayanaswamy [96] in time $\mathcal{O}^*(1.25298^k)$.

k	2^k	$1.2745^k k^4$	1.286^k	1.2738^k	1.25298^k
10	1 024	10^5	12	11	9
20	10^6	10^7	153	126	90
50	10^{15}	10^{12}	290 000	180 000	79 000
100	10^{30}	10^{18}	10^{11}	10^{10}	10^9
150	10^{45}	10^{24}	10^{16}	10^{15}	10^{14}
klam value	66	112	183	190	204

Table 2.1: Approximate efficiency and klam value of algorithms for VC.

Lower bounds Just as in classical complexity theory, we need the ability of ruling out certain types of algorithms, based on a strong hypothesis. The foundation of parameterized complexity was laid by Downey and Fellows [68], and it is an essential tool for algorithm designers, since one may be searching for a result that is not possible under our current understanding of the field. This means that achieving such an algorithm is as hard as getting an algorithm for a well-established problem, heavily studied by many researchers for decades. Additionally, like in the classical setting, the insights stemming from analyzing a problem in both perspectives, hardness and algorithmic, can lead to an easier path of obtaining results.

The basic working hypothesis[4] used in the parameterized setting is that Clique is not FPT, an assumption stronger than $\mathsf{P} \neq \mathsf{NP}$. Using reductions, we can prove such results for our problems. Let (I, k) and (I', k') denote instances of parameterized problems P and P', respectively. A *parameterized reduction* from P to P' is an algorithm that receives an instance of the former and outputs an instance of the latter, where

(i) (I, k) is a Yes-instance of P if, and only if, (I', k') is a Yes-instance of P';

(ii) $k' \leq g(k)$, for a computable function g;

(iii) the running time is $f(k) \cdot |I|^{\mathcal{O}(1)}$, for a computable function f.

[4]There is an equivalent argument with Turing machines.

Note that the second item, requiring that the parameter of P' must be bounded by a function of the parameter of P, implies that not every NP-hardness reduction translates into a parameterized one.

The W-*Hierarchy* organizes problems according to their hardness, in the hope of discriminating the different existent levels. A parameterized problem P belongs to W[t], for $t \geq 1$, if there is a parameterized reduction from P to a circuit satisfiability problem with weft[5] at most t [58]. There is also the familiar notion that a parameterized problem can belong to W[t] and W[t]-hard, making it W[t]-complete, for some $t \geq 1$. Our canonical problem Clique lies in the first class and is W[1]-complete, whereas Dominating Set is W[2]-complete. The hierarchy is organized as in Equation 2.1, and it is conjectured that each of the containments is proper [68]. Note that W[P] denotes the class obtained by having no restriction on depth.

$$\mathsf{FPT} \subseteq \mathsf{W[1]} \subseteq \mathsf{W[2]} \subseteq \cdots \subseteq \mathrm{W}[P] \subseteq \mathsf{XP} \tag{2.1}$$

As algorithm designers, if we are interested in ruling out a parameterized algorithm for a problem P, it is sufficient to show a parameterized reduction from a W[t]-hard problem to P, for some $t \geq 1$. A commonly-used conjecture in parameterized hardness results is the *Exponential Time Hypothesis* (ETH), that states 3-SAT cannot be solved in time $2^{(1-\epsilon)n}$, for $\epsilon > 0$, i.e., with subexponential running time in the number of variables. The initial intuition was that FPT $\neq$ W[1] is not suitable for obtaining precise estimates of lower bounds in the parameterized setting, later confirmed when ETH, which ultimately implies P $\neq$ NP, supported the existence of nearly-tight lower bounds for various problems [58]. A variant of ETH is the *Strong Exponential Time Hypothesis* (SETH), that gives even more refined lower bounds in the base of the exponent: it states that a version of SAT cannot be solved in time $\mathcal{O}((2-\epsilon)^n)$, for any $\epsilon > 0$ [103, 36, 58].

Being conjectures, one must examine its solidity in the field of study, nonetheless, improving a lower bound proven under ETH or SETH is as hard as improving the current understanding on the complexity of SAT. This is an active area of research, as many lower bounds, not only in the parameterized world, depend on other conjectures that produce tighter results, although with less consensus amongst theoreticians [161].

Design techniques Now, we give an overview on some design techniques for parameterized algorithms. These and more are thoroughly described in Niedermeier [140] and Cygan et al. [58].

Inspired in the idea of backtracking, a *bounded search tree* is a simple, yet effective, technique to obtain parameterized algorithms. Given an initial instance, the procedure guesses a small part of an optimal solution, simplifying the input to the next level: if, in each branching step, (*i*) at least one optimal solution is considered, (*ii*) the number of possible guesses is bounded by the parameter, and (*iii*) the instance is simplified, it is possible to bound the size of the search tree in terms of the parameter, obtaining an FPT algorithm. The $\mathcal{O}^*(2^k)$-time algorithm for VC is a simple application of bounded search

[5]The *weft* of a circuit is defined as the maximum number of large nodes on a path from an input node to the output node, where large nodes are those which have in-degree greater than 2.

trees, which can be further improved with kernelization.

The *iterative compression* technique can be used for solving problems whose objective is removing a subset of elements satisfying a certain condition, usually a subset of vertices or edges in graph problems. The procedure receives an instance and a solution with some known property, that is exploited to either slightly reduce its size or correctly state that no smaller solution exists. The key strategy to obtain fast algorithms is starting with the smallest solution possible and employing an efficient FPT compression routine. Reed et al. [150] were the first to employ such technique, wielding a $\mathcal{O}^*(3^k)$-time algorithm for Odd Cycle Transversal.

Probability plays an important part in some parameterized algorithms, and that is the case in *color coding*. Originally, it was used to detect some small subgraph pattern in an input graph, such as a path, cycle or graph with bounded treewidth, but since then, other applications emerged. The method colors objects randomly and searches for the wanted structure, that has some probability of being colored correctly; then, by repeating a sufficient number of times, we can find it with constant or high probability. An example is the randomized algorithm for Longest Path that solves the problem in time $\mathcal{O}^*((2e)^k)$, returning a path with constant probability.

Parameters come in many forms: they can be directly related to the problem at hand, e.g., the number of facilities in an instance of Facility Location, or its input graph, e.g., the maximum degree of a vertex. There are structural parameters, such as the *vertex cover number* (vc) or *feedback vertex set number* (fvs) of the input graph, that are used to further understand the difficulty of the problem in instances with these bounded parameters. Frequently, an algorithm with this kind of parameter is assumed to receive the structure as input; in practice, it is obtained with an FPT algorithm or approximation [28]. There are also structural parameters that capture the resemblance of the input graph to some type of graph, such as paths, trees or cliques. In Section 2.4, we explore the *treewidth*, the main parameter studied in this work, while in Section 2.5, we show how the *dynamic programming* paradigm is used to design parameterized algorithms, especially combined with treewidth.

2.3 Parameterized approximation algorithms

The Cobham-Edmonds thesis states that a problem is feasibly solvable if it admits an algorithm that is accurate and efficient [49, 71]. The previous sections gave two different approaches to solve NP-hard problems, each relaxing one of the characteristics we want in an algorithm. Since the dawn of parameterized complexity, there have been people studying the possibilities when the fields of approximation and parameterized algorithms come together [34, 69, 134]. However, a significant growth occurred in the last decade, when countless open questions were surfaced by the intensive study of parameterized complexity.

Indeed, efficient polynomial-time approximation schemes were studied *avant la lettre*, before the notion of FPT was evidently considered, since a PTAS and an EPTAS are, in fact, approximations running in XP and FPT time, respectively, parameterized by the

approximation factor[6]. This evidences the natural path of merging these fields, as the combined effort help us circumvent hardness results that affects each one separately.

Given a parameterized problem with instance (I, k), $\mathcal{A}$ is a *parameterized approximation algorithm* with ratio α if it runs in time $f(k) \cdot |I|^{O(1)}$ and produces a solution of value $\mathcal{A}(I) \leq \alpha \cdot \text{OPT}(I)$, for a computable function f. For $\epsilon > 0$, a family of algorithms $\{\mathcal{A}_\epsilon\}_{\epsilon>0}$ is a *parameterized approximation scheme* (PAS) if $\mathcal{A}_\epsilon$ approximates an optimal solution within $1 + \epsilon$ and runs in time $f(k, \epsilon) \cdot |I|^{g(\epsilon)}$, for constant ϵ and computable functions f and g; if ϵ is a parameter alongside k, then $\mathcal{A}_\epsilon$ must run in time $f(k, \epsilon) \cdot |I|^{\mathcal{O}(1)}$, and $\mathcal{A}_\epsilon$ is called an *efficient parameterized approximation scheme* (EPAS). An introduction to the subject and a recent survey can be found in Marx [134] and Feldmann et al. [75].

Approximate kernelization Some problems do not admit kernels of polynomial size, under common complexity assumptions. This is the case for Connected Vertex Cover, where the best one can hope for is a kernel with 2^k vertices [67, 58]. However, if we settle for a near-optimal solution, we can achieve a polynomial-size kernel [128]. As we broadened the definition of a parameterized algorithm to output approximate solutions, there has been work to do the same with kernels.

An *α-approximate kernelization algorithm* maps, in polynomial time, an instance (I, k) to an equivalent instance (I', k'), such that a β-approximate solution for I' can be converted, in polynomial time, into an $\alpha\beta$-approximate solution for I. As in the previous case, we must have $|I'|+k' \leq g(k)$, for a computable function g. For a complete exposition on this subject, including different types of approximate kernels, we refer the reader to Lokshtanov et al. [128].

Lower bounds The tools to develop lower bounds on the approximation of parameterized problems are an extension of the ones used to derive results in the parameterized world. Just as ETH and SETH were devised to obtain precise lower bounds to contrast with parameterized algorithms, there are attempts to improve the understanding of parameterized approximation algorithms.

The *Gap Exponential Time Hypothesis* (Gap-ETH) is a strengthening of ETH, that states an approximate version of 3-SAT cannot be solved in time $2^{o(n)}$ [65]. This hypothesis was used to obtain relevant lower bounds for classic problems, such as Clique and k-Median. The former was proven to not admit a parameterized $o(k)$-approximation [41], while the latter has no parameterized $(1 + 2/e - \epsilon)$-approximation, implying that the best algorithm is essentially tight [50].

Design techniques The design techniques employed in parameterized approximation algorithms come from both realms, often undergoing small adaptations to fit the new setting. As this field of study is relatively new, there is no unified theory categorizing the different types of algorithms. In clustering problems, for example, coresets consolidate clients into representatives, effectively reducing the set to a logarithmic size [50].

[6] In a recent talk, Fellows recounted the anecdote when, briefly after the publication of his seminal work on parameterized complexity, his colleague asked whether the parameter can really be anything, surfacing the relation between a PTAS and the parameterized world.

This technique found a fruitful field when combined with the enumeration of small sets and guessing values, dependent on the parameter. Other works are found in a recent survey [76].

Although most algorithms are focused on breaching W-hardness barriers, there is also the search for faster algorithms solving FPT problems, exchanging quality for efficiency. For instance, there is an algorithm for Vertex Cover that, given any exact algorithm running in $\mathcal{O}^*(\gamma^k)$, produces a $(1+\epsilon)$-approximate solution in time $\mathcal{O}^*(\gamma^{(1-\epsilon)k})$, for $\epsilon > 0$ [78]. Using the best exact algorithm and settling for a 1.1-approximate solution, we can improve the klam value of this problem from 190 to 211.

2.4 Treewidth

It is recurrent that solving a problem in a general graph is hard, but an efficient algorithm is known as soon as one restrains the input graph to a class, such as trees. Instead of considering only trees, the notion of treewidth classifies the tree-likeness of a graph, accompanied with its structural decomposition. One of the main uses of this tool is to design efficient algorithms when there is a high degree of similarity between the input graph and a tree, i.e., when the treewidth is small. Moreover, this algorithm is usually designed under the framework of dynamic programming. The concept was discovered a few times over the last decades and gained popularity through the work of Robertson and Seymour [151].

A person unfamiliar with treewidth could define the tree-likeness of a graph in various ways, perhaps counting the number of cycles or other estimate to transform it in a tree. Up to now, the most useful way to define it, both for algorithmic and theoretical purposes, is to decompose the graph into a special structure. This structure has the *separation property* of trees, roughly captured by the following: for a given vertex u of a tree, every path whose origin belongs to the subtree rooted at u going to a vertex that is not in the subtree, must pass through u. Next, we formally define this type of graph decomposition and the treewidth of a graph.

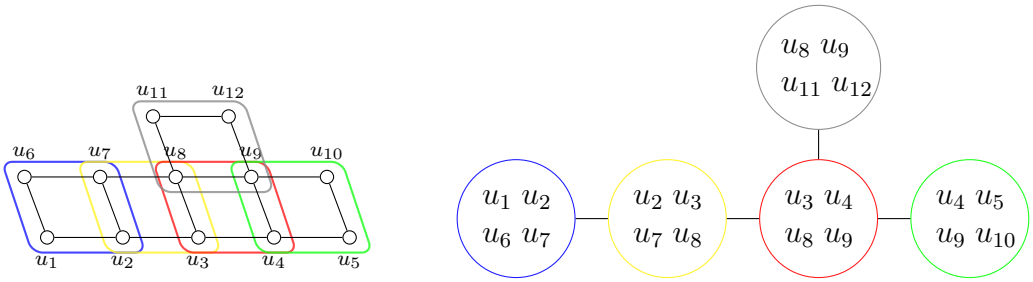
A *tree decomposition* of a graph G is a pair $(\mathcal{T}, B)$, where $\mathcal{T}$ is a tree and B is a function that associates each node t of $\mathcal{T}$ to a vertex subset B_t, called bag, such that:

(*i*) $\cup_{t \in V(\mathcal{T})} B_t = V(G)$, i.e., every vertex of G is in at least one bag;

(*ii*) $\forall\, (u,v) \in E(G), \exists\, t \in V(\mathcal{T}) : u, v \in B_t$, i.e., for each edge of G, there is a bag containing its endpoints;

(*iii*) $\forall\, u \in V(G)$, $T_u = \{t \in V(\mathcal{T}) : u \in B_t\}$ induces a connected subtree of $\mathcal{T}$, i.e., the nodes whose bags contain a vertex u form a connected subtree.

The width of a tree decomposition is $\max_{t \in V(\mathcal{T})} |B_t| - 1$ and the *treewidth* of G is the minimum width of any tree decomposition of the graph, and is denoted by tw. Other ways to define treewidth can be found in [23, 58]. Figure 2.2 exemplifies a tree decomposition alongside the obtained tree and bags; note that this graph admits a tree decomposition of width 2.



(a) Input graph. (b) Associated bags and tree.

Figure 2.2: Example of a tree decomposition.

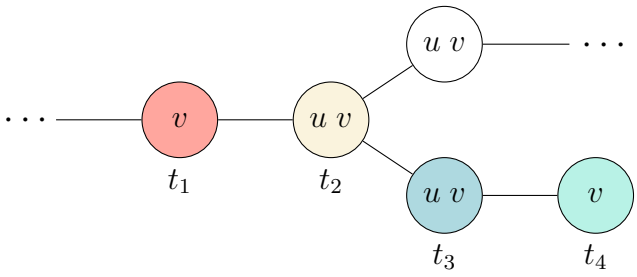


Figure 2.3: Node types of a nice tree decomposition.

In order to facilitate the design of dynamic programming algorithms, we use the concept of a *nice tree decomposition*, where $\mathcal{T}$ is a binary rooted tree and each node t can be one of four types:

(*i*) *leaf node*, in which $|B_t| \leq 1$;

(*ii*) *introduce node*, which has a child t' with $B_t = B_{t'} \cup \{u\}$, for $u \notin B_{t'}$;

(*iii*) *forget node*, which has a child t' with $B_t = B_{t'} \setminus \{u\}$, for $u \in B_{t'}$;

(*iv*) *join node*, which has children t' and t'' with $B_t = B_{t'} = B_{t''}$.

Note that, by the third property of a tree decomposition, every vertex of G is forgotten only once, but may be introduced several times. Frequently, it is helpful to use an extra type of node that introduces edges one by one:

(*v*) *introduce edge node*, which has a child t' with $B_t = B_{t'}$, annotated with an edge $\{u, v\}$ such that $u, v \in B_t$.

Figure 2.3 exemplifies the node types of a nice tree decomposition, where t_4 is a leaf node, t_3 is an introduce node, t_1 is a forget node and t_2 is a join node, while Figure 2.4 brings a nice tree decomposition of the graph given in Figure 2.2.

The adjacent concept of a *path decomposition* of graph G is a pair $(\mathcal{P}, B)$, where $\mathcal{P}$ is a path and B is a function that associates each node of $\mathcal{P}$ to a vertex subset B_t, respecting the same three properties of tree decompositions, except that the nodes whose bags contain a fixed vertex form a connected subpath of $\mathcal{P}$. As expected, the width of a path decomposition is $\max_{t \in V(\mathcal{P})} |B_t| - 1$ and the *pathwidth* of G is the minimum width of

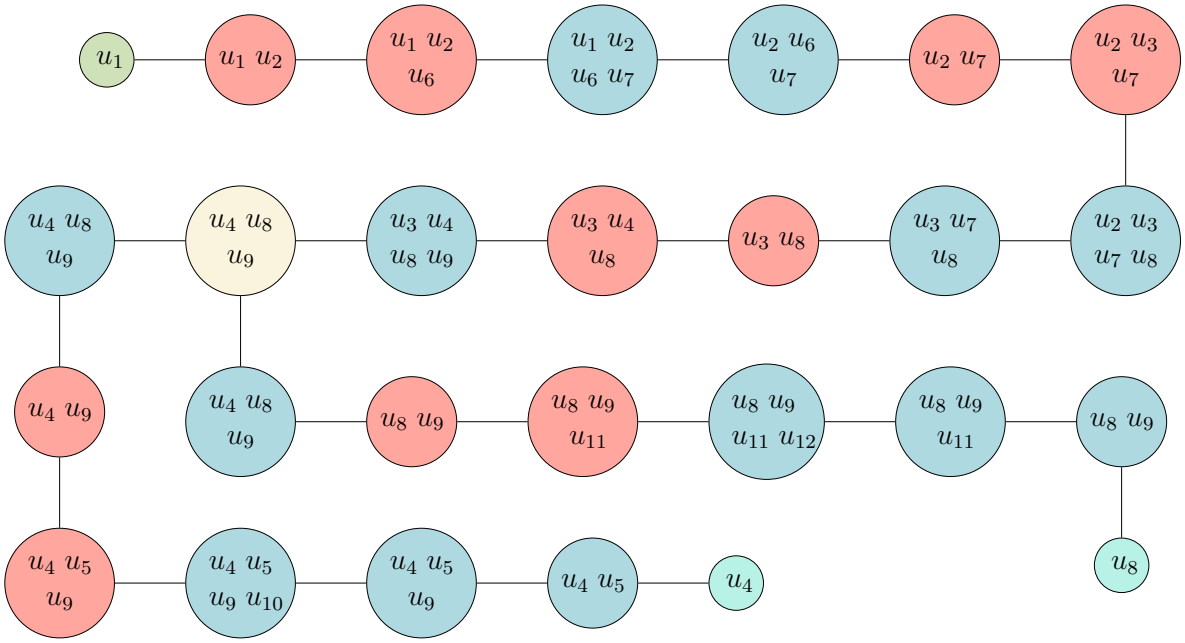


Figure 2.4: Example of a nice tree decomposition.

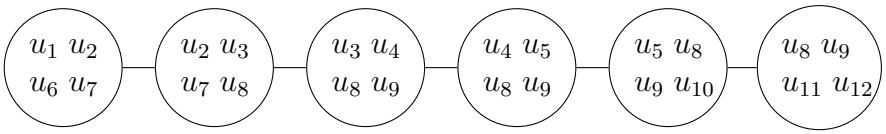


Figure 2.5: Example of a path decomposition.

any path decomposition of the graph, and is denoted by pw. A *nice path decomposition* has the same types of its tree counterpart, except for join nodes. Figures 2.5 and 2.6 present a path decomposition and its nice version of the graph of Figure 2.2.

As we have seen, nice decompositions gently introduce and forget vertices in a structured way, and thus facilitate the design of algorithms. The following theorem states that, given a decomposition, it is easy to obtain its nice counterpart of no larger width, only increasing the number of nodes.

Theorem 1 ([58, 24])**.** *Given a graph G and a tree decomposition $(\mathcal{T}, B)$ of width k, one can compute a nice tree decomposition of width k in time $\mathcal{O}(k^2 \cdot \max\{|V(\mathcal{T})|, |V(G)|\})$ that has $\mathcal{O}(k \cdot |V(G)|)$ nodes and is rooted at either a forget node or an introduce edge node.*

When solving some problems, it is useful to compute a nice tree decomposition of limited height, allowing for a small width increase.

Theorem 2. *Given a graph G and a tree decomposition $(\mathcal{T}, B)$ of width k, there is a polynomial-time algorithm that produces a nice tree decomposition of G of width at most $3k + 2$ and height $\mathcal{O}(k \log |V(\mathcal{T})|)$.*

Proof. Using the algorithm of Bodlaender and Hagerup [27], one can turn the given tree decomposition into a binary one $(\mathcal{T}', B')$ of width at most $3k+2$ and height $\mathcal{O}(\log |V(\mathcal{T})|)$. The niceness property is obtained by creating bags to introduce and forget vertices: for

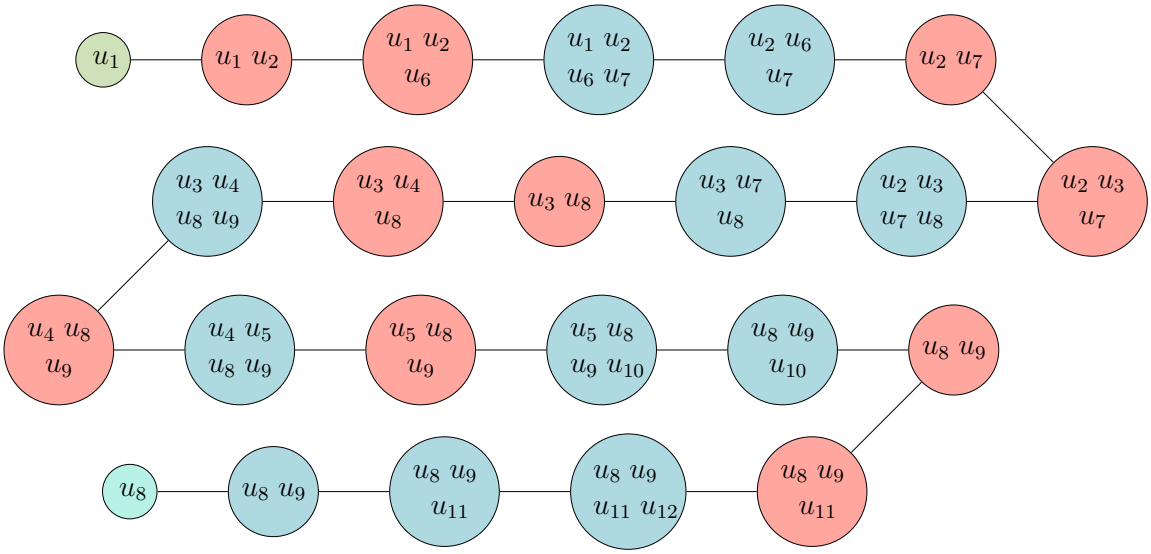


Figure 2.6: Example of a nice path decomposition.

each edge $(t, t') \in E(\mathcal{T}')$, we gradually turn B'_t into $B'_{t'}$, one vertex at a time. In each step, at most $2k$ bags are created. The resulting is a nice tree decomposition of G of width at most $3k + 2$ and height $\mathcal{O}(k \log |V(\mathcal{T})|)$. □

For a graph G and a node t of a nice tree decomposition $(\mathcal{T}, B)$ of G, let $V_t \subseteq V(G)$ be the subset of vertices introduced in the subtree rooted at t, including itself. It is useful to define a subgraph G_t of G, defined as

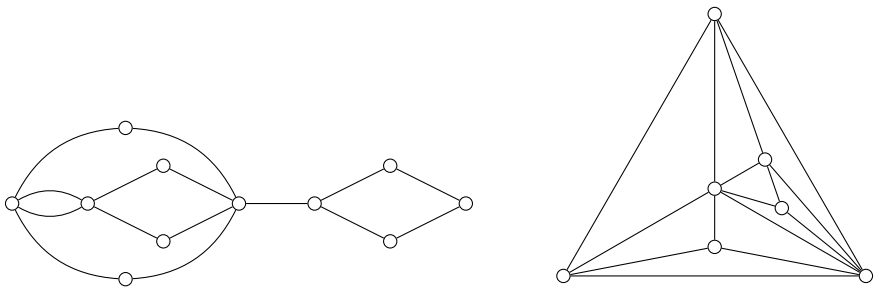
$$G_t = (V_t, E_t = \{e \in E(G) : e \text{ is introduced in the subtree rooted at } t\}).$$

Graphs of bounded treewidth A *k-tree* is a family of graphs defined inductively: a $(k + 1)$-clique is a k-tree, and a new one can be obtained by adding a vertex connected to k vertices that form a clique. A *partial k-tree* is a subgraph of a k-tree. The former is the family of graphs with treewidth at most k and the latter is the family of graphs with treewidth exactly k, i.e., any new edge would increase this value [58].

For a fixed k, the family of partial k-trees can be defined by a set of forbidden minors, where a minor of a graph G is a graph obtained from G by performing vertex and edge deletions and edge contractions [152]. The forbidden minors for $k = 1$ and $k = 2$ are K_3 and K_4, respectively; for $k = 3$, there are four forbidden minors. All of the following families have treewidth at most 3: outerplanar graphs, cactus graphs, series-parallel graphs, Halin graphs and Apollonian networks [21] (see Figure 2.7).

Some types of graphs existent in real-world settings have been observed to have low treewidth. Planken et al. [149] tackled the problem of computing all-pairs shortest paths[7], giving a quadratic-time algorithm for graphs of constant treewidth. Among the types of networks in the dataset, they considered subgraphs of the streets of New York with up to 5000 vertices. These graphs had tree decompositions of width around 30, which is in the range of practical tractability.

[7]Note this problem is in P.



(a) A series-parallel graph. (b) An Apollonian network.

Figure 2.7: Graphs of bounded treewidth.

A study on the treewidth estimation of large-scale instances (containing millions of vertices) found that graphs representing infrastructure such as public roads, bus routes and power grids often have low treewidth [131]. Another example is in the field of bioinformatics, in which graphs modelling macromolecules were observed to have treewidth 10 or lower [132].

Monadic Second-Order Logic The formalism of Monadic Second-Order Logic (MSO_2) describes properties of undirected graphs using quantifiers over elements and sets of elements. In the following, we show some examples (for a complete exposition refer to [58], Section 7.4). The following formula defines 3-Colorability, where, for a vertex u and an edge e of a graph G, $\texttt{inc}(u, e)$ is true if u is an endpoint of e.

$$
\begin{aligned}
\texttt{adj}(u, v) &\coloneqq (u \neq v) \wedge (\exists_{e \in E(G)} \texttt{inc}(u, e) \wedge \texttt{inc}(v, e)) \\
\texttt{independent}(X) &\coloneqq \forall_{u,v \in X} \neg \texttt{adj}(u, v) \\
\texttt{partition}(X_1, X_2, X_3) &\coloneqq \forall_{v \in V(G)} ((v \in X_1 \wedge v \notin X_2 \wedge v \notin X_3) \\
&\qquad \vee (v \notin X_1 \wedge v \in X_2 \wedge v \notin X_3) \\
&\qquad \vee (v \notin X_1 \wedge v \notin X_2 \wedge v \in X_3)) \\
\texttt{3colorability}(G) &\coloneqq \exists_{X_1, X_2, X_3 \subseteq V(G)} \texttt{partition}(X_1, X_2, X_3) \\
&\qquad \wedge \texttt{independent}(X_1) \\
&\qquad \wedge \texttt{independent}(X_2) \\
&\qquad \wedge \texttt{independent}(X_3)
\end{aligned}
$$

Many problems on graphs are expressible in MSO_2, such as Hamiltonian Cycle and Vertex Cover [56, 58]. An interesting use of this formalism is to show that a problem admits a parameterized algorithm on graphs of bounded treewidth.

Theorem 3 (Courcelle's Theorem [55])**.** *Assume that ϕ is a formula of* MSO_2 *and G is an n-vertex graph equipped with evaluation of all the free variables of ϕ. Suppose, moreover, that a tree decomposition of G of width t is provided. Then, there exists an algorithm that verifies whether ϕ is satisfied in G in time $f(||\phi||, t) \cdot n$, for some computable function f.*

Unfortunately, the running times of algorithms devised this way are composed of a tower of exponentials of unbounded height, and thus are prohibitive in practical scenarios.

However, the result is still of great importance, as one can quickly classify a problem by formulating it in MSO_2.

2.4.1 Computing the treewidth of a graph

As we have seen, the treewidth of a graph is the minimum width of any tree decomposition. Computing this value, however, is an NP-hard problem [7]; still, we need to obtain low-width tree decompositions for both our theoretical and practical uses. Bodlaender [22] gave an algorithm that, given a graph G an integer k, either outputs a tree decomposition of width k or concludes that the treewidth of G is greater than k; the running time of the algorithm is $\mathcal{O}^*(k^{\mathcal{O}(k^3)})$, thus, the problem is FPT. Determining the treewidth of a graph was one of the first problems to be analyzed under the lens of parameterized approximation, exchanging solution quality for time efficiency [127].

Over the years, many algorithms were developed with this objective, balancing the approximation factor, parameter and instance size dependencies on the running time (see Table 2.2 for a non-exhaustive list).

Paper	Approx. factor	$f(k)$	$g(n)$	Year
Arnborg et al. [7]	exact	$\mathcal{O}(1)$	n^{k+2}	1987
Lagergren [120]	$8k+7$	$2^{\mathcal{O}(k \log k)}$	$n \log^2 n$	1996
Bodlaender [22]	exact	$k^{\mathcal{O}(k^3)}$	n	1996
Fomin et al. [87]	exact	$\mathcal{O}(1)$	1.7347^n	2015
Bodlaender et al. [25]	$5k+4$	$2^{\mathcal{O}(k)}$	n	2016
Korhonen [115]	$2k+1$	$2^{\mathcal{O}(k)}$	n	2022

Table 2.2: Algorithms for computing treewidth.

There has been interest in devising heuristics to compute tree decompositions, since these are also used in multiple large-scale scenarios, such as compiler optimization [159] and shortest path computation [149]. Bodlaender and Koster [29] gave a detailed description of the main heuristics and further treewidth characterizations, such as triangulations and elimination orderings.

The PACE Challenge took a step in this direction, prompting researchers to compute treewidth in exact and heuristic tracks, with the objective of decreasing the gap between theory and practice [61]. The proposed instances have up to 1000 vertices in the exact track and up to 10 million vertices in the heuristic track, derived from road networks, satisfiability and probabilistic inference settings. The submitted solutions have since been used in various works [116, 12, 81]. Some applications also benefit from the computation of a lower bound on the treewidth of a graph, that, together with an upper bound given by a tree decomposition, form a treewidth estimation [131].

For our theoretical interests, we can frequently assume that a tree decomposition is given as part of the input, since we can use Bodlaender's algorithm to compute an exact one. As we have seen, this algorithm has only tw as parameter, thus, it does not spoil the status of a problem being FPT in respect to tw. When researchers are looking for the faster algorithm solving a problem parameterized by treewidth, obtaining a tree decomposition is also left out for better comparison.

2.4.2 Other parameters

For our main algorithmic results, we focus on treewidth; however, there exists many other parameters, each with its strengths and weaknesses. In the following, we briefly introduce some of these parameters.

Clique-width (cw) is a structural parameter that resembles treewidth, but kinder to dense graphs. Its decomposition is defined by four operators: creating a new vertex with a label, disjoint union of graphs, creating edges between every two vertices of a pair of labels and renaming a label. The clique-width of a graph is the minimum number of labels needed to construct it with these operators [57]. For unbounded clique-width graphs, it is NP-hard to obtain a decomposition; when the clique-width is bounded, there exists an approximation algorithm [85].

Treedepth (td) is a structural parameter that indicates the star-likeness of a graph. It is defined as the minimum height of a forest such that every edge of the original graph connects a pair of vertices in the forest having an ancestor-descendant relationship. It can be computed in polynomial time for some types of graphs [63] and there exists an approximation algorithm for general graphs [26].

A *feedback vertex set* (fvs) of a graph is a minimum set of vertices one needs to remove so the graph has no cycles, and a *vertex cover* (vc) of a graph is a minimum set of vertices that includes at least one endpoint of every edge. Both problems were shown to be NP-complete by Karp [109], and their use in parameterized complexity is extensive [58]. These graph parameters are often used to obtain parameterized algorithms for problems in which treewidth and other parameters are not strong enough [79, 114]. Other parameters we mention are twinwidth, branchwidth and rankwidth [31, 62, 144].

Figure 2.8 brings the relationships between some of these parameters, e.g., treewidth inherits algorithmic results from clique-width and hardness results from feedback vertex set, pathwidth, treedepth and vertex cover.

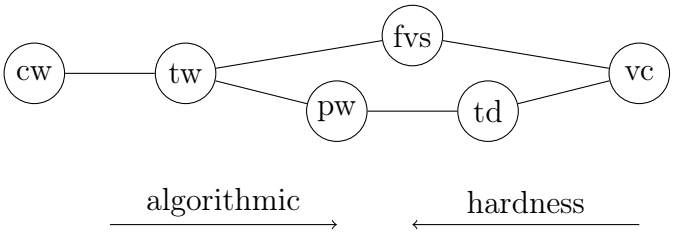


Figure 2.8: Relationships between parameters.

2.5 Dynamic programming over a tree decomposition

The design technique of *dynamic programming* is used all over computer science with the same two principles, being optimal substructure, in which an optimal solution can be obtained by combining optimal solutions for smaller problems, and overlapping subproblems, when a given subproblem is considered several times to build solutions for bigger problems [53]. These concepts align perfectly with our setting, allowing us to build a dynamic programming algorithm over a (nice) tree decomposition. A dynamic programming

table is encoded by a given node of the tree decomposition and, often times, some useful information about the produced solution; then, a solution for the global problem is found at the root of the tree decomposition [58]. This strategy has been around for several decades, when researchers generalized algorithmic results on trees for classic problems, such as Vertex Cover, Dominating Set and Independent Set [20, 8].

An algorithm designer uses the fact that any node of the tree decomposition is a separator of the subtree rooted at this node to the rest of the graph, then any choice made within this subtree is independent from the solutions for higher nodes, depending on the problem being solved. An algorithm is given by describing an specific procedure for each type of node of a nice tree decomposition: these are generally simple, since there are only a limited number of ways the vertices in the parent bag can interact with its children. Then, if the table has a limited number of entries, mainly dependent on the parameter, and each procedure takes FPT time, we obtain a parameterized algorithm by following the standard steps in dynamic programming of using the bottom-up approach and memoization.

According to Cygan et al. [58], the challenge in devising a dynamic programming algorithm over a tree decomposition boils down to giving a precise definition of a table state, indexed by a node of the decomposition and some information about the solution, together with the recursive formulas calculating its value. The correctness of the algorithm follows by showing two types of inequalities, for each node t of the decomposition: how an optimal solution for t relates with some solutions for its children, and how optimal solutions for its children relate with a solution for t.

Many problems admit algorithms parameterized by treewidth, such as Vertex Cover, Weighted Independent Set, Dominating Set and Max Cut, running in time $\mathcal{O}^*(2^{\mathcal{O}(\mathrm{tw})})$. Using this strategy, problems with connectivity requirements, such as Steiner Tree, Hamiltonian Path, Connected Dominating Set and Cycle Packing, admit algorithms running in time $\mathcal{O}^*(\mathrm{tw}^{\mathcal{O}(\mathrm{tw})})$ [58]. In Chapter 5, we use the framework of rank-based approach of Bodlaender et al. [24] to give faster algorithms for the connectivity problems we consider.

2.5.1 An example algorithm: Weighted Independent Set

In this subsection, we go over the algorithm for Weighted Independent Set, parameterized by treewidth, to illustrate the aforementioned directions on the design of this type of algorithm. In this problem, the input is a graph G and a vertex-weight function $w : V(G) \to \mathbb{R}_{\geq 0}$, and the objective is to find an independent set of maximum weight, that is, a subset of $V(G)$ such that no two vertices are adjacent. Recall that we also receive a nice tree decomposition $(\mathcal{T}, B)$ of G as input.

Given a node $t \in V(\mathcal{T})$ and $S \subseteq B_t$, the subproblem definition asks for a maximum-weight extension $\hat{S}$ such that $S \subseteq \hat{S} \subseteq V_t$, $S = \hat{S} \cap B_t$ and $\hat{S}$ is an independent set. Let $A_t(S)$ be an entry of this table. For a *leaf node* t, we have $A_t(\emptyset) = 0$. For an *introduce node* t with child t', such that $B_t = B_{t'} \cup \{u\}$, for $u \notin B_{t'}$, we have three cases: S is not an independent set, u is in S or u is not in S. Note that the first case considers the

possibility $u \in S$ and $N(u) \cap S \neq \emptyset$. Then,

$$A_t(S) = \begin{cases} -\infty & \text{if } S \text{ is not independent,} \\ A_{t'}(S) & \text{if } u \notin S, \\ A_{t'}(S \setminus \{u\}) + w(u) & \text{otherwise.} \end{cases}$$

For a *forget node* t with child t', such that $B_t = B_{t'} \setminus \{u\}$, for $u \in B_{t'}$, we take the best solution for t' either including or excluding u from S, considering S is valid. Then,

$$A_t(S) = \begin{cases} -\infty & \text{if } S \text{ is not independent,} \\ \max\{A_{t'}(S), A_{t'}(S \cup \{u\})\} & \text{otherwise.} \end{cases}$$

For a *join node* t with children t' and t'', such that $B_t = B_{t'} = B_{t''}$, we take the best solution for each child and subtract the weight of vertices in S, given S is valid. Then,

$$A_t(S) = \begin{cases} -\infty & \text{if } S \text{ is not independent,} \\ A_{t'}(S) + A_{t''}(S) - w(S) & \text{otherwise.} \end{cases}$$

The computation of the table starts from each leaf node, followed by other node types, as soon as their children have been calculated. Considering t_r is the root of the tree decomposition, a maximum-weight independent set of G can be found by computing $A_{t_r}(\emptyset)$. For a complete exposition, including the correctness proof, running time calculation and implementation details, see [58].

2.5.2 Storing large integers

For many algorithms parameterized by some graph width, such as treewidth, pathwidth or clique-width, a good portion of their running times come from the fact that they calculate and store large integers as part of their solutions. These integers may represent distances between nodes or the size of a set of vertices, but the common ground among these uses is that they are calculated by adding previous entries, stored on the dynamic programming table. A standard technique to improve the running time of such an algorithm is to store these integers approximately, effectively shrinking the table and consequently the search space [30].

For some $\delta > 0$, each value is rounded to the closest integer power of $(1 + \delta)$, then, by storing only the integers corresponding to the powers, one reduces a table with r^w entries to a table with $(\log_{(1+\delta)} r)^w$ entries, given that we want to represent integers in $\{1, \ldots, r\}$ and have w entities. A downside to this approach is that rounding errors pile up as the algorithm proceeds to higher nodes in the decomposition, and that some precision is lost. One can show that these errors are contained directly by the algorithm or apply the framework of approximate addition trees [121]. We used the former strategy in Chapter 3 and the latter in Chapter 4.

An *addition tree* is an abstract model that represents the computation of a number by successively adding two other previously computed numbers.

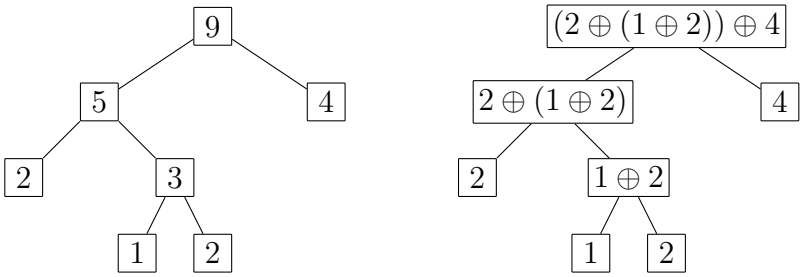


Figure 2.9: Addition tree and a corresponding approximate addition tree.

Definition 1. *An* addition tree *is a full binary tree such that each leaf u is associated to a non-negative integer input y_u, and each internal node u with children u' and u'' is associated to a computed number $y_u \coloneqq y_{u'} + y_{u''}$.*

One can replace the sum with some operator $\oplus$, which computes each such sum only approximately, up to an integer power of $(1+\delta)$, for some parameter $\delta > 0$. The resulting will be an *approximate addition tree* (see Figure 2.9).

Definition 2. *An* approximate addition tree *with parameter $\delta > 0$ is a full binary tree, where each leaf u is associated to a non-negative integer input z_u, and each internal node u with children u' and u'' is associated to a computed value $z_u \coloneqq z_{u'} \oplus z_{u''}$, where $a \oplus b \coloneqq 0$ if both a and b are zero, and $a \oplus b \coloneqq (1+\delta)^{\lceil \log_{1+\delta}(a+b) \rceil}$, otherwise.*

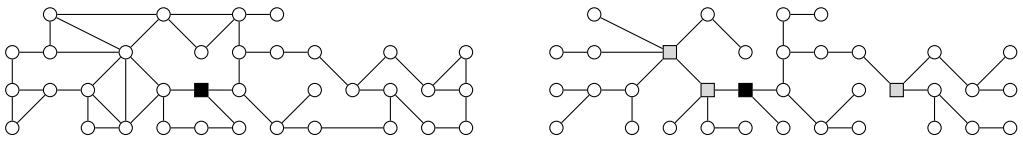
For simplicity, here we defined only a deterministic version of the approximate addition tree, since we can assume that the height of the decomposition is bounded by $\mathcal{O}(\mathrm{tw} \cdot \log |V(G)|)$. While the error of the approximate value can pile up as more operations are performed, Lampis [121] showed that, for some $\epsilon > 0$, as long as δ is not too large, the relative error can bounded by $1+\epsilon$.

Theorem 4 ([121])**.** *Given an approximate addition tree of height ℓ, if $\delta < \frac{\epsilon}{2\ell}$, then, for every node u of the tree, we have* $\max\left\{\frac{z_u}{y_u}, \frac{y_u}{z_u}\right\} < 1+\epsilon$.

Chapter 3

Star k-Hub Center

In the Star k-Hub Center (SkHC), we are given a connected edge-weighted graph and a special center vertex. The task is to select a set of hubs of size k, and define an assignment of vertices to hubs such that every pair of vertices is connected by a path going through their hubs and possibly the center. A path connecting vertices that are assigned to the same hub must visit only this hub, otherwise, it must visit the hubs and the center. The objective is to minimize the length of the longest path in a solution. An example of an instance and a corresponding solution are depicted in Figure 3, where the center is represented as a black square and the selected hubs as gray squares.



(a) Input graph with center. (b) Solution using 3 hubs.

Figure 3.1: An instance and a solution of SkHC.

This problem models a two-level hub system, where hubs are connected to a given center and each non-hub vertex must communicate with all other vertices through their assigned hubs. Historically, a solution has been described as a tree of depth two, rooted at the center, which means that non-hub vertices are directly connected to the hubs [165]. In practice, however, the input graph may not be complete, thus we allow the path between a pair of vertices to contain multiple vertices; note that one can obtain an instance to the mentioned version of the problem by computing the metric closure of the input graph. In the following, we assume that G is an edge-weighted undirected graph with non-negative integer weights. For vertices u and v of G, the length of a minimum-weight path from u to v is denoted by $d(u, v)$, and we assume, w.l.o.g., that $d(u, v) > 0$, for $u \neq v$, as otherwise one could simplify the graph. For vertices $(u_1, u_2, \ldots, u_p)$, define $d(u_1, u_2, \ldots, u_p) = d(u_1, u_2) + d(u_2, u_3) + \cdots + d(u_{p-1}, u_p)$.

Formally, an instance is composed of a connected edge-weighted graph G with n vertices, a center vertex $c \in V(G)$ and integers k and r. A solution is a pair (H, ϕ), consisting of a set of hubs $H \subseteq V(G) \setminus \{c\}$ of size k and an assignment of vertices to hubs $\phi : V(G) \setminus \{c\} \to H$, such that $\phi(u) = u$, for $u \in H$, and

$$\max_{u,v \in V(G) \setminus \{c\}} d(P_\phi(u,v)) \leq r,$$

where $P_\phi(u,v) = (u, \phi(u), c, \phi(v), v)$, if $\phi(u) \neq \phi(v)$, and, otherwise, $P_\phi(u,v) = (u, \phi(u), v)$.

This setting has applications in many-to-many distribution systems that considers a connection hierarchy, e.g., the backbone of the network and the access links, connected to the center and the vertices, respectively [73, 3]. This is often referred as a *star-star network*, since the center and its links to hubs form a star graph, as well as each hub and its links to vertices [165]. Helme and Magnanti [97] gave a practical application, where they designed a two-level satellite network. In this network, each pair of satellites communicate locally if they are connected to the same switch, and otherwise, they use an Earth station, connected to each of their switches.

This is one of the many hub location problems, a type of problem modeling transportation systems that are organized through hubs, which are intermediate transshipment entities that routes goods between origin–destination pairs [73, 4].

Related works The first works on two-level hub networks go back several decades, with Mirzaian [136] and Garcia [89], that presented integer linear programming formulations to build this type of network, considering installation and operating costs. More exact formulations were considered by Labbé and Yaman [119], that studied a version of SkHC that minimizes the cost of locating and assigning hubs plus a cost per distance of each link. Some adjacent problems have been considered in a multi-level network structure [37, 43, 51, 33, 94].

Yaman and Elloumi [165] introduce SkHC and a related problem that aims to minimize the total routing cost of the network, i.e., the sum of every path that connects two vertices, while maintaining the cost of each path below a given threshold. Both problems are concerned with generating a telecommunication network that values service quality, albeit with distinct objective functions. They present an NP-hardness proof of SkHC and give integer linear programming formulations for both problems, that are experimentally tested on instances with up to 50 vertices.

The problem was first considered in the approximation setting by Liang [126], who gave a 3.5-approximation algorithm. This is a purely combinatorial algorithm, using a guessing step to obtain relevant link costs, and, as an auxiliary algorithm, a 2-approximation of a version of k-Center that forbids centers. Also, this work shows the first NP-hardness proof for SkHC in metric spaces: unless P = NP, there is no $(1.25 - \epsilon)$-approximation algorithm, for any $\epsilon > 0$, with a reduction from Dominating Set.

This result was later strengthened by Chen et al. [48], proving that it is NP-hard to obtain an approximation factor of $1.5 - \epsilon$, for any $\epsilon > 0$, with a reduction from Set Cover. They also presented two algorithmic results, improving the previous one. The first is a linear-time 2-approximation that picks the k closest vertices to the center, and connects every other vertex to the closest hub. The second is a $\frac{5}{3}$-approximation running in time $\mathcal{O}(kn^4)$, that returns the best solution of two other algorithms, with opposing strategies to pick hubs. The variant in which there is no center vertex and each pair is connected only through its hubs is studied by Chen et al. [47], that provide similar

approximation algorithms. A problem related to SkHC is the well-studied k-Center, which receives an edge-weighted graph G and an integer k as input. The objective is to find a set of centers $K \subseteq V(G)$ of size k such that $\max_{u \in V(G)} \min_{v \in K} d(u, v)$ is minimized. A simple greedy algorithm has approximation factor 2, which is the best possible, unless $\mathsf{P} = \mathsf{NP}$ [95].

The lower bounds imposed on approximation algorithms for problems such as SkHC or k-Center imply that, unless $\mathsf{P} = \mathsf{NP}$, one cannot find optimal or even near-to-optimal solutions in polynomial time. An alternative is designing parameterized algorithms. The goal is to solve the problem exactly in super-polynomial time, restricting the non-polynomial factors of the running time to depend only on a parameter of the input [58]. It turns out that many location problems do not admit this type of algorithm for natural parameters, such as k-Center, that is $\mathsf{W[1]}$-hard for planar graphs of constant doubling dimension when the parameter is a combination of k, the highway dimension, the pathwidth and the skeleton dimension of the graph [19].

In the last decade, the strategy of combining parameterized and approximation algorithms to obtain near-to-optimal solutions has been gaining momentum [121, 111, 74, 76, 75]. For example, Katsikarelis et al. [111] presented an approximation scheme for k-Center parameterized by tw, the treewidth of the graph. They showed that, for any $\epsilon > 0$, there is an algorithm that runs in FPT time and produces a solution with k centers and value at most $(1 + \epsilon)r$, whenever a solution with k centers and value r exists. This technique is particularly useful when a problem has some form of intractability even in a parameterized setting, which is circumvented by computing an approximate solution in FPT time.

Our results and techniques We initiate the study of SkHC in the parameterized framework. We show that the problem is $\mathsf{W[1]}$-hard when the parameter is vc and k, a combination of the vertex cover number of the graph and the number of hubs, by using a reduction from k-Center. Recall that vc is an upper bound on tw, thus the hardness holds when SkHC is parameterized by tw and k as well. We also note that the problem does not admit a parameterized $(1.25 - \epsilon)$-approximation algorithm, for any $\epsilon > 0$, when the parameter is k. This is derived from the standard approximation hardness [126].

Two algorithmic results are presented. We give a parameterized algorithm, with parameter tw and r, that either produces a solution for SkHC of size k and value at most r, or decides that no such solution exists. The algorithm uses the technique of dynamic programming over a tree decomposition, and actually solves a slightly modified problem with a different structure. This problem additionally takes as input two functions relating vertices to distances: the first is an upper bound on the path connecting the vertex to the center, and the second is an upper bound on the path connecting the vertex to its assigned hub. We guess important distances of an optimal solution and reduce an instance of SkHC to an instance of the modified problem, which simplifies our algorithm. This is possible because every pair of vertices must be connected by a path of length at most r, since there is a unit of demand between every two vertices of the graph.

Building on the previous result, we give an efficient parameterized approximation scheme with parameter tw. More precisely, for any $\epsilon > 0$, we give an algorithm that finds

a solution with k hubs and value at most $(1+\epsilon)r$, or concludes that every solution has value greater than r. The algorithm runs in time $\mathcal{O}^*((\mathrm{tw}/\epsilon)^{\mathcal{O}(\mathrm{tw})})$. The ingredient to remove r as a parameter is to store the distances in the dynamic programming table in a approximate manner, as discussed in Subsection 2.5.2, as well as using a tree decomposition of limited height.

The remaining of this chapter is organized as follows: Section 3.1 brings our hardness results, Section 3.2 describes the algorithm parameterized by tw and r, Section 3.3 gives the EPAS with only tw as parameter, building on the previous algorithm, and Section 3.4 points future research directions.

3.1 Hardness results

As we have seen, SkHC is hard to approximate in polynomial time. More specifically, Chen et al. [48] established that it is NP-hard to obtain a $(1.5-\epsilon)$-approximation, for any $\epsilon > 0$. A possible strategy to improve the approximation factor is to consider an algorithm running in FPT time, that efficiently solves instances whose parameter is small. A natural choice is the number of hubs k. We note that, using the reduction of Liang [126] from Dominating Set, an algorithm parameterized by k could only achieve an approximation factor of 1.25.

Theorem 5. *For any $\epsilon > 0$, computing a $(1.25-\epsilon)$-approximation for* SkHC *parameterized by k is* W[2]*-hard.*

This lower bound prompts us to explore other parameters, as the goal is to get an optimal or close-to-optimal solution. A common choice among the structural parameters is the treewidth of the graph. The main result of this chapter is an efficient parameterized approximation scheme for SkHC, with treewidth as parameter. We present a hardness lower bound proving this is the best algorithmic result one could hope for, when parameterized by treewidth. More than that, it states the problem is W[1]-hard even if we consider as parameter the combination of the vertex cover number of the graph and the number of hubs. Since the treewidth of a graph is bounded by its vertex cover number, i.e., tw $\leq$ vc [77], this implies the problem is also W[1]-hard when parameterized by tw and k. We use the following hardness result of k-Center.

Theorem 6 ([111])**.** k-Center *is* W[1]*-hard parameterized by* vc *and k. Furthermore, if there is an algorithm for the problem in time $n^{o(\mathrm{vc}+k)}$, then* ETH *is false.*

Theorem 7. SkHC *is* W[1]*-hard parameterized by* vc *and k.*

Proof. We reduce k-Center to SkHC. Let $I = (G, k, r)$ be an instance for the decision version of k-Center. We build an instance $I' = (G', c, k', r')$ for the decision version of SkHC. Starting with G, add vertices h and z connected by an edge of weight r. Also, add a vertex c and an edge from c to each vertex $u \in V(G) \cup \{h\}$ of weight $r+1$. Observe that $d(h, z) = r$ and $d(u, c) = r+1$, for each $u \in V(G) \cup \{h\}$. Figure 3.2 shows the resulting graph G'. Now, let $k' = k+1$ be the number of hubs and $r' = 4r+2$ be the

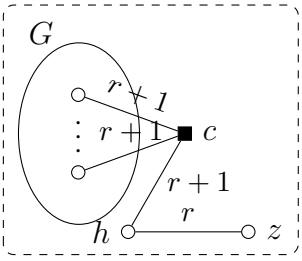


Figure 3.2: Graph G'.

bound on the value of a solution. We will show that I is a YES-instance if, and only if, I' is a YES-instance.

Let $K \subseteq V(G)$ be an optimal solution for I, such that $\max_{u \in V(G)} \min_{v \in K} d(u, v) \leq r$. Thus, there is a assignment $\phi : V(G) \to K$, such that $d(u, \phi(u)) \leq r$, for every $u \in V(G)$. Let $H = K \cup \{h\}$ be a set of $k + 1 = k'$ vertices, and define an assignment $\phi' : V(G') \setminus \{c\} \to H$ such that $\phi'(u) = h$ if $u \in \{h, z\}$, and otherwise $\phi'(u) = \phi(u)$. For a pair of vertices $u, v \in V(G') \setminus \{c\}$ with $\phi'(u) \neq \phi'(v)$, we have $d(u, \phi'(u), c, \phi'(v), v) \leq 4r+2$, since the distance between every hub and the center is exactly $r+1$ and the distance between every vertex and its hub is at most r. For the case in which $\phi'(u) = \phi'(v)$, we have $d(u, \phi'(u), v) \leq 2r$. Thus, I' is indeed a YES-instance.

Now, let H be a set of $k' = k + 1$ vertices, and $\phi' : V(G') \setminus \{c\} \to H$ be an assignment such that $d(u, \phi'(u), c, \phi'(v), v) \leq 4r + 2$, for every $u, v \in V(G')$, if $\phi'(u) \neq \phi'(v)$, and otherwise $d(u, \phi'(u), v) \leq 4r + 2$. Note that $\phi'(z) = h$, as otherwise we would have $d(z, \phi'(z), c, \phi'(u), u) > 4r + 2$, for some $u \in V(G) \setminus \{c\}$. Thus, one of the hubs must be h. Using similar arguments, z cannot be the hub assigned to h. As before, the distance between every hub and the center is exactly $r + 1$, then, for every $u \in V(G)$, we have $d(u, \phi'(u)) \leq 4r + 2 - d(\phi'(u), c) - d(c, \phi'(z)) - d(\phi'(z), z) = r$. We conclude that H induces a solution for I, and thus I is a YES-instance.

Finally, recall that k-CENTER is W[1]-hard parameterized by vc and k (Theorem 6) and observe that the reduction increases the vertex cover number of G by at most 2. Therefore, SkHC is W[1]-hard parameterized by vc and k. □

3.2 A parameterized algorithm

In this section, we present an algorithm for SkHC parameterized by tw and r that runs in time $\mathcal{O}^*(r^{\mathcal{O}(\text{tw})})$. More precisely, for an instance (G, c, k, r), we either find a set of k hubs and a corresponding assignment such that the path between each pair of vertices through the assigned hubs (and possibly the center) has length at most r, or decide that no such solution exists. Note that this path passes through the center if, and only if, the vertices are assigned to distinct hubs. The algorithm is based on a dynamic programming approach over a nice tree decomposition. We use this result as a foundation for the efficient parameterized approximation scheme presented in Section 3.3.

3.2.1 Reduction to a simplified problem

In order to define a dynamic programming table, rather than considering SkHC directly, we solve a slightly different problem, defined next. This formulation relies on the property that every pair of vertices has unit demand, which allows us to guess important vertices and distances of an optimal solution.

Definition 3. *Let $\ell_1, \ell_2 : V(G) \setminus \{c\} \to \mathbb{Z}_{\geq 0}$. An assignment $\phi : V(G) \setminus \{c\} \to H$, where $H \subseteq V(G) \setminus \{c\}$,* respects *$\ell_1$ and ℓ_2 if, for every $u \in V(G) \setminus \{c\}$, we have:*

$$d(u, \phi(u), c) \leq \ell_1(u) \quad \text{and} \quad d(u, \phi(u)) \leq \ell_2(u).$$

In the Modified Star k-Hub Center (MSkHC), an instance is composed of a graph G with center c, a number k and functions ℓ_1 and ℓ_2. A solution is an assignment of vertices to hubs, opening k hubs, that respects ℓ_1 and ℓ_2. Figure 3.3 shows the distances of a vertex to the center and to its hub, relating to ℓ_1 and ℓ_2.

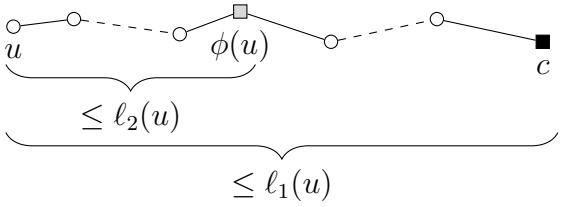


Figure 3.3: Relationship between distances and functions ℓ_1 and ℓ_2.

We show how to solve an instance of SkHC with an algorithm that solves the corresponding instance of this modified version.

Lemma 1. *Let I be an instance of* SkHC *and I' be the corresponding instance of* MSkHC*. It is possible to solve I in time $\mathcal{O}(n^4 \cdot f(I'))$, given an algorithm that solves I' in time $f(I')$.*

Proof. Let ϕ be the assignment of vertices to hubs of an optimal solution for an instance $I = (G, c, k, r)$ of SkHC. Let $u_1 = \arg\max_{u \in V(G) \setminus \{c\}} d(u, \phi(u), c)$ be the farthest vertex from the center and $u_2 = \arg\max_{u \in V(G) \setminus \{c\}} d(u, \phi(u))$ be the farthest vertex from its hub in this optimal solution. Moreover, let $\gamma_1 = d(u_1, \phi(u_1), c)$ and $\gamma_2 = d(u_2, \phi(u_2))$ be the respective distances. These can be found in a guessing step in time $\mathcal{O}(n^4)$. The instance of MSkHC is $I' = (G, c, k, \ell_1, \ell_2)$, created as follows. Let $N = \{u \in V(G) \setminus \{c\} : d(u, \phi(u_1)) \leq d(u_1, \phi(u_1))\}$. For every $u \in N$, we have $\ell_1(u) = \gamma_1$, and for all others, we have $\ell_1(u) = r - \gamma_1$. If $\gamma_2 \leq r/2$, then $\ell_2(u) = \gamma_2$ for every u, otherwise, $\ell_2(u_2) = \gamma_2$ and $\ell_2(u) = r - \gamma_2$, for every $u \neq u_2$. This setting is depicted in Figure 3.4.

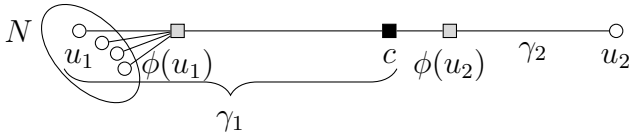


Figure 3.4: Important vertices and distances of an optimal solution.

Let (H, ϕ) be a solution for I. From construction, we know u_1 and u_2. We build a solution (H, ϕ') that respects ℓ_1 and ℓ_2. Let $\phi'(u) = \phi(u_1)$ for $u \in N$, then $d(u, \phi'(u), c) \leq d(u_1, \phi'(u_1), c) = \gamma_1 = \ell_1(u)$. For every other u, let $\phi'(u) = \phi(u)$, then, as this solution has value r, we have $d(u, \phi'(u), c) \leq r - d(u_1, \phi'(u_1), c) = r - \gamma_1 = \ell_1(u)$. If $\gamma_2 \leq r/2$, then $d(u, \phi'(u)) \leq \gamma_2 = \ell_2(u)$, for every u. Now, assume $\gamma_2 > r/2$. If u and u_2 are connected to the same hub, then $d(u, \phi'(u)) \leq r - d(u_2, \phi'(u_2)) = r - \gamma_2 = \ell_2(u)$. If u and u_2 have different hubs, we also have $d(u, \phi'(u)) \leq r - \gamma_2 = \ell_2(u)$, as otherwise $d(u, \phi'(u), c, \phi'(u_2), u_2) > d(u, \phi'(u)) + d(u_2, \phi'(u_2)) > r - \gamma_2 + \gamma_2 > r$, a contradiction.

For the other direction, let (H, ϕ) be a solution for I'. We prove this is a solution for I. Let u and v be vertices such that $\phi(u) \neq \phi(v)$, and assume, w.l.o.g., that $\phi(u) = \phi(u_1)$. Then, $d(u, \phi(u), c) \leq \ell_1(u) = \gamma_1$ and $d(v, \phi(v), c) \leq \ell_1(v) = r - \gamma_1$. If none of the hubs is $\phi(u_1)$, both their distances to c are bounded by $r - \gamma_1$. Since $\gamma_1 \geq r/2$, in either case the path of (u, v) costs at most r. Now, let u and v be vertices such that $\phi(u) = \phi(v)$, and assume $\gamma_2 > r/2$. Assume, w.l.o.g., that $u = u_2$, then $d(u, \phi(u)) \leq \ell_2(u) = \gamma_2$ and $d(v, \phi(v)) \leq \ell_2(v) = r - \gamma_2$, implying that the path of (u, v) costs at most r; otherwise, the path costs at most $2(r - \gamma_2) < r$. Now, assume $\gamma_2 \leq r/2$, then $d(w, \phi(w)) \leq \ell_2(w) = \gamma_2$ for every vertex w, thus the path of (u, v) costs at most r. □

3.2.2 The algorithm

In this section, we describe the algorithm in the form of recurrence relations, one for each node type of a nice tree decomposition. We suppose we receive an instance $(G, c, k, \ell_1, \ell_2)$ of MSkHC and a nice tree decomposition $(\mathcal{T}, B)$ of G of width tw.

Assume the center c is contained in the bag B_t, for each $t \in V(\mathcal{T})$, since otherwise we could add c to every bag and increase the width of the decomposition by at most one. For the ease of notation, define $X_t = B_t \setminus \{c\}$ and let V_t denote the non-center vertices of G, contained in bags of the subtree rooted at t. Also, define $G_t = G[V_t \cup \{c\}]$ as the subgraph induced by V_t and c.

Subproblem definition For each node t of the tree decomposition, we would like to find a set of hubs that corresponds to a partial solution for the global problem. Since vertices in V_t might be connected to hubs that are not introduced yet, they might be satisfied indirectly by vertices of X_t. To encode the distances from a vertex of X_t to the center and to its hub in a global solution, we consider colorings $\mathbf{c}, \mathbf{h} : X_t \to \{0, \ldots, r\}$, respectively. Thus, a value $\mathbf{c}(u)$ associated with a vertex u gives an upper bound on the length of a path from u to the center, while $\mathbf{h}(u)$ gives an upper bound on the length of a path from u to its hub. Vertices in $S \subseteq X_t$ are not connected to the center and a hub yet, they will be satisfied by higher nodes of the tree decomposition. Next, we define a partial solution.

Definition 4. *Let t be a node of the tree decomposition, and consider colorings $\mathbf{c}, \mathbf{h} : X_t \to \{0, \ldots, r\}$ and a subset $S \subseteq X_t$. A partial solution for t is a subset $H \subseteq V_t$ such that:*

(i) for $u \in X_t$, we have $u \in H$ if, and only if, $\mathbf{c}(u) = d(u, c)$ and $\mathbf{h}(u) = 0$; and

(*ii*) *for* $u \in X_t \setminus S$, *there is* $v \in H$ *such that* $d(u, v, c) \leq \mathbf{c}(u)$ *and* $d(u, v) \leq \mathbf{h}(u)$; *and*

(*iii*) *for* $u \in V_t \setminus X_t$, G_t *has a shortest path* P *from* u *to a vertex* v *such that:*

(*a*) $V(P) \cap X_t = \{v\}$, $d(u, v) + \mathbf{c}(v) \leq \ell_1(u)$ *and* $d(u, v) + \mathbf{h}(v) \leq \ell_2(u)$, *or*

(*b*) $V(P) \cap X_t = \emptyset$, $v \in H$, $d(u, v, c) \leq \ell_1(u)$ *and* $d(u, v) \leq \ell_2(u)$.

If item (*ii*) *or* (*iii*) *holds for some vertex* u, *then we say that* u *is* satisfied *by* v.

A partial solution for SkHC is depicted in Figure 3.5, where a curved edge represents a path whose internal vertices are not in X_t. Note that a regular edge may contain several internal vertices. In this example, u is satisfied directly by hub v, provided that $d(u, v, c) \leq \ell_1(u)$ and $d(u, v) \leq \ell_2(u)$. Similarly, u' is satisfied by hub v', but since $u' \in X_t$, this time we require that $d(u', v', c) \leq \mathbf{c}(u')$ and $d(u', v') \leq \mathbf{h}(u')$. Finally, u'' is satisfied indirectly by the non-hub vertex v'', provided that $d(u'', v'') + \mathbf{c}(v'') \leq \ell_1(u'')$ and $d(u'', v'') + \mathbf{h}(v'') \leq \ell_2(u'')$.

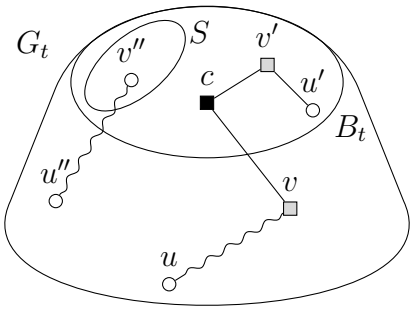


Figure 3.5: Partial solution in a node t of the tree decomposition.

Algorithm description Let A be the dynamic programming table used in the algorithm, where A_t is defined for each node t of the tree decomposition:

$$A_t : \{0, \ldots, r\}^{X_t} \times \{0, \ldots, r\}^{X_t} \times 2^{X_t} \to \mathbb{N} \cup \{\infty\}$$

An entry $A_t(\mathbf{c}, \mathbf{h}, S)$ is the minimum size of a partial solution for t w.r.t. $(\mathbf{c}, \mathbf{h}, S)$, and if there is no partial solution, then $A_t(\mathbf{c}, \mathbf{h}, S) = \infty$. Observe that finding a solution for the whole graph G corresponds to an entry of the subproblem defined at the root node with $S = \emptyset$. Next, we give recurrence relations to compute $A_t(\mathbf{c}, \mathbf{h}, S)$ for each node type.

For a *leaf node* t, the only vertex in the bag is the center, so there is no vertex to be satisfied and $A_t(\mathbf{c}, \mathbf{h}, S) = 0$ (see Figure 3.6).

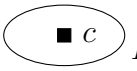


Figure 3.6: The leaf case of SkHC.

For an *introduce node* t with child t', we have $X_t = X_{t'} \cup \{u\}$, for some $u \notin X_{t'}$. Let $\mathbf{c}'$ and $\mathbf{h}'$ be the restrictions of $\mathbf{c}$ and $\mathbf{h}$ to $X_{t'}$, respectively, and $H \subseteq V_{t'}$ be the partial solution for t'. If $\mathbf{c}(u) = d(u, c)$ and $\mathbf{h}(u) = 0$, then u is a hub, and we define S' as the union of S and the vertices of $X_{t'}$ that are satisfied by u, according to item (ii) of Definition 4:

$$S' = S \cup \{v \in X_{t'} : d(v, u, c) \leq \mathbf{c}(v) \wedge d(v, u) \leq \mathbf{h}(v)\}.$$

It may be the case that $u \notin S$ and there is no hub that satisfies u, i.e., $d(u, v, c) > \mathbf{c}(u)$ and $d(u, v) > \mathbf{h}(u)$, for every $v \in H$, then we set the entry to ∞. Otherwise, there are two cases: $u \in S$ or $u \notin S$ and there is $v \in H$ such that $d(u, v, c) \leq \mathbf{c}(u)$ and $d(u, v) \leq \mathbf{h}(u)$, according to item (ii) of Definition 4. Figure 3.7 depicts the case u is chosen as a hub and the case there is a hub in H satisfying it.

(a) u is chosen as a hub.

(b) u is satisfied by a hub.

Figure 3.7: The introduce case of SkHC.

Observe that no vertex in $V_t \setminus X_t$ can be satisfied by u, as $B_{t'}$ is a separator, thus there is no direct path from such a vertex to u. We compute:

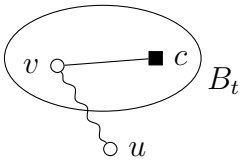
$$A_t(\mathbf{c}, \mathbf{h}, S) = \begin{cases} A_{t'}(\mathbf{c}', \mathbf{h}', S') + 1 & \text{if } u \text{ is a hub,} \\ \infty & \text{if } u \notin S \text{ and } u \text{ is not satisfied,} \\ A_{t'}(\mathbf{c}', \mathbf{h}', S') & \text{otherwise.} \end{cases}$$

Lemma 2. *The formula holds for introduce nodes.*

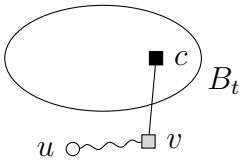
Proof. Define an indicator set U: if $\mathbf{c}(u) \neq d(u, c)$ and $\mathbf{h}(u) \neq 0$, then $U = \emptyset$, otherwise, $U = \{u\}$. Observe that a partial solution H for t induces a partial solution $H \setminus U$ for t'. To see this, note that, since $X_{t'}$ is a separator of G_t, any vertex v that is satisfied by u must be in $X_{t'}$, but then $v \in S'$. Thus, $A_{t'}(\mathbf{c}', \mathbf{h}', S') \leq A_t(\mathbf{c}, \mathbf{h}, S) - |U|$.

For the opposite direction, if H' is a partial solution for t', then $H' \cup U$ is a partial solution for t. It suffices to recall that, if there is $v \in S' \setminus S$, then v is satisfied by u. Thus, $A_{t'}(\mathbf{c}', \mathbf{h}', S') + |U| \geq A_t(\mathbf{c}, \mathbf{h}, S)$. □

For a *forget node* t with child t', we have $X_t = X_{t'} \setminus \{u\}$, for some $u \in X_{t'}$. Let $\mathbf{c}'$ and $\mathbf{h}'$ be extensions of $\mathbf{c}$ and $\mathbf{h}$ to $X_{t'}$. There are two cases: u is satisfied indirectly by some vertex of X_t, i.e., there is $v \in X_t$ with $d(u, v) + \mathbf{c}'(v) \leq \mathbf{c}'(u)$ and $d(u, v) + \mathbf{h}'(v) \leq \mathbf{h}'(u)$; or u is satisfied directly by a hub in the partial solution for t (see Figure 3.8).



(a) u is satisfied indirectly.



(b) u is satisfied by a hub.

Figure 3.8: The forget case of SkHC.

We compute:

$$A_t(\mathbf{c}, \mathbf{h}, S) = \min_{\mathbf{c}', \mathbf{h}'} \begin{cases} A_{t'}(\mathbf{c}', \mathbf{h}', S \cup \{u\}) & \text{if } u \text{ is satisfied indirectly} \\ A_{t'}(\mathbf{c}', \mathbf{h}', S) & \text{otherwise,} \end{cases}$$

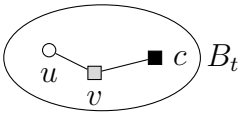
where $\mathbf{c}'$ and $\mathbf{h}'$ span over all extensions of $\mathbf{c}$ and $\mathbf{h}$ to $X_{t'}$.

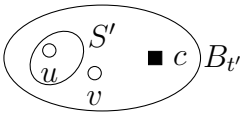
Lemma 3. *The formula holds for forget nodes.*

Proof. Let H be a partial solution for t, and suppose that u is satisfied by a vertex v. Let $\mathbf{c}'$ and $\mathbf{h}'$ be the extensions such that, if $v \in X_t$, we have $\mathbf{c}'(u) = d(u, v) + \mathbf{c}'(v)$ and $\mathbf{h}'(u) = d(u, v) + \mathbf{h}'(v)$, and, otherwise, we have $\mathbf{c}'(u) = d(u, v, c)$ and $\mathbf{h}'(u) = d(u, v, w)$, for a $w \in H$. Also, let $S' = S \cup \{u\}$ if $v \in S$, and $S' = S$, otherwise. Note that H is a partial solution for t' w.r.t. $(\mathbf{c}', \mathbf{h}', S')$, thus $A_t(\mathbf{c}, \mathbf{h}, S) \geq A_{t'}(\mathbf{c}', \mathbf{h}', S')$, which is considered in the recurrence.

For the other direction, let $A_{t'}(\mathbf{c}', \mathbf{h}', S')$ be an entry of the recurrence that achieves the minimum, and let H' be a corresponding partial solution for t'. If $u \notin S'$, then $S' = S$ and one can check that H' is a partial solution for t. Otherwise, there exists $v \in S$ such that $d(u, v) + \mathbf{c}'(v) \leq \mathbf{c}'(u)$ and $d(u, v) + \mathbf{h}'(v) \leq \mathbf{h}'(u)$, thus H' is also a partial solution for t in this case. It follows that $A_t(\mathbf{c}, \mathbf{h}, S) \leq A_{t'}(\mathbf{c}', \mathbf{h}', S')$. □

For a *join node* t with children t' and t'', we have $X_t = X_{t'} = X_{t''}$. A vertex in X_t that is satisfied directly in the subproblem for t might be satisfied indirectly in the subproblem for t' or t'', but not both (see Figure 3.9). Thus, the intersection of vertices satisfied indirectly in t' and t'' must be equal to the set of vertices satisfied indirectly in t.





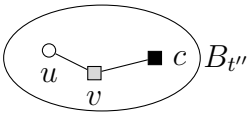


Figure 3.9: The join case of SkHC.

We compute:

$$A_t(\mathbf{c},\mathbf{h},S) = \min_{\substack{S',S''\subseteq X_t:\\ S'\cap S''=S}} \Big\{A_{t'}(\mathbf{c},\mathbf{h},S') + A_{t''}(\mathbf{c},\mathbf{h},S'') - |H_0|\Big\},$$

where H_0 is the set of vertices $u \in X_t$ such that $\mathbf{c}(u) = d(u,c)$ and $\mathbf{h}(u) = 0$, which corresponds to a common subset of hubs in the children's partial solutions.

Lemma 4. *The formula holds for join nodes.*

Proof. Let H be a partial solution for t, and define $H' = H \cap V_{t'}$ and $H'' = H \cap V_{t''}$. We consider an entry $(\mathbf{c},\mathbf{h},S')$ of the subproblem for t' and an entry $(\mathbf{c},\mathbf{h},S'')$ of the subproblem for t''. Observe that every vertex of $V_{t'}$ is satisfied by some vertex in H', with the exception of S and possibly some other vertices in X_t. Let S' be the union of S and the subset of vertices in X_t that are satisfied by some hub in $H'' \setminus X_t$. The construction of S'' is analogous. Notice that $H_0 = H \cap X_t$, and that H_0 is a common subset of H' and H''. Observe that H' and H'' are partial solutions for t' and t'', respectively. Then, $A_t(\mathbf{c},\mathbf{h},S) = |H| = |H'| + |H''| - |H_0| \geq A_{t'}(\mathbf{c},\mathbf{h},S') + A_{t''}(\mathbf{c},\mathbf{h},S'') - |H_0|$.

For the converse direction, consider the pair of subsets S' and S'' that achieve the minimum in the recurrence, and let H' and H'' be corresponding partial solutions for t' and t'', respectively, such that $|H'| = A_{t'}(\mathbf{c},\mathbf{h},S')$ and $|H''| = A_{t''}(\mathbf{c},\mathbf{h},S'')$. We claim that $H = H' \cup H''$ is a partial solution for t. Observe that H' satisfies vertices in $V_{t'} \setminus S'$ and H'' satisfies vertices in $V_{t''} \setminus S''$. Thus, H satisfies vertices in $(V_{t'} \cup V_{t''}) \setminus (S' \cap S'')$. Since $V_t = V_{t'} \cup V_{t''}$ and $S = S' \cap S''$, H is a partial solution for t. Then, $A_{t'}(\mathbf{c},\mathbf{h},S') + A_{t''}(\mathbf{c},\mathbf{h},S'') - |H_0| = |H'| + |H''| - |H_0| = |H| \geq A_t(\mathbf{c},\mathbf{h},S)$. □

Using the recurrence relations in a dynamic programming algorithm, one can compute a minimum size solution for the whole graph. This leads to the following theorem.

Theorem 8. *Given an instance (G,c,k,ℓ_1,ℓ_2) of* MSkHC *and a nice tree decomposition of G of width* tw*, there is an algorithm that computes a solution in time $\mathcal{O}^*(r^{\mathcal{O}(\mathrm{tw})})$, or concludes that there is no solution with k hubs.*

Proof. Let t_0 be the root of the tree decomposition, where $X_{t_0} = \emptyset$, and $\mathbf{c}_\emptyset$, $\mathbf{h}_\emptyset$ be empty colorings. Using Lemmas 2, 3 and 4, if $A_{t_0}(\mathbf{c}_\emptyset,\mathbf{h}_\emptyset,\emptyset) > k$, then there is no solution with k hubs. Otherwise, we have a partial solution $H \subseteq V_{t_0} = V(G)$ and a corresponding assignment $\phi : V(G) \setminus \{c\} \to H$, where item (iii) of Definition 4 holds for every vertex of G. It follows that (H,ϕ) respects ℓ_1 and ℓ_2, as for every $u \in V(G)$, there is $\phi(u) \in H$ such that $d(u,\phi(u),c) \leq \ell_1(u)$ and $d(u,\phi(u)) \leq \ell_2(u)$.

Now we analyze the running time of the algorithm. For each node t of the tree decomposition, there are at most $(r+1)^{|X_t|}$ distinct colorings for each of $\mathbf{c}$ and $\mathbf{h}$ and $2^{|X_t|}$ choices of S, resulting in $\mathcal{O}(r^{\mathcal{O}(\mathrm{tw})})$ entries per node. The running time to compute the value of an entry $A_t(\mathbf{c},\mathbf{h},S)$ is considered next. For leaf and introduce nodes, the operations are performed in time $\mathcal{O}(n)$. For a forget node, since the child node has exactly one extra vertex, we consider at most $r+1$ extensions for each of $\mathbf{c}$ and $\mathbf{h}$, which can be done in time $\mathcal{O}^*(r^2)$. For a join node, we list entries of the subproblems that correspond to pairs of subsets S' and S'' whose intersection is S, which can be done in

time $\mathcal{O}^*(4^{\mathrm{tw}})$. Since the nice tree decomposition has $\mathcal{O}(n \cdot \mathrm{tw})$ nodes, the overall time of the algorithm is $\mathcal{O}^*(r^{\mathcal{O}(\mathrm{tw})})$. □

3.3 An efficient parameterized approximation scheme

In this section, we present an EPAS for SkHC, parameterized by the treewidth of the graph. Namely, for any $\epsilon > 0$, we give an algorithm that receives an instance (G, c, k, r) of SkHC and either finds a solution of value at most $(1+\epsilon)r$, or concludes that every solution has value greater than r. The running time of the algorithm is $\mathcal{O}^*((\mathrm{tw}/\epsilon)^{\mathcal{O}(\mathrm{tw})})$. The previous algorithm depended exponentially on r because an entry of the dynamic programming table contains the exact distances from each vertex in the bag to the center and to its hub. This is a common issue for algorithms parameterized by graph widths, which often store and enumerate large integers.

The general structure of the EPAS is inspired by the framework of Lampis [121], that was later applied to k-Center by Katsikarelis et al. [111]. In these works, integers are stored approximately in a dynamic programming table, and are calculated by adding values from previously computed entries. This effectively shrinks the size of the table and renders a faster algorithm, while increasing the value of a solution by a small factor. The former author devised an abstract model of computation based on this idea, which is now part of the parameterized approximation toolkit (see Subsection 2.5.2). We use this method in Chapter 4, for a different hub location problem. For SkHC, however, we directly show how to obtain such an approximate solution with no more hubs than in the exact one.

3.3.1 Reduction to a simplified problem

As before, we consider an instance $(G, c, k, \ell_1, \ell_2)$ of MSkHC that is solved by a dynamic programming algorithm. However, now we require that a solution H respects ℓ_1 and ℓ_2 only approximately.

Definition 5. *Let $\epsilon > 0$ be a constant and consider an instance $I = (G, c, k, \ell_1, \ell_2)$ of* MSkHC*. An assignment $\phi : V(G) \setminus \{c\} \to H$, where $H \subseteq V(G) \setminus \{c\}$ with $|H| \leq k$,* ϵ-respects *ℓ_1 and ℓ_2 if, for every $u \in V(G) \setminus \{c\}$, we have $d(u, \phi(u), c) \leq (1+\epsilon)\ell_1(u)$ and $d(u, \phi(u)) \leq (1+\epsilon)\ell_2(u)$. Moreover, we say (H, ϕ) is an* ϵ-solution *for I.*

We show that finding an ϵ-solution for MSkHC is sufficient to find a $(1+\epsilon)$-approximation for SkHC.

Lemma 5. *If there is an ϵ-solution for an instance of* MSkHC*, then there is a $(1+\epsilon)$-approximation for the corresponding instance of* SkHC*.*

Proof. Suppose that we want to find a $(1+\epsilon)$-approximation for an instance $I = (G, c, k, r)$ of SkHC, for $0 < \epsilon \leq 1$. Let (H, ϕ) be an optimal solution for I, and let u_1, u_2, γ_1 and γ_2 be as in the proof of Lemma 1, i.e., the farthest vertices from the center and its hub in this optimal solution, respectively, and their distances. Recall that $N = \{u \in V(G) \setminus \{c\} : d(u, \phi(u_1)) \leq d(u_1, \phi(u_1))\}$. We guess the length of γ_1 and γ_2 up to a factor,

so assume that we are given lower bounds γ_1' and γ_2', such that $\gamma_1' \leq \gamma_1 \leq (1+\epsilon)\gamma_1'$ and $\gamma_2' \leq \gamma_2 \leq (1+\epsilon)\gamma_2'$. This can be done in time $\mathcal{O}^*(\log_{(1+\epsilon)} r)$.

The instance I' we build for MSkHC is similar to that in Lemma 1, but we allow vertices to have a greater distance to the center and its hub. We have $\ell_1(u) = (1+\epsilon)\gamma_1'$, for $u \in N$, and $\ell_1(u) = r - \gamma_1'$, otherwise. If $\gamma_2 \leq r/2$, we have $\ell_2(u) = (1+\epsilon)\gamma_2'$, for every u; otherwise, we have $\ell_2(u_2) = (1+\epsilon)\gamma_2'$ and $\ell_2(u) = r - \gamma_2'$, for every $u \neq u_2$.

Clearly, (H, ϕ) is an ϵ-solution for I', thus, we can find an ϵ-solution for this instance using the given algorithm. Since the resulting is an ϵ-solution, for every $u \in V(G) \setminus \{c\}$, we have $d(u, \phi(u), c) \leq (1+\epsilon)\ell_1(u)$ and $d(u, \phi(u)) \leq (1+\epsilon)\ell_2(u)$. We verify that this solution has the desired approximation factor for I. Let ϕ be the assignment function of this solution.

Let u and v be such that $\phi(u) \neq \phi(v)$. Suppose, w.l.o.g., that $u = u_1$, then

$$\begin{aligned} d(u, \phi(u), c, \phi(v), v) &\leq (1+\epsilon)(\ell_1(u) + \ell_1(v)) \\ &= (1+\epsilon)((1+\epsilon)\gamma_1' + r - \gamma_1') \\ &= (1+\epsilon)(r + \epsilon\gamma_1') \\ &\leq (1+\epsilon)^2 r, \end{aligned}$$

using $\gamma_1' \leq r$. If $u, v \neq u_1$, then

$$\begin{aligned} d(u, \phi(u), c, \phi(v), v) &\leq (1+\epsilon)(\ell_1(u) + \ell_1(v)) \\ &= 2(1+\epsilon)(r - \gamma_1') \\ &\leq 2(1+\epsilon)(r - \gamma_1/(1+\epsilon)) \\ &\leq (1+2\epsilon) r, \end{aligned}$$

using $\gamma_1' \geq \gamma_1/(1+\epsilon)$ and $r - \gamma_1 \leq r/2$. Now, let u and v be such that $\phi(u) = \phi(v) = w$. Suppose $\gamma_2 \leq r/2$, then

$$\begin{aligned} d(u, w, v) &\leq (1+\epsilon)(\ell_2(u) + \ell_2(v)) \\ &= 2(1+\epsilon)^2 \gamma_2' \\ &\leq (1+\epsilon)^2 r, \end{aligned}$$

using $\gamma_2' \leq \gamma_2 \leq r/2$. Now, suppose $\gamma_2 > r/2$ and, w.l.o.g., let $u = u_2$. We have

$$\begin{aligned} d(u, w, v) &\leq (1+\epsilon)(\ell_2(u) + \ell_2(v)) \\ &= (1+\epsilon)((1+\epsilon)\gamma_2' + r - \gamma_2') \\ &= (1+\epsilon)(r + \epsilon\gamma_2') \\ &\leq (1+\epsilon)^2 r, \end{aligned}$$

using $\gamma_2' \leq \gamma_2 \leq r$. If $u, v \neq u_2$, then

$$\begin{aligned} d(u,w,v) &\leq (1+\epsilon)(\ell_2(u)+\ell_2(v)) \\ &= 2(1+\epsilon)(r-\gamma_2') \\ &\leq 2(1+\epsilon)(r-\gamma_2/(1+\epsilon)) \\ &\leq (1+2\epsilon)r, \end{aligned}$$

using $\gamma_2' \geq \gamma_2/(1+\epsilon)$. Therefore, since $\max\{1+2\epsilon, (1+\epsilon)^2\} \leq 1+3\epsilon$, this solution is a $(1+3\epsilon)$-approximation for I. □

3.3.2 The algorithm

For a fixed $\delta > 0$, we will approximate a distance by an integer power of $(1+\delta)$. For some vertex $u \in V(G) \setminus \{c\}$, we consider a set of numbers containing zero, $d(u,c)$ and the integer powers of $(1+\delta)$ that are less than or equal to $(1+\epsilon)r$, defined as:

$$\Sigma_u = \{0, d(u,c)\} \cup \{(1+\delta)^i : i \in \mathbb{Z}_{\geq 0},\ (1+\delta)^i \leq (1+\epsilon)r\}.$$

In the subproblem definition, rather than considering values in $\{0, \ldots, r\}$, we will enumerate only values in Σ_u, for a vertex u. This implies that the colorings found for the subproblem corresponding to a node of the tree decomposition are computed only within a factor $(1+\delta)$. But, because the partial solutions for the children nodes are already approximate, the approximation error is multiplied in each node of the tree up to the root. This suggests the following definition, that is a relaxed version of Definition 4.

Definition 6. *Let t be a node of the tree decomposition of height h. Consider colorings $\mathbf{c}_\delta, \mathbf{h}_\delta$ on X_t such that $\mathbf{c}_\delta(u), \mathbf{h}_\delta(u) \in \Sigma_u$, for $u \in X_t$, and a subset $S \subseteq X_t$. A δ-partial solution for t is a subset $H \subseteq V_t$ such that:*

(i) for every $u \in X_t$, we have $u \in H$ if, and only if, $\mathbf{c}_\delta(u) = d(u,c)$ and $\mathbf{h}_\delta(u) = 0$; and

(ii) for every $u \in X_t \setminus S$, there is $v \in H$ such that $d(u,v,c) \leq (1+\delta)\mathbf{c}_\delta(u)$ and $d(u,v) \leq (1+\delta)\mathbf{h}_\delta(u)$; and

(iii) for every $u \in V_t \setminus X_t$, G_t has a shortest path P from u to a vertex v such that:

(a) $V(P) \cap X_t = \{v\}$, $d(u,v) + \mathbf{c}_\delta(v) \leq (1+\delta)^h \ell_1(u)$ and $d(u,v) + \mathbf{h}_\delta(v) \leq (1+\delta)^h \ell_2(u)$, or

(b) $V(P) \cap X_t = \emptyset$, $v \in H$, $d(u,v,c) \leq (1+\delta)^h \ell_1(u)$ and $d(u,v) \leq (1+\delta)^h \ell_2(u)$.

If item (ii) or (iii) holds for some vertex u, then we say that u is δ-satisfied *by v.*

Instead of finding a δ-partial solution of minimum size, for each entry $(\mathbf{c}_\delta, \mathbf{h}_\delta, S)$, our algorithm computes a δ-partial solution for t of bounded size $A_t^\delta(\mathbf{c}_\delta, \mathbf{h}_\delta, S)$ and, if we cannot find a δ-partial solution, we set $A_t^\delta(\mathbf{c}_\delta, \mathbf{h}_\delta, S) = \infty$. We show that, for each entry of the table, the size of the constructed δ-partial solution is no greater than the size of a minimum partial solution for a corresponding instance of the exact subproblem.

Namely, in Lemma 6, for each entry $A_t(\mathbf{c}, \mathbf{h}, S)$ for the exact subproblem, we compute a corresponding entry $A_t^\delta(\mathbf{c}_\delta, \mathbf{h}_\delta, S)$ for the relaxed subproblem, where:

$$\mathbf{c}_\delta(u) = \begin{cases} d(u,c) & \text{if } \mathbf{c}(u) = d(u,c), \\ (1+\delta)^{\lfloor \log_{(1+\delta)} \mathbf{c}(u) \rfloor} & \text{otherwise.} \end{cases}$$

$$\mathbf{h}_\delta(u) = \begin{cases} 0 & \text{if } \mathbf{h}(u) = 0, \\ (1+\delta)^{\lfloor \log_{(1+\delta)} \mathbf{h}(u) \rfloor} & \text{otherwise.} \end{cases}$$

Lemma 6. *Consider an entry $A_t(\mathbf{c}, \mathbf{h}, S)$ of the exact dynamic programming table. There is an algorithm that, in time $\mathcal{O}^*((\log_{(1+\delta)} r)^{\mathcal{O}(\mathrm{tw})})$, computes a δ-partial solution for t w.r.t. $(\mathbf{c}_\delta, \mathbf{h}_\delta, S)$ and size $A_t^\delta(\mathbf{c}_\delta, \mathbf{h}_\delta, S) \leq A_t(\mathbf{c}, \mathbf{h}, S)$.*

Proof. We use the same recurrence relations as those given in Section 3.2, except that now we refer to the notions of δ-partial solutions of Definition 6, instead of the notions of partial solutions of Definition 4. We will show the lemma by induction on the height h of t. For leaf nodes, X_t contains only the center, so $A_t^\delta(\mathbf{c}_\delta, \mathbf{h}_\delta, S) = 0$, and the lemma holds trivially. Thus, assume that t is not a leaf. Next, we consider each node type separately.

Forget node In this case, t has a child t' with height h' such that $X_t = X_{t'} \setminus \{u\}$, for some $u \in X_{t'}$, and $G_t = G_{t'}$. Consider a partial solution H for t such that $|H| = A_t(\mathbf{c}, \mathbf{h}, S)$. Next, we consider an instance of the subproblem for t' w.r.t. $(\mathbf{c}', \mathbf{h}', S')$ such that $\mathbf{c}'$ and $\mathbf{h}'$ are extensions of $\mathbf{c}$ and $\mathbf{h}$, respectively, and $S' \subseteq X_{t'}$ is a superset of S. We consider two subcases separately.

(*i*) First, assume that u is satisfied by some vertex $y \in X_t$ such that $d(u,y) + \mathbf{c}(y) \leq \ell_1(u)$ and $d(u,y) + \mathbf{h}(y) \leq \ell_2(u)$. In this case, we define $\mathbf{c}'(u) = d(u,y) + \mathbf{c}(y)$, $\mathbf{h}'(u) = d(u,y) + \mathbf{h}(y)$ and $S' = S \cup \{u\}$. Observe that H is a partial solution for t', thus $A_{t'}(\mathbf{c}', \mathbf{h}', S') \leq A_t(\mathbf{c}, \mathbf{h}, S)$.

For the other direction, let H'_δ be a δ-partial solution for t' such that $|H'_\delta| = A_{t'}^\delta(\mathbf{c}'_\delta, \mathbf{h}'_\delta, S')$. We claim that each vertex v of $V_t \setminus S$ is δ-satisfied in the subproblem for t. This is clear if v is δ-satisfied by some vertex of $H'_\delta \cup X_t$ in the subproblem for t'. In the only remaining possibility, v is δ-satisfied by u, thus $d(v,u) + \mathbf{c}'_\delta(u) \leq (1+\delta)^{h'} \ell_1(v)$ and $d(v,u) + \mathbf{h}'_\delta(u) \leq (1+\delta)^{h'} \ell_2(v)$. It follows that v can be δ-satisfied by $y \in X_t$, since

$$\begin{aligned} d(v,y) + \mathbf{c}_\delta(y) &\leq d(v,u,y) + \mathbf{c}(y) \\ &= d(v,u) + \mathbf{c}'(u) \\ &\leq (1+\delta)(d(v,u) + \mathbf{c}'_\delta(u)) \\ &\leq (1+\delta)(1+\delta)^{h'} \ell_1(v) \\ &= (1+\delta)^h \ell_1(v), \end{aligned}$$

where we used $\mathbf{c}'_\delta(u) \leq \mathbf{c}'(u) \leq (1+\delta)\mathbf{c}'_\delta(u)$, and $h' < h$. Analogously, $d(v,y) + \mathbf{h}_\delta(y) \leq (1+\delta)^h \ell_2(v)$.

(*ii*) Now, assume that u is not satisfied by any vertex in X_t. Then, u is satisfied directly by some vertex $w \in H$. In this case, we define $\mathbf{c}'(u) = d(u, w, c)$, $\mathbf{h}'(u) = d(u, w)$ and $S' = S$. Observe that H is a partial solution for t', thus $A_{t'}(\mathbf{c}', \mathbf{h}', S') \leq A_t(\mathbf{c}, \mathbf{h}, S)$.

For the other direction, let H'_δ be a δ-partial solution for t' such that $|H'_\delta| = A^\delta_{t'}(\mathbf{c}'_\delta, \mathbf{h}'_\delta, S')$. We claim that each vertex v of $V_t \setminus S$ is δ-satisfied in the subproblem for t. Again, it suffices to analyze the case that v is δ-satisfied by u in the subproblem for t', thus $d(v, u) + \mathbf{c}'_\delta(u) \leq (1+\delta)^{h'}\ell_1(v)$ and $d(v, u) + \mathbf{h}'_\delta(u) \leq (1+\delta)^{h'}\ell_2(v)$. In this case, since $u \notin S'$, u is δ-satisfied by some vertex $w \in H'_\delta$ such that $d(u, w, c) \leq (1+\delta)\mathbf{c}'_\delta(u)$ and $d(u, w) \leq (1+\delta)\mathbf{h}'_\delta(u)$. Thus,

$$\begin{aligned} d(v, w, c) &\leq d(v, u, w, c) \\ &\leq d(v, u) + (1+\delta)\mathbf{c}'_\delta(u) \\ &\leq d(v, u) + (1+\delta)((1+\delta)^{h'}\ell_1(v) - d(v, u)) \\ &\leq (1+\delta)(1+\delta)^{h'}\ell_1(v) \\ &\leq (1+\delta)^h\ell_1(v). \end{aligned}$$

Analogously, $d(v, w) \leq (1+\delta)^h\ell_2(v)$. It follows that v is δ-satisfied by w.

In either case, the entry $A^\delta_{t'}(\mathbf{c}'_\delta, \mathbf{h}'_\delta, S')$ is considered by the recurrence relation, thus $A^\delta_t(\mathbf{c}_\delta, \mathbf{h}_\delta, S) \leq A^\delta_{t'}(\mathbf{c}'_\delta, \mathbf{h}'_\delta, S')$. Since $A^\delta_{t'}(\mathbf{c}'_\delta, \mathbf{h}'_\delta, S') \leq A_{t'}(\mathbf{c}', \mathbf{h}', S')$ by the induction hypothesis, and we also have $A_{t'}(\mathbf{c}', \mathbf{h}', S') \leq A_t(\mathbf{c}, \mathbf{h}, S)$, the lemma holds.

Introduce node In this case, t has a child t' such that $X_{t'} = X_t \setminus \{u\}$, for some $u \in X_t$. Let $\mathbf{c}'$ and $\mathbf{h}'$ be the restrictions of $\mathbf{c}$ and $\mathbf{h}$ to $X_{t'}$, respectively. We define an indicator set U and a set S'. If $\mathbf{c}(u) \neq d(u, c)$ and $\mathbf{h}(u) \neq 0$, then let $U = \emptyset$ and $S' = S$; otherwise, let $U = \{u\}$ and define $S' = S \cup \{v \in X_{t'} : d(v, u, c) \leq \mathbf{c}(v) \wedge d(v, u) \leq \mathbf{h}(v)\} \setminus \{u\}$, i.e., S' corresponds to the union of S with the vertices of $X_{t'}$ which can be satisfied by u. Consider a partial solution H for t such that $|H| = A_t(\mathbf{c}, \mathbf{h}, S)$ and note that $H' = H \setminus U$ is a partial solution for t' such that $|H'| = |H| - |U|$, then $A_{t'}(\mathbf{c}', \mathbf{h}', S') \leq A_t(\mathbf{c}, \mathbf{h}, S) - |U|$.

For the other direction, let H'_δ be a δ-partial solution for t' such that $|H'_\delta| = A^\delta_{t'}(\mathbf{c}'_\delta, \mathbf{h}'_\delta, S')$. We claim that $H_\delta = H'_\delta \cup U$ is a δ-partial solution for t. Because H'_δ is a δ-partial solution for t' w.r.t. $(\mathbf{c}'_\delta, \mathbf{h}'_\delta, S')$, it suffices to show that each vertex $v \in S' \setminus S$ is δ-satisfied in the subproblem for t. By construction, $S' \setminus S$ is non-empty only if $\mathbf{c}_\delta(u) = d(u, c)$ and $\mathbf{h}_\delta(u) = 0$, but then $u \in H_\delta$. Thus $d(v, u, c) \leq \mathbf{c}(v) \leq (1+\delta)\mathbf{c}_\delta(v)$ and $d(v, u) \leq \mathbf{h}(v) \leq (1+\delta)\mathbf{h}_\delta(v)$, and indeed v is δ-satisfied. It follows that $A^\delta_t(\mathbf{c}_\delta, \mathbf{h}_\delta, S) \leq A^\delta_{t'}(\mathbf{c}'_\delta, \mathbf{h}'_\delta, S') + |U|$. Since $A^\delta_{t'}(\mathbf{c}'_\delta, \mathbf{h}'_\delta, S') \leq A_{t'}(\mathbf{c}', \mathbf{h}', S')$ by the induction hypothesis, the lemma holds.

Join node In this case, t has children t' and t'' such that $X_t = X_{t'} = X_{t''}$. Let H be a partial solution for t such that $|H| = A_t(\mathbf{c}, \mathbf{h}, S)$, and define $H' = H \cap V_{t'}$ and $H'' = H \cap V_{t''}$. Observe that a vertex in X_t might be satisfied only by a hub in $H'' \setminus X_t$. Thus, in the subproblem corresponding to t', we have to consider additional vertices that are satisfied indirectly. Formally, define S' as the union of S and all vertices $v \in X_t$ that are *not* δ-satisfied by some vertex $u \in X_t \cup H'$. We define S'' symmetrically.

Let H_0 be the set of vertices $v \in X_t$ such that $\mathbf{c}(v) = d(v, c)$ and $\mathbf{h}(v) = 0$. Observe that H' is a partial solution for t' and H'' is a partial solution for t'' such that $|H| = |H'| + |H''| - |H' \cap H''| = |H'| + |H''| - |H_0|$. Thus, $A_{t'}(\mathbf{c}, \mathbf{h}, S') + A_{t''}(\mathbf{c}, \mathbf{h}, S'') - |H_0| \leq A_t(\mathbf{c}, \mathbf{h}, S)$.

For the other direction, let H'_δ be a δ-partial solution for t' such that $|H'_\delta| = A^\delta_{t'}(\mathbf{c}_\delta, \mathbf{h}_\delta, S')$, and H''_δ be a δ-partial solution for t'' such that $|H''_\delta| = A^\delta_{t''}(\mathbf{c}_\delta, \mathbf{h}_\delta, S'')$. Note that $H_\delta = H'_\delta \cup H''_\delta$ is a δ-partial solution for t. Because the pair of entries $A^\delta_{t'}(\mathbf{c}_\delta, \mathbf{h}_\delta, S')$ and $A^\delta_{t''}(\mathbf{c}_\delta, \mathbf{h}_\delta, S'')$ are considered in the recurrence relation, $A^\delta_t(\mathbf{c}_\delta, \mathbf{h}_\delta, S) \leq A^\delta_{t'}(\mathbf{c}_\delta, \mathbf{h}_\delta, S') + A^\delta_{t''}(\mathbf{c}_\delta, \mathbf{h}_\delta, S'') - |H_0|$. Since $A^\delta_{t'}(\mathbf{c}_\delta, \mathbf{h}_\delta, S') \leq A_{t'}(\mathbf{c}, \mathbf{h}, S')$ and $A^\delta_{t''}(\mathbf{c}_\delta, \mathbf{h}_\delta, S'') \leq A_{t''}(\mathbf{c}, \mathbf{h}, S'')$ by the induction hypothesis, the lemma holds.

The running time of the algorithm is dominated by the number of table entries for a node of the tree decomposition, which is $\mathcal{O}^*((\log_{(1+\delta)} r)^{\mathcal{O}(\mathrm{tw})})$. □

We have yet to prove that finding a δ-partial solution for the root node is sufficient to find an ϵ-solution for MSkHC. First, let $\hat{h}$ be the height of the tree decomposition, and define $\delta = \frac{\epsilon}{2(\hat{h}+1)}$. To bound the approximation error, we use the following auxiliary result. Then, we observe that a δ-partial solution for the root node is an ϵ-solution.

Lemma 7. $(1+\delta)^{\hat{h}+1} \leq 1 + \epsilon$.

Proof. By direct calculation,

$$(1+\delta)^{\hat{h}+1} = \Big(1 + \frac{\epsilon}{2(\hat{h}+1)}\Big)^{\hat{h}+1} \leq e^{\frac{\epsilon}{2}} \leq 1 + \epsilon,$$

where we use $\Big(1 + \frac{x}{n}\Big)^n \leq e^x$ and $e^{\frac{x}{2}} \leq 1 + x$, for $x \leq 2$. □

Lemma 8. *If H is a δ-partial solution for the root node of the tree decomposition, then H is an ϵ-solution.*

Proof. Let H be a δ-partial solution for t_0, and recall that $\hat{h}$ is the height of this node. Consider a fixed vertex $u \in V(G) \setminus \{c\}$. Since t_0 is empty, Definition 6 implies that u is δ-satisfied by some vertex $v \in H$. Then, $d(u, v, c) \leq (1+\delta)^{\hat{h}} \ell_1(u)$ and $d(u, v) \leq (1+\delta)^{\hat{h}} \ell_2(u)$. Therefore, the statement follows by Lemma 7. □

Putting everything together leads to the main result of this chapter.

Theorem 9. *There is an efficient parameterized approximation scheme for* SkHC*, parameterized by the treewidth of the graph.*

Proof. First, we give an algorithm that computes an ϵ-solution for an instance $(G, c, k, \ell_1, \ell_2)$ of MSkHC. Using the algorithm of Lemma 6, in time $\mathcal{O}^*((\log_{(1+\delta)} r)^{\mathcal{O}(\mathrm{tw})})$, one can compute a δ-partial solution H for the root node, and, with Lemma 8, we have that H is an ϵ-solution for this instance. The running time can be simplified:

$$\mathcal{O}(\log_{(1+\delta)} r) = \mathcal{O}\Big(\frac{\log r}{\log(1+\delta)}\Big) = \mathcal{O}\Big(\frac{\log r}{\delta}\Big) = \mathcal{O}\Big(\frac{\mathrm{tw} \cdot \log r \cdot \log n}{\epsilon}\Big),$$

since $\delta = \mathcal{O}(\frac{\epsilon}{\mathrm{tw} \cdot \log n})$ and $3\log(1+\delta) \geq \delta$ for $\delta \leq 2$.

Now, we consider the corresponding instance (G, c, k, r) of SkHC. Without loss of generality, we assume this instance has bounded aspect ratio, i.e., that r is polynomially bounded in n/ϵ, and thus $\log r = \mathcal{O}(\log(n/\epsilon))$. If not, then one can construct a corresponding instance with bounded aspect ratio such that an approximation for the modified instance induces an approximation for the original instance using standard tools (e.g., [9, 50]). It suffices to find a constant-factor approximation A for the optimal value r (using the 2-approximation of Chen et al. [48]), then constructing a copy of G such that each edge (u, v) has weight $d'(u, v) = \left\lceil \frac{d(u,v)}{\epsilon A/n^2} \right\rceil$.

To find a $(1 + \epsilon)$-approximation for SkHC, we use Lemma 5, which increases the running time by an extra factor $\mathcal{O}^*(\log_{(1+\epsilon)} r)$. The overall running time is:

$$\mathcal{O}^*(\log_{(1+\epsilon)} r \cdot (\log_{(1+\delta)} r)^{\mathcal{O}(\mathrm{tw})}) = \mathcal{O}^*((\mathrm{tw}/\epsilon)^{\mathcal{O}(\mathrm{tw})}).$$

To verify the last equality, note that if $\mathrm{tw} \leq \sqrt{\log n}$, then

$$(\log n/\epsilon)^{\mathcal{O}(\mathrm{tw})} = (1/\epsilon)^{\mathcal{O}(\mathrm{tw})}(\log n)^{\mathcal{O}(\sqrt{\log n})} = \mathcal{O}^*((1/\epsilon)^{\mathcal{O}(\mathrm{tw})});$$

otherwise, $\log n < \mathrm{tw}^2$, and then $(\log n/\epsilon)^{\mathcal{O}(\mathrm{tw})} = \mathcal{O}^*((\mathrm{tw}/\epsilon)^{\mathcal{O}(\mathrm{tw})})$. □

3.4 Future work

A possible research direction for SkHC is investigating parameterizations discussed in Subsection 2.4.2: we highlight the strategy of bounding the treewidth of an input graph in terms of another parameter, like clique-width, as done in Katsikarelis et al. [111], and then running the existing algorithm parameterized by treewidth. There is also the matter of finding either an improved approximation algorithm or a tighter inapproximability lower bound, given the gap is currently $1.5 - \epsilon$ to $\frac{5}{3}$, for $\epsilon > 0$.

Chapter 4

Multiple Allocation k-Hub Center

In the Multiple Allocation k-Hub Center (MAkHC), we are given a connected edge-weighted graph G, sets of clients $\mathcal{C}$ and hub locations $\mathcal{H}$, where $V(G) = \mathcal{C} \cup \mathcal{H}$, a set of demands $\mathcal{D} \subseteq \mathcal{C}^2$ and a positive integer k. A solution is a set of hubs $H \subseteq \mathcal{H}$ of size k such that every demand (a, b) is satisfied by a path starting in a, going through some vertex of H, and ending in b. The objective is to minimize the largest length of a path:

$$\max_{(a,b)\in\mathcal{D}} \min_{h\in H} d(a,h) + d(h,b),$$

where $d(u, v)$ denotes the length of a shortest path between vertices u and v. In the decision version of the problem, we are also given a non-negative number r and the goal is to determine whether there exists a solution of value at most r.

In classical location theory, the optimal strategy is usually to connect a client to its closest open facility [100, 44, 101, 54]. We consider the scenario in which pairs of clients need to be connected: rather than connecting each pair directly, one might select a set of hubs that act as consolidation points to take advantage of economies of scale [107, 39, 93, 108]. In this case, each origin-destination demand is served by a path starting at the origin, going through one of the selected hubs and ending at the destination. Using consolidation points reduces the cost of maintaining the network, as a large number of goods is often transported through few hubs, and a small fleet of vehicles is sufficient to serve the network [38].

Related works The first modern studies on hub location problems were done by O'Kelly [142, 143], when models and applications were surveyed. Since then, most papers focused on integer linear programming and heuristic methods [4, 73]. Some approximation algorithms were considered for single allocation median variants [104, 5, 92, 15]. The analogous of MAkHC with median objective was considered by Bordini and Vignatti [32], who presented a (4α)-approximation algorithm that opens $\left(\frac{2\alpha}{2\alpha-1}\right)k$ hubs, for $\alpha > 1$.

The problem we study is closely related to the well-known k-Center, where, given an edge-weighted graph G, one wants to select a set of k vertices, called centers, so that the maximum distance from each vertex to the closest center is minimized [101, 95]. In the corresponding decision version, one also receives a number r, and asks whether there is a solution of value at most r. By creating a demand (u, u) for each vertex u of G,

one reduces k-Center to MAkHC, thus MAkHC can be seen as a generalization of k-Center. In fact, MAkHC even generalizes the k-Supplier, a variant of k-Center whose vertices are partitioned into clients and locations, only clients need to be served, and centers must be selected from the set of locations [102].

A simple greedy algorithm gives a 2-approximation for k-Center, that is the best one can hope for, since finding an approximation with smaller factor is NP-hard [95]. Analogously, there is a best-possible 3-approximation for k-Supplier [102]. These results have been extended to MAkHC as well, which also admits a 3-approximation [145]. This algorithm applies the bottleneck strategy of Hochbaum and Shmoys [102], that guesses the value of an optimal solution; then, the algorithm either outputs a certificate that there is no solution of this value, or gives an approximate solution for this instance.

As we have seen, an alternative is to consider the problem from the perspective of parameterized algorithms, that insist on finding an exact solution, but allow running times with a non-polynomial factor that depends only on a certain parameter of the input. Feldmann and Marx [76] showed that k-Center is W[1]-hard for planar graphs of constant doubling dimension when the parameter is a combination of k, the highway dimension and the pathwidth of the graph. Blum [19] showed that the hardness holds even if we additionally parameterize by the skeleton dimension of the graph. Under the assumption that FPT $\neq$ W[1], this implies that k-Center does not admit an FPT algorithm for any of these parameters, even if restricted to planar graphs of constant doubling dimension.

Demaine et al. [62] give an FPT algorithm for k-Center, parameterized by k and r for planar and map graphs. All these characteristics seem necessary for an exact FPT algorithm, as even finding a $(2-\epsilon)$-approximation with $\epsilon > 0$ for the general case is W[2]-hard for parameter k [74]. If we remove the solution value r and parameterize only by k, the problem remains W[1]-hard if we restrict the instances to planar graphs [76], or if we add structural graph parameters, such as the vertex-cover number or the feedback-vertex-set number, and thus, also treewidth or pathwidth [111].

In order to confront the previous barriers, Katsikarelis et al. [111] provide an EPAS for k-Center with different parameters w, i.e., for every $\epsilon > 0$, one can compute a $(1+\epsilon)$-approximation in time $f(\epsilon, w) \cdot n^{O(1)}$, where w is either the clique-width or treewidth of the graph. More recently, Feldmann and Marx [76] have also given an EPAS for k-Center when it is parameterized by k and the doubling dimension, which can be a more appropriate parameter for transportation networks than r.

Our results and techniques We initiate the study of MAkHC under the perspective of parameterized algorithms. We start by showing that, for any $\epsilon > 0$, there is no parameterized $(3-\epsilon)$-approximation for the problem when the parameter is k, the value r is bounded by a constant and the graph is unweighted, unless FPT = W[2]. For planar graphs, finding a good constant-factor approximation remains hard in the polynomial sense, as we show that it is NP-hard to find a $(3-\epsilon)$-approximation for MAkHC in this case, even if the maximum degree is 3.

To challenge the approximation lower bound, one might envisage an FPT algorithm by considering an additional structural parameter, such as vertex-cover and feedback-vertex-set numbers or treewidth. However, this is unlikely to lead to an exact FPT algorithm,

as we note that the hardness results for k-Center [111, 76, 19] extend to MAkHC. Namely, we show that, unless FPT = W[1], MAkHC does not admit an FPT algorithm when parameterized by a combination of k, the highway and skeleton dimensions and the pathwidth of the graph, even if restricted to planar graphs of constant doubling dimension; or when parameterized by k and the vertex-cover number. Instead, we aim at finding an approximation with factor strictly smaller than 3 that runs in FPT time.

In this chapter, we present a $(2+\epsilon)$-approximation for MAkHC parameterized by the treewidth of the graph, for $\epsilon > 0$, running in time $\mathcal{O}^*((\mathrm{tw}/\epsilon)^{\mathcal{O}(\mathrm{tw})})$. Moreover, we give a parameterized $(2+\epsilon)$-approximation for MAkHC when the input graph is planar and unweighted, parameterized by k and r.

Our main result is a non-trivial dynamic programming algorithm over a tree decomposition, that follows the spirit of the algorithm by Demaine et al. [62]. We assume that we are given a tree decomposition of the graph and consider both k and r as part of the input. Thus, for each node t of this decomposition, we can guess the distance from each vertex in the bag of t to its closest hub in some (global) optimal solution H^*. The subproblem is computing the minimum number of hubs to satisfy each demand in the subgraph G_t, corresponding to t.

Compared to k-Center and k-Supplier, however, MAkHC has two additional sources of difficulty. First, the cost to satisfy a demand cannot be computed locally, as it is the sum of two shortest paths, each from a client in the origin-destination pair to some hub in H^* that satisfies that pair. Second, the set of demand pairs $\mathcal{D}$ is given as part of the input, whereas every client must be served in k-Center or in k-Supplier. If we knew the subset of demands D_t^* that are satisfied by some hub in $H^* \cap V(G_t)$, then one could solve every subproblem in a bottom-up fashion, so that every demand would have been satisfied in the subproblem corresponding to the root of the decomposition.

Guessing D_t^* leads to an FPT algorithm parameterized by tw, r and $|\mathcal{D}|$, which is unsatisfactory as the number of demands might be large in practice. Rather, for each node t of the tree decomposition, we compute deterministically two sets of demands $D_t, S_t \subseteq \mathcal{D}$ that enclose D_t^*, that is, that satisfy $D_t \subseteq D_t^* \subseteq D_t \cup S_t$. By filling the dynamic programming table using D_t instead of D_t^*, we can obtain an algorithm that runs in FPT time on parameters tw and r, and that finds a 2-approximate solution.

The key insight for the analysis is that the minimum number of hubs in G_t that are necessary to satisfy each demand in D_t by a path of length at most r is a lower bound on $|H^* \cap V(G_t)|$. At the same time, the definition of the set of demands S_t ensures that each such demand can be satisfied by a path of length at most $2r$ using a hub that is close to a vertex in the bag of t. This is the main technical contribution of this chapter, and we believe these ideas might find usage in algorithms for similar problems whose solution costs have non-local components.

Using only these ideas, however, is not enough to get rid of r as a parameter, as we need to enumerate the distance from each vertex in a bag to its closest hub. A common method to shrink a dynamic programming table with large integers is storing only an approximation of each number, causing the solution value to be computed approximately. This eliminates the parameter r from the running time, but adds a term ϵ to the approximation factor. This technique is now standard and has been applied multiple times for

graph width problems [62, 83, 121, 111, 16], as we have seen in Subsection 2.5.2.

Specifically, we employ the framework of approximate addition trees [121]. For some $\delta > 0$, we approximate each value $\{1, \ldots, r\}$ of an entry in the dynamic programming table by an integer power of $(1 + \delta)$, and show that each such value is computed by an addition tree and corresponds to an approximate addition tree. By results in [121], we can readily set δ appropriately so that the number of distinct entries is polynomially bounded and each value is approximated within factor $(1 + \epsilon)$.

The remaining of this chapter is organized as follows: Section 4.1 brings our hardness results, Section 4.2 describes the algorithm parameterized by treewidth, which is analyzed in Section 4.3, Section 4.4 shows the algorithm for the planar case, and Section 4.5 points future research directions.

4.1 Hardness results

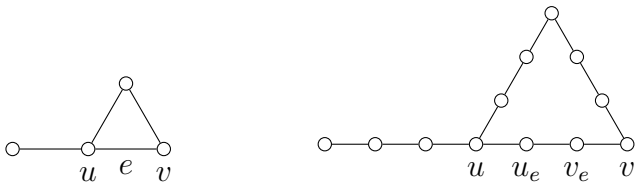
In this section, we observe that approximating MAkHC is hard, both in the classical and parameterized senses. First, we show that approximating the problem by a factor better than 3 is NP-hard, even if the input graph is planar and unweighted. This result strengthens the previous known lower bound and matches the approximation factor of the greedy algorithm [145].

Theorem 10. *For every $\epsilon > 0$, if there is a $(3 - \epsilon)$-approximation for* MAkHC *when the graph is planar and unweighted, then* P = NP.

Proof. We present a reduction from Vertex Cover (VC), whose task is to find a subset of k vertices that contains at least one endpoint of every edge of the graph. More specifically, we consider a particular version of the problem.

Claim 1. Vertex Cover *is* NP*-hard even if the graph is planar, triangle-free and has maximum degree 3.*

Proof. We self-reduce the problem from the case the input graph is planar and with maximum degree 3, which is known to be NP-hard [90]. Given an instance (G, k) of vertex cover, create another instance (G', k'), where G' is obtained by subdividing each edge of G in three parts, and $k' = k + |E(G)|$. Let u_e and v_e be new vertices added for the subdivision of an edge $e = (u, v) \in E(G)$ and that are incident with u and v, respectively. This reduction is depicted in Figure 4.1.



(a) Graph G. (b) Graph G'.

Figure 4.1: Self reduction of VC.

Assume S is a vertex cover for G of size k, and build a vertex cover S' for G' as follows. Initialize S' with a copy of S and, for each edge $e = (u, v)$ of G, add v_e to S', if $u \in S$, and add u_e, otherwise. Note that S' is a vertex cover of G' of size k'. For the other direction, assume S' is a vertex cover of G' with size k', and define $S = S' \setminus \{u_e, v_e : e \in E(G)\}$. If, for some edge (u_e, v_e) of G', both u_e and v_e are in S', then $S' \setminus \{u_e\} \cup \{u\}$ is a vertex cover of G'. Thus, assume for every such edge (u_e, v_e), either u_e or v_e is in S'. It follows that S is a vertex cover of G of size k. □

Given an instance (G, k) of VC, build an instance $(G, \mathcal{C}, \mathcal{H}, \mathcal{D}, k)$ of MAkHC, where $\mathcal{C} = \mathcal{H} = V(G)$ and $\mathcal{D} = E(G)$. Observe that there exists a vertex cover S of size k in G if, and only if, the solution S for MAkHC has value 1. Suppose that the optimal value is greater than 1, then it would have to be at least 3, since the graph has no triangles. Then, for $\epsilon > 0$, a $(3-\epsilon)$-approximation for MAkHC can decide whether the optimal value is 1, thus deciding whether there is a vertex cover of size k in G. □

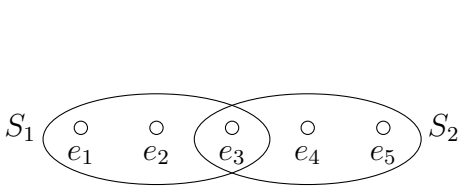
From this reduction, one may observe that the previous result holds even for the case where the maximum degree is 3 and the optimal value is bounded by 3.

To find a better approximation guarantee, one might resource to a parameterized approximation algorithm. The natural candidates for parameters of MAkHC are the number of hubs k and the value r of an optimal solution. The next theorem states that this choice of parameters does not help, as it is W[2]-hard to find a parameterized approximation with factor better than 3, when the parameter is k, the value r is bounded by a constant and G is unweighted.

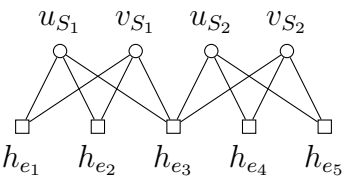
Theorem 11. *For every $\epsilon > 0$, if there is a parameterized $(3-\epsilon)$-approximation for* MAkHC *with parameter k, then* FPT = W[2].

Proof. The result will follow by a reduction from Hitting Set (HS), which is known to be W[2]-hard [68]. We show that a $(3-\epsilon)$-approximation for MAkHC can decide an instance of HS, implying that FPT = W[2]. Recall that in HS, we are given a set $\mathcal{U}$, a family of sets $\mathcal{F} \subseteq 2^{\mathcal{U}}$ and an integer k, and the objective is to decide whether there exists a set $H \subseteq \mathcal{U}$ of size k that intersects every set of $\mathcal{F}$.

Given an instance $I = (\mathcal{U}, \mathcal{F}, k)$ of HS, we build an instance $I' = (G, \mathcal{C}, \mathcal{H}, \mathcal{D}, k)$ of MAkHC: for each element $e \in \mathcal{U}$, create a vertex h_e in G and add it to $\mathcal{H}$; for each set $S \in \mathcal{F}$, create vertices u_S and v_S in G, add them to $\mathcal{C}$, create a demand (u_S, v_S) in $\mathcal{D}$ and connect u_S and v_S to vertices $\{h_e : e \in S\}$. This reduction is depicted in Figure 4.2.



(a) Example instance of HS.



(b) Reduced instance of MAkHC.

Figure 4.2: Reduction from HS to MAkHC.

Consider a hitting set H of size k, and let $H' = \{h_e : e \in H\}$ be a set of hubs of size k. This set of hubs satisfies every demand in $\mathcal{D}$ with cost 2, since for every $S \in \mathcal{F}$, there is $e \in S \cap H$ and thus $h_e \in H'$. In the other direction, consider a set of hubs H' of size k that satisfies every demand in $\mathcal{D}$ with cost 2, and let $H = \{e : h_e \in H'\}$ be a set of elements of size k. For each set $S \in \mathcal{F}$, there exists a corresponding demand (u_S, v_S) in $\mathcal{D}$ that is satisfied by a hub $h_e \in H'$ with cost 2. Since the length of this path is 2, h_e must be a neighbor of u_S and v_S in G, then $e \in S \cap H$. It follows that H is a hitting set for I.

We have shown that I is a YES-instance if, and only if, the optimal value of I' is 2. Now, if the optimal value of I' is greater than 2, then it would have to be at least 6. Indeed, if a demand (u_S, v_S) is satisfied by a hub $h_e \in H'$ with cost greater than 2, then h_e is not a neighbor of u_S. But G is bipartite and u_S and h_e are at different parts, then $d(u_S, h_e) \geq 3$. Analogously, we have $d(v_S, h_e) \geq 3$, and thus $d(u_S, h_e) + d(v_S, h_e) \geq 6$. We conclude that a $(3 - \epsilon)$-approximation can decide whether the optimal value of I' is 2, thus deciding whether I is a YES-instance. □

Due to the previous hardness results, a parameterized algorithm for MAkHC must consider different parameters, or assume a particular case of the problem. We focus on the treewidth of the graph, that is one of the most studied structural parameters [58], and the particular case of planar graphs. This setting is unlikely to lead to an (exact) FPT algorithm, though, as the problem is W[1]-hard, even if we combine these conditions. The next theorem follows directly from a result of Blum [19], since MAkHC is a generalization of k-CENTER.

Theorem 12. *Even on planar graphs with edge lengths of constant doubling dimension,* MAkHC *is* W[1]*-hard for the combined parameter* $(k, \mathrm{pw}, h, \kappa)$*, where* pw *is the pathwidth,* h *is the highway dimension and* κ *is the skeleton dimension of the graph.*

Proof. Given an instance $I = (G, k)$ of k-CENTER, we build an instance $I' = (G, \mathcal{C}, \mathcal{H}, \mathcal{D}, k)$ of MAkHC where $\mathcal{C} = \mathcal{H} = V(G)$ and $\mathcal{D} = \{(u, u) : u \in V(G)\}$. Now, note that there is a solution of value at most r for I if, and only if, there is a solution of value $2r$ for I'. The theorem follows, as we do not change the graph or the number of hubs k. □

Note that MAkHC inherits other hardness results of k-CENTER, thus it is W[1]-hard when parameterized by a combination of k and the vertex-cover number [111]. Recall that the treewidth is a lower bound on the pathwidth, thus the previous theorem implies that the problem is also W[1]-hard for planar graphs when parameterized by a combination of tw and k.

In order to circumvent these hardness results, we give a $(2+\epsilon)$-approximation algorithm for MAkHC for arbitrary graphs that is parameterized by tw, breaking the approximation barrier of 3. Then, we complement this result with a $(2+\epsilon)$-approximation for unweighted planar graphs parameterized by k and r.

4.2 Algorithm

In this section, we give a $(2+\epsilon)$-approximation parameterized by treewidth. In what follows, we assume that we receive an instance of MAkHC and a nice tree decomposition $(\mathcal{T}, B)$ of the input graph G of width tw and height bounded by $\mathcal{O}(\text{tw} \cdot \log |V(G)|)$. Also, we assume that G contains all edges connecting pairs $u, v \in B_t$ for each node t. Moreover, we are given an integer r bounded by $\mathcal{O}((1/\epsilon)|V(G)|)$. Our goal is to design a dynamic programming algorithm that computes the minimum number of hubs that satisfy each demand with a path of length r. The overall idea is similar to that of the algorithm for k-Center by Demaine et al. [62], except that we consider a tree decomposition, instead of a branch decomposition, and that the computed solution will satisfy demands only approximately. We use a preprocessing step to simplify our algorithm and notation.

Preprocessing For an instance of MAkHC and a demand $(a, b) \in \mathcal{D}$, define G_{ab} as the induced subgraph of G with vertex set $V(G_{ab}) = \{v \in V(G) : d(a, v) + d(v, b) \leq r\}$.

Notice that if a solution H has a hub $h \in V(G_{ab})$, then the length of a path serving (a, b) that crosses h is at most r. In this case, we say that demand (a, b) is *satisfied* by h with cost r. Thus, in an optimal solution H^* of MAkHC, for every $(a, b) \in \mathcal{D}$, the set $H^* \cap V(G_{ab})$ must be non-empty.

Also, if there is $v \in V(G)$ such that $d(a, v) + d(v, b) > r$ for every $(a, b) \in \mathcal{D}$, then v does not belong to any (a, b)-path of length at most r, and can be safely removed from G. From now on, assume that we have preprocessed G in polynomial time, such that for every $v \in V(G)$,

$$\min_{(a,b)\in\mathcal{D}} d(a, v) + d(v, b) \leq r.$$

Moreover, we assume that each edge has an integer weight and that the optimal value, OPT, is bounded by $\mathcal{O}(\frac{1}{\epsilon}|V(G)|)$, for a given constant $\epsilon > 0$. If not, then we solve another instance for which this holds and that has optimal value $\text{OPT}' \leq (1+\epsilon)\text{OPT}$ using standard rounding techniques [164]. It suffices finding a constant-factor approximation of value $A \leq 3\text{OPT}$ [145], and defining a new distance function such that $d'(u, v) = \left\lceil \frac{3|V(G)|}{\epsilon A} d(u, v) \right\rceil$.

Subproblem definition Consider some fixed global optimal solution H^* and a node t of the tree decomposition. Let us discuss possible candidates for a subproblem definition. The subgraph G_t corresponding to t in the decomposition contains a subset of H^* that satisfies a subset D_t^* of the demands. The shortest path serving each demand with a hub of $H^* \cap V(G_t)$ is either completely contained in G_t, or it must cross some vertex of the bag B_t. Thus, as in [62], we guess the distance i from each vertex u in B_t to the closest hub in H^*, and assign "color" $\downarrow i$ to u to mean that the corresponding shortest path is in G_t, and color $\uparrow i$ otherwise.

Since the number of demands may be large, we cannot include D_t^* as part of the subproblem definition. For k-Center, if the shortest path serving a vertex in G_t crosses a vertex $u \in B_t$, then the length of this path can be bounded locally using the color of u,

and the subproblem definition may require serving all vertices. For MAkHC, however, there might be demands (a, b) such that a is in G_t, while b is not, thus the coloring of B_t is not sufficient to bound the length of a path serving (a, b).

Instead of guessing D_t^*, for each coloring c of B_t, we require that only a subset $D_t(c)$ must be satisfied in the subproblem, and they can be satisfied by a path of length at most $2r$. Later, we show that the other demands in D_t^* are already satisfied by the hubs corresponding to the coloring of B_t. More specifically, we would like to compute $A_t(c)$ as the minimum number of hubs in G_t that satisfy each demand in $D_t(c)$ with a path of length at most $2r$ and that respect the distances given by c.

Since we preprocessed the graph, there must be a hub in H^* to each vertex of B_t at distance at most r. Thus, the number of distinct colorings to consider for each t is bounded by $r^{\mathcal{O}(\text{tw})}$. To get an algorithm parameterized only by tw, we need one more ingredient: in the following, the value of each color is stored approximately as an integer power of $(1 + \delta)$, for some $\delta > 0$. Later, using the framework of approximate addition trees (see Subsection 2.5.2), for any constant $\epsilon > 0$, we can set δ such that the number of subproblems is bounded by $\mathcal{O}^*((\text{tw}/\epsilon)^{\mathcal{O}(\text{tw})})$, and demands are satisfied by a path of length at most $(1 + \epsilon)2r$.

The set of approximate colors is

$$\Sigma = \{\downarrow 0\} \cup \{\uparrow i, \downarrow i \ : \ j \in \mathbb{Z}_{\geq 0}\,,\, i = (1+\delta)^j\,,\, i \leq (1+\epsilon)r\,\}.$$

A coloring of B_t is represented by a function $c : B_t \to \Sigma$. For each coloring c, we compute a set of demands that are "satisfied" by c.

Definition 7. *Define $S_t(c)$ as the set of demands (a, b) for which there exists $u \in B_t$ with $c(u) \in \{\uparrow i, \downarrow i\}$ and such that $d(a, u) + 2i + d(u, b) \leq (1 + \epsilon)2r$.*

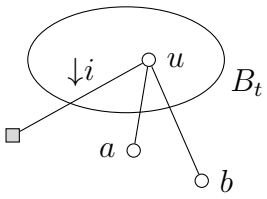


Figure 4.3: Demand (a, b) is satisfied by a hub close to B_t.

The intuition is that a demand $(a, b) \in S_t(c)$ can be satisfied by a hub close to u by a path of length at most $(1 + \epsilon)2r$ (see Figure 4.3). Also, we compute a set of demands that must be served by a hub in G_t in the global optimal solution (see Figure 4.4).

Definition 8. *Define $D_t(c)$ as the set of demands (a, b) such that $(a, b) \notin S_t(c)$ and either: (i) $a, b \in V(G_t)$; or (ii) $a \in V(G_t)$, $b \notin V(G_t)$ and there is $h \in V(G_{ab}) \cap V(G_t)$ such that $d(h, V(G_{ab}) \cap B_t) > r/2$.*

We will show in Lemmas 12 and 13 that $D_t(c) \subseteq D_t^* \subseteq D_t(c) \cup S_t(c)$, thus we only need to take care of demands in $D_t(c)$ in the subproblem. Formally, for each node t of the tree decomposition and coloring c of B_t, our algorithm computes a number $A_t(c)$ and a set of hubs $H \subseteq \mathcal{H} \cap V(G_t)$ of size $A_t(c)$ that satisfies the conditions below.

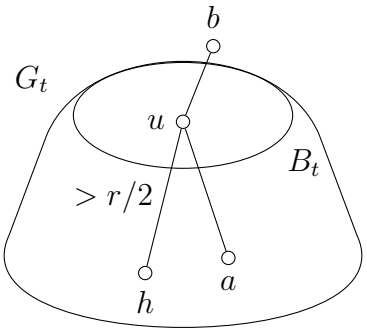


Figure 4.4: A demand in $D_t(c)$.

(C1) For every $u \in B_t$, if $c(u) = \downarrow i$, then there exists $h \in H$ and a shortest path P from u to h of length at most i such that $V(P) \subseteq V(G_t)$;

(C2) For every $(a, b) \in D_t(c)$, $\min_{h \in H} d(a, h) + d(h, b) \leq (1 + \epsilon)2r$.

If the algorithm does not find one such set, then it assigns $A_t(c) = \infty$.

Algorithm description We describe how to compute $A_t(c)$ for each node type (recall the meaning of the operator $\oplus$ in Subsection 2.5.2). For a *leaf node* t, we have $V(G_t) = \emptyset$, then $H = \emptyset$ satisfies the conditions, and we set $A_t(c_\emptyset) = 0$, where $c_\emptyset$ denotes the empty coloring.

For an *introduce node* t with child t', let u be the introduced vertex, such that $B_t = B_{t'} \cup \{u\}$, for $u \notin B_{t'}$. Let $I_t(c)$ be the set of colorings c' of $B_{t'}$ such that c' is the restriction of c to $B_{t'}$ and, if $c(u) = \downarrow i$ for some $i > 0$, there is $v \in B_{t'}$ with $c'(v) = \downarrow j$ such that $i = d(u, v) \oplus j$ (see Figure 4.5). Note that this set is either a singleton or is empty. If $I_t(c)$ is empty, discard c.

(a) u is chosen as a hub. (b) u connects to a hub.

Figure 4.5: The introduce case of MAkHC.

Define:

$$A_t(c) = \min_{\substack{c' \in I_t(c): \\ D_t(c) \subseteq D_{t'}(c')}} \begin{cases} A_{t'}(c') + 1 & \text{if } c(u) = \downarrow 0, \\ A_{t'}(c') & \text{otherwise.} \end{cases}$$

Let H' be the solution corresponding to $A_{t'}(c')$: we output $H = H' \cup \{u\}$ if $c(u) = \downarrow 0$, or $H = H'$ otherwise.

For a *forget node* t with child t', let u be the forgotten vertex, such that $B_t = B_{t'} \setminus \{u\}$, for $u \in B_{t'}$. Let $F_t(c)$ be the set of colorings c' of $B_{t'}$ such that c is the restriction of c'

to B_t and, if $c'(u) = \uparrow i$, then there is $v \in B_t$ such that $c(v) = \uparrow j$ and $i = d(u, v) \oplus j$ (see Figure 4.6). If $F_t(c)$ is empty, discard c.

(a) u is close to a hub in G_t. (b) u connects to a hub not in G_t.

Figure 4.6: The forget case of MAkHC.

Define:

$$A_t(c) = \min_{\substack{c' \in F_t(c): \\ D_t(c) \subseteq D_{t'}(c') \cup S_{t'}(c')}} A_{t'}(c').$$

We output as solution the set $H = H'$, where H' corresponds to the solution of the selected subproblem in t'.

For a *join node* t with children t' and t'', we have $B_t = B_{t'} = B_{t''}$. Let $J_t(c)$ be the set of pairs of colorings (c', c'') of B_t such that, for every $u \in B_t$, when $c(u)$ is $\downarrow 0$ or $\uparrow i$, then $c'(u) = c''(u) = c(u)$; else, if $c(u)$ is $\downarrow i$, then $(c'(u), c''(u))$ is either $(\uparrow i, \downarrow i)$ or $(\downarrow i, \uparrow i)$. If $J_t(c)$ is empty, discard c. Figure 4.7 depicts the case a vertex u is satisfied by a hub down the subproblem (node t''), thus its coloring in t' must be $c'(u) = \uparrow i$; vertex u', on the other hand, is satisfied by a hub up the subproblem, thus we have $c(u') = c'(u') = c''(u') = \uparrow i'$.

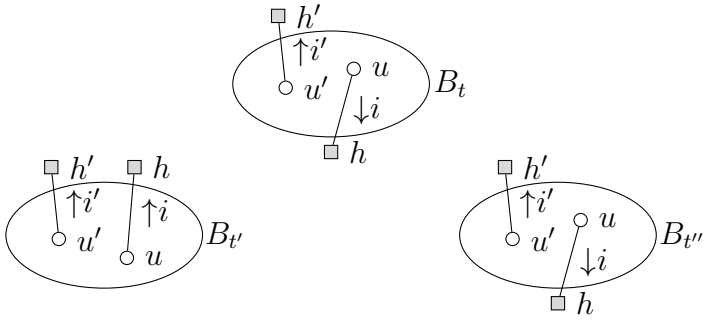


Figure 4.7: The join case of MAkHC.

Define:

$$A_t(c) = \min_{\substack{(c', c'') \in J_t(c): \\ D_t(c) \subseteq D_{t'}(c') \cup D_{t''}(c'')}} A_{t'}(c') + A_{t''}(c'') - h(c),$$

where $h(c)$ is the number of vertices u in B_t such that $c(u) = \downarrow 0$. We output a solution $H = H' \cup H''$, where H' and H'' are the solutions corresponding to t' and t'', respectively.

In the next lemma, we show that the algorithm indeed produces a solution of bounded size that satisfies both conditions.

Lemma 9. *If $A_t(c) \neq \infty$, then the algorithm outputs a set $H \subseteq \mathcal{H} \cap V(G_t)$, with $|H| \leq A_t(c)$, that satisfies* (C1) *and* (C2).

Proof. We prove the lemma by induction on the height of node t, thus assume the lemma holds for nodes below t. For *leaves*, the algorithm outputs an empty set, satisfying both conditions.

For an *introduce node* t with child t' and $u \in B_t \setminus B_{t'}$, let H' be the solution corresponding to t' with coloring c'. Since c' is the restriction of c to $B_{t'}$, condition (C1) is satisfied for every $v \in B_{t'}$, by induction. If $c(u) = \downarrow 0$, then it is satisfied for u, since, in this case, $u \in H$. Else, if $c(u) = \downarrow i$ for $i > 0$, then it is also satisfied, since in this case there is $v \in B_{t'}$ with $c'(v) = \downarrow j$ such that $i = d(u, v) \oplus j$. Condition (C2) is satisfied as well, since $D_t(c) \subseteq D_{t'}(c')$, and H' satisfies (C2).

For a *forget node* t with child t' and $u \in B_{t'} \setminus B_t$, we have that $H = H'$, where H' is the solution to t' corresponding to some coloring c' of $B_{t'}$. Since c is the restriction of c' to B_t, H satisfies (C1) by induction. For (C2), let $(a, b) \in D_t(c)$, and remember that $D_t(c) \subseteq D_{t'}(c') \cup S_{t'}(c')$. If $(a, b) \in D_{t'}(c')$, then this demand is satisfied by H with cost at most $(1 + \epsilon)2r$. Else, $(a, b) \in S_{t'}(c')$, but $(a, b) \notin S_t(c)$. Thus, for the forgotten vertex u, we have $c'(u) \in \{\uparrow i, \downarrow i\}$ and $d(a, u) + 2i + d(u, b) \leq (1 + \epsilon)2r$. We consider two cases:

- If $c'(u) = \downarrow i$, then, since H satisfies (C1), there is $h \in H$ such that the distance from u to h is at most i. Thus condition (C2) is satisfied, because

$$d(a, h) + d(h, b) \leq d(a, u) + 2i + d(u, b) \leq (1 + \epsilon)2r.$$

- If $c'(u) = \uparrow i$, there is $v \in B_t$ with $c(v) = \uparrow j$ and $i = d(u, v) \oplus j$. We get

$$\begin{aligned} d(a, v) + 2j + d(v, b) &\leq d(a, u) + 2(d(u, v) + j) + d(u, b) \\ &\leq d(a, u) + 2i + d(u, b) \\ &\leq (1 + \epsilon)2r, \end{aligned}$$

  where we used $d(u, v) + j \leq d(u, v) \oplus j = i$ in the second inequality. But this means that $(a, b) \in S_t(c)$, which is a contradiction.

For a *join node* t with children t' and t'', let H' and H'' be solutions for the subproblems at t' and t'' corresponding to the selected pair of colorings c' and c''. We claim that $H = H' \cup H''$ satisfies both conditions. For (C1), note that if $c(u) = \downarrow i$, for some $u \in B_t$, then $c'(u) = \downarrow i$ or $c''(u) = \downarrow i$. For (C2), note that, for $(a, b) \in D_t(c)$, we have $(a, b) \in D_{t'}(c') \cup D_{t''}(c'')$, and thus this demand is satisfied with cost at most $(1+\epsilon)2r$ by a vertex in H' or H''. □

Let t_0 be the root of the tree decomposition and $c_\emptyset$ be the empty coloring. Since the bag corresponding to the root node is empty, we have $S_{t_0}(c_\emptyset) = \emptyset$ and thus $D_{t_0}(c_\emptyset) = \mathcal{D}$. Therefore, if $A_{t_0}(c_\emptyset) \leq k$, Lemma 9 implies that the set of hubs H computed by the algorithm is a feasible solution that satisfies each demand with cost at most $(1 + \epsilon)2r$. In the next section, we bound the size of H by the size of the global optimal solution H^*.

4.3 Analysis

For each node t of the tree decomposition, we want to show that the number of hubs computed by the algorithm for some coloring c of B_t is not larger than the number of hubs of H^* contained in G_t, that is, we would like to show that $A_t(c) \leq |H^* \cap V(G_t)|$ for some c. If the distances from each vertex $u \in B_t$ to its closest hub in H^* were stored exactly, then the partial solution corresponding to H^* would induce one such coloring c_t^*, and we could show the inequality for this particular coloring.

More precisely, for each $u \in V(G)$, let $h^*(u)$ be a hub of H^* such that $d(u, h^*(u))$ is minimum and $P^*(u)$ be a corresponding shortest path. Assume that each $P^*(u)$ is obtained from a shortest path tree to $h^*(u)$ and that it has the minimum number of edges among the shortest paths. The *signature* of H^* corresponding to a partial solution in G_t is a function c_t^* on B_t such that

$$c_t^*(u) = \begin{cases} \downarrow d(u, h^*(u)) & \text{if } V(P^*(u)) \subseteq V(G_t), \\ \uparrow d(u, h^*(u)) & \text{otherwise.} \end{cases}$$

Since distances are stored approximately as integer powers of $(1+\delta)$, the function c_t^* might not be a valid coloring. Instead, we show that the algorithm considers a coloring $\bar{c}_t$ with roughly the same values of c_t^* and that its values are computed by approximate addition trees. We say that an addition tree and an approximate addition tree are *corresponding* if they are isomorphic and have the same input values. Also, recall that a coloring c of B_t is *discarded* by the algorithm if the set $I_t(c)$, $F_t(c)$ or $J_t(c)$ corresponding to t is empty.

Lemma 10. *Let ℓ_{t_0} be the height of the tree decomposition. There exists a coloring $\bar{c}_t$ that is not discarded by the algorithm such that, for every $u \in B_t$, the values $c_t^*(u)$ and $\bar{c}_t(u)$ are computed, respectively, by an addition tree and a corresponding approximate addition tree of height at most $2\ell_{t_0}$.*

Proof. A *partial addition tree* is a pair (T, p), where T is an addition tree and p is a leaf of T. The vertex p represents a subtree that computes a pending value x_p, and may be replaced by some other (partial) addition tree that computes this value.

For some node t, let ℓ_t be the height of t and define U_t as the set of vertices $u \in B_t$ such that $c_t^*(u) = \uparrow i$, for some i. We say that a vertex $v \in V(G_t) \setminus U_t$ is *t-complete* according to the following cases:

- if $V(P^*(v)) \subseteq V(G_t)$ and $v \in B_t$, then $d(v, h^*(v))$ is computed by an addition tree of height at most ℓ_t;
- if $V(P^*(v)) \subseteq V(G_t)$ and $v \notin B_t$, then $d(v, h^*(v))$ is computed by an addition tree of height at most $2\ell_t$;
- if $V(P^*(v)) \not\subseteq V(G_t)$, then $d(v, h^*(v))$ is computed by a partial addition tree (T, p) of height at most ℓ_t such that $x_p = d(w, h^*(w))$, for some $w \in U_t$.

We will show by induction on the height of t that every $v \in V(G_t) \setminus U_t$ is t-complete. The claim holds trivially for leaves, thus suppose that t is not a leaf.

Assume t is an *introduce node* with child t', and let u be the introduced vertex. Since $U_{t'} \subseteq U_t$, if $v \in V(G_{t'}) \setminus U_{t'}$, then v is t-complete by the induction hypothesis. Else, we have $v = u$ and, since $v \notin U_t$, $c_t^*(v) = \downarrow d(v, h^*(v))$. Thus, there is $w \in B_{t'} \setminus U_{t'}$ such that $c_{t'}^*(w) = \downarrow d(w, h^*(w))$ and $d(v, h^*(v)) = d(v, w) + d(w, h^*(w))$. Since $d(w, h^*(w))$ can be computed by an addition tree of height at most $\ell_{t'}$, this implies that $d(v, h^*(v))$ can be computed by an addition tree of height at most $\ell_{t'} + 1 \leq \ell_t$.

Now, assume t is a *forget node* with child t', and let u be the forgotten vertex. Since $V(G_t) = V(G_{t'})$, if $V(P^*(v)) \subseteq V(G_{t'})$, then v is t-complete by the induction hypothesis. Otherwise, by the induction hypothesis, $d(v, h^*(v))$ is computed by a partial addition tree (T, p) of height at most $\ell_{t'}$ such that $x_p = d(w', h^*(w'))$, for some $w' \in U_{t'}$. If $w' \in U_t$, then v is t-complete. So, assume $w' \in U_{t'} \setminus U_t$, which implies that w' is the forgotten vertex u and $c_{t'}^*(u) = \uparrow d(u, h^*(u))$. Thus, $P^*(u)$ crosses some vertex $w \in U_t$ such that $d(u, h^*(u)) = d(u, w) + d(w, h^*(w))$. It follows that $d(u, h^*(u))$ can be computed by a partial addition tree (T_u, p_u) of height 1 such that $x_{p_u} = d(w, h^*(w))$. Therefore, we can replace the vertex p by the subtree T_u, and the height of T becomes at most $\ell_{t'} + 1 \leq \ell_t$.

Finally, assume t is a *join node* with children t' and t'', and recall that $B_t = B_{t'} = B_{t''}$. If $v \in B_t$, then $V(P^*(v)) \subseteq V(G_{t'})$ or $V(P^*(v)) \subseteq V(G_{t''})$, because B_t induces a clique and $P^*(v)$ is a shortest path with minimum number of edges. Thus, v is t-complete by the induction hypothesis. Otherwise, $v \in V(G_t) \setminus B_t$. Assume $v \in V(G_{t'}) \setminus B_t$, as the other case is analogous. By the induction hypothesis for t', $d(v, h^*(v))$ is computed by a partial addition tree (T', p) of height at most $\ell_{t'}$ such that $x_p = d(w, h^*(w))$ for some $w \in U_{t'}$. If $w \in U_t$, then v is t-complete. Thus, assume $w \notin U_t$, which implies that $V(P^*(w)) \subseteq V(G_t)$. Again, since B_t induces a clique, $P^*(w)$ is included in $V(G_{t'})$ or $V(G_{t''})$, but since $w \in U_{t'}$, we have $V(P^*(w)) \subseteq V(G_{t''})$. It follows that $c_{t''}^*(w) = \downarrow d(w, h^*(w))$. By the induction hypothesis for t'', $d(w, h^*(w))$ is computed by an addition tree T'' of height at most $\ell_{t''}$. Therefore, we can replace the vertex p by the subtree T'', and the height of T' becomes at most $\ell_{t'} + \ell_{t''} \leq 2\ell_t$. This completes the induction.

For the root node t_0, we have $B_{t_0} = \emptyset$, thus for every $v \in V(G)$, the distance $d(v, h^*(v))$ is computed by an addition tree T_v of height at most $2\ell_{t_0}$. Let $\bar{T}_v$ be the approximate addition tree corresponding to T_v, and define $\bar{d}(v)$ as the output of $\bar{T}_v$. For every node t, and $u \in B_t$, if $c_t^*(u) = \downarrow d(u, h^*(u))$, define $\bar{c}_t(u) = \downarrow \bar{d}(u)$; else, define $\bar{c}_t(u) = \uparrow \bar{d}(u)$. By repeating the arguments above, and replacing the addition operator by $\oplus$, one can show that, for every t, the coloring $\bar{c}_t$ is not discarded by the algorithm. □

By setting $\delta = \epsilon/(2\ell_{t_0} + 1)$, Theorem 4 implies the next lemma.

Lemma 11. *For every $u \in B_t$, if $c_t^*(u) \in \{\uparrow x, \downarrow x\}$ and $\bar{c}_t(u) \in \{\uparrow i, \downarrow i\}$, then $i \leq (1+\epsilon)x$.*

Recall that H^* is a fixed global optimal solution that satisfies each demand with cost r. Our goal is to bound $A_t(\bar{c}_t) \leq |H^* \cap V(G_t)|$ for every node t, thus we would like to determine the subset of demands D_t^* that are necessarily satisfied by hubs $H^* \cap V(G_t)$ in the subproblem definition. This is made precise in the following.

Definition 9. $D_t^* = \{(a, b) \in \mathcal{D} : \min_{h \in H^* \setminus V(G_t)} d(a, h) + d(h, b) > r\}$.

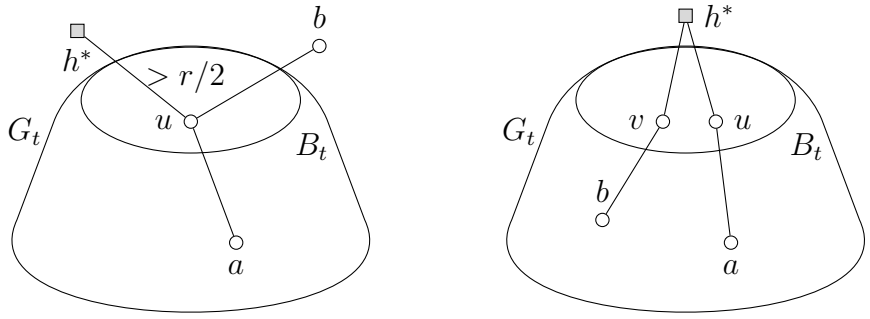
Since we cannot determine D_t^*, we show that, for each node t, the algorithm outputs a solution H for the subproblem corresponding to $A_t(\bar{c}_t)$ that satisfies every demand in $D_t(\bar{c}_t)$. In Lemma 12, we show that every demand in $D_t(\bar{c}_t)$ is also in D_t^*, as, otherwise, there could be no solution with size bounded by $|H^* \cap V(G_t)|$. Conversely, we show in Lemma 13 that a demand in D_t^* that is not in $D_t(\bar{c}_t)$ must be in $S_t(\bar{c}_t)$, thus all demands are satisfied.

Lemma 12. $D_t(\bar{c}_t) \subseteq D_t^*$.

Proof. Let $(a, b) \in D_t(\bar{c}_t)$ and consider an arbitrary hub $h^* \in H^*$ that satisfies (a, b) with cost r. We will show that $h^* \in V(G_t)$, and thus $(a, b) \in D_t^*$. For the sake of contradiction, assume that $h^* \in V(G) \setminus V(G_t)$.

First we claim that $d(h^*, V(G_{ab}) \cap B_t) > r/2$. If not, then let $u \in V(G_{ab}) \cap B_t$ be a vertex with $\bar{c}_t(u) \in \{\uparrow i, \downarrow i\}$ such that $d(u, h^*) \leq r/2$. Because the closest hub to u has distance at least $i/(1+\epsilon)$, we have $i \leq (1+\epsilon)d(u, h^*) \leq (1+\epsilon)r/2$, but since $u \in V(G_{ab})$, this implies that $(a, b) \in S_t(\bar{c}_t)$, and thus $(a, b) \notin D_t(\bar{c}_t)$ (see Figure 4.8(a)). Then, it follows that indeed $d(h^*, V(G_{ab}) \cap B_t) > r/2$.

Now we show that it cannot be the case that $a, b \in V(G_t)$. Suppose that $a, b \in V(G_t)$. Consider the shortest path from a to h^*, and let u be the last vertex of this path that is in $V(G_t)$. Since B_t separates $V(G_t) \setminus B_t$ from $V(G) \setminus V(G_t)$, it follows that $u \in B_t$. From the previous claim, $d(h^*, u) > r/2$, and thus $d(h^*, a) > r/2$. Analogously, there is $v \in B_t$ with $d(h^*, b) > d(h^*, v) > r/2$, but then $d(a, h^*) + d(h^*, b) > r$, which contradicts the fact that h^* satisfies (a, b) with cost r (see Figure 4.8(b)). This contradiction comes from supposing that $a, b \in V(G_t)$. Thus, either a or b is not in $V(G_t)$.



(a) (a, b) has no close hub. (b) a and b are not both in G_t.

Figure 4.8: Exemplifying the assumptions.

Assume without loss of generality that $a \in V(G_t)$ and $b \notin V(G_t)$. From the definition of $D_t(\bar{c}_t)$, we know that there exists $h \in V(G_{ab}) \cap V(G_t)$ such that $d(h, V(G_{ab}) \cap B_t) > r/2$. Let P be a path from a to b crossing h^* with length at most r. Similarly, since $h \in V(G_{ab})$, there exists a path Q from a to b crossing h with length at most r. Let u be the last vertex of P with $u \in B_t$, and let v be the last vertex of Q with $v \in B_t$ (see Figure 4.9). Concatenating P and Q leads to a closed walk of length at most $2r$. This walk crosses u,

h^*, v and h, and thus

$$\begin{aligned} 2r &\geq d(a,h^*) + d(h^*,b) + d(a,h) + d(h,b) \\ &= d(u,h^*) + d(h^*,v) + d(v,h) + d(h,u) \\ &> 2r, \end{aligned} \tag{4.1}$$

where we used the fact that each term in (4.1) is greater than $r/2$. This is a contradiction, so $h^* \in V(G_t)$ and then $(a,b) \in D_t^*$.

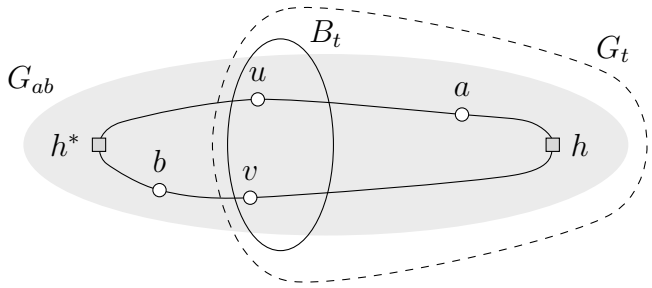


Figure 4.9: Closed walk formed by P and Q.

□

Lemma 13. *$D_t^* \subseteq D_t(\bar{c}_t) \cup S_t(\bar{c}_t)$.*

Proof. Let $(a,b) \in D_t^*$. Assume $(a,b) \notin S_t(\bar{c}_t)$, as otherwise we are done. If $a, b \in V(G_t)$, we have $(a,b) \in D_t(\bar{c}_t)$. Thus, suppose without loss of generality that $a \in V(G_t)$ and $b \notin V(G_t)$. Since $(a,b) \in D_t^*$, there is $h^* \in H^* \cap V(G_{ab}) \cap V(G_t)$. Let $u \in V(G_{ab}) \cap B_t$ with $\bar{c}_t(u) \in \{\uparrow i, \downarrow i\}$ for some i. Because $(a,b) \notin S_t(\bar{c}_t)$, we have $i > (1+\epsilon)r/2$. But the distance from u to the closest hub in H^* is at least $i/(1+\epsilon)$, thus $i \leq (1+\epsilon)d(u,h^*)$. It follows that $d(u,h^*) > r/2$. Therefore, $h^* \in V(G_{ab}) \cap V(G_t)$ and $d(h^*, V(G_{ab}) \cap B_t) > r/2$, and then $(a,b) \in D_t(\bar{c}_t)$. □

Before bounding the number of hubs opened by the algorithm, we prove some auxiliary results.

Lemma 14. *If t is an introduce node with child t', then $D_t(\bar{c}_t) \subseteq D_{t'}(\bar{c}_{t'})$.*

Proof. We claim that $D_t^* \setminus D_{t'}^* \subseteq S_t(\bar{c}_t)$. Let u be introduced vertex, and note that $V(G_t) \setminus V(G_{t'}) = \{u\}$. If $(a,b) \in D_t^* \setminus D_{t'}^*$, by definition, we know that $\min_{h \in H^* \setminus V(G_t)} d(a,h) + d(h,b) > r$, but $\min_{h \in H^* \setminus V(G_{t'})} d(a,h) + d(h,b) \leq r$. This can only happen if $u \in H^*$, so $\bar{c}_t(u) = \downarrow 0$, and then $(a,b) \in S_t(\bar{c}_t)$.

Since $D_{t'}^* \subseteq D_t^*$ and $S_{t'}(\bar{c}_{t'}) \subseteq S_t(\bar{c}_t)$, the claim implies $D_t^* \setminus S_t(\bar{c}_t) \subseteq D_{t'}^* \setminus S_{t'}(\bar{c}_{t'})$. Using Lemmas 12 and 13, we get

$$D_t(\bar{c}_t) \subseteq D_t^* \setminus S_t(\bar{c}_t) \subseteq D_{t'}^* \setminus S_{t'}(\bar{c}_{t'}) \subseteq D_{t'}(\bar{c}_{t'}).$$

□

Lemma 15. *If t is a forget node with child t', then $D_t(\bar{c}_t) \subseteq D_{t'}(\bar{c}_{t'}) \cup S_{t'}(\bar{c}_{t'})$.*

Proof. In this case $V(G_t) = V(G_{t'})$, thus $D^*_t = D^*_{t'}$. Using Lemmas 12 and 13,

$$D_t(\bar{c}_t) \subseteq D^*_t = D^*_{t'} \subseteq D_{t'}(\bar{c}_{t'}) \cup S_{t'}(\bar{c}_{t'}).$$

□

Lemma 16. *If t is a join node with children t' and t'', then $D_t(\bar{c}_t) \subseteq D_{t'}(\bar{c}_{t'}) \cup D_{t''}(\bar{c}_{t''})$.*

Proof. We claim that $D^*_t \setminus (D^*_{t'} \cup D^*_{t''}) \subseteq S_t(\bar{c}_t)$. Let $(a,b) \in D^*_t \setminus (D^*_{t'} \cup D^*_{t''})$ and suppose, for a contradiction, that $(a,b) \notin S_t(\bar{c}_t)$. Then, for every $h^* \in H^*$ and $u \in V(G_{ab}) \cap B_t$, we have $d(h^*, u) > r/2$. Since $(a,b) \in D^*_t$, but $(a,b) \notin D^*_{t''}$, there is $h' \in H^* \cap V(G_{t'}) \setminus B_t$ that satisfies (a,b). Similarly, there is $h'' \in H^* \cap V(G_{t''}) \setminus B_t$. Now, $h', h'' \in V(G_{ab})$, but the diameter of G_{ab} is at most r, thus $d(h', h'') \leq r$. Since B_t separates h' and h'', there is $u \in B_t$ with $d(h', u) + d(u, h'') \leq r$. Thus, either $d(h', u) \leq r/2$ or $d(h'', u) \leq r/2$, a contradiction. This implies $(a,b) \in S_t(\bar{c}_t)$.

Observe that $D^*_{t'} \cup D^*_{t''} \subseteq D^*_t$ and $S_t(\bar{c}_t) = S_{t'}(\bar{c}_{t'}) = S_{t''}(\bar{c}_{t''})$. Combining with Lemmas 12 and 13, we get

$$D_t(\bar{c}_t) \subseteq D^*_t \setminus S_t(\bar{c}_t) \subseteq (D^*_{t'} \setminus S_{t'}(\bar{c}_{t'})) \cup (D^*_{t''} \setminus S_{t''}(\bar{c}_{t''})) \subseteq D_{t'}(\bar{c}_{t'}) \cup D_{t''}(\bar{c}_{t''}).$$

□

Combining Lemma 10 and Lemmas 14–16, we can show that the algorithm does not open too many hubs.

Lemma 17. $A_t(\bar{c}_t) \leq |H^* \cap V(G_t)|$.

Proof. Assume the lemma holds for the children of t. For a *leaf node*, the output set is empty, and the inequality is satisfied trivially.

Let t be an *introduce node* with child t' and $u \in B_t \setminus B_{t'}$. From Lemmas 10 and 14, we know that $\bar{c}_{t'} \in I_t(\bar{c}_t)$ and $D_t(\bar{c}_t) \subseteq D_{t'}(\bar{c}_{t'})$. Thus, if $\bar{c}_t(u) = \downarrow 0$, we have $u \in H^*$ and $A_t(\bar{c}_t) = A_{t'}(\bar{c}_{t'}) + 1 \leq |H^* \cap V(G_{t'})| + 1 = |H^* \cap V(G_t)|$. Otherwise, $A_t(\bar{c}_t) = A_{t'}(\bar{c}_{t'}) \leq |H^* \cap V(G_{t'})| = |H^* \cap V(G_t)|$.

Let t be a *forget node* with child t' and $u \in B_{t'} \setminus B_t$. From Lemmas 10 and 15, we know that $\bar{c}_{t'} \in F_t(\bar{c}_t)$ and $D_t(\bar{c}_t) \subseteq D_{t'}(\bar{c}_{t'}) \cup S_{t'}(\bar{c}_{t'})$. Thus, $A_t(\bar{c}_t) \leq A_{t'}(\bar{c}_{t'}) \leq |H^* \cap V(G_{t'})| = |H^* \cap V(G_t)|$.

Let t be a *join node* with children t' and t''. From Lemmas 10 and 16, we know that $(\bar{c}_{t'}, \bar{c}_{t''}) \in J_t(\bar{c}_t)$ and $D_t(\bar{c}_t) \subseteq D_{t'}(\bar{c}_{t'}) \cup D_{t''}(\bar{c}_{t''})$. Let H' and H'' be the output solutions corresponding to t' and t'', respectively. We have

$$\begin{aligned} |H| &= |H'| + |H''| - |H' \cap H''| \\ &\leq A_{t'}(\bar{c}_{t'}) + A_{t''}(\bar{c}_{t''}) - h(\bar{c}_t) \\ &\leq |H^* \cap V(G_{t'})| + |H^* \cap V(G_{t''})| - h(\bar{c}_t) \\ &= |H^* \cap V(G_t)|. \end{aligned}$$

□

Now we can state the main result of this chapter.

Theorem 13. *For every $\epsilon > 0$, there is a parameterized $(2+\epsilon)$-approximation algorithm for* MAkHC *running in time $\mathcal{O}^*((\mathrm{tw}/\epsilon)^{\mathcal{O}(\mathrm{tw})})$.*

Proof. Consider a preprocessed instance $(G, \mathcal{C}, \mathcal{H}, \mathcal{D}, k)$ of MAkHC, in which the optimal value OPT is an integer bounded by $\mathcal{O}(\frac{1}{\epsilon}|V(G)|)$. We run the dynamic programming algorithm for each $r = 1, 2, \ldots$, and output the first solution with no more than k hubs. Next, we show that the dynamic programming algorithm either correctly decides that there is no solution of cost r that opens k hubs, or finds a solution of cost $(1+\epsilon)2r$ that opens k hubs. Thus, when the main algorithm stops, $r \leq \mathrm{OPT}$, and the output is a $(2+\epsilon')$-approximation, for a suitable ϵ'.

Assume H^* is a solution that satisfies each demand with cost r with minimum size. Recall that t_0 is the root of the tree decomposition and $c_\emptyset$ is the coloring of an empty bag. If $A_{t_0}(c_\emptyset) \leq k$, then Lemma 9 states that the dynamic programming algorithm outputs a set of hubs H of size at most k that satisfies each demand in $D_{t_0}(c_\emptyset) = \mathcal{D}$ with cost $(1+\epsilon)2r$. Otherwise, $k < A_{t_0}(c_\emptyset)$, and Lemma 17 implies $k < A_{t_0}(c_\emptyset) \leq |H^* \cap V(G_{t_0})| = |H^*|$. Thus, by the minimality of H^*, there is no solution of cost r that opens k hubs.

Finally, we bound the running time. Let $n = |V(G)|$. The tree decomposition has $\mathcal{O}(\mathrm{tw} \cdot n)$ nodes and, for each node t, the number of colorings is $|\Sigma|^{\mathcal{O}(\mathrm{tw})}$. Also, each recurrence can be computed in time $\mathcal{O}^*(|\Sigma|^{\mathcal{O}(\mathrm{tw})})$. Since $r = \mathcal{O}(\frac{n}{\epsilon})$ and $\delta = \Theta\big(\frac{\epsilon}{\mathrm{tw}\cdot\log n}\big)$, the size of Σ is

$$\begin{aligned}|\Sigma| &= \mathcal{O}\left(\log_{1+\delta} r\right) = \mathcal{O}\left(\frac{\log r}{\log(1+\delta)}\right) = \mathcal{O}\left(\frac{\log n + \log(1/\epsilon)}{\delta}\right) \\ &= \mathcal{O}\left((\mathrm{tw}/\epsilon)\left(\log^2 n + \log n \log(1/\epsilon)\right)\right) = \mathcal{O}\left((\mathrm{tw}/\epsilon)^2 \log^2 n\right).\end{aligned}$$

Notice that $\mathcal{O}(\log^{\mathcal{O}(\mathrm{tw})} n) = \mathcal{O}^*(2^{\mathcal{O}(\mathrm{tw})})$, thus the total running time is bounded by $\mathcal{O}^*\left(|\Sigma|^{\mathcal{O}(\mathrm{tw})}\right) = \mathcal{O}^*\left((\mathrm{tw}/\epsilon)^{\mathcal{O}(\mathrm{tw})}\right)$. □

4.4 The planar case

In this section, we give a $(2+\epsilon)$-approximation algorithm parameterized by k and r, when the input is restricted to unweighted planar graphs. This algorithm can be seen as another way to challenge the approximation lower bound presented in Section 4.1. Indeed, by Theorem 11, finding a $(3-\epsilon)$-approximation parameterized by k and r is W[2]-hard for unweighted graphs, even when r is a constant. Thus, we restrict the input to planar graphs, but get a better approximation factor.

The algorithm is built upon the bidimensionality framework, and is inspired by the work for k-Center by Demaine et al. [62]. In the following, let $(G, \mathcal{C}, \mathcal{H}, \mathcal{D}, k, r)$ be a positive instance of MAkHC such that G is an unweighted planar graph.

Lemma 18. *If G has a $(\rho \times \rho)$-grid as minor, then $\rho \leq \sqrt{k}(2r+1) + 2r$.*

Proof. We begin with a series of definitions.

Let F be a $(\rho \times \rho)$-grid, where

$$V(F) = \{1, \ldots, \rho\} \times \{1, \ldots, \rho\} \quad \text{and}$$
$$E(F) = \{((x, y), (x', y')) : |x - x'| + |y - y'| = 1\}.$$

Let V_{ext} be the set of vertices of F whose degrees are smaller than 4. We assume the vertices of V_{ext} belong to the external face of some embedding of F and call the other faces *internal*.

Let V_{int} be the set of vertices of F that have distance at least r from every vertex in V_{ext}. Note that V_{int} induces a subgraph $F[V_{int}]$ that is a subgrid of F with $|V_{int}| = (\rho - 2r)^2$.

Define $\delta((x, y), (x', y')) = \max\{|x - x'|, |y - y'|\}$ and let $d_F((x, y), (x', y'))$ be the distance from (x, y) to (x', y') in F.

Let J be the supergraph of F with the same set of vertices and with the additional set of (diagonal) edges:

$$\{((x, y), (x + 1, y + 1)),\ ((x, y + 1), (x + 1, y)) : 1 \leq x, y < \rho\}.$$

Let R be a subgraph of J and let $d_R(u, v)$ be the length of a shortest path from u to v in R. Observe that, for every $u, v \in V(R)$, we have $\delta(u, v) \leq d_R(u, v)$. Define $N_R^\ell(u) = \{v : d_R(u, v) \leq \ell\}$.

For a $(x, y) \in V(F)$ and an integer ℓ, define

$$B_\ell((x, y)) = \{(x', y') : \delta((x, y), (x', y')) \leq \ell\}.$$

Now, consider a sequence of edge contractions and removals which transforms G into a minor isomorphic to F using a maximal number of edge contractions. Let H be the result of applying only the contractions of that sequence to G, and consider an embedding of H in the plane that corresponds to an embedding of F. Partition the edges of H in three sets: the edges that occur in F, the set E_1 that connect non-adjacent vertices of an internal face of F, and the set E_2 with all other edges. Note that edges in E_2 are only incident with vertices in V_{ext}.

Call R the graph we obtain by adding edges E_1 to F, and note that R is a subgraph of J. Then, for a vertex u of R and an integer ℓ, we have that $N_R^\ell(u) \subseteq B_\ell(u)$. Observe that the set of edges of H is $E(R) \cup E_2$. For a vertex $u \in V_{int}$, we claim that $N_H^r(u) \subseteq B_r(u)$. This holds because paths of length at most r starting at a vertex of V_{int} do not use edges of E_2 and, as a consequence, $N_H^r(u) = N_R^r(u)$.

Let S be a solution for the instance of MAkHC. Observe that the distance between every client and a hub of S is at most r, since every vertex is in some set $V(G_{ab})$. Also, note that, for vertices u and v of G associated with vertices u' and v' of H, $d_H(u', v') \leq d_G(u, v)$, as H is obtained from G using only edge contractions.

Define a set of vertices:

$$Y = V_{int} \cap \{((2r + 1)i + r + 1, (2r + 1)j + r + 1) : i, j \in \mathbb{Z}_{\geq 0}\}.$$

The size of this set is $|Y| \geq \left\lceil \frac{\rho - 2r}{2r+1} \right\rceil^2 \geq \left(\frac{\rho - 2r}{2r+1} \right)^2$.

For distinct $y, y' \in Y$, we have $B_r(y) \cap B_r(y') = \emptyset$. Also, there must exist a hub in S that is associated with some vertex in $N_H^r(y) \subseteq B_r(y)$. Therefore, each $y \in Y$ is associated to one unique hub in S, and finally,

$$k \geq |Y| \geq \left(\frac{\rho - 2r}{2r+1}\right)^2.$$

□

Corollary 1. $\mathrm{tw}(G) \leq 6\sqrt{k}(2r+1) + 12r + 1$.

Proof. Robertson, Seymour and Thomas [153] prove that, if G has no $((\rho{+}1){\times}(\rho{+}1))$-grid as a minor, then $\mathrm{tw}(G) \leq 6(\rho{+}1){-}5$. Let ρ be the largest integer for which G has a $(\rho{\times}\rho)$-grid as a minor. Then, using Lemma 18, we have that $\mathrm{tw}(G) \leq 6\sqrt{k}(2r{+}1){+}12r{+}1$. □

Using the previous bound and Theorem 13, we obtain an algorithm for the planar case.

Theorem 14. *For every $\epsilon > 0$, there is a parameterized $(2+\epsilon)$-approximation algorithm for* MAkHC *when the parameters are k and r, and the input graph is unweighted and planar.*

4.5 Future work

In this chapter, we focused on giving an improved algorithm for MAkHC that, albeit runs in FPT time, bypasses the lower bound of 3 on the factor of polynomial-time approximation algorithms. From the parameterized side, we can rule out an exact algorithm for various parameters, such as vertex-cover number, highway dimension and the number of hubs. We give a $(2+\epsilon)$-approximation parameterized by treewidth, for $\epsilon > 0$, which still leaves an interesting gap to be explored by either algorithmic or hardness results.

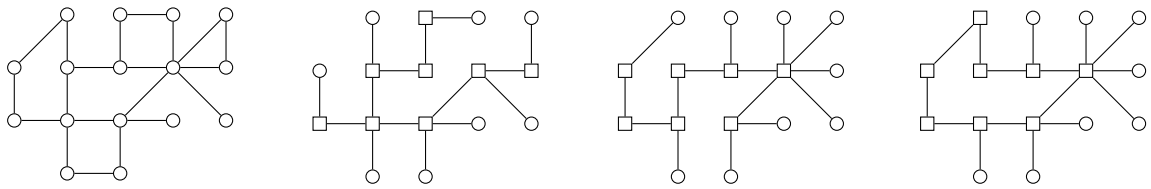
An orthogonal research direction is applying our strategy to deal with the set of demands to similar problems, for which the vertices that need to be connected are given in the input. Instead of trying to connect the exact subset of demands of a subproblem, we settle for connecting a subset of these vertices and relaxing the cost requirement for the remaining ones. In our analysis, this increased the cost of a solution by a factor of at most 2, but different ideas and connectivity constraints might lead to other guarantees.

Chapter 5

Spanning Tree-Star and variants

In this chapter, we consider a family of problems that aim to design a minimum-cost network respecting a prespecified configuration. For a graph G and a connected spanning subgraph H of G, if a vertex u has only one neighbor in H, we say that u is an *external vertex*, otherwise, it is an *internal vertex*. An edge $\{u, v\} \in E(H)$ is an *internal edge* if u and v are internal vertices, otherwise, it is an *external edge*. Let $\mathcal{I}(H)$ and $\mathcal{X}(H)$ be the sets of internal and external edges of H, respectively.

In the Spanning Tree-Star (STS), we are given a connected undirected graph G and two edge-cost functions $c, d : E(G) \to \mathbb{N}$, and a solution is a connected spanning subgraph[1] H of G, such that the internal vertices of H induce a tree. The goal is to find a solution that minimizes $\sum_{e \in \mathcal{I}(H)} c(e) + \sum_{e \in \mathcal{X}(H)} d(e)$. We also consider variants that ask for other types of networks, namely, the Spanning Path-Star (SPS) and the Spanning Cycle-Star (SCS), where the internal vertices induce, respectively, a path and a cycle. Figure 5.1 shows the types of network we want for the same input graph, where internal and external vertices are represented in a solution as squares and circles, respectively.



(a) Input graph. (b) Solution for STS. (c) Solution for SPS. (d) Solution for SCS.

Figure 5.1: Different types of networks of STS, SPS and SCS.

Related works These network design problems have been studied in the last decades under a diverse nomenclature, mostly with exact formulations and computational experiments. They often model complex real-world applications that arise in hierarchical vehicle routing, optical fiber networks and rapid transit systems [2, 118, 10].

The first studies on STS were carried in the context of designing telecommunication networks for digital data services, using integer programming approaches [123, 122]. Subsequent papers dealt with the problem applying similar techniques [113, 129, 6].

[1]The solution for STS is a tree – subgraph is used to accommodate variant definitions.

Labbé et al. [118] formulate SCS as a mixed-integer linear program and give several valid inequalities. The capacitated version of the problem is studied by Baldacci et al. [10]. Several papers consider the problem with multiple depots, in the sense that a solution is a collection of cycle-star networks, each containing one open depot [99, 156, 11]. Simonetti et al. [154] empirically test their formulation for SPS, that so far was only used as a hardness source to other problems [158]. There are also related problems in which the objective is to find an underlying network spanning a given set of vertices [112, 124, 147].

The problems we investigate have interesting results in the parameterized world. In the Minimum Leaf Spanning Tree the task is to find a spanning tree that minimizes the number of leaves; there is an XP algorithm for this problem and its directed variant, parameterized by clique-width, running in time $\mathcal{O}(n^{2^{\mathcal{O}(\text{cw})}})$ [88]. The associated task of finding a spanning tree with at least k leaves admits an algorithm running in time $\mathcal{O}^*(3.4575^k)$. Note that STS generalizes both these problems.

Dinneen and Khosravani [64] were the first to consider STS, SPS and SCS in the parameterized setting. They provided algorithms for SPS, parameterized by pathwidth or treewidth, which are expanded to STS and SCS, running in time roughly $\mathcal{O}^*(5^{\text{tw}} \cdot \text{tw}^{2\text{tw}})$. The variant of SPS for directed graphs, in which a solution must span only a given set of terminals, is studied by Okada et al. [141], that gave an algorithm with a similar running time.

For some time, researchers could not find improved algorithms for connectivity problems, such as Connected Dominating Set and Steiner Tree, whose best algorithms had the usual running time of $\mathcal{O}^*(\text{tw}^{\mathcal{O}(\text{tw})})$, when parameterized by treewidth. The major belief was that the global connectivity requirement imposed algorithms to keep track of all the ways a partial solution interacts with a separator of the tree decomposition. The first technique to solve these problems in single-exponential time was developed by Cygan et al. [59], however, the use of randomization raised the question whether deterministic algorithms were possible.

Bodlaender et al. [24] answered this query positively, and introduced the general framework of *rank-based approach* to solve connectivity problems deterministically, using the idea of representative partial solutions. This strategy is capable of speeding up dynamic programming algorithms, given that they are formulated with a set of operators and connectivity is expressed via sets of partitions. The authors applied the framework to fundamental problems, giving algorithms for Steiner Tree and Traveling Salesman running in time $\mathcal{O}^*(2^{3.6\text{tw}})$ and $\mathcal{O}^*(2^{3.3\text{tw}})$, respectively.

The framework has been expanded to consider more width parameters, such as branchwidth, clique-width and rankwidth [148, 17, 18]. The ideia of representative partial solutions is as follows: given sets of partial solutions $\mathcal{S}$ and $\mathcal{S}'$, we say $\mathcal{S}'$ *represents* $\mathcal{S}$ if, whenever a solution from $\mathcal{S}$ can be completed into an optimal solution, there is a solution in $\mathcal{S}'$ that can also be completed into an optimal solution. At each node of the tree decomposition, we build partial solutions using a dynamic programming formulation; then, the framework provides a representative set of these partial solutions of limited size, which ultimately speeds up the algorithm, since this set will be used by higher nodes of the decomposition.

Our results and techniques We give improved parameterized results for the considered family of network design problems. We primarily investigate the tree variant, from which we derive other algorithmic results. Using the rank-based approach, we give parameterized algorithms for STS running in time $\mathcal{O}^*(4^{1.7\text{pw}})$ and $\mathcal{O}^*(4^{3.5\text{tw}})$, with pathwidth and treewidth as parameters, respectively. Note that no single-exponential time algorithm was previously known for this problem. Moreover, using standard complexity-theoretic assumptions, we prove that: (*i*) there is no $g(k)$-approximation parameterized by the cost k of an optimal solution (*ii*) the best base of the exponent in the running time when parameterized by pathwidth one can hope for is 4, and (*iii*) we are unlikely to obtain an algorithm parameterized by clique-width. We also show that these positive and some negative results can be extended to SPS and SCS.

As a secondary contribution, we expand the collection of connectivity problems that can be solved with the rank-based approach. Our algorithms identify and label vertices according to their roles in a partial solution, that depends on its neighbors. In STS, for example, a vertex of a partial solution has four possible roles: internal or external vertex of a connected component, an external vertex of a small component, or an isolated vertex. These labels change whenever an edge is included in a partial solution or two partial solutions are joined, which alters the contribution of edges to the total cost of a solution. In the path and cycle variants of the problem, more labels are needed to represent all the roles present in a partial solution, as the structure of the network formed by internal vertices is less flexible, causing vertices to interact in a more intricate way.

The remaining of this chapter is organized as follows: Section 5.1 introduces the rank-based approach, Section 5.2 brings our hardness results for the studied problems, Section 5.3 gives the algorithms for STS parameterized by pathwidth and treewidth, and Section 5.4 extends these results to the path and cycle variants.

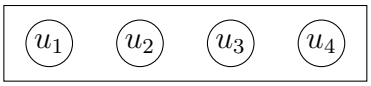
5.1 The rank-based approach

In this section, we introduce the framework of Bodlaender et al. [24] to solve connectivity problems parameterized by treewidth.

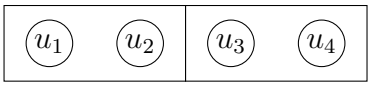
A dynamic programming algorithm stores partial solutions in a table, using them to compose solutions for bigger subproblems. Suppose there are partial solutions a and a' in this table such that, for each extension of a to a complete solution $a \cdot b$, a' can be extended in the same way $a' \cdot b$, with cost at most that of $a \cdot b$. In this way, we can delete a and effectively reduce the size of the table. The rank-based approach uses this idea to obtain single-exponential time parameterized algorithms for connectivity problems, with the treewidth of the input graph as parameter.

In order to decrease the size of a dynamic programming table A, for each newly computed entry, an algorithm `reduce` that computes a representative set of solutions is invoked, and this entry is stored instead. This technique was introduced by Bodlaender et al. [24], along with a collection of operators that allows one to apply the same framework to any dynamic programming algorithm, given that it is defined with these operators.

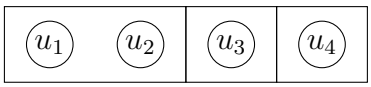
Let U be a ground set. Denote by $\Pi(U)$ the set of all partitions of U. For a partition $p \in \Pi(U)$, we call each part of p a block, and let $\#\texttt{blocks}(p)$ denote the number of blocks of p. For $X \subseteq U$, $U[X]$ is the partition of U where one block is X and all others are singletons. For partitions $p, q \in \Pi(U)$, $p \sqsubseteq q$ if, and only if, every block of q is a subset of some block of p, i.e., q refines p. Observe that, for $u, v \in U$, we have $p \sqsubseteq U[\{u, v\}]$ if, and only if, u and v are in the same block in p. Figure 5.2 exemplifies a ground set and two partitions p and q, where $p \sqsubseteq q$.



(a) A ground set U.



(b) A partition $p = \{\{u_1, u_2\}, \{u_3, u_4\}\}$.



(c) A partition $q = U[\{u_1, u_2\}]$.

Figure 5.2: Example of partitions.

For partitions $p, q \in \Pi(U)$, the join operation is denoted as $p \sqcup q$, and is defined in graph terms: build a graph with U as the vertex set and add an edge between vertices within the same block of p and q; the result of the join operation is the partition of U into the connected components of this graph. For $X \subseteq U \subseteq X'$ and $p \in \Pi(U)$, let $p_{\downarrow X} \in \Pi(X)$ be the partition obtained by removing all elements not in X from p, and let $p_{\uparrow X'} \in \Pi(X')$ be the partition obtained from p by adding singletons for every element in $X' \setminus U$.

A set of weighted partitions $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$ is a family of pairs, where each element is a partition of U and a non-negative integer weight. These pairs represent the connectivity and the cost of partial solutions, where vertices are connected according to the associated partition, i.e., vertices in the same block are in the same connected component. The operators presented in the following can be naturally applied to connectivity problems.

Definition 10 (Bodlaender et al. [24])**.** *Let U be a ground set and $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$, and define $\texttt{rmc}(\mathcal{A}) = \{(p, w) \in \mathcal{A} : \nexists (p, w') \in \mathcal{A} \wedge w' < w\}$. Consider the following operators:*

- $\texttt{union}(\mathcal{A}, \mathcal{B})$ *combines two sets of weighted partitions and discards dominated partitions, i.e., removes non-minimal weight copies:*

$$\texttt{union}(\mathcal{A}, \mathcal{B}) = \texttt{rmc}(\mathcal{A} \cup \mathcal{B}),$$

  where $\mathcal{B} \subseteq \Pi(U) \times \mathbb{N}$.

- $\texttt{insert}(X, \mathcal{A})$ *inserts elements $X \subseteq U$ into $\mathcal{A}$ as singletons:*

$$\texttt{insert}(X, \mathcal{A}) = \{(p_{\uparrow U \cup X}, w) : (p, w) \in \mathcal{A}\}.$$

- $\texttt{shift}(w', \mathcal{A})$ *increases the weight of each partition by w':*

$$\texttt{shift}(w', \mathcal{A}) = \{(p, w + w') : (p, w) \in \mathcal{A}\}.$$

- $\texttt{glue}(\{u, v\}, \mathcal{A})$ *combines, in each partition, the sets containing* u *and* v*, inserting them if needed:*

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

where $\hat{U} = U \cup \{u, v\}$*.*

- $\texttt{project}(X, \mathcal{A})$ *removes all elements of* $X \subseteq U$ *from each partition, discarding those where this would reduce the number of blocks:*

$$\texttt{project}(X, \mathcal{A}) = \texttt{rmc}(\{(p_{\downarrow \bar{X}}, w) : (p, w) \in \mathcal{A} \wedge \forall u \in X, \exists v \in \bar{X} : p \sqsubseteq U[\{u, v\}]\}),$$

where $\bar{X} = U \setminus X$*.*

- $\texttt{join}(\mathcal{A}, \mathcal{B})$ *extends all partitions to the same base set, joining blocks that have a common element, with weight equal to the sum of the weights:*

$$\texttt{join}(\mathcal{A}, \mathcal{B}) = \texttt{rmc}(\{(p_{\uparrow \hat{U}} \sqcup q_{\uparrow \hat{U}}, w + w') : (p, w) \in \mathcal{A} \wedge (q, w') \in \mathcal{B}\}),$$

where $\mathcal{B} \subseteq \Pi(U') \times \mathbb{N}$ *and* $\hat{U} = U \cup U'$*.*

Operators `glue` *and* `join` *with a subscript infer a* `shift` *operation with that value, i.e.,*

$$\texttt{glue}_w(\{u, v\}, \mathcal{A}) = \texttt{shift}(w, \texttt{glue}(\{u, v\}, \mathcal{A}))$$
$$\texttt{join}_w(\mathcal{A}, \mathcal{B}) = \texttt{shift}(w, \texttt{join}(\mathcal{A}, \mathcal{B})).$$

Proposition 1 (Bodlaender et al. [24])**.** *The operators* `union`*,* `insert`*,* `shift`*,* `glue` *and* `project` *execute in time* $\mathcal{O}(S \cdot |U|^{\mathcal{O}(1)})$*, where* S *is the size of the operation input, while* `join` *executes in time* $\mathcal{O}(|\mathcal{A}| \cdot |\mathcal{B}| \cdot |U|^{\mathcal{O}(1)})$*.*

The following theorem states that we can always find a reasonably small representative set of weighted partitions. Then, this set should allow us to extend to an optimal solution, given that one of the solutions present in the original set extends to it.

Theorem 15 (Bodlaender et al. [24])**.** *There exists an algorithm* `reduce` *that given a set of weighted partitions* $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$*, outputs in time* $|\mathcal{A}|2^{(\omega-1)|U|}|U|^{\mathcal{O}(1)}$ *a set of weighted partitions* $\mathcal{A}' \subseteq \mathcal{A}$ *such that* $\mathcal{A}'$ *represents* $\mathcal{A}$ *and* $|\mathcal{A}'| \leq 2^{|U|-1}$*, where* ω *denotes the matrix multiplication exponent.*

Computational experiment Briefly after the work of Bodlaender et al. [24], a computational experiment on the effectiveness of the rank-based approach was performed by Fafianie et al. [72], using Steiner Tree as benchmark. They compared the running times and the number of generated partial solutions of the naive dynamic programming algorithm, that is described via the operators of Definition 10, and the algorithm that additionally applies the `reduce` algorithm after each step of the recurrence. The instances were obtained from a diverse set of experimental studies of the problem, with graphs up to a thousand vertices and edges, whose tree decompositions have width up to 13.

They observed that the rank-based approach significantly speeds up the algorithm, indicating that the extra time used to reduce the size of the table pays off, as fewer partial solutions need to be analyzed in higher nodes of the decomposition. A modification that is relevant to experimental practitioners is to apply the `reduce` algorithm only when the size of the current set of weighted partitions is above the upper bound given in Theorem 15, which further improves the results.

Other notation For a predicate P, let $[P]$ be 1 if P is true, and 0 otherwise. For a vector $\mathbf{s}$, $\mathbf{s}[u \to \alpha]$ is a vector whose entries are the same as $\mathbf{s}$, except for u, that has value α. Denote $\mathbf{s}|_X$ the vector $\mathbf{s}$ restricted to the domain X.

5.2 Hardness results

In this section, we prove lower bounds for the studied problems in both classical and parameterized complexity. We employ well-known complexity assumptions that provide quantitative information about the running times to give, together with the algorithmic results of Sections 5.3 and 5.4, a tight understanding on the complexity of these problems, up to small factors in the exponent.

Our results for STS are based on a reduction from Dominating Set (DS), where the input is an undirected graph G and $k \in \mathbb{N}$, and the task is to find $S \subseteq V(G)$ of size k such that $N[S] = V(G)$. The following theorem for STS uses the total inapproximability of DS (Theorem 16), that rules out any parameterized approximation running in time $f(k) \cdot n^{o(k)}$ [110]. Note that the result holds for metric distance functions.

Theorem 16 (Karthik et al. [110])**.** *Assuming* ETH*, there is no $g(k)$-approximation algorithm for* DS *running in time $f(k) \cdot n^{o(k)}$, parameterized by the size of an optimal solution, for computable functions f and g.*

Theorem 17. *Assuming* ETH*, there is no $g(k)$-approximation algorithm for* STS *running in time $f(k) \cdot n^{o(k)}$, parameterized by the cost of an optimal solution, for computable functions f and g.*

Proof. Consider the following parameterized reduction from DS. Let $I = (G, k)$ be an instance of DS, from which we build an instance $I' = (G', c, d)$ of STS. Graph G' has vertices $V(G) \cup \{u_0\}$ and edges $E(G) \cup E_0$, where $E_0 = \{\{u_0, u\} : u \in V(G)\}$; edge costs are defined as $c(e) = d(e) = 1$, for $e \in E_0$, and $c(e) = 1$, $d(e) = 0$, for $e \in E(G)$ (see Figure 5.3). We will prove that I is a Yes-instance if, and only if, I' costs k.

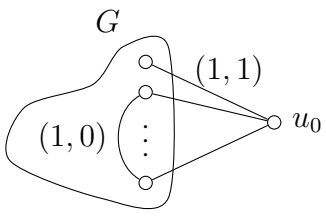


Figure 5.3: Reduction from DS to STS.

Let S be a dominating set of G with size k. Consider the function $\phi : V(G) \setminus S \to S$ that associates, for each vertex not in S, a neighbor in S. It follows that the tree T with edges $\{\{u_0, u\} : u \in S\} \cup \{\{u, \phi(u)\} : u \in V(G) \setminus S\}$ is a solution for I' with cost k, since T spans G and edges in the first and second sets cost, respectively, 1 and 0. Note that the first set can be composed of both internal and external edges, whereas the second is composed of only external edges. For the other direction, let T be a solution for I' with cost k. Let $p : V(G) \to V(G')$ be a function that indicates the parent of a vertex by performing a depth-first search in T, starting from u_0. We modify T by reassigning every internal vertex of T to u_0, thus removing edges $\{\{u, p(u)\} : u \text{ is an internal vertex of } T\}$ and adding edges $\{\{u_0, u\} : u \text{ is an internal vertex of } T\}$; this modification incurs no extra cost in the solution, as both types of edges have cost 1. Let $S = \{u : \{u_0, u\} \in E(T')\}$ be the set of vertices incident to u_0 in T'. Since T' costs k, we know that $|S| = k$, and moreover, as T' spans G, every vertex in $V(G) \setminus S$ has a neighbor in S. Then, S is a dominating set of G with size k. Therefore, the result follows from the total inapproximability of DS. □

Using the same reduction, we prove that STS is as hard to approximate as Set Cover in the classical setting [66], and that the best base of the exponent when parameterized by pathwidth is 4, from the lower bound of the connected version of the problem [59]. The parameterized reduction also works for pathwidth, as the universal vertex in the constructed graph increases the parameter by a constant.

Theorem 18. *It is* NP*-hard to obtain a* $(1 - \epsilon) \ln n$*-approximation for* STS*, for* $\epsilon > 0$*.*

Theorem 19. *Assuming* SETH*,* STS *cannot be solved in time* $\mathcal{O}^*((4 - \epsilon)^{\mathrm{pw}})$*, for* $\epsilon > 0$*.*

The success of the rank-based approach in providing single-exponential time algorithms for connectivity problems, parameterized by treewidth, prompted researchers to use the framework with further structural parameters, such as the algorithms for Steiner Tree and Connected Dominating Set, parameterized by clique-width and branch-width [148, 17]. We prove that we are unlikely to obtain an algorithm for STS parameterized by clique-width, as we have for treewidth, using the lower bound given by Fomin et al. [84], stating that Hamiltonian Path (HP) cannot be solved in time $f(\mathrm{cw}) \cdot n^{o(\mathrm{cw})}$, for a computable function f.

Theorem 20. *Assuming* ETH*, there is no* $(1.5 - \epsilon)$*-approximation for* STS *running in time* $f(\mathrm{cw}) \cdot n^{o(\mathrm{cw})}$*, for* $\epsilon > 0$*, where* n *is the number of vertices and* f *is a computable function.*

Proof. Consider the following parameterized reduction from HP. Let $I = (G)$ be an instance of HP, from which we build an instance $I' = (G, c, d)$ of STS, where $c(e) = 0$ and $d(e) = 1$, for $e \in E(G)$. We will prove that I is a Yes-instance if, and only if, an optimal solution for I' costs 2.

Let P be a hamiltonian path of G; it follows that $T = P$ is a solution for I' with cost 2, since there are only two external edges in T and it spans G. For the other direction, let T be a solution for I' with cost 2. Since a spanning tree with only two leaves is a hamiltonian path, $P = T$ is a solution for I. For $\epsilon > 0$, assume we have a $(1.5 - \epsilon)$-approximation

algorithm for STS, running in time $f(\mathrm{cw}) \cdot |I'|^{o(\mathrm{cw})}$, for a computable function f. Since a solution for I' has integer cost, this algorithm is able to decide HP in time $f(\mathrm{cw}) \cdot |I|^{o(\mathrm{cw})}$, therefore, the theorem follows. □

For the path variant, from a similar reduction from HP, we can rule out a $f(n)$-approximation algorithm, for a polynomial-time computable function f, and show that there is no algorithm parameterized by clique-width.

5.3 The tree variant

In this section, we give a parameterized algorithm for Spanning Tree-Star that runs in time $\mathcal{O}^*(4^{3.5\mathrm{tw}})$, based on the dynamic programming over a tree decomposition technique and the rank-based approach. As a byproduct of this result, we obtain an algorithm parameterized by the pathwidth of the input graph and another for particular classes of graphs. We assume we are given a nice tree decomposition $(\mathcal{T}, B)$ of the input graph. The algorithm identifies and labels vertices according to their roles in a partial solution, which are modified as new edges are included or two partial solution are joined.

Subproblem definition Let $\mathcal{U} = \{\mathfrak{i}, \mathfrak{e}, \mathfrak{m}, \mathfrak{z}\}$ be a set of labels and $\mathcal{U}' = \mathcal{U} \setminus \{\mathfrak{z}\}$. For a bag B_t of the tree decomposition, the vector $\mathbf{s} \in \mathcal{U}^{B_t}$ indicates the role of vertices in a partial solution: an internal vertex has label $\mathfrak{i}$; an external vertex that is connected to an internal vertex has label $\mathfrak{e}$; an external vertex that is connected to an external vertex has label $\mathfrak{m}$, and an isolated vertex has label $\mathfrak{z}$. Figure 5.4 shows the possible labels for a vertex and its respective neighbors, where internal and external vertices are represented as squares and circles, respectively. Note that an internal vertex can be connected to more than two vertices, either of internal or external types.

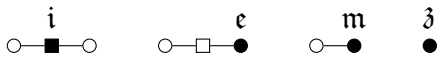


Figure 5.4: Possible labels of a vertex in STS.

Every connected pair of vertices in a partial solution must be given valid labels, that is, the neighbors of a vertex must have compatible labels to the presented definition. Let D be the set of valid pairs of labels:

$$D = \{(\mathfrak{i}, \mathfrak{i}), (\mathfrak{i}, \mathfrak{e}), (\mathfrak{e}, \mathfrak{i}), (\mathfrak{m}, \mathfrak{m})\}.$$

For $\mathbf{s} \in \mathcal{U}^{B_t}$ and $W \subseteq \mathcal{U}$, let $\mathbf{s}^{-1}(W) = \{u \in B_t : \mathbf{s}(u) \in W\}$. Define the dynamic

programming table and the set of partial solutions as:

$$
\begin{aligned}
A_t(\mathbf{s}) &= \{(p, \min_{X \in \mathcal{E}_t(p,\mathbf{s})} w(X)) : p \in \Pi(\mathbf{s}^{-1}(\mathcal{U}')) \wedge \mathcal{E}_t(p,\mathbf{s}) \neq \emptyset\} \\
\mathcal{E}_t(p,\mathbf{s}) &= \{X \subseteq E_t : \forall u \in B_t, u \in V(G[X]) \iff \mathbf{s}(u) \neq \mathfrak{z} \\
&\quad \wedge \forall u, v \in \mathbf{s}^{-1}(\mathcal{U}'), p \sqsubseteq U[uv] \iff u, v \text{ are in the same c.c. of } G[X] \\
&\quad \wedge \forall u, v \in \mathbf{s}^{-1}(\mathcal{U}'), \{u, v\} \in X \implies (\mathbf{s}(u), \mathbf{s}(v)) \in D \\
&\quad \wedge \#\texttt{blocks}(p) = \texttt{cc}(G[X]) \\
&\quad \wedge G[X] \text{ is acyclic}\}
\end{aligned}
$$

A partial solution $X \in \mathcal{E}_t(p, \mathbf{s})$ for STS is a forest $G[X]$ in which the intersection of $G[X]$ with B_t corresponds to the blocks of p, and vertices with label $\mathfrak{z}$ are not spanned by X. An example partial solution is depicted in Figure 5.5, where the vertices of the current bag are partitioned in $\{\{u_1, u_2, u_3\}, \{u_4\}, \{u_5, u_6, u_7\}\}$, with labels $\mathbf{s}^{-1}(\mathfrak{i}) = \{u_2, u_6\}$, $\mathbf{s}^{-1}(\mathfrak{e}) = \{u_1, u_3, u_5, u_7\}$, $\mathbf{s}^{-1}(\mathfrak{m}) = \{u_4\}$ and $\mathbf{s}^{-1}(\mathfrak{z}) = \{u_8\}$. Note that u_8 is not in the partition as it is an isolated vertex.

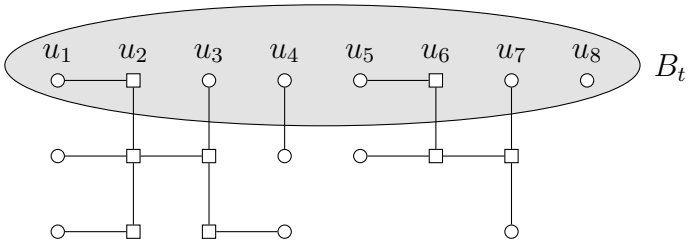


Figure 5.5: An example partial solution of STS.

Algorithm description In the following, we describe how to compute partial solutions for each node of the tree decomposition, according to its type. For a *leaf node* t, we have that $B_t = \emptyset$, then $A_t(\emptyset) = \{(\emptyset, 0)\}$.

Consider an *introduce vertex node* t with child t' such that $B_t = B_{t'} \cup \{u\}$, for $u \notin B_{t'}$. Since u is being introduced, there are no incident edges yet, then we only consider the case u is isolated.

$$
A_t(\mathbf{s}) = \begin{cases} A_{t'}(\mathbf{s}|_{B_{t'}}) & \text{if } \mathbf{s}(u) = \mathfrak{z}, \\ \emptyset & \text{otherwise.} \end{cases}
$$

Consider a *forget vertex node* t with child t' such that $B_t = B_{t'} \setminus \{u\}$, for $u \in B_{t'}$. In this case, we take every partial solution that includes u in a tree such that it has a neighbor in some block of the partition, since `project` removes partitions in which u is a singleton.

$$
A_t(\mathbf{s}) = \bigcup_{a \in \{\mathfrak{e},\mathfrak{i},\mathfrak{m}\}} \texttt{project}(u, A_{t'}(\mathbf{s}[u \to a]))
$$

Consider an *introduce edge node* t with child t' such that $B_t = B_{t'}$, annotated with edge

$e = \{u, v\}$. In order to include this edge in a partial solution, we analyze all label pairs that lead to their values in $\mathbf{s}$. Each valid label pair of connected vertices $(\mathbf{s}(u), \mathbf{s}(v)) \in D$ defines a set of pairs $C(\mathbf{s}(u), \mathbf{s}(v))$ that lead to their values when an edge between these types of vertices is included. For instance, if u and v are set as internal vertices in $\mathbf{s}$, we consider every partial solution whose labels are in $C(\mathfrak{i}, \mathfrak{i})$. Note that these are the only possible labels of u and v that result in them being internal vertices, as we include e in the partial solution.

$$
\begin{aligned}
C(\mathfrak{i}, \mathfrak{i}) &= \{(\mathfrak{e}, \mathfrak{e}), (\mathfrak{e}, \mathfrak{i}), (\mathfrak{e}, \mathfrak{m}), (\mathfrak{i}, \mathfrak{e}), (\mathfrak{i}, \mathfrak{i}), (\mathfrak{i}, \mathfrak{m}), (\mathfrak{m}, \mathfrak{e}), (\mathfrak{m}, \mathfrak{i}), (\mathfrak{m}, \mathfrak{m})\} \\
C(\mathfrak{i}, \mathfrak{e}) &= \{(\mathfrak{e}, \mathfrak{z}), (\mathfrak{i}, \mathfrak{z}), (\mathfrak{m}, \mathfrak{z})\} \\
C(\mathfrak{e}, \mathfrak{i}) &= \{(\mathfrak{z}, \mathfrak{e}), (\mathfrak{z}, \mathfrak{i}), (\mathfrak{z}, \mathfrak{m})\} \\
C(\mathfrak{m}, \mathfrak{m}) &= \{(\mathfrak{z}, \mathfrak{z})\}
\end{aligned}
$$

Figure 5.6 gives a graphical representation of these sets, where we display every label combination of u and v along with the resulting labels when e is included in the solution; vertices u and v are colored as black and gray, respectively.

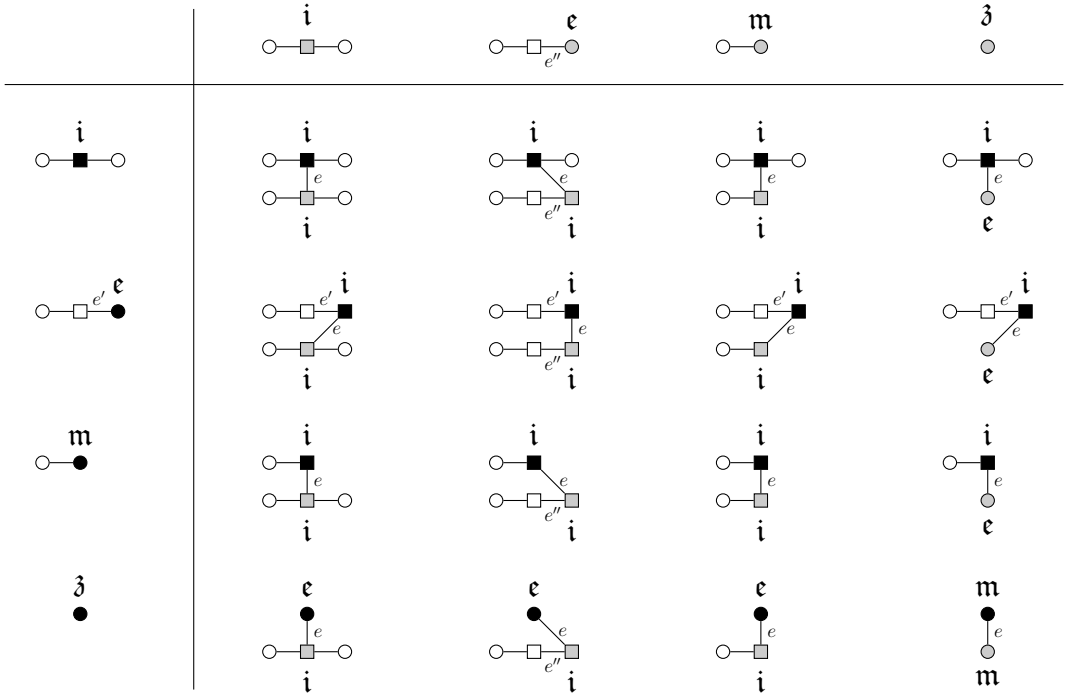


Figure 5.6: Label combinations in STS.

When an edge $\{u, v\}$ is included in a partial solution, we calculate its effect in the cost with $\Delta(\ell_1, \ell_2) = \Delta_0(\ell_1, \ell_2) + \Delta_1(\ell_1, \ell_2)$, for $(\ell_1, \ell_2) \in C(\mathbf{s}(u), \mathbf{s}(v))$, according to Table 5.1. The first term represents the cost of the introduced edge, considering it is either an external or an internal edge, whilst the second term accounts for changing the value of an incident edge from external to internal. This happens when at least one of u or v have label $\mathfrak{e}$, and we define e' and e'' as the incident edges of u and v, respectively, before e is included.

$$A_t(\mathbf{s}) = A_{t'}(\mathbf{s}) \cup \bigcup_{(\ell_1,\ell_2)\in C(\mathbf{s}(u),\mathbf{s}(v))} \texttt{glue}_{\Delta(\ell_1,\ell_2)}(\{u,v\}, A_{t'}(\mathbf{s}[u\to\ell_1, v\to\ell_2]))$$

ℓ_1	ℓ_2	$\ell_1 \otimes \ell_2$	$\Delta_0(\ell_1,\ell_2)$	$\Delta_1(\ell_1,\ell_2)$
$\mathfrak{i}$	$\mathfrak{i}$	$\mathfrak{i}$	$c(e)$	0
$\mathfrak{i}$	$\mathfrak{e}$	$\mathfrak{i}$	$c(e)$	$c(e'') - d(e'')$
$\mathfrak{i}$	$\mathfrak{m}$	$\mathfrak{i}$	$c(e)$	0
$\mathfrak{i}$	$\mathfrak{z}$	$\mathfrak{i}$	$d(e)$	0
$\mathfrak{e}$	$\mathfrak{i}$	$\mathfrak{i}$	$c(e)$	$c(e') - d(e')$
$\mathfrak{e}$	$\mathfrak{e}$	$\mathfrak{i}$	$c(e)$	$c(e') + c(e'') - d(e') - d(e'')$
$\mathfrak{e}$	$\mathfrak{m}$	$\mathfrak{i}$	$c(e)$	$c(e') - d(e')$
$\mathfrak{e}$	$\mathfrak{z}$	$\mathfrak{e}$	$d(e)$	$c(e') - d(e')$
$\mathfrak{m}$	$\mathfrak{i}$	$\mathfrak{i}$	$c(e)$	0
$\mathfrak{m}$	$\mathfrak{e}$	$\mathfrak{i}$	$c(e)$	$c(e'') - d(e'')$
$\mathfrak{m}$	$\mathfrak{m}$	$\mathfrak{i}$	$c(e)$	0
$\mathfrak{m}$	$\mathfrak{z}$	$\mathfrak{m}$	$d(e)$	0
$\mathfrak{z}$	$\mathfrak{i}$	$\mathfrak{i}$	$d(e)$	0
$\mathfrak{z}$	$\mathfrak{e}$	$\mathfrak{e}$	$d(e)$	$c(e'') - d(e'')$
$\mathfrak{z}$	$\mathfrak{m}$	$\mathfrak{m}$	$d(e)$	0
$\mathfrak{z}$	$\mathfrak{z}$	$\mathfrak{z}$	$d(e)$	0

Table 5.1: Operator $\otimes$ and costs in STS.

Consider a *join node* t with children t' and t'' such that $B_t = B_{t'} = B_{t''}$. In this case, we combine two vectors $\mathbf{l}$ and $\mathbf{r}$ of partial solutions into $\mathbf{s}$ according to the operator $\otimes : \mathcal{U}^2 \to \mathcal{U}$, that is applied for each vertex $u \in B_t$, such that $\mathbf{s}(u) = \mathbf{l}(u) \otimes \mathbf{r}(u)$. As we are joining two partial solutions, a vertex u can have distinct labels in t' and t'', e.g., if $\mathbf{s}(u) = \mathfrak{i}$, we need to consider various combinations of $\mathbf{l}(u)$ and $\mathbf{r}(u)$ that lead to such a label. We also need to account for possible changes in the edge costs, since joining a vertex that has label $\mathfrak{e}$ in one of the children nodes entails the adjustment of the weight of the resulting partitions, just as in introduce edge nodes. This is done by the `shift` operation, embedded in the `join` below, summing all $\Delta_1(\mathbf{l}(u), \mathbf{r}(u))$, for $u \in B_t$. For the ease of notation, define $\Delta_1(\mathbf{l}, \mathbf{r}) = \sum_{u\in B_t} \Delta_1(\mathbf{l}(u), \mathbf{r}(u))$. Note that Figure 5.6 is also helpful in this case, considering u and v are the same vertex, but coming from each of the children.

$$A_t(\mathbf{s}) = \bigcup_{\mathbf{l},\mathbf{r}\in\mathcal{U}^{B_t}:\mathbf{l}\otimes\mathbf{r}=\mathbf{s}} \texttt{join}_{\Delta_1(\mathbf{l},\mathbf{r})}(A_{t'}(\mathbf{l}), A_{t''}(\mathbf{r}))$$

Lemma 19. *The algorithm correctly computes an optimal solution for* STS.

Proof. The correctness follows from induction on a node t of the tree decomposition $\mathcal{T}$. Consider the induction hypothesis: given nodes t and t' of the tree decomposition, such that t' is a descendant of t with $\mathbf{s} \in \mathcal{U}^{B_{t'}}$, the recurrence correctly calculates a set $A_{t'}(\mathbf{s})$

of weighted partitions for t'. For a given partition $p \in \Pi(\mathbf{s}^{-1}(\mathcal{U}'))$, a partial solution $X \in \mathcal{E}_{t'}(p, \mathbf{s})$ consists of a forest, in which every connected component of $G[X]$ has at least one vertex in $B_{t'}$ and vertices are correctly labeled.

A *leaf node* t with $B_t = \emptyset$ is the base case, for which the induction hypothesis holds. Now we consider other node types, and assume the induction hypothesis holds for their descendants.

An *introduce vertex node* t has child t' with $B_t = B_{t'} \cup \{u\}$, for $u \notin B_{t'}$. In this case, the algorithm takes the set of weighted partitions of t' when u is set as an isolated vertex, and, otherwise, takes the empty set, as there is no valid partial solution. This is caused by the increase of the number of blocks of a partition and not the number of connected components in the graph induced by the solution. Assume X is a partial solution for t that is cheaper than every solution considered by the algorithm. Since u has no edges in G_t, it is an isolated vertex, then X is also a partial solution for t'. This contradicts the optimality of the induction hypothesis, as the algorithm uses the same set of partial solutions of t' for t, when u is set to $\mathfrak{z}$.

A *forget node* t has child t' with $B_t = B_{t'} \setminus \{u\}$, for $u \in B_{t'}$. The recurrence extends a given $\mathbf{s}$ to the cases u is set as $\mathfrak{e}$, $\mathfrak{i}$ or $\mathfrak{m}$ in the child node, and the set of weighted partitions for t is then the union of those sets of t'. The `project` operation considers only partial solutions whose partitions maintain the number of blocks when u is removed, which ensures u has a neighbor in some block of the partition. Assume X is a cheaper partial solution for t that is not considered by the algorithm. Since X is valid, u is not an isolated vertex. It follows that the recurrence correctly computes such a solution by regarding every possible label of u in the child node, which is correctly calculated (IH).

An *introduce edge node* t has child t' with $B_t = B_{t'}$, annotated with an edge $\{u, v\}$. In case this edge is not included in the solution, the algorithm simply takes the set of weighted partitions of the child t', otherwise, we must consider every possible label of u and v. Now, if we use $\{u, v\}$ in the solution and, for example, set u and v as $\mathfrak{i}$ and $\mathfrak{e}$ in $\mathbf{s}$, respectively, we need to consider every label pair $(\ell_1, \ell_2) \in C(\mathfrak{i}, \mathfrak{e})$ that leads to their label in $\mathbf{s}$. Then, the `glue` operation combines the blocks containing u and v into one, representing the newly created connected component; moreover, the cost $\Delta(\ell_1, \ell_2)$ of the operation is added to the weight of every partition. Analogous steps are taken when u and v have labels $(\mathfrak{i}, \mathfrak{i})$, $(\mathfrak{e}, \mathfrak{i})$ or $(\mathfrak{m}, \mathfrak{m})$ in the vector $\mathbf{s}$. Aiming for a contradiction, let X be a cheaper partial solution that is not considered by the algorithm, using edge $\{u, v\}$ in such a way that u is an internal and v is an external vertex. Let ℓ_1 and ℓ_2 be the labels of u and v, respectively, before including the edge in X. Using the induction hypothesis, we correctly calculate this solution for the child node t'. Since the algorithm considers every label combination that leads to u and v being these types of vertices, listed in $C(\mathfrak{i}, \mathfrak{e})$, and take into account the correct costs $\Delta(\ell_1, \ell_2)$ of the operation, X cannot be cheaper than the solutions obtained by the algorithm. The analysis is analogous when u and v have labels $(\mathfrak{i}, \mathfrak{i})$, $(\mathfrak{e}, \mathfrak{i})$ or $(\mathfrak{m}, \mathfrak{m})$ in the vector $\mathbf{s}$.

A *join node* t has children t' and t'' with $B_t = B_{t'} = B_{t''}$. In order to calculate the set of weighted partitions for t, we combine every vector $\mathbf{l}$ and $\mathbf{r}$, of t' and t'', respectively, into the given vector $\mathbf{s}$, using the operator $\otimes$. The `join` operator is applied to the partitions of the children, effectively placing every vertex that is in the same connected component

of the combined partial solution in the resulting block of a partition. The weight of this partition, that is, the cost of the combined partial solution, is the sum of the weights of the children considering the possible changes in the costs of the existing edges, when at least one of the children have label $\mathfrak{e}$, expressed by $\Delta_1(\mathbf{l}, \mathbf{r})$. Aiming for a contradiction, assume X is a cheaper partial solution for t that is not calculated by the algorithm. According to our definition of a nice tree decomposition, each edge is introduced once, above the last join node that has both endpoints, and each vertex is forgotten once, then the edges between vertices in B_t are not introduced yet, and $V(G_{t'}) \cap V(G_{t''}) = B_t$; this implies that X can be partitioned into valid partial solutions X' and X'', for t' and t'', respectively. The partial solution X' for t' induces a vector $\mathbf{l} \in \mathcal{U}^{B_{t'}}$ that is considered by the algorithm (IH), and the analogous holds for X'' and t'', as we try every vector combination. Since we correctly account for possible changes in the edge costs, it follows that such solution X is considered by the algorithm for node t. □

Recall that the best base of the exponent one can achieve is 4, as seen in Theorem 19. Now we are ready for the main result of this section.

Theorem 21. *There is an algorithm for* STS *that runs in time* $\mathcal{O}^*(4^{3.5\mathrm{tw}})$.

Proof. The algorithm proceeds by computing the intermediate sets $\tilde{A}_t$, for each node $t \in V(\mathcal{T})$, according to the described recurrence relations. Then, $A_t = \texttt{reduce}(\tilde{A}_t)$ is stored, the reduced-size representative set of weighted partitions of such node, to be used in bigger subproblems.

The `reduce` operator runs in time $|\tilde{A}_t| 2^{(\omega-1)|U|} |U|^{\mathcal{O}(1)}$ and produces a set A_t of size at most $2^{|U|-1}$, where $U = \mathbf{s}^{-1}(\mathcal{U}')$ and $\omega < 2.3727$ [163]. In the following, we calculate the size of the intermediate set of a node t, considering it is a join node, since these are larger than other node types.

$$\begin{aligned}
|\tilde{A}_t(\mathbf{s})| &\leq \sum_{\mathbf{l} \otimes \mathbf{r} = \mathbf{s}} 2^{|\mathbf{l}^{-1}(\mathcal{U}')|} 2^{|\mathbf{r}^{-1}(\mathcal{U}')|} \\
&= \prod_{u \in B_t} \sum_{\mathbf{l}(u) \otimes \mathbf{r}(u) = \mathbf{s}(u)} 2^{[\mathbf{l}(u) \in \mathcal{U}'] + [\mathbf{r}(u) \in \mathcal{U}']} \\
&= 40^{|\mathbf{s}^{-1}(\mathfrak{i})|} 4^{|\mathbf{s}^{-1}(\mathfrak{e})|} 4^{|\mathbf{s}^{-1}(\mathfrak{m})|},
\end{aligned}$$

where the first equality expands into products for each vector coordinate, and use the following observation:

$$\sum_{\mathbf{l}(u) \otimes \mathbf{r}(u) = \mathbf{s}(u)} 2^{[\mathbf{l}(u) \in \mathcal{U}'] + [\mathbf{r}(u) \in \mathcal{U}']} = \begin{cases} 40 & \text{if } \mathbf{s}(u) = \mathfrak{i}, \\ 4 & \text{if } \mathbf{s}(u) = \mathfrak{e}, \\ 4 & \text{if } \mathbf{s}(u) = \mathfrak{m}, \\ 1 & \text{if } \mathbf{s}(u) = \mathfrak{z}. \end{cases}$$

Take $\mathbf{s}(u) = \mathfrak{e}$ as an example: the label pairs that lead to $\mathfrak{e}$ when applied the operator are $(\mathfrak{e}, \mathfrak{z})$ and $(\mathfrak{z}, \mathfrak{e})$, since $\mathfrak{e} \otimes \mathfrak{z} = \mathfrak{z} \otimes \mathfrak{e} = \mathfrak{e}$; then, the previous sum for this label would be:

$$2^{[\mathfrak{e} \in \mathcal{U}'] + [\mathfrak{z} \in \mathcal{U}']} + 2^{[\mathfrak{z} \in \mathcal{U}'] + [\mathfrak{e} \in \mathcal{U}']} = 2^{1+0} + 2^{0+1} = 4.$$

The running time to compute $A_t(\mathbf{s})$, for every $\mathbf{s} \in \mathcal{U}^{B_t}$, is

$$\begin{aligned}\sum_{\mathbf{s}\in\mathcal{U}^{B_t}} |\tilde{A}_t(\mathbf{s})| 2^{(\omega-1)|\mathbf{s}^{-1}(\mathcal{U}')|}\mathrm{tw}^{\mathcal{O}(1)} &= \sum_{i_{\mathfrak{i}}+i_{\mathfrak{e}}+i_{\mathfrak{m}}+i_{\mathfrak{z}}=|B_t|} \binom{|B_t|}{i_{\mathfrak{i}}, i_{\mathfrak{e}}, i_{\mathfrak{m}}, i_{\mathfrak{z}}} 40^{i_{\mathfrak{i}}} 4^{i_{\mathfrak{e}}} 4^{i_{\mathfrak{m}}} 1^{i_{\mathfrak{z}}} 2^{(\omega-1)\mathrm{tw}}\mathrm{tw}^{\mathcal{O}(1)} \\ &\leq (49 \cdot 2^{(\omega-1)})^{\mathrm{tw}}\mathrm{tw}^{\mathcal{O}(1)} \\ &\leq 4^{3.5\mathrm{tw}}\mathrm{tw}^{\mathcal{O}(1)},\end{aligned}$$

where we use $|\mathbf{s}^{-1}(\mathcal{U}')| \leq \mathrm{tw}$ and the multinomial theorem. Since there are $\mathcal{O}(n \cdot \mathrm{tw})$ nodes in $\mathcal{T}$, the algorithm runs in time $\mathcal{O}^*(4^{3.5\mathrm{tw}})$. □

Further algorithmic results The presented algorithm can be easily extended to use pathwidth as parameter and to consider particular classes of graphs as input. In the first case, we assume a path decomposition of width pw is given as input (recall this type of decomposition has no join nodes). Then, the size of the intermediate sets of weighted partitions is at most $2^{|\mathbf{s}^{-1}(\mathcal{U}')|} \leq 2^{\mathrm{pw}}$. Since the operator $\otimes$ is not needed, the running time is improved compared to the algorithm parameterized by treewidth. Note that this result is close to the lower bound given in Theorem 19.

Theorem 22. *There is an algorithm for* STS *that runs in time* $\mathcal{O}^*(4^{1.7\mathrm{pw}})$.

Following directly from our positive results parameterized by pathwidth and treewidth, and Theorems 23 and 24, we obtain algorithms for graphs of bounded degree and planar graphs.

Theorem 23 ([82])**.** *For any graph* G *with* n *vertices and maximum degree 3,* $\mathrm{pw}(G) \leq \frac{n}{6}$.

Theorem 24 ([86])**.** *For any planar graph* G *with* n *vertices,* $\mathrm{tw}(G) + 1 \leq 3.183\sqrt{n}$.

Theorem 25. *For graphs of maximum degree 3, there is an algorithm for* STS *that runs in time* $\mathcal{O}(1.49^n)$.

Theorem 26. *For planar graphs, there is an algorithm for* STS *that runs in time* $\mathcal{O}(2^{\mathcal{O}(\sqrt{n})})$.

5.4 The path and cycle variants

In this section, we present a parameterized algorithm for Spanning Path-Star, using treewidth as parameter, that runs in time $\mathcal{O}^*(2^{8.4\mathrm{tw}})$. We also give an algorithm parameterized by pathwidth, that runs in time $\mathcal{O}^*(2^{4.38\mathrm{pw}})$, and two others specialized in particular classes of graphs. These results follow the same structure as the one for STS, thus, we highlight the points that need to be adapted.

For the tree variant, four types of vertices were codified as labels to represent a partial solution, namely, internal vertices, external vertices that are connected to either an internal or an external vertex, and isolated vertices. In this case, more information is needed to correctly describe the role of a vertex in a bag of the tree decomposition, as vertices interact in a more intricate way. The algorithm for the cycle variant of the problem is also devised in this section.

In SPS, given a graph G and two edge-cost functions, the objective is to find a minimum-cost spanning subgraph H of G in which its internal vertices induce a path, where the internal and external edges are priced differently. We have that every vertex not in the path is connected to exactly one vertex in the path, and hence the problem is also known as the Minimum Spanning Caterpillar. The path formed by the internal vertices is also called the spine of the solution.

An instance of the problem is composed of a graph G, edge-cost functions $c, d : E(G) \to \mathbb{N}$ and a nice tree decomposition $\mathcal{T}$ of G. It is useful to assume the decomposition is rooted at an introduce edge node, which can be done efficiently (Theorem 1). We guess the endpoints of the spine of an optimal solution are the vertices u_0 and u_n, creating the edge $\{u_0, u_n\}$ in G. In the algorithm for the cycle variant of the problem, we look for an edge that is part of the cycle of an optimal solution, guessing only existing edges of G. This guessing step adds an extra quadratic term to the running time of the algorithm.

Subproblem definition Consider labels $\mathcal{U} = \{\mathfrak{i}_0, \mathfrak{i}_1, \mathfrak{i}_2, \mathfrak{e}_0, \mathfrak{e}_1, \mathfrak{e}_2, \mathfrak{m}, \mathfrak{z}\}$, and let $\mathcal{U}' = \mathcal{U} \setminus \{\mathfrak{z}\}$. For a node $t \in V(\mathcal{T})$, we use a vector $\mathbf{s} \in \mathcal{U}^{B_t}$ to describe the role of vertices in the bag of t, then, for $u \in B_t$, its role in the partial solution is represented by $\mathbf{s}(u)$: if u is an internal vertex connected to none, one or two internal vertices, it has label $\mathfrak{i}_0$, $\mathfrak{i}_1$ or $\mathfrak{i}_2$, respectively; if u is an external vertex connected to a vertex of type $\mathfrak{i}_0$, $\mathfrak{i}_1$ or $\mathfrak{i}_2$, it has label $\mathfrak{e}_0$, $\mathfrak{e}_1$ or $\mathfrak{e}_2$, respectively; if u is an external vertex connected to an external vertex, it has label $\mathfrak{m}$, and if u is an isolated vertex, it has label $\mathfrak{z}$. Figure 5.7 shows the types of labels of a vertex in a partial solution, where internal and external vertices are represented as squares and circles, respectively.

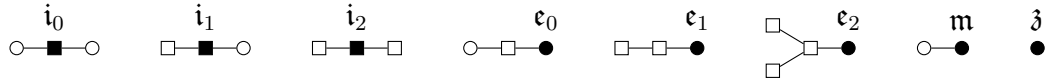


Figure 5.7: Possible labels of a vertex in SPS.

Any connected pair of vertices of a partial solution must be given valid labels, that is, the neighbors of a vertex must have compatible labels to the presented definition. Let D be the set that contains the valid pairs of labels.

$$D = \{(\mathfrak{i}_0, \mathfrak{e}_0), (\mathfrak{i}_1, \mathfrak{i}_1), (\mathfrak{i}_1, \mathfrak{i}_2), (\mathfrak{i}_1, \mathfrak{e}_1), (\mathfrak{i}_2, \mathfrak{i}_1), (\mathfrak{i}_2, \mathfrak{i}_2), (\mathfrak{i}_2, \mathfrak{e}_2), (\mathfrak{e}_0, \mathfrak{i}_0), (\mathfrak{e}_1, \mathfrak{i}_1), (\mathfrak{e}_2, \mathfrak{i}_2), (\mathfrak{m}, \mathfrak{m})\}.$$

For $t \in V(\mathcal{T})$ and a vector $\mathbf{s} \in \mathcal{U}^{B_t}$, define the dynamic programming table and the set of partial solutions as:

$$A_t(\mathbf{s}) = \{(p, \min_{X \in \mathcal{E}_t(p,\mathbf{s})} w(X)) : p \in \Pi(\mathbf{s}^{-1}(\mathcal{U}')) \wedge \mathcal{E}_t(p, \mathbf{s}) \neq \emptyset\}$$

$$\begin{aligned}\mathcal{E}_t(p, \mathbf{s}) = \{&X \subseteq E_t : \forall u \in B_t,\ u \in V(G[X]) \iff \mathbf{s}(u) \neq \mathfrak{z} \\ &\wedge \forall u, v \in \mathbf{s}^{-1}(\mathcal{U}'),\ p \sqsubseteq U[uv] \iff u \text{ and } v \text{ are in the same c.c. of } G[X] \\ &\wedge \forall u, v \in \mathbf{s}^{-1}(\mathcal{U}'), \{u, v\} \in X \implies (\mathbf{s}(u), \mathbf{s}(v)) \in D \\ &\wedge \#\texttt{blocks}(p) = \texttt{cc}(G[X]) \\ &\wedge G[X] \text{ is acyclic}\}\end{aligned}$$

A partial solution $X \in \mathcal{E}_t(p, \mathbf{s})$ for SPS is a forest $G[X]$ in which the intersection of the connected components of $G[X]$ with B_t correspond to the blocks of p, any two connected vertices must be given one of the valid label pairs of D, and vertices with label $\mathfrak{z}$ are not spanned by X. An optimal solution is then found at the root node, whose partition is $\{\{u_0, u_n\}\}$, with $\mathbf{s}(u_0) = \mathbf{s}(u_n) = \mathfrak{i}_1$, since we want to find a single connected component with these vertices as the endpoints of the spine. In the cycle variant of the problem, we set the label of these vertices to $\mathfrak{i}_2$ and account for the weight of this edge in the total cost of the solution.

Algorithm description In the following, we describe the algorithm for SPS, highlighting the differences between the one presented in the previous section. A *leaf node* contains only a partial solution formed by an empty partition and weight zero, as expected. In an *introduce vertex node*, as in STS, valid solutions are obtained from the child's solution when the introduced vertex is set as isolated, since it has no introduced edges yet; then, all other labels lead to an invalid partial solution. In a *forget node*, one can only forget external vertices and internal vertices that already have two other neighbors set as internal vertices, i.e., vertices with labels $\mathfrak{e}_0$, $\mathfrak{e}_1$, $\mathfrak{e}_2$ and $\mathfrak{i}_2$. This ensures that the last two internal vertices are the endpoints of the solution's spine, as intended.

In an *introduce edge node* annotated with edge $e = \{u, v\}$, we consider every label pair of u and v in the child's solution that leads to their values in $\mathbf{s}$ when the edge is included in the partial solution. As before, each pair in D defines a set of pairs $C(\mathbf{s}(u), \mathbf{s}(v))$ to be analyzed:

$$
\begin{aligned}
C(\mathfrak{i}_0, \mathfrak{e}_0) &= \{(\mathfrak{i}_0, \mathfrak{z}), (\mathfrak{m}, \mathfrak{z})\} \\
C(\mathfrak{i}_1, \mathfrak{i}_1) &= \{(\mathfrak{i}_0, \mathfrak{i}_0), (\mathfrak{i}_0, \mathfrak{m}), (\mathfrak{m}, \mathfrak{i}_0), (\mathfrak{m}, \mathfrak{m})\} \\
C(\mathfrak{i}_1, \mathfrak{i}_2) &= \{(\mathfrak{i}_0, \mathfrak{i}_1), (\mathfrak{i}_0, \mathfrak{e}_0), (\mathfrak{i}_0, \mathfrak{e}_1), (\mathfrak{m}, \mathfrak{i}_1), (\mathfrak{m}, \mathfrak{e}_0), (\mathfrak{m}, \mathfrak{e}_1)\} \\
C(\mathfrak{i}_1, \mathfrak{e}_1) &= \{(\mathfrak{i}_1, \mathfrak{z}), (\mathfrak{e}_0, \mathfrak{z}), (\mathfrak{e}_1, \mathfrak{z})\} \\
C(\mathfrak{i}_2, \mathfrak{i}_1) &= \{(\mathfrak{i}_1, \mathfrak{i}_0), (\mathfrak{e}_0, \mathfrak{i}_0), (\mathfrak{e}_1, \mathfrak{i}_0), (\mathfrak{i}_1, \mathfrak{m}), (\mathfrak{e}_0, \mathfrak{m}), (\mathfrak{e}_1, \mathfrak{m})\} \\
C(\mathfrak{i}_2, \mathfrak{i}_2) &= \{(\mathfrak{i}_1, \mathfrak{i}_1), (\mathfrak{i}_1, \mathfrak{e}_0), (\mathfrak{e}_0, \mathfrak{i}_1), (\mathfrak{i}_1, \mathfrak{e}_1), (\mathfrak{e}_1, \mathfrak{i}_1), (\mathfrak{e}_0, \mathfrak{e}_0), (\mathfrak{e}_0, \mathfrak{e}_1), (\mathfrak{e}_1, \mathfrak{e}_0), (\mathfrak{e}_1, \mathfrak{e}_1)\} \\
C(\mathfrak{i}_2, \mathfrak{e}_2) &= \{(\mathfrak{i}_2, \mathfrak{z})\} \\
C(\mathfrak{e}_0, \mathfrak{i}_0) &= \{(\mathfrak{z}, \mathfrak{i}_0), (\mathfrak{z}, \mathfrak{m})\} \\
C(\mathfrak{e}_1, \mathfrak{i}_1) &= \{(\mathfrak{z}, \mathfrak{i}_1), (\mathfrak{z}, \mathfrak{e}_0), (\mathfrak{z}, \mathfrak{e}_1)\} \\
C(\mathfrak{e}_2, \mathfrak{i}_2) &= \{(\mathfrak{z}, \mathfrak{i}_2)\} \\
C(\mathfrak{m}, \mathfrak{m}) &= \{(\mathfrak{z}, \mathfrak{z})\}
\end{aligned}
$$

Figure 5.8 shows a graphical representation of the label combinations of u and v and the resulting labels after edge e is included, where u and v are represented, respectively, as black and gray vertices. Note that $\mathfrak{i}_2$ only produces a valid combination with $\mathfrak{z}$, while $\mathfrak{e}_2$ has no valid combinations. In this formulation, we observe that changing the label of a vertex causes the labels of the neighboring vertices to change as well. Say u and v have labels $\mathfrak{e}_0$ and $\mathfrak{i}_0$, respectively, before e is included in the partial solution, then, including the edge changes their labels to $\mathfrak{i}_2$ and $\mathfrak{i}_1$, respectively. Moreover, the vertex connected to

u changes to $\mathfrak{i}_1$ and its neighbors to $\mathfrak{e}_1$, and the vertices connected to v change to $\mathfrak{e}_1$. This straightforward adaptation of labels in $\mathbf{s}$ need to occur each time a new edge is included in the partial solution, so every label pair of connected vertices remain valid.

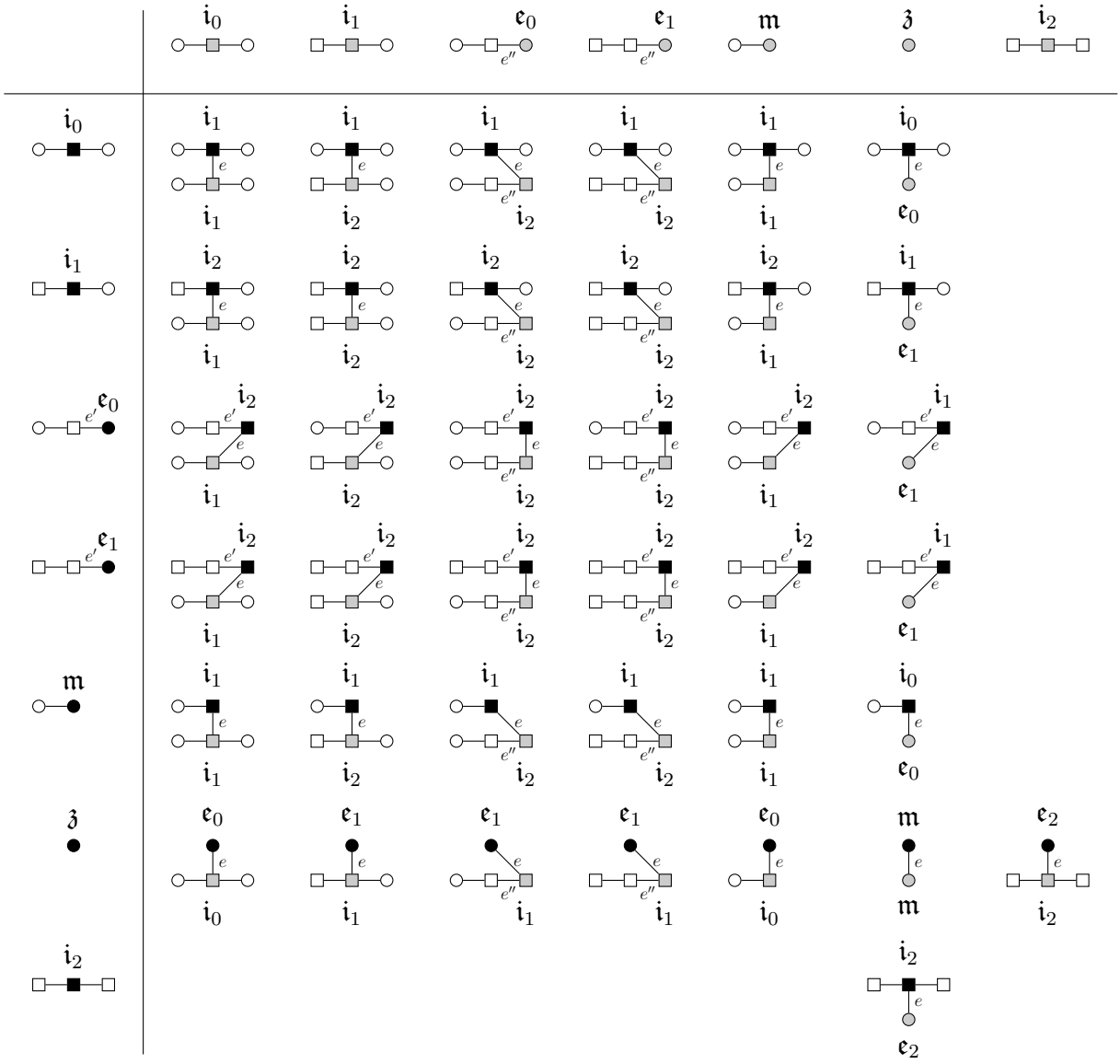


Figure 5.8: Label combinations in SPS.

Whenever an edge e is included, we need to calculate the change in the total cost of a partial solution, that has value $\Delta(\ell_1, \ell_2) = \Delta_0(\ell_1, \ell_2) + \Delta_1(\ell_1, \ell_2)$, where $(\ell_1, \ell_2) \in C(\mathbf{s}(u), \mathbf{s}(v))$ are the possible labels of u and v, respectively, before e is included in the solution. The first term is the cost of the included edge and the second term accounts for the change in cost of an incident edge from external to internal. Table 5.2 summarizes these costs for every label pair contained in the sets described by D, where e' and e'' are the edges of u and v, respectively, when they are set as external vertices.

A *join node* needs to combine partial solutions to two different subgraphs of G, then, with $\mathbf{l}$ and $\mathbf{r}$ as the vectors describing the role of vertices in the bags of the children, we need to consider the vector $\mathbf{s}$ for the current node, such that $\mathbf{s}(u) = \mathbf{l}(u) \otimes \mathbf{r}(u)$, according

to the operator $\otimes$ defined in Table 5.2. Joining these solutions effectively changes the labels of various vertices, e.g., if a vertex u is labeled as $\mathfrak{e}_0$ and $\mathfrak{i}_0$ in $\mathbf{l}$ and $\mathbf{r}$, respectively, the resulting label of u will be $\mathfrak{e}_0 \otimes \mathfrak{i}_0 = \mathfrak{i}_1$; just as in introduce edge nodes, neighboring vertices also need to adapt their labels. Note that, while $\mathfrak{e}_2$ vertices do not admit edge inclusion, they can be joined with $\mathfrak{z}$ vertices. The cost of a partial solution also needs adjustment when an incident edge changes from being external to internal, as in STS, caused by joining the children solutions. This cost is given by $\Delta_1(\ell_1, \ell_2)$ in Table 5.2, where ℓ_1 and ℓ_2 are the labels of the vertex in the bags of the children.

Theorem 27. *There is an algorithm for* SPS *that runs in time* $\mathcal{O}^*(2^{8.4\mathrm{tw}})$.

Proof. The correctness arguments are similar as the ones presented in Theorem 21, so we concentrate on the running time of the algorithm. This running time is also dominated by the size of the intermediate sets of join nodes:

$$\begin{aligned}
|\tilde{A}_t(\mathbf{s})| &\leq \sum_{\mathbf{l}\otimes\mathbf{r}=\mathbf{s}} 2^{|\mathbf{l}^{-1}(\mathcal{U}')|} 2^{|\mathbf{r}^{-1}(\mathcal{U}')|} \\
&= \prod_{u\in B_t} \sum_{\mathbf{l}(u)\otimes\mathbf{r}(u)=\mathbf{s}(u)} 2^{[\mathbf{l}(u)\in\mathcal{U}']+[\mathbf{r}(u)\in\mathcal{U}']} \\
&= 20^{|\mathbf{s}^{-1}(\mathfrak{i}_0)|} 52^{|\mathbf{s}^{-1}(\mathfrak{i}_1)|} 40^{|\mathbf{s}^{-1}(\mathfrak{i}_2)|} 4^{|\mathbf{s}^{-1}(\mathfrak{e}_0)|} 4^{|\mathbf{s}^{-1}(\mathfrak{e}_1)|} 4^{|\mathbf{s}^{-1}(\mathfrak{e}_2)|} 4^{|\mathbf{s}^{-1}(\mathfrak{m})|},
\end{aligned}$$

where we use the following observation:

$$\sum_{\mathbf{l}(u)\otimes\mathbf{r}(u)=\mathbf{s}(u)} 2^{[\mathbf{l}(u)\in\mathcal{U}']+[\mathbf{r}(u)\in\mathcal{U}']} = \begin{cases} 20 & \text{if } \mathbf{s}(u) = \mathfrak{i}_0, \\ 52 & \text{if } \mathbf{s}(u) = \mathfrak{i}_1, \\ 40 & \text{if } \mathbf{s}(u) = \mathfrak{i}_2, \\ 4 & \text{if } \mathbf{s}(u) = \mathfrak{e}_0, \\ 4 & \text{if } \mathbf{s}(u) = \mathfrak{e}_1, \\ 4 & \text{if } \mathbf{s}(u) = \mathfrak{e}_2, \\ 4 & \text{if } \mathbf{s}(u) = \mathfrak{m}, \\ 1 & \text{if } \mathbf{s}(u) = \mathfrak{z}. \end{cases}$$

Then, the total running time is

$$\begin{aligned}
&\sum_{\mathbf{s}\in\mathcal{U}^{B_t}} |\tilde{A}_t(\mathbf{s})| 2^{(\omega-1)|\mathbf{s}^{-1}(\mathcal{U}')|} \mathrm{tw}^{\mathcal{O}(1)} \\
&\quad= \sum_{i_{\mathfrak{i}_0}+\cdots+i_{\mathfrak{z}}=|B_t|} \binom{|B_t|}{i_{\mathfrak{i}_0}, \ldots i_{\mathfrak{z}}} 20^{i_{\mathfrak{i}_0}} 52^{i_{\mathfrak{i}_1}} 40^{i_{\mathfrak{i}_2}} 4^{i_{\mathfrak{e}_0}} 4^{i_{\mathfrak{e}_1}} 4^{i_{\mathfrak{e}_2}} 4^{i_{\mathfrak{m}}} 1^{i_{\mathfrak{z}}} 2^{(\omega-1)\mathrm{tw}} \mathrm{tw}^{\mathcal{O}(1)} \\
&\quad\leq (129 \cdot 2^{(\omega-1)})^{\mathrm{tw}} \mathrm{tw}^{\mathcal{O}(1)} \\
&\quad\leq 2^{8.4\mathrm{tw}} \mathrm{tw}^{\mathcal{O}(1)},
\end{aligned}$$

where we use $|\mathbf{s}^{-1}(\mathcal{U}')| \leq \mathrm{tw}$. Since $|V(\mathcal{T})| \in \mathcal{O}(n \cdot \mathrm{tw})$, the algorithm runs in time $\mathcal{O}^*(2^{8.4\mathrm{tw}})$. □

Further algorithmic results As before, the presented algorithm can be easily extended to consider the pathwidth of the input graph as parameter and other particular classes of graphs.

Theorem 28. *There is an algorithm for* SPS *that runs in time* $\mathcal{O}^*(2^{4.38\mathrm{pw}})$.

Theorem 29. *For graphs of maximum degree 3, there is an algorithm for* SPS *that runs in time* $\mathcal{O}(1.66^n)$.

Theorem 30. *For planar graphs, there is an algorithm for* SPS *that runs in time* $\mathcal{O}(2^{\mathcal{O}(\sqrt{n})})$.

5.5 Future work

The algorithm presented for STS improved the running time to solve it from $\mathcal{O}^*(\mathrm{tw}^{\mathcal{O}(\mathrm{tw})})$ to $\mathcal{O}^*(4^{3.5\mathrm{tw}})$, being the first single-exponential time algorithm for this problem. The existence of our $\mathcal{O}^*((4-\epsilon)^{\mathrm{pw}})$ lower bound constitutes a good research opportunity to tighten this running time for both pathwidth and treewidth. Our application of the rank-based approach proved its versatility when we used more labels to represent a solution than the seminal paper, analyzing multiple cases emerging from their interaction. We left open the analysis of the Steiner version of these problems under the rank-based approach.

ℓ_1	ℓ_2	$\ell_1 \otimes \ell_2$	$\Delta_0(\ell_1, \ell_2)$	$\Delta_1(\ell_1, \ell_2)$
$\mathfrak{i}_0$	$\mathfrak{i}_0$	$\mathfrak{i}_0$	$c(e)$	0
$\mathfrak{i}_0$	$\mathfrak{i}_1$	$\mathfrak{i}_1$	$c(e)$	0
$\mathfrak{i}_0$	$\mathfrak{e}_0$	$\mathfrak{i}_1$	$c(e)$	$c(e'') - d(e'')$
$\mathfrak{i}_0$	$\mathfrak{e}_1$	$\mathfrak{i}_1$	$c(e)$	$c(e'') - d(e'')$
$\mathfrak{i}_0$	$\mathfrak{m}$	$\mathfrak{i}_0$	$c(e)$	0
$\mathfrak{i}_0$	$\mathfrak{z}$	$\mathfrak{i}_0$	$d(e)$	0
$\mathfrak{i}_1$	$\mathfrak{i}_0$	$\mathfrak{i}_1$	$c(e)$	0
$\mathfrak{i}_1$	$\mathfrak{i}_1$	$\mathfrak{i}_2$	$c(e)$	0
$\mathfrak{i}_1$	$\mathfrak{e}_0$	$\mathfrak{i}_2$	$c(e)$	$c(e'') - d(e'')$
$\mathfrak{i}_1$	$\mathfrak{e}_1$	$\mathfrak{i}_2$	$c(e)$	$c(e'') - d(e'')$
$\mathfrak{i}_1$	$\mathfrak{m}$	$\mathfrak{i}_1$	$c(e)$	0
$\mathfrak{i}_1$	$\mathfrak{z}$	$\mathfrak{i}_1$	$d(e)$	0
$\mathfrak{i}_2$	$\mathfrak{z}$	$\mathfrak{i}_2$	$d(e)$	0
$\mathfrak{e}_0$	$\mathfrak{i}_0$	$\mathfrak{i}_1$	$c(e)$	$c(e') - d(e')$
$\mathfrak{e}_0$	$\mathfrak{i}_1$	$\mathfrak{i}_2$	$c(e)$	$c(e') - d(e')$
$\mathfrak{e}_0$	$\mathfrak{e}_0$	$\mathfrak{i}_2$	$c(e)$	$c(e') + c(e'') - d(e') - d(e'')$
$\mathfrak{e}_0$	$\mathfrak{e}_1$	$\mathfrak{i}_2$	$c(e)$	$c(e') + c(e'') - d(e') - d(e'')$
$\mathfrak{e}_0$	$\mathfrak{m}$	$\mathfrak{i}_1$	$c(e)$	$c(e') - d(e')$
$\mathfrak{e}_0$	$\mathfrak{z}$	$\mathfrak{e}_0$	$d(e)$	$c(e') - d(e')$
$\mathfrak{e}_1$	$\mathfrak{i}_0$	$\mathfrak{i}_1$	$c(e)$	$c(e') - d(e')$
$\mathfrak{e}_1$	$\mathfrak{i}_1$	$\mathfrak{i}_2$	$c(e)$	$c(e') - d(e')$
$\mathfrak{e}_1$	$\mathfrak{e}_0$	$\mathfrak{i}_2$	$c(e)$	$c(e') + c(e'') - d(e') - d(e'')$
$\mathfrak{e}_1$	$\mathfrak{e}_1$	$\mathfrak{i}_2$	$c(e)$	$c(e') + c(e'') - d(e') - d(e'')$
$\mathfrak{e}_1$	$\mathfrak{m}$	$\mathfrak{i}_1$	$c(e)$	$c(e') - d(e')$
$\mathfrak{e}_1$	$\mathfrak{z}$	$\mathfrak{e}_1$	$d(e)$	$c(e') - d(e')$
$\mathfrak{e}_2$	$\mathfrak{z}$	$\mathfrak{e}_2$	$-$	0
$\mathfrak{m}$	$\mathfrak{i}_0$	$\mathfrak{i}_0$	$c(e)$	0
$\mathfrak{m}$	$\mathfrak{i}_1$	$\mathfrak{i}_1$	$c(e)$	0
$\mathfrak{m}$	$\mathfrak{e}_0$	$\mathfrak{i}_1$	$c(e)$	$c(e'') - d(e'')$
$\mathfrak{m}$	$\mathfrak{e}_1$	$\mathfrak{i}_1$	$c(e)$	$c(e'') - d(e'')$
$\mathfrak{m}$	$\mathfrak{m}$	$\mathfrak{i}_0$	$c(e)$	0
$\mathfrak{m}$	$\mathfrak{z}$	$\mathfrak{m}$	$d(e)$	0
$\mathfrak{z}$	$\mathfrak{i}_0$	$\mathfrak{i}_0$	$d(e)$	0
$\mathfrak{z}$	$\mathfrak{i}_1$	$\mathfrak{i}_1$	$d(e)$	0
$\mathfrak{z}$	$\mathfrak{i}_2$	$\mathfrak{i}_2$	$d(e)$	0
$\mathfrak{z}$	$\mathfrak{e}_0$	$\mathfrak{e}_0$	$d(e)$	$c(e'') - d(e'')$
$\mathfrak{z}$	$\mathfrak{e}_1$	$\mathfrak{e}_1$	$d(e)$	$c(e'') - d(e'')$
$\mathfrak{z}$	$\mathfrak{e}_2$	$\mathfrak{e}_2$	$-$	0
$\mathfrak{z}$	$\mathfrak{m}$	$\mathfrak{m}$	$d(e)$	0
$\mathfrak{z}$	$\mathfrak{z}$	$\mathfrak{z}$	$d(e)$	0

Table 5.2: Operator $\otimes$ and costs in SPS.

Chapter 6

Final Remarks

In this thesis, we investigated two families of pair connectivity problems, focusing on algorithmic and hardness results on the fields of approximation and parameterized complexity. These problems find usage in several areas of combinatorial optimization, such as location theory and network design, and although we scrutinize them with a theoretical point of view, the gained insights can often be transposed to practical scenarios.

We have used the dynamic programming over a tree decomposition algorithm design technique in our positive results, leveraging consolidated works and introducing novel ideas. The versatility of this technique is demonstrated by the range of solution quality guarantees we give: a $(2+\epsilon)$-approximation, an approximation scheme and an exact algorithm, all obtained with the same underlying framework, running in FPT time. The preliminaries chapter brings basic definitions and examples of such types of algorithms, and with the application in later chapters, we believe this constitutes a modest introduction to the field of parameterized approximation algorithms.

For SkHC, we give the first parameterized results for the problem: it is W[1]-hard when the parameter is the combination of the vertex cover number of the graph and the number of hubs, and there is no parameterized $(1.25-\epsilon)$-approximation algorithm when the parameter is the number of hubs, for $\epsilon > 0$; on the positive side, we give an efficient parameterized approximation scheme with treewidth as parameter, that either builds a close-to-optimal solution or concludes there is no solution with such cost. This algorithm reduces the problem to one in which we are additionally given two functions bounding the cost of the paths from a vertex to the center and to the assigned hub. Then, we go over the dynamic programming formulation, building such paths for vertices and calculating distances in an approximate fashion. A distance value is approximated by an integer power of $(1+\delta)$, reducing the number of values from $\mathcal{O}(n)$ to $\mathcal{O}(\mathrm{tw} \cdot \log n)$, which allows us to enumerate them. In the end, we show the values computed approximately are not far from the exact ones, given our choice of δ and the height of the tree decomposition. Note this is a tight algorithm for treewidth, since it is W[1]-hard to give a parameterized algorithm, and ours is a parameterized $(1+\epsilon)$-approximation for this parameter.

For MAkHC, we give hardness results for different sets of parameters: when we have the number of hubs as parameter, the cost of the solution is bounded by a constant and the graph is unweighted, there is no parameterized $(3-\epsilon)$-approximation, for $\epsilon > 0$; for the number of hubs, the highway and skeleton dimensions and the pathwidth of the graph

as parameters, or when parameterized by the number of hubs and the vertex cover of the graph, the problem does not admit a parameterized algorithm, unless $\mathsf{FPT} = \mathsf{W}[2]$. Our algorithmic result also revolves around approximate distances and a tree decomposition, but with different challenges compared to the previous problem: since in this case the set of demands is given as part of the input, we cannot assume every vertex in a subproblem needs to be connected, and enumeration is not possible unless the number of demands is part of the parameter. We solve this issue by requiring a special subset of demands to be satisfied within twice the optimal cost, and then we show the rest of demands are already satisfied by the same set of hubs; this makes the algorithm a parameterized $(2 + \epsilon)$-approximation. As in the previous algorithm, we store distances approximately, but now we employ the framework of approximate addition trees, which simplifies the process of bounding the maximum error from a value computed in a tree decomposition of limited height. We hope the presented algorithmic and hardness results for the hub location family pave the way for more research in the field of parameterized complexity for these types of problems.

For STS, we give single-exponential time parameterized algorithms for either pathwidth or treewidth running in time $\mathcal{O}^*(4^{1.7\mathrm{pw}})$ and $\mathcal{O}^*(4^{3.5\mathrm{tw}})$, improving upon results in the literature. We employ the recent rank-based approach in our results, responsible for keeping track of the connected components of a solution and efficiently storing them. In a solution, each vertex is given a label to represent its current state in respect to their neighbors, adapting as new edges and vertices are introduced. For SPS and SCS, we show an adaption of the given algorithm already renders similar results for these variants, using a different label set to describe the connectivity requirements.

Bibliography

[1] Marek Adamczyk, Jarosław Byrka, Jan Marcinkowski, Syed Meesum, and Michał Włodarczyk. Constant factor FPT approximation for capacitated k-median. *arXiv preprint arXiv:1809.05791*, 2018.

[2] Gur Saran Adhar. Optimum interval routing in k-caterpillars and maximal outer planar networks. In *Applied Informatics*, pages 692–696. Citeseer, 2003.

[3] Sibel Alumur, James Campbell, Ivan Contreras, Bahar Kara, Vladimir Marianov, and Morton O'Kelly. Perspectives on modeling hub location problems. *European Journal of Operational Research*, 291(1):1–17, 2021.

[4] Sibel Alumur and Bahar Kara. Network hub location problems: the state of the art. *European Journal of Operational Research*, 190(1):1–21, 2008.

[5] Ryuta Ando and Tomomi Matsui. Algorithm for single allocation problem on hub-and-spoke networks in 2-dimensional plane. In *International Symposium on Algorithms and Computation*, pages 474–483. Springer, 2011.

[6] Rafael Andrade and Jefferson Gurguri. Models and cutting-plane strategies for the tree-star problem. *Electronic Notes in Discrete Mathematics*, 69:261–268, 2018.

[7] Stefan Arnborg, Derek Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[8] Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete applied mathematics*, 23(1):11–24, 1989.

[9] Sanjeev Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 2–11. IEEE, 1996.

[10] Roberto Baldacci, Mauro Dell'Amico, and Salazar González. The capacitated m-ring-star problem. *Operations research*, 55(6):1147–1162, 2007.

[11] Roberto Baldacci, Alessandro Hill, Edna Hoshino, and Andrew Lim. Pricing strategies for capacitated ring-star problems based on dynamic programming algorithms. *European Journal of Operational Research*, 262(3):879–893, 2017.

[12] Max Bannach and Sebastian Berndt. Positive-instance driven dynamic programming for graph searching. In *Workshop on Algorithms and Data Structures*, pages 43–56. Springer, 2019.

[13] Mokhtar Bazaraa, John Jarvis, and Hanis Sherali. *Linear programming and network flows*. John Wiley & Sons, 2008.

[14] Marcelo Benedito, Lucas Melo, and Lehilton Pedrosa. A parameterized approximation algorithm for the multiple allocation k-hub center. In *Latin American Symposium on Theoretical Informatics*, pages 141–156. Springer, 2022.

[15] Marcelo Benedito and Lehilton Pedrosa. Approximation algorithms for median hub location problems. *Journal of Combinatorial Optimization*, 38(2):375–401, 2019.

[16] Marcelo Benedito and Lehilton Pedrosa. An efficient parameterized approximation scheme for the star k-hub center. *Procedia Computer Science*, 195:49–58, 2021.

[17] Benjamin Bergougnoux and Mamadou Kanté. Fast exact algorithms for some connectivity problems parameterized by clique-width. *Theoretical Computer Science*, 782:30–53, 2019.

[18] Benjamin Bergougnoux and Mamadou Moustapha Kanté. More applications of the d-neighbor equivalence: acyclicity and connectivity constraints. *SIAM Journal on Discrete Mathematics*, 35(3):1881–1926, 2021.

[19] Johannes Blum. W[1]-hardness of the k-center problem parameterized by the skeleton dimension. *Journal of Combinatorial Optimization*, pages 1–20, 2021.

[20] Hans Bodlaender. Dynamic programming on graphs with bounded treewidth. In *International Colloquium on Automata, Languages, and Programming*, pages 105–118. Springer, 1988.

[21] Hans Bodlaender. Planar graphs with bounded treewidth. *Tech. Rep. RUU-CS-88-14*, 1988.

[22] Hans Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[23] Hans Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical computer science*, 209(1-2):1–45, 1998.

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

[25] Hans Bodlaender, Pål Drange, Markus Dregi, Fedor Fomin, Daniel Lokshtanov, and Michał Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016.

[26] Hans Bodlaender, John Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995.

[27] Hans Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM Journal on Computing*, 27(6):1725–1746, 1998.

[28] Hans Bodlaender and Arie Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.

[29] Hans Bodlaender and Arie Koster. Treewidth computations i. upper bounds. *Information and Computation*, 208(3):259–275, 2010.

[30] Hans Bodlaender, Petra Schuurman, and Gerhard Woeginger. Scheduling of pipelined operator graphs. *Journal of Scheduling*, 15(3):323–332, 2012.

[31] Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width iii: max independent set, min dominating set, and coloring. *arXiv preprint arXiv:2007.14161*, 2020.

[32] Camile Frazão Bordini and André Luís Vignatti. An approximation algorithm for the p-hub median problem. *Electronic Notes in Discrete Mathematics*, 62:183–188, 2017.

[33] Jaroslaw Byrka and Bartosz Rybicki. Improved LP-rounding approximation algorithm for k-level uncapacitated facility location. In *International Colloquium on Automata, Languages, and Programming*, pages 157–169. Springer, 2012.

[34] Liming Cai and Jianer Chen. On fixed-parameter tractability and approximability of NP optimization problems. *journal of computer and system sciences*, 54(3):465–474, 1997.

[35] Liming Cai, Jianer Chen, Rodney Downey, and Michael Fellows. Advice classes of parameterized tractability. *Annals of pure and applied logic*, 84(1):119–138, 1997.

[36] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *International Workshop on Parameterized and Exact Computation*, pages 75–85. Springer, 2009.

[37] James Campbell. Integer programming formulations of discrete hub location problems. *European Journal of Operational Research*, 72(2):387–405, 1994.

[38] James Campbell. Hub Location and the p-Hub Median Problem. *Operations Research*, 44(6):923–935, 1996.

[39] James Campbell, Andreas Ernst, and Mohan Krishnamoorthy. Hub location problems. *Facility location: application and theory*, 2002.

[40] Jean-Luc Chabert and Évelyne Barbin. *A history of algorithms: from the pebble to the microchip*, volume 23. Springer, 1999.

[41] Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From Gap-ETH to FPT-inapproximability: Clique, dominating set, and more. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 743–754. IEEE, 2017.

[42] Sunil Chandran and Fabrizio Grandoni. Refined memorization for vertex cover. *Information Processing Letters*, 93(3):125–131, 2005.

[43] Pierre Chardaire, J.-L. Lutton, and Alain Sutter. Upper and lower bounds for the two-level simple plant location problem. *Annals of operations research*, 86:117–140, 1999.

[44] Moses Charikar, Sudipto Guha, Éva Tardos, and David Shmoys. A Constant-Factor Approximation Algorithm for the k-Median Problem. *Journal of Computer and System Sciences*, 65(1):129–149, 2002.

[45] Jianer Chen, Iyad Kanj, and Weijia Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41(2):280–301, 2001.

[46] Jianer Chen, Iyad Kanj, and Ge Xia. Improved parameterized upper bounds for vertex cover. In *International symposium on mathematical foundations of computer science*, pages 238–249. Springer, 2006.

[47] Li-Hsuan Chen, Dun-Wei Cheng, Sun-Yuan Hsieh, Ling-Ju Hung, Chia-Wei Lee, and Bang Ye Wu. Approximation algorithms for single allocation k-hub center problem. In *Proceedings of the 33rd Workshop on Combinatorial Mathematics and Computation Theory (CMCT 2016)*, pages 13–18, 2016.

[48] Li-Hsuan Chen, Dun-Wei Cheng, Sun-Yuan Hsieh, Ling-Ju Hung, Chia-Wei Lee, and Bang Ye Wu. Approximation algorithms for the star k-hub center problem in metric graphs. In *International Computing and Combinatorics Conference*, pages 222–234. Springer, 2016.

[49] Alan Cobham. The intrinsic computational difficulty of functions. *Logic, methodology and philosophy of science, Proceedings of the 1964 International Congress*, 1965.

[50] Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT approximations for k-median and k-means. *arXiv preprint arXiv:1904.12334*, 2019.

[51] Ivan Contreras, Elena Fernández, and Alfredo Marín. The tree of hubs location problem. *European Journal of Operational Research*, 202(2):390–400, 2010.

[52] Stephen Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[53] Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[54] Gérard Cornuéjols, George Nemhauser, and Laurence Wolsey. The uncapacitated facility location problem. Technical report, Cornell University Operations Research and Industrial Engineering, 1983.

[55] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

[56] Bruno Courcelle and Joost Engelfriet. *Graph structure and monadic second-order logic: a language-theoretic approach*, volume 138. Cambridge University Press, 2012.

[57] Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *Journal of computer and system sciences*, 46(2):218–270, 1993.

[58] Marek Cygan, Fedor Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4. Springer, 2015.

[59] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Joham van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 150–159. IEEE, 2011.

[60] Martin Davis. *The undecidable: Basic papers on undecidable propositions, unsolvable problems and computable functions*. Courier Corporation, 2004.

[61] Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The pace 2017 parameterized algorithms and computational experiments challenge: The second iteration. In *12th international symposium on parameterized and exact computation (IPEC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[62] Erik Demaine, Fedor Fomin, Mohammad Hajiaghayi, and Dimitrios Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Transactions on Algorithms*, 1(1):33–47, 2005.

[63] Jitender Deogun, Ton Kloks, Dieter Kratsch, and Haiko Müller. On the vertex ranking problem for trapezoid, circular-arc and other graphs. *Discrete Applied Mathematics*, 98(1-2):39–63, 1999.

[64] Michael Dinneen and Masoud Khosravani. A linear time algorithm for the minimum spanning caterpillar problem for bounded treewidth graphs. In *International Colloquium on Structural Information and Communication Complexity*, pages 237–246. Springer, 2010.

[65] Irit Dinur. Mildly exponential reduction from gap 3SAT to polynomial-gap label-cover. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 23, 2016.

[66] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 624–633, 2014.

[67] Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and IDs. *ACM Transactions on Algorithms (TALG)*, 11(2):1–20, 2014.

[68] Rodney Downey and Michael Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

[69] Rodney Downey, Michael Fellows, and Catherine McCartin. Parameterized approximation problems. In *International Workshop on Parameterized and Exact Computation*, pages 121–129. Springer, 2006.

[70] Zvi Drezner and Horst Hamacher. *Facility location: applications and theory*. Springer Science & Business Media, 2001.

[71] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

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

[73] Reza Farahani, Masoud Hekmatfar, Alireza Arabani, and Ehsan Nikbakhsh. Hub location problems: A review of models, classification, solution techniques, and applications. *Computers & Industrial Engineering*, 64(4):1096–1109, 2013.

[74] Andreas Feldmann. Fixed-parameter approximations for k-center problems in low highway dimension graphs. *Algorithmica*, 81(3):1031–1052, 2019.

[75] Andreas Feldmann, C.S. Karthik, Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020.

[76] Andreas Feldmann and Dániel Marx. The parameterized hardness of the k-center problem in transportation networks. *Algorithmica*, 82(7):1989–2005, 2020.

[77] Michael Fellows, Bart Jansen, and Frances Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European Journal of Combinatorics*, 34(3):541–566, 2013.

[78] Michael Fellows, Ariel Kulik, Frances Rosamond, and Hadas Shachnai. Parameterized approximation via fidelity preserving transformations. In *International Colloquium on Automata, Languages, and Programming*, pages 351–362. Springer, 2012.

[79] Michael Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In *Algorithms and Computation: 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings 19*, pages 294–305. Springer, 2008.

[80] Carlos Eduardo Ferreira, C.G. Fernandes, F.K. Miyazawa, J.A.R. Soares, J.C. Pina Jr., K.S. Guimarães, M.H. Carvalho, M.R. Cerioli, P. Feofiloff, R. Dahab, et al. Uma introdução sucinta a algoritmos de aproximação. *Colóquio Brasileiro de Matemática-IMPA, Rio de Janeiro-RJ*, 2001.

[81] Johannes Fichte, Markus Hecher, and Valentin Roland. Parallel model counting with cuda: Algorithm engineering for efficient hardware utilization. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[82] Fedor Fomin, Serge Gaspers, Saket Saurabh, and Alexey Stepanov. On two techniques of combining branching and treewidth. *Algorithmica*, 54(2):181–207, 2009.

[83] Fedor Fomin, Petr Golovach, Daniel Lokshtanov, and Saket Saurabh. Algorithmic lower bounds for problems parameterized by clique-width. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 493–502. SIAM, 2010.

[84] Fedor Fomin, Petr Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM Journal on Computing*, 43(5):1541–1563, 2014.

[85] Fedor Fomin and Tuukka Korhonen. Fast FPT-approximation of branchwidth. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 886–899, 2022.

[86] Fedor Fomin and Dimtirios Thilikos. A simple and fast approach for solving problems on planar graphs. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 56–67. Springer, 2004.

[87] Fedor Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM Journal on Computing*, 44(1):54–87, 2015.

[88] Robert Ganian, Petr Hlineny, and Jan Obdrzálek. Clique-width: When hard does not mean impossible. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011.

[89] R. Garcia. An optimization method for multiplexer locations. *Operations-Research-Spektrum*, 7(2):111–114, 1985.

[90] Michael Garey and David Johnson. The rectilinear steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

[91] Michael Garey and David Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[92] Dongdong Ge, Simai He, Yinyu Ye, and Jiawei Zhang. Geometric rounding: a dependent randomized rounding scheme. *Journal of Combinatorial Optimization*, 22(4):699–725, 2010.

[93] Shahin Gelareh and David Pisinger. Fleet deployment, network design and hub location of liner shipping companies. *Transportation Research Part E: Logistics and Transportation Review*, 47(6):947–964, 2011.

[94] Nader Ghaffarinasab. A tabu search heuristic for the bi-objective star hub location problem. *International Journal of Management Science and Engineering Management*, 15(3):213–225, 2020.

[95] Teofilo Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[96] David Harris and N.S. Narayanaswamy. A faster algorithm for vertex cover parameterized by solution size. *arXiv preprint arXiv:2205.08022*, 2022.

[97] Marcia Helme and Thomas Magnanti. Designing satellite communication networks by zero—one quadratic programming. *Networks*, 19(4):427–450, 1989.

[98] Lane Hemaspaandra. SIGACT news complexity theory column 74. *SIGACT News*, 43(2):51–52, 2012.

[99] Alessandro Hill and Stefan Voß. An equi-model matheuristic for the multi-depot ring star problem. *Networks*, 67(3):222–237, 2016.

[100] Dorit Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on computing*, 11(3):555–556, 1982.

[101] Dorit Hochbaum and David Shmoys. A Best Possible Heuristic for the k-Center Problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[102] Dorit Hochbaum and David Shmoys. A Unified Approach to Approximation Algorithms for Bottleneck Problems. *J. ACM*, 33(3):533–550, 1986.

[103] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[104] Masaru Iwasa, Hiroo Saito, and Tomomi Matsui. Approximation algorithms for the single allocation problem in hub-and-spoke networks and related metric labeling problems. *Discrete Applied Mathematics*, 157(9):2078–2088, 2009.

[105] Patrick Jaillet, Gao Song, and Gang Yu. Airline network design and hub location problems. *Location science*, 4(3):195–212, 1996.

[106] Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM (JACM)*, 50(6):795–824, 2003.

[107] Bahar Kara and Barbaros Tansel. On the single-assignment p-hub center problem. *European Journal of Operational Research*, 125(3):648–655, 2000.

[108] H. Karimi and M. Bashiri. Hub covering location problems with different coverage types. *Scientia Iranica*, 18(6):1571–1578, 2011.

[109] Richard Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[110] C.S. Karthik, Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. *arXiv preprint arXiv*, 1711, 2017.

[111] Ioannis Katsikarelis, Michael Lampis, and Vangelis Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discrete Applied Mathematics*, 264:90–117, 2019.

[112] Samir Khuller and An Zhu. The general steiner tree-star problem. *Information Processing Letters*, 84(4):215–220, 2002.

[113] Arnaud Knippel and Viet Hung Nguyen. On tree star network design. In *Proceedings of International Network Optimization Conference INOC*, 2007.

[114] Yasuaki Kobayashi and Hisao Tamaki. Treedepth parameterized by vertex cover number. In *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[115] Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–192. IEEE, 2022.

[116] Tuukka Korhonen, Jeremias Berg, and Matti Järvisalo. Solving graph problems via potential maximal cliques: An experimental evaluation of the bouchitté–todinca algorithm. *Journal of Experimental Algorithmics (JEA)*, 24:1–19, 2019.

[117] B. Korte, J. Vygen, B. Korte, and J. Vygen. *Combinatorial optimization*, volume 1. Springer, 2011.

[118] Martine Labbé, Gilbert Laporte, Inmaculada Rodríguez Martín, and Juan Jose Salazar Gonzalez. The ring star problem: Polyhedral analysis and exact algorithm. *Networks: An International Journal*, 43(3):177–189, 2004.

[119] Martine Labbé and Hande Yaman. Solving the hub location problem in a star–star network. *Networks: An International Journal*, 51(1):19–33, 2008.

[120] Jens Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *Journal of Algorithms*, 20(1):20–44, 1996.

[121] Michael Lampis. Parameterized approximation schemes using graph widths. In *International Colloquium on Automata, Languages, and Programming*, pages 775–786. Springer, 2014.

[122] Youngho Lee, Steve Chiu, and Jennifer Ryan. A branch and cut algorithm for a steiner tree-star problem. *INFORMS Journal on Computing*, 8(3):194–201, 1996.

[123] Youngho Lee, Lu Lu, Yuping Qiu, and Fred Glover. Strong formulations and cutting planes for designing digital data service networks. *Telecommunication Systems*, 2(1):261–274, 1993.

[124] Markus Leitner, Ivana Ljubić, Juan-José Salazar-González, and Markus Sinnl. An algorithmic framework for the exact solution of tree-star problems. *European Journal of Operational Research*, 261(1):54–66, 2017.

[125] Leonid Anatolevich Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973.

[126] Hongyu Liang. The hardness and approximation of the star p-hub center problem. *Operations Research Letters*, 41(2):138–141, 2013.

[127] Daniel Lokshtanov, Pranabendu Misra, M.S. Ramanujan, Saket Saurabh, and Meirav Zehavi. FPT-approximation for FPT problems. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 199–218. SIAM, 2021.

[128] Daniel Lokshtanov, Fahad Panolan, M.S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 224–237, 2017.

[129] Abilio Lucena, Luidi Simonetti, and Alexandre Salles da Cunha. The tree-star problem: A formulation and a branch-and-cut algorithm. *Electronic Notes in Discrete Mathematics*, 52:285–292, 2016.

[130] Gustavo Malkomes, Matt Kusner, Wenlin Chen, Kilian Weinberger, and Benjamin Moseley. Fast distributed k-center clustering with outliers on massive data. In *Advances in Neural Information Processing Systems*, pages 1063–1071, 2015.

[131] Silviu Maniu, Pierre Senellart, and Suraj Jog. An experimental study of the treewidth of real-world graph data. In *22nd International Conference on Database Theory (ICDT 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[132] Bertrand Marchand, Yann Ponty, and Laurent Bulteau. Tree diet: reducing the treewidth to unlock FPT algorithms in RNA bioinformatics. *Algorithms for Molecular Biology*, 17(1):1–17, 2022.

[133] Rafael Martí, Panos Pardalos, and Mauricio Resende. *Handbook of Heuristics*. Springer International Publishing, 2018.

[134] Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.

[135] Pitu Mirchandani and Richard Francis. *Discrete location theory*. Wiley-Interscience, 1990.

[136] Andranik Mirzaian. Lagrangian relaxation for the star-star concentrator location problem: Approximation algorithm and bounds. *Networks*, 15(1):1–20, 1985.

[137] George Nemhauser and Laurence Wolsey. Integer programming and combinatorial optimization. *Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin*, 20:8–12, 1988.

[138] Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

[139] Rolf Niedermeier. Some prospects for efficient fixed parameter algorithms. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 168–185. Springer, 1998.

[140] Rolf Niedermeier. *Invitation to fixed-parameter algorithms*. Oxford University Press, 2006.

[141] Taku Okada, Akira Suzuki, Takehiro Ito, and Xiao Zhou. Algorithm for the minimum caterpillar problem with terminals. *Proceedings of the 6th Annual Meeting of Asian Association for Algorithms and Computation*, 2013.

[142] Morton O'Kelly. The location of interacting hub facilities. *Transportation Science*, 20(2):92–106, 1986.

[143] Morton O'Kelly. A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research*, 32(3):393–404, 1987.

[144] Sang-il Oum. Rank-width: Algorithmic and structural results. *Discrete Applied Mathematics*, 231:15–24, 2017.

[145] Lehilton Pedrosa, Vinícius dos Santos, and Rafael Schouery. Uma aproximação para o problema de alocação de terminais. In *Anais do I Encontro de Teoria da Computação*. SBC, 2016.

[146] Dan Pelleg and Andrew Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 277–281, 1999.

[147] Armando Honorio Pereira, Geraldo Robson Mateus, and Sebastián Urrutia. Branch-and-cut algorithms for the p-arborescence star problem. *International Transactions in Operational Research*, 29(4):2374–2400, 2020.

[148] Willem Pino, Hans Bodlaender, and Johan Van Rooij. Cut and count and representative sets on branch decompositions. In *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[149] Leon Planken, Mathijs de Weerdt, and Roman van der Krogt. Computing all-pairs shortest paths by leveraging low treewidth. *Journal of artificial intelligence research*, 43:353–388, 2012.

[150] Bruce Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

[151] Neil Robertson and Paul Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

[152] Neil Robertson and Paul Seymour. Graph minors. xx. wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004.

[153] Neil Robertson, Paul Seymour, and Robin Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 62(2):323–348, 1994.

[154] Luidi Simonetti, Yuri Frota, and Cid de Souza. An exact method for the minimum caterpillar spanning problem. In *CTW*, pages 48–51, 2009.

[155] Aravind Srinivasan. Approximation algorithms via randomized rounding: a survey. *Series in Advanced Topics in Mathematics, Polish Scientific Publishers PWN*, pages 9–71, 1999.

[156] Kaarthik Sundar and Sivakumar Rathinam. Multiple depot ring star problem: a polyhedral study and an exact algorithm. *Journal of Global Optimization*, 67(3):527–551, 2017.

[157] El-Ghazali Talbi. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

[158] Jinsong Tan and Louxin Zhang. Approximation algorithms for the consecutive ones submatrix problem on sparse matrices. In *International Symposium on Algorithms and Computation*, pages 835–846. Springer, 2004.

[159] Mikkel Thorup. All structured programs have small tree width and good register allocation. *Information and Computation*, 142(2):159–181, 1998.

[160] Adriano Vasconcelos, Carlos Nassi, and Luiz Lopes. The uncapacitated hub location problem in networks under decentralized management. *Computers & Operations Research*, 38(12):1656–1666, 2011.

[161] Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[162] Vijay Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[163] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 887–898, 2012.

[164] David Williamson and David Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[165] Hande Yaman and Sourour Elloumi. Star p-hub center problem and star p-hub median problem with bounded path lengths. *Computers & Operations Research*, 39(11):2725–2732, 2012.

[166] Ta-Hui Yang. Stochastic air freight hub location and flight routes planning. *Applied Mathematical Modelling*, 33(12):4424–4430, 2009.