An Algorithmic Approach to Local and Global Resource Allocations

Mong-Jen Kao[1]

[1]Department of Computer Science and Information Engineering, National Taiwan University. d97021@csie.ntu.edu.tw

Contents

Part I

Resource Allocation as an Algorithmic Problem

Resource management is from all aspects one of the most important issues to be addressed and whose attracted attention is never sufficient. Ranging from individuals, companies of moderate property, to international enterprises or even government policy makers, the need to manage and allocate the limited amount of resources in an efficient and effective way that leads to a profitable outcome is a self-proving story which every manager will eventually understand and tell.

One natural way to describe this scenario is to view resource management as a process of distributing resources to supply demanding targets, which could possibly be individual objects or their combinations of certain forms. The resources could either be generated by an universal source producer or may come from multiple objects. With respect to different outcomes led by different ways of assigning the resources, we expect a common measurement on the quality of each different outcome in order to distinguish how good and how profitable it is. Then, among possible combinations of different assignments, we, as a manager of the resources, wish to choose the particular assignment which results in the best outcome. Therefore, resource management, or, resource planning, is in fact a combinatorial optimization problem in a sense that, we have a set of choices each with its own target value and we wish to select a choice from the set such that the resulting target value is as large as possible.

In some cases, there exists locality constraints between the supply and the demand, and it only makes senses to assign the resource to demanding objects from its vicinity. For example, in wireless ad-hoc networks, a peer can only be reached after entering the area it guards, and we will not be able to receive any signal (resource) outside our vicinity. Another example comes from the scenario depicted by the well-known facility location problem. As the owner of the brand of a chain coffee shop, we want to set up service points in order to attract as many customers as possible. When setting up a service point, we get to serve the customers in the neighboring area. As customers are unlikely to visit a distant coffee shop, we want to achieve a reasonable balance between the budget spent and the customers received.

In other cases when we do not have locality constraints but geographical conditions instead come into play, the resources are packed and delivered through intermediate stops via underlying backbone connections to demanding targets. For example, in an international manufacture enterprise, the administrative staff has to plan the overall production line. Starting from setting up local manufacturers, locating worldwide retailers, to the delivery of the products from the manufacturers to the retailers, not only the locations of the facilities matter but also the way how the products are delivered will have great impact on the overall profit. In this case, in addition to placing the facilities at suitable spots, we also have to arrange the underlying backbone for which the products are shipped carefully to guarantee a safe and efficient delivery.

For yet another example for which the resource assignment, location of the facilities, and the underlying backbone network are equally important, consider the real-time multicast streaming network. We have a real-time streaming source of limited capacity and a considerably large set of clients who wish to receive this stream. As the total amount of demand exceeds greatly from the capacity the streaming source can provide, we either have to choose a set of lucky clients who will monopolize the stream, or, we have to identify a set of proxies providers to further casting the messages. In both situations, carefully planning and maintaining the underlying connections on the message delivering paths is a necessity to ensure the streaming quality.

In this dissertation, we attempt to address the problem of resource allocation from an algorithmic perspective. Specifically, the following two categories of problems are considered.

(i) Local Resource Allocation. The first category of problems we investigate in this dissertation concern the scenario when we have locality constraints and the demand-supply relations only exist between adjacent objects under the given locality conditions. A fundamental and also the most commonly used concept to depicting this scenario is given as follows.

For each pair of objects, the question of whether one belongs to the vicinity of the other or not is represented by a binary relation between these two objects. Depending on the context, this binary relation could be either symmetric or asymmetric, i.e., in a symmetric binary relation, if item $\mathcal{P}$ belongs to the vicinity of item $\mathcal{Q}$, then item $\mathcal{Q}$ also belongs to the vicinity of item $\mathcal{P}$, and vice versa. Note that, although many of the daily visible examples, such as road networks, cable networks, or even social networks, suggest symmetric binary relations, there are situations which give asymmetric ones. For example, in the transportation network, the vicinity for which an individual can reach within a limited amount of time is strictly determined by the power of the provided vehicle, and therefore the vicinity relation could be asymmetric if the vehicles used by two individuals are not identical.

The amount of supply demanded by each individual object is quantized as a non-negative real number. To provide the supply to demanding objects, a given subset of the objects can be activated by paying a prescribed amount of prices, also quantized as non-negative real numbers, to generate the supply. The amount of supply an object can provide after activation, or, the *capacity* of an object, is quantized as a non-negative real number as well. The general objective of this problem is to properly serve the demanding objects in a cost-efficient way, in a sense that we wish to spend as least as possible while serving as much demand as possible. As a combinatorial optimization problem, when the objective is to entirely supply the demanding objects, we wish to minimize the total activation cost. Instead, when we have limited amount of budget to spend, we wish to maximize the utilization, i.e., to fully supply as many objects as possible. Considering the general objective of serving the demanding objects, there are several interesting problems, which are also of practical importance, to address.

First, when the amount of supply each object can generate after activation is unlimited while the amount of demand as well as the activation cost for each object is unit, the problem is exactly one of the most fundamental and extensively-studied problem in *algorithmic graph theory* [77], which is also known as the Dominating Set problem [42] in the literature. Starting from this problem, an ample body of work has been proposed, studying both the problem itself and possible extensions, including further structural restrictions on the set of activated supply providers [87], slight relaxations on the locality constraints as well as the demand-supply relations [41,63], and generalizations on the set of parameters, i.e., parameters other than units and infinities [62].

Second, when the overhead of supply delivery is non-negligible, and we have to pay a corresponding extra price for each demand-supply assignment, then the problem becomes the well-known Facility Location problem, which has occupied an important place in operations research since the early sixties. During the past decades, a considerable amount of work regarding both the facility location problem and its variations has been proposed [73,84].

When we are allowed to activate the supplying source multiple times in exchange of more supply, then we say the scenario is *capacitized*. For example, in the construction of cellular phone service network, depending on the number of potential customers, we will need different base stations. In suburban areas, we prefer stations with smaller capacity and less expensive cost while in the downtown, we need stations of larger capacity. From time to time, when the number of customers increases and the base station of a certain area is about to saturate, we will have to increase the amount of supply for that area. One way to resolve this issue is to replace the original base station with a new one which is more powerful. On the other hand, we can simply place another station in addition to the existing one. The above example demonstrates the possibility of *capacitation*. In fact, the concept of capacitation has attracted an intensive attention during the last decade, and what follows is a series of remarkable work, considering resource allocation problems of all sorts, including dominating set, facility location, and etc [20,53–55].

In the first part of this dissertation, with respect to local resource allocation, or, more pre-

cisely, resource allocation with locality constraints, we place our focus on a natural generalization of the dominating set problem under the concept of capacitation, namely, the Capacitated Domination problem. Under the objective of finding optimal resource assignments, a comprehensive study for this problem with respect to different scenarios is presented. The readers are referred to Part II for an elaborate discussion of this topic.

(ii) Global Resource Planning via Backbone Construction. In the second half of this dissertation, we attempt to address resource planning from an overall perspective. Considering the intimate relation between the assignment of supply and the delivery that follows, when the overhead to carrying supply from the sources to the demanding objects is non-negligible, the planning of the underlying backbone structure will play a relatively important role in the quality of the overall assignment. As a preliminary step to the investigation of the intricate global resource planning with non-negligible transmitting overheads, I placed my focus on the design of a quality underlying backbone with respect to different structural considerations. This issue falls exactly into the holy grail of *network design*, which has occupied a central and focused place in the history of computer science and which has also been studied extensively from both theoretical and empirical aspects.

In general, when it comes to network design, we expect a way of connecting a given set of sites by a subset of possible connecting links such that certain requirements are met. For example, a typical problem in wireless sensor network design is to connect the set of sensors, which are powered by batteries of limited capacity, in a way that desired data can be collected while certain criteria, such as sensor lifetimes, stability, as well as the number of sensors required, are optimized under certain given objectives. Similar problems arise in other applications such as communication networks, transportation networks, VLSI chip routing design, etc.

In different applications, the constraints and requirements introduce themselves in different ways, ranging from the way how the network is connected to the way how the quality of the network is measured. In order to deal with such diversity, the network design problem can be formulated in various ways. The space from which the sites are drawn is often the Euclidean plane, but other metrics or distance functions are also possible. Another design issue comes from the degrees of fault tolerance, which is a restriction to the network structure in advance. For instance, trees are the simplest connected network structure which provides no fault tolerance at all, in the sense that, when a link fails, the entire network is no longer connected and at least one site is unreachable from some other site. When we wish to raise the degree of fault tolerance, more links between the sites will be required as alternatives, and the corresponding cost to plan the network will be higher. On the other hand, the quality of a network can be measured in other ways as well. For example, in wireless sensor networks, to adapt with limited transmission power of each sensor device, we require the distances, or, lengths of the links between the sensors and the base stations to be upper-limited. For other scenarios in sensor networks, we may require the number of links connecting to each sensor to be small, in order to reflect the limited computing capability of each device.

In communication networks, the transmission time between any two nodes is often a major criterion. One typical measure is the maximum transmission time between pairs of nodes. This corresponds to the longest path between any two nodes. The problem of constructing a spanning network of minimum diameter and also its variations have been studied extensively in the literature [43, 44, 82]. Constrained by the geographical natures of the locations where the sites are placed, this measure usually does not reflect the quality of the overall connections, as it focuses merely on the transmitting time of the worst pair. Hence, another interesting measure is proposed to consider the ratio between the distance of each pair in the network and the corresponding distance in the original metric. In other words, in this measure, we are interested in the detour of each pair of the sites in the network. The problem of planning a network with small pairwise detours is known as the Spanner problem [70]. When we turn to the average

performance of a network, it is natural to consider the weighted sum of pairwise distances, where the weight associated with each pair depends on the underlying application [2, 51, 92].

The efficiency and the latency for each delivery are fundamental to the performance evaluation of most networks, including transportation networks, flow networks, and communication networks. In this scenario, we wish to route the paths between the sites such that minimum capacity of each link, or, the *bandwidth*, along each path is as large as possible, in a sense that more can be delivered in each transport. This is called Bottleneck Path problem [83, 86]. In addition to constructing a quality network, issues regarding maintenance or modification, e.g., removing a malfunctioned link, introducing a new site, or, partially alter the links between the sites of a given region under certain conditions in a dynamic way is also important to consider. These issues correspond to the *robustness* of the network, i.e., the ability of the network against the removal of cites or edges dynamically, due to random failures of cites or links.

For this part, a focused study on the construction of a quality underlying backbone network with respect to two different object measurements is presented. First, without considering the expense, we show that, provided an arbitrary metric, a spanning network with good average performance guarantees can be constructed and maintained semi-dynamically and efficiently, using the minimum number of links. Second, taking the expense into consideration, we move the focus further to the cost-efficient network constructions. Situations under different constraints and requirements are considered.

Organization of this Dissertation. Chapter §1 presents the definitions to the notations used throughout the content and preliminary ground knowledges that are necessary to thoroughly depict the ideas and the algorithms.

Part II and Part III formally introduce the aforementioned issues which have been addressed within the scopes of local resource allocation and global resource planning via backbone construction, respectively. In particular, algorithmic ideas with novelties for the addressed problems which lead to efficient solutions with quality guarantees will be the main dish to taste. For situations seemingly lacking of good solutions, mathematical proofs are provided to show the non-existence of efficient and quality solutions unless all such problems can be solved and the entire computing theory collapses.

At the end, we will conclude with possible future directions to proceed as well as open problems left to consider within the same scopes.

Chapter 1

Notations and Ground Knowledge *

1.1 Graph Theory

1.2 Linear Programs & Duality

1.3 Parameterized Complexity Theory

Chapter 2

Local Demand and Supply: the Capacitated Domination Problem

The Dominating Set problem has been one of the most fundamental and well-known problems in both graph theory and combinatorial optimization. Given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and an integer k, the dominating set problem asks for a subset $\mathbf{D} \subseteq \mathbf{V}$ whose cardinality does not exceed k such that every vertex in the graph either belongs to this set or has a neighbor which does. As this problem is known to be **NP**-hard, approximation algorithms have been proposed in the literature [9, 45, 52]. On the one hand, greedy algorithms are shown to achieve a guaranteed ratio of $(\ln n)$ [52], where n is the number of vertices. The ratio is later proven to be tight by Feige [30]. On the other hand, algorithms based on dual-fitting provide a guaranteed ratio of Δ [45], where Δ is the maximum degree of the vertices in the graph. A polynomial-time approximation scheme for planar graphs was given by Baker [9].

In terms of *parameterized complexity*, dominating set has its special place as well [25, 31, 71]. In contrast to the Vertex Cover problem, which is *fixed-parameter tractable* (FPT) with respect to solution size, dominating set is proven to be $\mathbf{W}[\mathbf{2}]$-complete, in the sense that no fixed-parameter tractable algorithm with respect to solution size exists unless $\mathbf{FPT} = \mathbf{W}[\mathbf{2}]$. Although dominating set is a fundamentally hard problem in the parameterized **W**-hierarchy, it has been used as a benchmark problem for sub-exponential time parameterized algorithms [5, 23, 32] and linear size kernels have been obtained for planar graphs [6, 31, 40, 71], and more generally, for graphs that exclude a fixed graph as a minor.

In addition, a vast body of work has been proposed in the literature, considering possible variations from purely theoretical aspects to practical applications. See [42, 77] for a detailed survey. In particular, variations of dominating set problem occur in numerous practical settings, ranging from strategic decisions, such as locating radar stations or emergency services, to computational biology and to voting systems. For example, Haynes et al. [41] considered Power Domination problem in electricity networks [41, 63] while Wan et al. [87] considered Connected Dominating Set problem in wireless ad hoc networks.

A series of study on capacitated covering problems was initiated by Guha et al., [39], which addressed the Capacitated Vertex Covering problem from a scenario of Glycomolecule ID (GMID) placement. Under the concept of *capacitation*, each vertex can be activated in order to serve the demanding edges by paying the corresponding activation cost. Several follow-up papers have appeared since then, studying both this topic and related variations [22, 34, 35]. In particular, Chuzhoy [22] proved that, when the multiplicities of the vertices are limited, i.e., the number of times a vertex can be activated is limited, and the demand of each each is unit, this problem is already at least as hard as Set Cover problem, and a logarithmic approximation is further provided. Furthermore, when the demand is arbitrary and inseparable but the activation cost is unit, they showed that this problem cannot be approximated at all, unless $\mathbf{P} = \mathbf{NP}$. When

the multiplicities are not limited but the edge demand is unit, they gave a 3-approximation for this problem, which was later improved to 2 by Gandhi et al [34].

These problems are also closely related to the Capacitated Facility Location problem, which considers the demand assignment in a metric space together with assignment cost and activation cost and which has also drawn a lot of attention since 1990s. Approximation algorithms based on *LP-rounding* and *primal-dual* analysis are provided. [50, 84]. See also [20, 73].

Motivated by a local service-requirement assignment scenario, we considered a generalization of the dominating set problem, called Capacitated Domination problem, in a series of work [53–55], which is defined below.

Problem Definition

Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a graph with three non-negative parameters defined on each vertex $u \in V$, which will be referred to as the cost, the capacity, and the demand, further denoted by $w(u)$, $c(u)$, and $d(u)$, respectively. The demand of a vertex stands for the amount of service it requires from its adjacent vertices, including the vertex itself, while the capacity of a vertex represents the amount of service each multiplicity (copy) of that vertex can provide.

The demand assignment function $f\colon \mathbf{V} \times \mathbf{V} \to \mathcal{R}^+ \cup \{0\}$ is a function which maps pairs of vertices to non-negative real numbers. Intuitively, $f(u, v)$ denotes the amount of demand of u that is assigned to v.

Definition 2.1 (feasible demand assignment function)**.** A demand assignment function $f\colon \mathbf{V} \times \mathbf{V} \to \mathcal{R}^+ \cup \{0\}$ is said to be feasible if

$$\sum_{u \in \mathbf{N_G}[v]} f(v, u) \geq d(v),$$

for each $v \in \mathbf{V}$. That is, the demand of each vertex is fully-assigned to its closed neighbors.

Given a demand assignment function f, the corresponding capacitated dominating multi-set $\mathbf{D}(f)$ is defined as follows. For each vertex $v \in \mathbf{V}$, the *multiplicity* of v in $\mathbf{D}(f)$ is defined to be

$$x_f(v) = \left\lceil \frac{\sum_{u \in \mathbf{N_G}[v]} f(u, v)}{c(v)} \right\rceil.$$

The cost of the assignment function f, denoted $w(f)$, is defined to be

$$w(f) = \sum_{u \in \mathbf{V}} w(u) \cdot x_f(u).$$

Problem 1 (Capacitated Domination problem)**.** Given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with cost, capacity, and demand defined on each vertex, the capacitated domination problem asks for a feasible demand assignment function f such that $w(f)$ is minimized.

Depending on the way how the demand is assigned, there are different models to consider. By inseparable demand model we require that $f(u, v)$ is either 0 or $d(u)$, for each edge $(u, v) \in E$, while in separable demand model we do not have such constraints. Literally, in inseparable demand model we require that the demand of a vertex must be fully-served merely by one of its closed neighbors. Note that, a straight reduction from Subset Sum problem shows that, when the demand is inseparable, it is already **NP**-hard to compute a feasible demand assignment function from a given capacitated dominating multi-set, which is already known to be feasible. Therefore it is essential to require that the demand assignment function be specified when dealing with this problem.

<table>
<tr><th></th><th>Inseparable demand</th><th>Separable demand</th></tr>
<tr><td rowspan="3">General graphs</td><td rowspan="3">$(\ln n)$-approx.</td><td>$(4\ln n+2)$-approx.</td></tr>
<tr><td>$(2\ln n+1)$-approx., unit cost</td></tr>
<tr><td>Δ^*-approx.</td></tr>
<tr><td rowspan="4">Graphs of bounded treewidth</td><td colspan="2">$\mathbf{W}[\mathbf{1}]$-hard, with respect to treewidth</td></tr>
<tr><td colspan="2">FPT w.r.t. treewidth, max. capacity, and max. demand</td></tr>
<tr><td>$O\left(2^{2k(\log M+1)+\log k}n\right)$</td><td>$O\left(2^{(2M+2N+1)\log k}n\right)$</td></tr>
<tr><td colspan="2">M: maximum capacity, N: maximum demand, k: treewidth</td></tr>
<tr><td>Outerplanar graphs</td><td></td><td>Constant factor approx.</td></tr>
<tr><td rowspan="2">Planar graphs</td><td>$\left(\frac{3}{2}-\epsilon\right)$-approx. threshold</td><td>Constant factor approx.</td></tr>
<tr><td colspan="2">$(1+\epsilon)$ approx. in pseudo-polynomial time</td></tr>
<tr><td>Trees</td><td>Exact algo. in linear time</td><td>Poly-time approx. scheme</td></tr>
</table>

Table 2.1: An overall summary on the results provided in this dissertation for capacitated domination problem, classified by the demand models and graph classes.

Summary of the Content to Appear

In Chapter §4, we consider this problem on general graphs. Logarithmic approximations with respect to both separable and inseparable demand models are presented. Furthermore, a sophisticated charging scheme based on linear program duality which leads to a Δ^*-approximation for separable demand is also presented. This establishes asymptotically matching bounds to the classical dominating set problem up to constant factors.

From the perspective of parameterization, we consider graphs of bounded treewidth in Chapter §5. First, this problem is showed to be $\mathbf{W}[\mathbf{1}]$-hard when parameterized by treewidth, regardless of demand assigning model. Then, an exact fixed-parameter tractable algorithm with respect to treewidth and the maximum capacity of the vertices is provided. Then, as for the approximation side, we present a constant factor approximation algorithm for separable demand model on outerplanar graphs, based on a hierarchical perspective on outerplanar graphs, a further analysis on the primal linear program, and a refined charging scheme.

In Chapter §6, as for an overview to the problem complexity on planar graphs, both pseudo-polynomial time approximation schemes and constant factor approximations for this problem are presented, by generalizing the aforementioned results under a standard framework due to Baker [9]. Although the former one follows the standard approach, the second one, however, requires slight modification and careful augmentation of the algorithm. Furthermore, a $\left(\frac{3}{2}-\epsilon\right)$-approximation threshold for planar graphs when the demand is inseparable is provided, for any $\epsilon > 0$.

In Chapter §7, we consider this problem on trees. First, a linear time algorithm for inseparable demand is presented. Based on the **NP**-hardness proof for separable demand, the bottleneck

of this problem on trees is formulated as a combinatorial optimization problem named Relaxed Knapsack problem. Then, a polynomial time approximation scheme for separable demand model on trees is presented, based on an fully-poly approximation scheme for the relaxed knapsack problem. A summary of the aforementioned results to appear is depicted in Table. 2.1.

Chapter 3

Supplying Global Demand: Quality Backbone Design & Maintenance

In this part, we move our focus to the theme of global resource planning and considered the construction and maintenance of a quality backbone network. Two different categories of problems are studied. First, disregarding the cost required for each link, we discuss the problem of how good a tree metric can achieve in terms of average performance. This problem is closely related to low-stretch metric embedding problem and is interesting by its own flavor from the line of research proposed in the literature. As the structure of a tree imposes great constraints on the resulting pairwise distances, any embedding of a metric into a tree metric is known to have maximum pairwise stretch of $\Omega(\log n)$. we show, however, from the perspective of average performance, there exist tree metrics which preserve the sum of pairwise distances of the given metric up to a small constant factor, for which can be proven to be no worse than twice what we can possibly expect. Second, when the given metric is extracted from Euclidean space of finite dimension d, we show the existence of spanning trees for the given point set such that the sum of pairwise distances is preserved up to a constant which depends only on d.

Second, taking the expense to build the network into account, we switch our focus further and discuss the problem of cost-efficient network construction. A framework of bi-objective optimization problems, where one objective is to be maximized while the other is to be minimized, is considered. Under this objective, the problems of computing a maximum cost-efficiency network under various additional constraints are investigated. In the following, we introduce these two concepts separately in more detail.

3.1 Metric Embeddings of Low DWA-Stretch

The problem of approximating a given metric by a metric which is structurally simpler has been a central issue to the theory of finite metric embedding and has been studied extensively in the past decades. A particularly simple metric of interest, which also favors from the algorithmic perspective, is a tree metric. By a tree metric we mean a metric induced by the shortest distances between pairs of points in a tree containing the given points. Generally we would require the distances in the given metric not to be underestimated in the target metric, which is crucial for most of the applications, and we would like to bound the increase of the distances, distortion, or stretch, from above. See [2, 11, 14, 29]. On the other hand, a similar and equally important problem in network design is to find a tree spanning the network, represented by a graph, that provides a good approximation to the shortest path metric defined in the graph [2, 7, 27].

Let $\mathbf{M} = (\mathbf{V}, d)$ and $\mathbf{M}' = (\mathbf{V}, d')$ be two metrics over the same point set $\mathbf{V}$ such that

$d'(u,v) \geq d(u,v)$ for all $u, v \in \mathbf{V}$. For each $u, v \in \mathbf{V}$, let

$$stretch(u,v) = \frac{d'(u,v)}{d(u,v)}$$

be the *pairwise stretch*, or *distortion*, between the pair u and v. Different notions have been suggested to quantify how well the distances of $\mathbf{M}$ are preserved in $\mathbf{M}'$, e.g.,

1. Maximum pairwise stretch [70], defined by $\max_{u,v\in\mathbf{V}} stretch(u,v)$, which is closely related to the extensively studied *Spanner* problems.

2. Average pairwise stretch [2, 27], defined as

$$\frac{1}{\binom{|\mathbf{V}|}{2}} \cdot \sum_{u,v\in\mathbf{V}} stretch(u,v).$$

3. Distance-weighted average stretch [51, 74, 92], defined as

$$\frac{1}{\sum_{u,v\in\mathbf{V}} d(u,v)} \sum_{u,v\in\mathbf{V}} d(u,v) \cdot stretch(u,v) = \frac{\sum_{u,v\in\mathbf{V}} d'(u,v)}{\sum_{u,v\in\mathbf{V}} d(u,v)}.$$

   This measure makes sense in real-time scenarios when it is less desirable and more costly to raise the distances of distant pairs than that of close pairs. For example, the effect of raising the delay of a pair from 2 seconds to 10 seconds is less tolerable than raising the delay of another pair from 20 ms to 100 ms. Throughout this part we will also refer to the sum of pairwise distances as the routing cost following the terminology used in the literature.

Problem Definition

In this part, we considere the problem of how well a tree is able to preserve the sum of pairwise distances, or, the distance-weighted average stretch, of an underlying metric. To be more precise, let $\mathbf{M} = (\mathbf{V}, d)$ and $\mathbf{M}' = (\mathbf{V}', d')$ be two metrics. $\mathbf{M}'$ is said to dominates $\mathbf{M}$ if $\mathbf{V}' \supseteq \mathbf{V}$ and for all $u, v \in \mathbf{V}$, we have $d'(u,v) \geq d(u,v)$. As for the construction of good embeddings, we consider the following two problems in Chapter §8.

Problem 2 (Tree Metric Embeddings of Low DWA-Stretch problem)**.** Let $\mathbf{M} = (\mathbf{V}, d)$ be a given metric and $\mathcal{D}(\mathbf{M})$ be the set of dominating tree metrics of $\mathbf{M}$. What is

$$\inf_{(\mathbf{V}',d')\in\mathcal{D}(\mathbf{M})} \frac{\sum_{u,v\in\mathbf{V}} d'(u,v)}{\sum_{u,v\in\mathbf{V}} d(u,v)} \ ?$$

Problem 3 (Euclidean Spanning Tree of Low DWA-Stretch problem)**.** Let $\mathbf{V}$ be a set of points in the Euclidean space $\mathcal{R}^d$, $|\overline{u,v}|$ be the straight-line distance between two points $u, v \in \mathbf{V}$, $\mathbf{T}(\mathbf{V})$ be the set of spanning trees of $\mathbf{V}$, and d_T be the distance function of T, for any $T \in \mathbf{T}(\mathbf{V})$. What is

$$\inf_{T\in\mathbf{T}(\mathbf{V})} \frac{\sum_{u,v\in\mathbf{V}} d_T(u,v)}{\sum_{u,v\in\mathbf{V}} |\overline{u,v}|} \ ?$$

Although we can consider the Euclidean metric extracted from $\mathbf{V}$ as we did in Problem 2, dominating tree metrics of it do not necessarily correspond to any spanning tree of V. In fact, if we apply the approaches for Problem 2 directly, the lack of balance guarantee in each partition can make the resulting pairwise distances arbitrary large.

State-of-the-Art

Embedding metrics into tree metrics was introduced in the context of *probabilistic embedding* by Alon et al., [7]. What follows was a series of notable work. Bartal [11] considered probabilistic embeddings and proved that any metric can be probabilistically approximated by tree metrics with expected maximum distortion $O(\log^2 n)$. This result was later improved to $O(\log n \log\log n)$ [12]. Bartal also observed that any probabilistic embedding into a tree has distortion at least $\Omega(\log n)$. This gap was closed by Fakcharoenphol et al., [29], who showed that for any metric, there exists tree metrics with $O(\log n)$ distortion.

Problem 4 (Tree Metric Embeddings of Low Weighted Average Stretch problem)**.** Given a metric $\mathbf{M} = (\mathbf{V}, d)$ and a weight function $w : \mathbf{V} \times \mathbf{V} \to \mathcal{R}^+$, find a dominating tree metric T of $\mathcal{M}$ such that $\sum_{u,v\in\mathbf{V}} w_{uv} \cdot d_T(u,v) \leq \alpha \sum_{u,v\in\mathbf{V}} w_{uv} \cdot d(u,v)$.

As Charikar et al., [18] showed by linear program duality that computing probabilistic embeddings of a given metric and Problem 4 described above are in fact dual problems, the series of work led by Bartal [11, 12, 27, 29] has provided improved approximation results for a large set of problems, including buy-at-bulk network design problem, vehicle routing problem, metric labeling problem, group Steiner tree problem, minimum cost communication network problem. Refer to [12, 18] for more detail and applications.

Kleinberg, Slivkins, and Wexler [57] initiated the study of *partial embedding* and *scaling distortion*, which can be regarded as embedding with relaxed guarantees. In a series of following work, Abraham et al., [1,4] proved that any finite metric embeds probabilistically in a tree metric such that the distortion of $(1-\epsilon)$ portion of the pairs is bounded by $O(\log \frac{1}{\epsilon})$, for any $0 < \epsilon < 1$. They also observed a lower bound of $\Omega(\sqrt{1/\epsilon})$, which is closed by Abraham et al., in [2].

In particular, Abraham et al., [4] showed that any metric can be probabilistically embedded into a tree metric such that the ratio between the expected sum of pairwise distances is $O(\log \Phi)$, where Φ is the effective aspect ratio of given distribution. This provides an upper-bound to Problem 2 we considered. However, the guarantee they provided is loose due to the constant inherited from the guarantee on scaling distortion. See also [1–3]. Rabinovich [74] showed that it is possible to embed certain special graph metrics into real line such that distance-weighted average stretch is bounded by a constant.

For approximating arbitrary graph metrics by their spanning trees, a simple $\Omega(n)$ lower bound in terms of maximum stretch is known for n-cycles [75]. Alon, Karp, Peleg, and West [7] considered a distribution over spanning trees and proved an upper bound of $2^{O(\sqrt{\log n \log\log n})}$ on the expected distortion. Elkin et al., [27] showed how a spanning tree with $O(\log^2 n \log\log n)$ average stretch (over the set of edges) can be computed in polynomial time. In terms of average pairwise stretch, Abraham et al., [2] showed the existence of a spanning tree such that, for any $0 < \epsilon < 1$, the distortion of an $(1-\epsilon)$ fraction of the pairs is bounded by $O(\sqrt{1/\epsilon})$. Note that this implies an $O(1)$ average pairwise stretch.Smid [85] gave a simpler proof for this result when the metric is Euclidean.

In terms of sum of pairwise distances in graphs (routing cost), Johnson et al., [51] showed that computing the spanning tree of minimum routing cost is **NP**-hard. Polynomial time approximations as well as approximation schemes have been proposed by Wong [88] and Wu et al., [92]. Despite the efforts devoted, however, no general guarantees have been made on the ratio between the routing cost of the optimal spanning tree and that of the underlying graphs. Other reasonable variations have been considered as well, i.e., sum-requirement routing trees problem, product-requirement routing trees problem, and multi-sources routing trees problem [89–91].

3.2 Cost-Efficiency Maximization

Realistic network construction problems are often characterized by complex constraints and multiple, possibly conflicting, objectives, and are therefore formulated within the framework of *Multi-Criteria Optimizations* [19].

There are several ways to model the optimality in the presence of multiple objective, for example, *local optimality* and *aggregate optimality*. Let Π be a multi-objective optimization problem, $\mathbf{I}$ be the set of the instances of Π, and $f_i \colon \mathbf{I} \to \mathbf{R}$, $1 \leq i \leq k$, be the set of objective functions of Π to be maximized. An instance $x \in \mathbf{I}$ is called *local optimal* if there is no other instance $y \in \mathbf{I}$ such that $f_j(y) > f_j(x)$ for at least one index $1 \leq j \leq k$ while $f_i(y) \geq f_i(x)$ for all $1 \leq i \leq k$. In other words, x is local optimal if we cannot improve a single objective without diminishing at least one of the other objectives.

While the local optimality seems to capture the notion of best solutions intuitively, there may be many instances which are locally optimal for a multi-objective optimization problem. One common way to further measuring the quality is to combine the objectives into a single aggregated objective, followed by optimizing it as a single-criterion objective. Typical aggregate functions include weighted sum or weighted extremums, that is, we try to maximize

$$F(x) := \sum_{i=1}^{k} w_i \cdot f_i(x) \quad \text{or} \quad G(x) := \max\{w_i \cdot f_i(x) \mid 1 \leq i \leq k\} \,,$$

where $w_i \in \mathbf{R}$ are constants. These weighted aggregate functions, however, must be guided in that the decision maker has to supply a set of suitable weights at the risk of arbitrariness.

In this part, a framework of *bi-criteria* network construction problems is considered. This is motivated by a scenario of investment where one objective, the profit, is to be maximized while the other, the expense, is to be minimized. In addition, a target upper bound on the cost and a lower bound on the profit are given. In other words, we are given a budget limit and a target profit. There two objectives are aggregated by their ratio. This features two main advantages over other aggregate functions. First, we do not need to supply any weights, and if we did, it would not alter the notion of optimality. Second, any optimal solution with respect to the ratio is also locally optimal with respect to these two objectives. In economics, this ratio, which is termed as return on investment, is a common measure for assessing the quality of investments.

Problem Definition

The framework is defined as follows. Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a graph, which will be referred to as the *host*. Throughout the remaining content I will use n and m to represent the cardinality of $\mathbf{V}$ and $\mathbf{E}$, respectively. In addition, a weight function $w : E \to \mathbf{Z}$ representing the profits and a length function $\ell : E \to \mathbf{N}$ representing the costs of the edges are given as two parameters as well. As a shorthand I will also denote $w(e)$ and $\ell(e)$ by w_e and ℓ_e, respectively. For any subgraph $\mathbf{H} \subseteq \mathbf{G}$, which I will refer to as a *pattern* of $\mathbf{G}$, define

$$w(\mathbf{H}) := \sum_{e \in \mathbf{E}(\mathbf{H})} w_e \quad \text{and} \quad \ell(\mathbf{H}) := \sum_{e \in \mathbf{E}(\mathbf{H})} \ell_e$$

as the total weight and total length of the edge set of a subgraph H, respectively.

Definition 3.1 (*Viable Patterns*)**.** Given two integers $\mathcal{W} \in \mathbf{Z}$ and $\mathcal{L} \in \mathbf{N}$, a pattern $\mathbf{H}$ of a host $\mathbf{G}$ is said to be $\mathcal{W}$*-viable* if

$$w(\mathbf{H}) \geq \mathcal{W}.$$

Similarly, it is $\mathcal{L}$*-viable* if

$$\ell(\mathbf{H}) \leq \mathcal{L}.$$

$\mathbf{H}$ is called $(\mathcal{W}, \mathcal{L})$*-viable* if it is both $\mathcal{W}$-viable and $\mathcal{L}$-viable.

Definition 3.2 (*Cost-efficiency*)**.** For any (*pattern*) $\mathbf{H}$, the cost-efficiency of $\mathbf{H}$ is defined as

$$\varrho(\mathbf{H}) = \frac{w(\mathbf{H})}{\ell(\mathbf{H})}.$$

For a given host graph $\mathbf{G}$ with two target integers $\mathcal{W}$ and $\mathcal{L}$, the ultimate objective of the addressed framework is to find a connected $(\mathcal{W}, \mathcal{L})$-viable pattern $\mathbf{H}$ with maximum cost-efficiency. In particular, we consider the following problem in Chapter §10 as an opening of the stage curtain.

Problem 5 (Bi-constrained Maximum Cost-Efficiency Pattern problem)**.** Given a host graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, a weight function $w\colon \mathbf{E} \to \mathbf{Z}$, a length function $\ell\colon \mathbf{E} \to \mathbf{N}$, and target integers $\mathcal{W} \in \mathbf{Z}$ and $\mathcal{L} \in \mathbf{N}$, the bi-constrained maximum cost-efficiency pattern problem is to find a connected $(\mathcal{W}, \mathcal{L})$-viable pattern $\mathbf{H}$ of $\mathbf{G}$ which has the maximum cost-efficiency among all possible patterns.

Problem 5 captures the essential idea of the considered framework. In most situations, however, we will not have a strict budget limits when constructing the network. Instead, there is usually a flexibility to a certain extent to deduce the overall profit in exchange of more budget. For instance, it may be possible to spend more than the budget limit by loaning additional money at the cost of some interest. The following definition provides a possible way to model this concept.

Definition 3.3 (*Penalized Cost-Efficiency*)**.** For a pattern $\mathbf{H}$ of $\mathbf{G}$ and target values $\mathcal{W}$ and $\mathcal{L}$, the $\mathcal{L}$-*deviation* of $\mathbf{H}$ with respect to the length target $\mathcal{L}$ is defined as

$$\delta(\mathbf{H}) := \max\{0, \ell(\mathbf{H}) - \mathcal{L}\},$$

and the corresponding *penalized cost-efficiency* is defined as

$$\widetilde{\varrho}(\mathbf{H}) := \frac{w(\mathbf{H})}{\ell(\mathbf{H}) + c \cdot \delta(\mathbf{H})},$$

where c is some non-negative constant.

Note that, the defined penalized cost-efficiency provides a simplest way of modelling the *penalization*. Depending on the situation, one can also define penalization of other forms. Provided the concept of penalization, the following problem is further considered.

Problem 6 (Relaxed Maximum Cost-Efficiency Pattern problem)**.** Given a host graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, a weight function $w\colon \mathbf{E} \to \mathbf{Z}$, a length function $\ell\colon \mathbf{E} \to \mathbf{N}$, and target values $\mathcal{W} \in \mathbf{Z}$ and $\mathcal{L} \in \mathbf{N}$, the relaxed maximum cost-efficiency pattern problem is to find a connected $\mathcal{W}$-viable pattern $\mathbf{H}$ of $\mathbf{G}$ which has the maximum *penalized cost-efficiency* among all possible patterns.

The requirement of a connected pattern provides a structural constrain of the simplest form. In the last of this dissertation, we consider the *Steiner Constraints* as a further step. In particular, in Chapter §11, the following problem is considered.

Problem 7 (Maximum Cost-Efficiency Steiner Pattern problem)**.** Given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with a weight function $w\colon \mathbf{E} \to \mathbf{Z}$ and a length function $\ell\colon \mathbf{E} \to \mathbf{N}$, and a set of terminals $\mathbf{S} \subseteq \mathbf{V}$, the maximum cost-efficiency Steiner pattern problem is to find a connected pattern $\mathbf{H}$ of $\mathbf{G}$ such that $\mathbf{S} \subseteq \mathbf{V}$ and $\varrho(\mathbf{H})$ is maximized among all possible patterns containing the terminals $\mathbf{S}$.

State-of-the-Art

An overview of recent developments in *multi-objective optimization* is given in [19]. Bálint [10] proves inapproximability for bi-objective network optimization problems, where the task is to minimize the diameter of a spanning subgraph with respect to a given length on the edges, subject to a limited budget on the total cost of the edges. Marathe et al. [68] study bi-objective network design problems with two minimization objectives. Given a limited budget on the first, they provide a *polynomial time approximation scheme* for minimizing the second objective among a set of feasible graphs. The considered objectives include total edge weight, diameter and maximum degree.

The study of dense segments in bi-weighted sequences arises from the investigation of non-uniformity of nucleotide composition with genomic sequences [49, 67] and has received considerable attention in bio-informatics. For this problem, we are given a sequence of pairs (a_i, b_i) and we wish to find a subsequence I with length bounded by $\mathcal{L}_{min} \leq \sum_{i \in I} b_i \leq \mathcal{L}_{max}$ that maximizes the density

$$\frac{\sum_{i \in I} a_i}{\sum_{i \in I} b_i},$$

where $\mathcal{L}_{min}$ and $\mathcal{L}_{max}$ are two given constants. For uniform lengths, Lin et al. [64] give an $O(n \log \mathcal{L}_{min})$ algorithm, which is improved to $O(n)$ by Goldwasser et al. [38]. A linear time algorithm for the non-uniform case is given by Chung and Lu [21]. Lee et al. [61] show how to select a subsequence whose density is closest to a given density δ in $O(n \log^2 n)$ time. Without the upper bound on the length B they present an optimal $O(n \log n)$-time algorithm.

Subsequently, this problem has been generalized to graphs. Previous work on this problem focuses mostly on the cases where the host is a tree subject to the two-sided constraint on the length of the solution. Hsieh et al. [47, 48] show that a maximum density path in a tree subject to lower and upper length bounds can be computed in time $O(\mathcal{L}_{max} n)$ and that it is NP-hard to find a maximum density subtree in a tree, for which they also presented an $O(\mathcal{L}_{max}^2 n)$ time algorithm. Wu et al. [93, 94] improve on this by presenting an optimal algorithm for computing a maximum density path in a tree in time $O(n \log n)$ in the presence of both a lower and upper length bounds. They also give an $O(n \log^2 n)$ algorithm for finding a *heaviest path* in a tree in the presence of length constraints [94], which is improved to $O(n \log n)$ by Liu and Chao [65] .

Problems involving Steiner constraints have been widely studied in computer science for a long time. For instance, it is known that the Steiner tree problem is **NP**-hard [37] and can be approximated within a factor of 1.55 [80]. When parameterized by the number of terminals, this problem is *fixed-parameter tractable* [26], when parameterized by the number of non-terminals in the solution it is $\mathbf{W}[\mathbf{2}]$-hard. The latter result is attributed to Bodlaender and can be found in [66]. For the special case that the set of terminals contains all vertices of the graph, Chandrasekaran [17] shows that a spanning tree with maximum density can be computed in polynomial time.

3.3 Summary of the Content to Appear

In Chapter §8, a direct approach to tackle Problem 2 as well as a provably small upper-bound is presented. Specifically, we adopt the notion of *hierarchically well-separated trees* (HSTs), introduced by Bartal [12] and Fakcharoenphol [29], and show that, for any given metric $\mathbf{M}$, there exists a 2-HST, $\mathbf{M}'$, such that the distance-weighted average stretch of $\mathbf{M}'$ is bounded by 14.24. The main ingredient of this result is a special point-set decomposition which relates two seemingly-unrelated quantities, namely, the diameter of the point set and the sum of pairwise distances between two separated subsets.

Bi-constrained Maximum Cost-Efficiency Pattern Problem

G	H	Constr.	Results	Reference
$tw = 2$	path	bi-constr.	NP-hard	Thm. 10.1
tree	path	bi-constr.	$\mathcal{O}(n \log^3 n)$	Thm. 10.3
tree +k edges	path	bi-constr	$\mathcal{O}(2^k k^2 n \log^2 n + n \log^3 n)$	Thm. 10.5
$tw = k$	minor-closed	bi-constr	$2^{\mathcal{O}(k^2 + k \log N + N)} \lvert\mathcal{F}\rvert\, Ln$	Thm. 10.6
$tw = k$	minor-closed	relaxed	$2^{\mathcal{O}(k^2 + k \log N + N)} \lvert\mathcal{F}\rvert\, m/\varepsilon^2 \log B$	Cor. 10.7

Maximum Cost-Efficiency Steiner Subgraph

G	H	Constr.	Results	Reference
$\star$	matching	V	$\mathcal{O}((m + n \log n) n \log (nM))$	Cor. 11.3
tree	tree with k leaves	$\lvert S\rvert \geq 1$	$\mathcal{O}(k^2 n \log (nM))$	Thm. 11.4
$\star$	path	$\lvert S\rvert = 1$	NP-hard, $\notin$ APX	Thm. 11.5
$\star$	$\star$, $\lvert V(H)\rvert \leq k$	$\lvert S\rvert = 1$	W[1]-hard	Thm. 11.7
planar	$\star$, $\lvert V(H)\rvert \leq k$	$\lvert S\rvert = 1$	FPT	Thm. 11.8
$\star$	path, $\lvert V(H)\rvert \leq k$	$\lvert S\rvert \geq 1$	$\mathcal{O}((2^{k-s} m + 3^{k-s}) s^2 \log (nM))$	Thm. 11.9
$\star$	tree	$\lvert S\rvert \geq 1$	NP-hard	Thm. 11.10

Table 3.1: Summary of the results to appear for the cost-efficiency maximization problem. The symbol $\star$ denotes an arbitrary graph.

If we do not require HSTs, it is also possible to apply our technique and construct the so-called *ultra-metrics*, which is introduced by Abraham [2] and Bartal [13], with a similar stretch, 3.56. This provides a better and explicit guarantee than that provided in [4] (from ≥ 64). For the negative side, we show that there exist metrics for which no dominating tree metrics can preserve the sum of pairwise distances to a factor better than 2. This shows that our result is within twice the best one can achieve.

As a side-product, we prove the existence of spanning trees with $O(d\sqrt{d})$ distance-weighted average stretch for any point set in Euclidean space $\mathcal{R}^d$. To this end, the point-set cutting lemma is used to decompose the points recursively. In order to guarantee a constant blow-up in the diameter of the spanning tree, however, instead of allowing arbitrary cuts, we show that it is always possible to make a balanced decomposition such that the diameters of the partitioned sets stay balanced. Our result provides a good guarantee when the dimension of the given Euclidean graph is low, which is true for most communication network. Although it is possible to apply the framework of [2,3] to obtain a spanning tree of constant distance-weighted average stretch, the constant hidden inside is huge ($> 10^5$) that makes it practically less useful. Both of the aforementioned proofs are constructive.

In Chapter §10, we discuss the Bi-constrained Maximum Cost-Efficiency Pattern problem. First, in terms of problem complexity, we prove that this problem is **NP**-hard, even if the host has treewidth 2 and the pattern is a path. Then we show how a cost-efficient path in a tree can be computed efficiently. This algorithm is extended to graphs that can be turned into a tree by

removing k edges, which implies this problem is **FPT** with respect to the difference between the number of edges and the number of vertices. As a further step, we show how this problem can be solved when the host graph has bounded treewidth and the pattern to be computed is restricted to a given minor-closed family of graphs. This algorithm applies for the Relaxed Maximum Cost-Efficiency Pattern problem as well. Then, a general framework which leads to a fully-polynomial time approximation scheme for the relaxed maximum cost-efficiency pattern problem is presented, provided that algorithms whose running time is pseudo-polynomial in the maximum length is available.

In Chapter §11, we consider the maximum cost-efficiency problem under different structural constraints. First, by adopting a generic technique from Chandrasekaran [17], we show how the maximum cost-efficiency perfect matching problem can be solved efficiently. Then, a polynomial time algorithm is presented for finding a maximum cost-efficiency subtree with k leaves in a tree. Then, we switch the focus to Steiner constraints and proved that this problem is **NP**-hard and cannot be approximated to any constant factor unless $\mathbf{P} = \mathbf{NP}$, even if the pattern is a path and the terminal set $\mathbf{S}S$ contains only one vertex. Furthermore, when parameterized by the number of vertices of the pattern, the maximum cost-efficiency pattern problem is proven to be $\mathbf{W}[\mathbf{1}]$-hard. In contrast, an fixed-parameter tractable algorithm for planar graphs is provided. Then, the problem of computing a maximum cost-efficiency path is proven to be fixed-parameter tractable when parameterized by the number of vertices on the path in general graphs. However, finding a maximum cost-efficiency Steiner tree is then proved to be **NP**-hard. Table 3.1 provides an overall summary for the results obtained for these problems.

Part II

Capacitated Domination Problem

Chapter 4

Approximation Algorithms for General Graphs

This chapter presents approximation algorithms for the capacitated domination problem on general graphs. Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be the input graph, whereas $n = |\mathbf{V}|$ is the number of vertices and $\Delta^* = deg[\mathbf{G}]$ is the maximum closed degree. Depending on the way how the approximation ratio is bounded, the results we presented are two folds.

(1) In terms of the number of vertices, logarithmic approximation algorithms with respect to different demand models are presented. In particular, for inseparable demand model, a $(\ln n)$-approximation in §4.1 is provided. For separable demand model, a $(4 \ln n + 2)$-approximation in §4.2 is presented. Furthermore, when the weight of each vertex is identical, we provide an improved approach in §4.3 which achieves an approximation ratio of $(2 \ln n + 1)$. This is achieved by properly determining the greedy choice. The idea used for inseparable demand model is relatively conceivable, nevertheless, it provides a hint towards separable demands for which a proper choice is less obvious and requires efforts to bound the solution quality.

(2) In terms of maximum degree of the input graph, we present a Δ^*-approximation algorithm for separable demand model in §4.4. The idea is to apply a sophisticated primal-dual fitting and charging argument on a linear program for this problem.

4.1 $\ln n$-Approximation for Inseparable Demand

Let $\mathbf{U}$ be the set of vertices which have not yet been served (dominated). Initially, we have $\mathbf{U} = \mathbf{V}$. For each vertex $u \in \mathbf{V}$, let $\mathbf{N}_{ud}[u] = \mathbf{U} \cap \mathbf{N}[u]$ be the set of undominated vertices in the closed neighborhood of u.

In each iteration, the algorithm chooses a vertex of the greatest efficiency from $\mathbf{V}$, where the efficiency of a vertex, say u, is defined by the largest effectiveness-cost ratio, which is number of vertices to be dominated by u over the total cost required by this assignment, among all possible demand assignments from $\mathbf{N}_{ud}[u]$ to u. To be precise, let $v_{u,1}, v_{u,2}, \ldots, v_{u,|\mathbf{N}_{ud}[u]|}$ denote the undominated neighbors of u, $\mathbf{N}_{ud}[u]$, sorted in non-descending order of their demands. The efficiency of u is defined to be

$$\delta(u) = \max_{1 \le i \le |\mathbf{N}_{ud}[u]|} \frac{i}{w(u) \cdot x_u(i)}, \quad \text{where} \quad x_u(i) = \left\lceil \frac{\sum_{1 \le j \le i} d(v_{u,j})}{c(u)} \right\rceil$$

is the number of copies of u necessary to dominate $v_{u,1}, v_{u,2}, \ldots, v_{u,i}$. $\delta(u)$ is defined to be zero if $\mathbf{N}_{ud}[u]$ is empty. Let u_0 be the vertex of maximum efficiency, and $\delta^{-1}(u_0)$ denote the

ALGORITHM *Inseparable-Log-Approx*

1: $\mathbf{U} \longleftarrow \mathbf{V}$
2: **while** $\mathbf{U} \neq \phi$ **do**
3: Pick a vertex from $\mathbf{V}$ with the greatest efficiency, say u.
4: Assign the demand of $\{v_{u,1}, v_{u,2}, \ldots, v_{u,\delta^{-1}(u)}\}$ to u.
5: $\mathbf{U} \longleftarrow \mathbf{U} \backslash \{v_{u,1}, v_{u,2}, \ldots, v_{u,\delta^{-1}(u)}\}$.
6: **end while**
7: return the demand assignment function as the solution.

Figure 4.1: The high-level description of the $(\ln n)$-approximation for the inseparable demand model.

corresponding index such that the ratio

$$\frac{i}{w(u_0) \cdot x_{u_0}(i)}$$

is maximized. The algorithm removes the set of vertices to be dominated, $v_{u_0,1}, v_{u_0,2}, \ldots, v_{u_0,\delta^{-1}(u_0)}$, from $\mathbf{U}$ and assigns their demands to u_0. This process continues until $\mathbf{U} = \phi$. A high-level description of this algorithm is presented in Fig. 4.1.

Since the algorithm only removes vertices from $\mathbf{U}$ when their demand is assigned, it always produces a feasible demand assignment function. In the following, we argue that this assignment function is also a $(\ln n)$-approximation.

For each iteration, say, j, let $\mathcal{O}pt(j)$ denote the cost of the optimal demand assignment function for the remaining problem instance. Clearly, we have $\mathcal{O}pt(j) \leq \mathcal{O}pt$, where $\mathcal{O}pt$ is the cost of the optimal demand assignment function for the original problem instance. Let the cardinality of $\mathbf{U}$ at the beginning of iteration j be n_j, and let $k_j = n_j - n_{j+1}$ be the number of vertices that are newly dominated in iteration j.

Denote by $\mathcal{S}(j)$ the cost we spend in iteration j. Assume that the algorithm repeats for m iterations. We have the following lemma.

Lemma 4.1. *For each j, $1 \leq j \leq m$, we have*

$$\mathcal{S}(j) \leq \frac{k_j}{n_j} \cdot \mathcal{O}pt(j)$$

Proof. Since we always choose the vertex with the maximum efficiency, this efficiency is no less than the efficiency of each vertex chosen in $\mathcal{O}pt(j)$, and therefore no less than any weighted average of them, including $n_j/\mathcal{O}pt(j)$. Therefore we have

$$\frac{k_j}{\mathcal{S}(j)} \geq \frac{n_j}{\mathcal{O}pt(j)},$$

and the lemma follows. □

Theorem 4.2. *Algorithm* **Inseparable-Log-Approx** *computes a $(\ln n)$-approximation for the capacitated domination problem with inseparable demands in $O(n^3)$ time, where n is the number of vertices of the input graph.*

Proof. It suffices to prove that this algorithm produces a logarithmic approximation. Take the summation over each $\mathcal{S}(j)$ and observe that $n_{j+1} = n_j - k_j$, we have

$$\sum_{1\le j\le m} \mathcal{S}(j) \le \sum_{1\le j\le m} \frac{k_j}{n_j}\cdot \mathcal{O}pt(j) \le \left(\sum_{1\le j\le n} \frac{1}{j}\right)\cdot \mathcal{O}pt \le \ln n\cdot \mathcal{O}pt,$$

where the second inequality follows from the fact that

$$\frac{k_j}{n_j} \le \frac{1}{n_j} + \frac{1}{n_j-1} + \frac{1}{n_j-2} + \cdots + \frac{1}{n_j-k_j+1} = \sum_{n_{j+1}<i\le n_j} \frac{1}{i}.$$

To see that the time complexity is $O(n^3)$, notice that it requires $O(n)$ time to compute a most efficient move for each vertex, which leads to an $O(n^2)$ computation for the most efficient choice in each iteration. The number of iterations is upper bounded by $O(n)$ since at least one vertex is satisfied in each iteration. □

4.2 $(4\ln n + 2)$-Approximation for Separable Demand

This section shows how a $(4\ln n + 2)$-approximation can be computed when the demand is separable. As the demand may be partially assigned during the algorithm, for each vertex $u \in \mathbf{V}$, we denote by $rd(u)$ the amount of demand of u that has not yet been served. For convenience we also refer to this quantity, $rd(u)$, as *residue demand* of u, and to the remaining fraction, $rd(u)/d(u)$, as the *effectiveness* of u. Initially $rd(u)$ is set to be $d(u)$, and will be updated accordingly when a fraction of the residue demand is assigned. The vertex u is said to be dominated when $rd(u) = 0$.

In each iteration, the algorithm performs two stages of greedy choices. First, the algorithm chooses the vertex of the most efficiency from $\mathbf{V}$, where the efficiency is defined in a similar fashion as in the previous section with some modification due to the separability of the demand.

For each vertex $u \in \mathbf{V}$, let $\mathbf{N}_{ud}[u] = \{v_{u,1}, v_{u,2}, \ldots, v_{u,|\mathbf{N}_{ud}[u]|}\}$ denote the set of undominated neighbors of u, sorted in non-descending order with respect to their demands. Let j_u, $0 \le j_u \le |\mathbf{N}_{ud}[u]|$, be the largest integer such that $c(u) \ge \sum_{i=1}^{j_u} rd(v_{u,i})$. Literally, we choose the largest index j_u such that the residue demand of the first j_u vertices in the sorted list could be served by one single copy of u. Let

$$\mathcal{X}(u) = \sum_{i=1}^{j_u} \frac{rd(v_{u,i})}{d(v_{u,i})}$$

be the corresponding sum of effectiveness. In addition, to effectively use the remaining capacity provided by this single copy of u, we let

$$Y(u) = \frac{c(u) - \sum_{i=1}^{j_u} rd(v_{u,i})}{d(v_{u,j_u+1})}$$

if $j_u < |\mathbf{N}_{ud}[u]|$ and $Y(u) = 0$ otherwise. Since we select the vertices from the sorted order of the their demands, one can easily verify that this always results in the maximum effectiveness among all possible combinations. The efficiency of the vertex u is defined to be

$$\frac{\mathcal{X}(u) + Y(u)}{w(u)}.$$

Second, the algorithm maintains for each vertex $u \in \mathbf{V}$ a subset of vertices, denoted by $\mathbf{P}(u)$, which consists of vertices that have served the demand of u before u is dominated. In other words, for each $v \in \mathbf{P}(u)$ we have a non-zero demand assignment from u to v. During the

ALGORITHM *Separable-Log-Approx*

1: $rd(u) \longleftarrow d(u)$, and $\mathbf{P}(u) \longleftarrow \phi$ for each $u \in \mathbf{V}$.
2: **while** there exist vertices with non-zero residue demand **do**
3: // 1^{st} **greedy choice**
4: Pick a vertex in $\mathbf{V}$ with the largest efficiency, say u.
5: **if** j_u equals 0 **then**
6: Assign this amount $c(u) \cdot \left\lfloor \frac{rd(v_{u,1})}{c(u)} \right\rfloor$ of residue demand of $v_{u,1}$ to u.
7: $\mathbf{P}(v_{u,1}) \longleftarrow \{u\}$
8: **else**
9: Assign the residue demands of the vertices in $\{v_{u,1}, v_{u,2}, \ldots, v_{u,j_u}\}$ to u.
10: **if** $j_u < |\mathbf{N}_{ud}[u]|$ **then**
11: Assign this amount $c(u) - \sum_{i=1}^{j_u} rd(v_{u,i})$ of residue demand from v_{u,j_u+1} to u.
12: $\mathbf{P}(v_{u,j_u+1}) \longleftarrow \mathbf{P}(v_{u,j_u+1}) \cup \{u\}$
13: **end if**
14: **end if**
15:
16: // 2^{nd} **greedy choice**
17: **if** there is a vertex u with $0 < rd(u) < \frac{1}{2} \cdot d(u)$ **then**
18: Satisfy u by doubling the demand assignment of u to vertices in $\mathbf{P}(u)$.
19: **end if**
20: **end while**
21: return the demand assignment function as the output.

Figure 4.2: The $(4 \ln n + 2)$-approximation for the separable demand model.

iterations, whenever there exists a vertex u whose residue demand falls below half of its original demand, i.e., $0 < rd(u) < \frac{1}{2} \cdot d(u)$, after the first greedy choice, the algorithm immediately doubles the demand assignment of u to the vertices in $\mathbf{P}(u)$. Note that in this way, we can completely serve the demand of u since

$$\sum_{v \in \mathbf{P}(u)} f(u, v) > \frac{1}{2} \cdot d(u).$$

This procedure repeats until every vertex of the graph is dominated. A high-level description of this algorithm is presented in Figure 4.2.

Below we analyse the algorithm *Separable-Log-Approx*. First we argue that this algorithm always produces a feasible demand assignment function. We begin with the following lemma.

Lemma 4.3. *After each iteration, the residue demand of each unsatisfied vertex is at least half of its original demand.*

Proof. Clearly, this lemma holds in the beginning when the demand of each vertex is not yet assigned. For later stages, we argue that the algorithm properly maintains the set $\mathbf{P}(u)$ for each vertex $u \in \mathbf{V}$ such that in our second greedy choice, whenever there exists a vertex u with $0 < rd(u) < \frac{1}{2} \cdot d(u)$, it is always sufficient to double the demand assignment $f(u, v)$ for each $v \in \mathbf{P}(u)$. If $\mathbf{P}(u)$ is only modified under the condition $0 < j_v < |\mathbf{N}_{ud}[v]|$, (see also line 12 in Fig. 4.2), then $\mathbf{P}(u)$ contains exactly the set of vertices that have partially served u. Therefore we have $\sum_{v \in \mathbf{P}(u)} f(u, v) > \frac{1}{2} d(u)$, and it is sufficient to double the demand assignment in this case. If $\mathbf{P}(u)$ is reassigned under the condition $j_v = 0$ at some stage, then

we have $c(v) < rd(u) \leq d(u)$. Since we assign this amount

$$c(v) \cdot \left\lfloor \frac{rd(u)}{c(v)} \right\rfloor$$

of residue demand of u to v, this leaves at most half amount of the original residue demand of u, which is also no larger than $c(v)$. Therefore u will be immediately dominated by doubling this assignment in the same iteration. □

By the above description, we conclude that the algorithm produces a feasible demand assignment function. In the following we show that the demand assignment function is indeed a $(4 \ln n + 2)$-approximation. Let the cost incurred by the first greedy choice be $\mathcal{S}_1$ and the cost by the second choice be $\mathcal{S}_2$. To see that the solution achieves the desired approximation guarantee, notice that $\mathcal{S}_2$ is bounded from above by $\mathcal{S}_1$, for what we do in the second choice is merely to satisfy the residue demand of a vertex, if there exists one, by doubling its previous demand assignment.

It remains to bound the cost $\mathcal{S}_1$. For each iteration j, let u_j be the chosen vertex of the maximum efficiency,

$$n_j = \sum_{u \in \mathbf{V}} \frac{rd(u)}{d(u)}$$

be the sum of remaining effectiveness of each vertex at the beginning of this iteration, and $\mathcal{O}pt(j)$ be the cost of the corresponding optimal demand assignment function of this remaining problem instance. Denote by $\mathcal{S}_{1,j}$ the cost incurred by the first greedy choice in iteration j. Assume that the algorithm repeats for m iterations. We have the following lemma.

Lemma 4.4. *For each j, $1 \leq j \leq m$, we have*

$$\mathcal{S}_{1,j} \leq \frac{n_j - n_{j+1}}{n_j} \cdot \mathcal{O}pt(j),$$

where $n_j - n_{j+1}$ is the effectiveness covered by u_j in iteration j.

Proof. The optimality of our choice in each iteration is obvious since we consider the elements of $\mathbf{N}_{ud}[u]$ in sorted order according to their demands. Note that only in the case $c(u) < rd(v_{u,1})$, the algorithm could possibly take more than one copy. In this case the efficiency of our choice remains unchanged since the cost and the effectiveness covered by u grows by the same factor. Therefore the efficiency of our choice, $(n_j - n_{j+1})/\mathcal{S}_{1,j}$, is always no less than the efficiency of each chosen vertex in the optimal solution, and therefore no less than any of their weighted averages, including $n_j/\mathcal{O}pt(j)$. Therefore this lemma follows. □

By Lemma 4.4 above, we have

$$\sum_{j=1}^{m} \mathcal{S}_{1,j} \leq \sum_{j=1}^{m-1} \frac{n_j - n_{j+1}}{n_j} \cdot \mathcal{O}pt(j) + \frac{n_m}{n_m} \cdot \mathcal{O}pt(m) \leq \left(\sum_{j=1}^{m-1} \frac{\lceil n_j - n_{j+1} \rceil}{\lfloor n_j \rfloor} + 1 \right) \cdot \mathcal{O}pt,$$

where the second inequality follows form the fact that $\lfloor r \rfloor \leq r \leq \lceil r \rceil$ for any real number r and $\mathcal{O}pt(j) \leq \mathcal{O}pt$ for each $1 \leq j \leq m$.

Lemma 4.5. *We have $n_j - n_{j+1} \geq \frac{1}{2}$ for each $1 \leq j \leq m$.*

Proof. For iteration j, $1 \leq j \leq m$, let u_j be the chosen vertex of the maximum efficiency. Observe that $v_{u_j,1}$ will be satisfied after this iteration. By Lemma 4.3, we have

$$rd(v_{u_j,1}) \geq \frac{1}{2} d(v_{u_j,1}).$$

Therefore the effectiveness covered in each iteration is at least half. □

Lemma 4.6. *We have*

$$\sum_{j=1}^{m-1} \frac{\lceil n_j - n_{j+1} \rceil}{\lfloor n_j \rfloor} \leq 2 \cdot \ln n.$$

Proof. Note that we have $n_j \geq 1$ for all $j < m$, since, whenever $n_j < 1$, the remaining effectiveness will be covered in the same iteration according to Lemma 4.3 and Lemma 4.5. We will argue that each item of this series together constitutes at most two harmonic series.

By expanding the summand we have

$$\frac{\lceil n_j - n_{j+1} \rceil}{\lfloor n_j \rfloor} \leq \frac{1}{\lfloor n_j \rfloor} + \frac{1}{\lfloor n_j \rfloor - 1} + \ldots + \frac{1}{\lfloor n_j \rfloor - \lceil n_j - n_{j+1} \rceil + 1} \tag{4.1}$$

Since

$$\begin{aligned} \lfloor n_{j+1} \rfloor &= \lfloor n_j - (n_j - n_{j+1}) \rfloor \\ &\leq \lfloor n_j \rfloor - \lfloor n_j - n_{j+1} \rfloor \leq \lfloor n_j \rfloor - \lceil n_j - n_{j+1} \rceil + 1 \ , \end{aligned}$$

possible repetitions of the expanded items only occur at the first item and the last item of Eq. (4.1) if we expand each summand from the summation. By Lemma 4.5, the decrease between n_j and n_{j+1} is at least half. Therefore, each repeated item,

$$\frac{1}{\lfloor n_j \rfloor - \lceil n_j - n_{j+1} \rceil + 1},$$

will never occur more than twice in the expansion, and we can conclude that

$$\sum_{j=1}^{m-1} \frac{\lceil n_j - n_{j+1} \rceil}{\lfloor n_j \rfloor} \leq 2 \cdot \ln n.$$

□

Regarding the Running Time of this Algorithm. A naïve implementation of this algorithm will lead to a running time cubic in the number of vertices. However, by exploiting the property we assumed during the iterations that the undominated neighbors of each vertex are sorted in non-descending order according their demands, we can improve the running time of the algorithms to $O(n^2 \log n)$, which is also the time required to build the sorted list of the closed neighborhood for each vertex.

The idea is to maintain the efficiency of each vertex in a binary max-heap. In each iteration, we extract the vertex of the greatest efficiency from the heap, perform the demand assignments suggested by the most efficient move, and update the efficiencies of the vertices affected by each demand assignment.

To this end, for each vertex, we maintain a pointer to the vertex in its closed neighborhood that corresponds to the last item in the most efficient move. The pointer stored for each vertex will iterate over its closed neighborhood in sorted order at most once upon updates. Whenever the residue demand of a vertex, say u, gets assigned, we update the efficiencies as well as the most efficient moves of its closed neighborhood accordingly. To be more precise, for each $v \in \mathbf{N}[u]$, depending on the relative position of u and the vertex to which the pointer of v points in $\mathbf{N}[v]$, we have two cases. If u lies after the pointer, then no updates are required. Otherwise we iterate the pointer of v according to our predefined notion of efficiency.

Since at least one vertex will be dominated and at most one vertex will be partially assigned in each iteration, the number of partial assignment will be no more than n. For each demand assignment, the number of updates required is bounded by the cardinality of the closed neighborhood, which is $O(n)$. Therefore, the total number of updates is $O(n^2)$. Since the pointer

we maintained for each vertex is iterated over its closed neighborhood at most once, the time required for all the updates is also bounded by $O(n^2)$. We conclude our result in the following theorem.

Theorem 4.7. *Algorithm* **Separable-Log-Approx** *computes a* $(4 \ln n + 2)$*-approximation in* $O(n^2 \log n)$ *time, where* n *is the number of vertices of the input graph, for the capacitated domination problem with separable demand model.*

Proof. The feasibility of our algorithm follows from the above discussion. By Lemma 4.6, the cost of the demand assignment function returned by our algorithm is bounded by

$$\mathcal{S}_1 + \mathcal{S}_2 \leq 2 \cdot \mathcal{S}_1 = 2 \cdot \sum_{j=1}^{m} \mathcal{S}_{1,j} \leq 2 \cdot (2 \ln n + 1) \cdot \mathcal{O}pt.$$

Regarding the time complexity, building the sorted list for each vertex takes $O(n^2 \log n)$ time. In each iteration, it takes $O(\log n)$ time to extract and to maintain the heap property, provided that the efficiency of each vertex is updated. The total time required to perform the update is $O(n^2)$ from the above discussion. Therefore the overall time complexity is $O(n^2 \log n)$. □

4.3 $(2 \ln n + 1)$-Approximation for Separable Demand with Unit Cost

We show that, when the demand is separable and each vertex has uniform cost, then we can compute a $(2 \ln n + 1)$-approximation in polynomial time. To this end, we first make a greedy reduction on the problem instance by spending at most the cost of $\mathcal{O}pt$ such that it takes at most one copy to serve each remaining undominated vertex. Then we show that a $(2 \ln n)$-approximation can be computed for this reduced problem instance, based on the same framework provided in the last section.

For each $u \in \mathbf{V}$, let $g_u \in \mathbf{N}[u]$ be the vertex with the maximum capacity. First, for each $u \in V$, we assign this amount

$$c(g_u) \cdot \left\lfloor \frac{d(u)}{c(g_u)} \right\rfloor$$

of the demand of u to g_u. Let the cost of this assignment be $\mathcal{S}$, then we have the following lemma.

Lemma 4.8. *We have* $\mathcal{S} \leq \mathcal{O}pt$*, where* $\mathcal{O}pt$ *is the cost of the optimal demand assignment function.*

Proof. Notice that, when fractional multiplicities are allowed, an optimal demand assignment, denoted f^*, can be obtained by assigning all the demand of u to g_u, for each vertex u. Let

$$w_{frac}(f^*) = \sum_{u \in V} w(u) \cdot \frac{\sum_{v \in \mathbf{N}[u]} f^*(v, u)}{c(u)}$$

be the total cost incurred by f^*, allowing fractional multiplicities. Since $S \leq w_{frac}(f^*)$ and $w_{frac}(f^*) \leq \mathcal{O}pt$, the lemma follows. □

In the following, we will assume that $d(u) \leq c(g_u)$, for each $u \in \mathbf{V}$. The algorithm provided in Section §4.2 is slightly modified. In particular, for the second greedy choice, whenever $rd(u) < d(u)$ for some vertex $u \in \mathbf{V}$, we immediately assign the residue demand of u to g_u. A high-level description of this algorithm is presented in Fig. 4.3.

Lemma 4.9. *We have* $n_j - n_{j+1} \geq 1$ *for each* $1 \leq j \leq m$.

ALGORITHM *Unit-Weight-Separable-Log-Approx*

1: For each $u \in \mathbf{V}$, assign $c(g_u) \cdot \left\lfloor \frac{d(u)}{c(g_u)} \right\rfloor$ demands of u to g_u, where $g_u \in \mathbf{N}[u]$ has the maximum capacity.
2: Reset the demands of the instance by setting $d(u) \longleftarrow rd(u)$ for each $u \in \mathbf{V}$.
3: **while** there exist vertices with non-zero residue demand **do**
4: // 1^{st} **greedy choice**
5: Pick a vertex in $\mathbf{V}$ with the most efficiency, say u.
6: Assign the demands of the vertices in $\{v_{u,1}, v_{u,2}, \ldots, v_{u,j_u}\}$ to u.
7: **if** $j_u < |\mathbf{N}_{ud}[u]|$ **then**
8: Assign this amount $c(u) - \sum_{i=1}^{j_u} rd(v_{u,i})$ of the residue demand of v_{u,j_u+1} to u.
9: **end if**
10:
11: // 2^{nd} **greedy choice**
12: **if** there is a vertex u with $0 < rd(u) < d(u)$ **then**
13: Satisfy u by assigning the residue demand of u to g_u.
14: **end if**
15: **end while**
16: return the demand assignment function as the output.

Figure 4.3: The high-level description for the improved $(2 \ln n + 1)$-approximation for the separable demand model with unit weight.

Proof. Observe that in each iteration, at least one vertex is dominated and the residue demand of each vertex is either 0 or equal to its original demand. □

We conclude the result in the following theorem.

Theorem 4.10. *Algorithm* **Unit-Weight-Separable-Log-Approx** *computes a* $(2 \ln n + 1)$*-approximation in* $O(n^2 \log n)$ *time for the capacitated domination problem with separable demand model and unit weight, where* n *is the number of vertices in the input graph.*

Proof. We adopt the notation from the previous section. Clearly, $\mathcal{S}_2$ is bounded above by $\mathcal{S}_1$, as we always take one copy for the first greedy choice and at most one copy for the second greedy choice in each iteration. By Lemma 4.9 and the fact that n_j is integral for each $1 \leq j \leq m$, we have

$$\sum_{j=1}^{m} \mathcal{S}_{1,j} \leq \sum_{j=1}^{m} \frac{n_j - n_{j+1}}{n_j} \cdot \mathcal{O}pt(j) \leq \ln n \cdot \mathcal{O}pt,$$

and

$$\mathcal{S} + \mathcal{S}_1 + \mathcal{S}_2 \leq \mathcal{O}pt + 2 \cdot \sum_{j=1}^{m} \mathcal{S}_{1,j} \leq (2 \ln n + 1) \cdot \mathcal{O}pt.$$

□

4.4 Δ^*-approximation for Separable Demand

In the following, we present a primal-dual algorithm which gives a Δ^*-approximation for the capacitated domination problem with separable demand on general graphs, where Δ^* is the closed degree of the input graph. The algorithm is based on a sophisticated dual fitting and charging scheme.

4.4.1 Primal/Dual Linear Programs

We formulate this problem as an integer linear program (ILP), which is given below in (4.2). The first inequality ensures the feasibility of the demand assignment function f. In the second inequality, we model the multiplicity function x_f as defined. The third constraint, $d(v)x(u) - f(v,u) \geq 0$, which seems unnecessary in the problem formulation, is required to bound the integrality gap between the optimal solution of this ILP and that of its relaxation, in which we allow the variables $x(v_i)$ to take any non-negative real values. To see that this additional constraint does not alter the optimality of any optimal solution, we have the following lemma.

$$
\begin{aligned}
&\text{Minimize} \quad \sum_{u \in \mathbf{V}} w(u)x(u) && && (4.2)\\
&\text{subject to} \\
&\quad \sum_{v \in \mathbf{N}[u]} f(u,v) - d(u) \geq 0, && u \in \mathbf{V} \\
&\quad c(u)x(u) - \sum_{v \in \mathbf{N}[u]} f(v,u) \geq 0, && u \in \mathbf{V} \\
&\quad d(v)x(u) - f(v,u) \geq 0, && v \in \mathbf{N}[u], u \in \mathbf{V} \\
&\quad f(u,v) \geq 0, \;\; x(u) \in \mathbb{Z}^+ \cup \{0\}, && u, v \in \mathbf{V}
\end{aligned}
$$

Lemma 4.11. *Let f be an arbitrary optimal demand assignment function. We have $d(v) \cdot x_f(u) - f(v,u) \geq 0$ for all $u \in \mathbf{V}$ and $v \in \mathbf{N}[u]$.*

Proof. Without loss of generality, we may assume that $d(v) \geq f(v,u)$. For otherwise, we set $f(v,u)$ to be $d(v)$ and the resulting assignment would be feasible and the cost can only be better. If $x_f(u) = 0$, then we have $f(v,u) = 0$ by definition, and this inequality holds trivially. Otherwise, if $x_f(u) \geq 1$, then $d(v) \cdot x_f(u) - f(v,u) \geq f(v,u) \cdot (x_f(u) - 1) \geq 0$. □

However, without this constraint, the integrality gap can be arbitrarily large. This is illustrated by the following example. Let $\alpha > 1$ be an arbitrary constant, and $\mathbf{T}(\alpha)$ be an n-vertex star, where each vertex has unit demand and unit cost. The capacity of the central vertex is set to be n, which is sufficient to cover the demand of the entire graph, while the capacity of each of remaining $n-1$ petal vertices is set to be αn.

Lemma 4.12. *Without the additional constraint $d(v)x(u) - f(v,u) \geq 0$, the integrality gap of the ILP (4.2) on $\mathbf{T}(\alpha)$ is α, where $\alpha > 1$ is an arbitrary constant.*

Proof. The optimal dominating set consists of a single multiplicity of the central vertex with unit cost, while the optimal fractional solution is formed by spending $\frac{1}{\alpha n}$ multiplicity at a petal vertex for each unit demand from the vertices of this graph, making an overall cost of $\frac{1}{\alpha}$ and therefore an arbitrarily large integrality gap. □

Indeed, with the additional constraint applied, we can refrain from unreasonably assigning a small amount of demand to any vertex in any fractional solution. Take a petal vertex, say v, from $\mathbf{T}(\alpha)$ as example, given that $d(v) = 1$ and $f(v,v) = 1$, this constraint would force $x(v)$ to be at least 1, which prevents the aforementioned situation from being an optimal fractional solution.

The dual program of the relaxation of (4.2) is given below in (4.3). Note that, by the linear program duality, any feasible solution to (4.3) will serve as a lower bound to any feasible solution of (4.2).

For the remaining of this chapter, for any graph $\mathbf{G}$, we denote the optimal values to the integer linear program (4.2) and to its relaxation by $\mathcal{O}pt(\mathbf{G})$ and $\mathcal{O}pt_{frac}(\mathbf{G})$, respectively. Note that $\mathcal{O}pt_{frac}(\mathbf{G}) \leq \mathcal{O}pt(\mathbf{G})$.

$$\text{Maximize} \quad \sum_{u \in \mathbf{V}} d(u) y_u \tag{4.3}$$

subject to

$$c(u) z_u + \sum_{v \in \mathbf{N}[u]} d(v) g_{u,v} \leq w(u), \qquad u \in \mathbf{V}$$

$$y_u \leq z_v + g_{v,u}, \qquad v \in \mathbf{N}[u], \ \ u \in \mathbf{V}$$

$$y_u \geq 0, \ \ z_u \geq 0, \ \ g_{v,u} \geq 0, \qquad v \in \mathbf{N}[u], \ \ u \in \mathbf{V}$$

4.4.2 The Greedy Charging Algorithm

We describe an approach to obtaining a feasible solution to (4.3). The idea is to begin with a trivial solution with all the variables set to zero and then raise the variables to achieve a local maximum. During the process, the feasibility of the solution is maintained, and the set of vertices becomes partially served. When a local maximum of the variables y_u is reached, we also have a feasible demand assignment function as well. Then we bound the objective value of the solution by distributing the cost we spent to each unit demand we served. In the following, we describe the whole process in more detail.

During the process, we will maintain a vertex subset, $\mathbf{V}^\phi$, which contains the set of vertices with non-zero unassigned demand. For each $u \in \mathbf{V}$, let $d^\phi(u) = \sum_{v \in \mathbf{N}[u] \cap \mathbf{V}^\phi} d(v)$ denote the amount of unassigned demand from the closed neighbors of u. We distinguish between two cases. If $c(u) < d^\phi(u)$, then we say that u is heavily-loaded. Otherwise, u is lightly-loaded. During the process, some heavily-loaded vertices might turn into lightly-loaded due to the demand assignments of its closed neighbors. For each of these vertices, say v, we will maintain a vertex subset $\mathbf{D}^*(v)$, which contains the set of unassigned vertices in $\mathbf{N}[v] \cap \mathbf{V}^\phi$ when v is about to fall into lightly-loaded. For other vertices, $\mathbf{D}^*(v)$ is defined to be an empty set.

Initially, $\mathbf{V}^\phi \equiv \{u : u \in \mathbf{V}, d(u) \neq 0\}$ and all the dual variables are set to be zero. We increase the dual variable y_u simultaneously, for each $u \in \mathbf{V}^\phi$. To maintain the dual feasibility, as we increase y_u, we have to raise either z_v or $g_{v,u}$, for each $v \in \mathbf{N}[u]$. If v is heavily-loaded, then we raise z_v. Otherwise, we raise $g_{v,u}$. Note that, during this process, for each vertex u that has a closed neighbor in $\mathbf{V}^\phi$, the left-hand side of the inequality $c(u)z_u + \sum_{v \in \mathbf{N}[u]} d(v) g_{u,v} \leq w(u)$ is constantly raising. As soon as one of the inequalities $c(u)z_u + \sum_{v \in \mathbf{N}[u]} d(v) g_{u,v} \leq w(u)$ is met with equality (saturated) for some vertex $u \in \mathbf{V}$, we perform the following operations.

If u is lightly-loaded, we assign all the unassigned demand from $\mathbf{N}[u] \cap \mathbf{V}^\phi$ to u. In this case, there are still $c(u) - d^\phi(u)$ units of capacity free at u. We assign the unassigned demand from $\mathbf{D}^*(u)$, if there is any, to u until either all the demand from $\mathbf{D}^*(u)$ is assigned or all the free capacity in u is used. On the other hand, if u is heavily-loaded, we mark it as heavy and delay the demand assignment from its closed neighbors.

Then we set $\mathbf{Q}_u \equiv \mathbf{N}[u] \cap \mathbf{V}^\phi$ and remove $\mathbf{N}[u]$ from $\mathbf{V}^\phi$. Note that, due to the definition of d^ϕ, even when u is heavily-loaded, we still update $d^\phi(p)$ for each $p \in \mathbf{V}$ with $N[p] \cap \mathbf{N}[u] \neq \phi$, if needed, as if the demand was assigned. During the above operation, some heavily-loaded vertices might turn into lightly-loaded due to the demand assignments (or simply due to the update of d^ϕ). For each of these vertices, say v, we set $\mathbf{D}^*(v) \equiv \mathbf{N}[v] \cap (\mathbf{V}^\phi \cup \mathbf{Q}_u)$. Intuitively, $\mathbf{D}^*(v)$ contains the set of unassigned vertices from $\mathbf{N}[v] \cap \mathbf{V}^\phi$ when v is about to fall into lightly-loaded.

ALGORITHM Δ^*-*Approx-Greedy-Charging*

$\mathbf{V}^\phi \longleftarrow \{u : u \in \mathbf{V}, d(u) \neq 0\}$, $\mathbf{V}^* \longleftarrow \mathbf{V}$.
$d^\phi(u) \longleftarrow \sum_{v \in \mathbf{N}[u]} d(v)$, for each $u \in \mathbf{V}$.
$w^\phi(u) \longleftarrow w(u)$, for each $u \in \mathbf{V}$.
Let $\mathbf{Q}$ be a first-in-first-out queue.
while $\mathbf{V}^\phi \neq \phi$ **do**
  $r_v \longleftarrow w^\phi(v)/\min\{c(v), d^\phi(v)\}$, for each $v \in \mathbf{V}^*$.
  $u \longleftarrow arg\min\{r_v : v \in \mathbf{V}^*\}$. [$u$ is the next vertex to be saturated.]
  $w^\phi(v) \longleftarrow w^\phi(v) - w^\phi(u)$, for each $v \in \mathbf{V}^*$.

  if $d^\phi(u) \leq c(u)$ **then**
    Assign the demand from $\mathbf{N}[u] \cap \mathbf{V}^\phi$ to u.
    Assign $c(u) - d^\phi(u)$ amount of unassigned demand from $\mathbf{D}^*(u)$, if there is any, to u.
  else
    Attach u to the queue $\mathbf{Q}$ and mark u as *heavy*.
  end if
  Let $\mathcal{S}_u \longleftarrow \mathbf{N}[u] \cap \mathbf{V}^\phi \cup \{u\}$.
  Remove $\mathbf{N}[u]$ from $\mathbf{V}^\phi$ and update the corresponding $d^\phi(v)$ for $v \in \mathbf{V}$.
  For each $v \in \mathbf{V}^*$ such that $d^\phi(v) = 0$, remove v from $\mathbf{V}^*$.
  for all vertex v becomes lightly-loaded in this iteration **do**
    $\mathbf{D}^*(v) \longleftarrow \mathbf{N}[v] \cap (\mathbf{V}^\phi \cup \mathcal{S}_u)$.
  end for
end while
while $\mathbf{Q} \neq \phi$ **do**
  Extract a vertex from the head of $\mathbf{Q}$, say u.
  Assign the unassigned demand from $\mathbf{N}[u]$ to u.
end while

Figure 4.4: The high-level pseudo-code for the primal-dual algorithm.

This process continues until $\mathbf{V}^\phi = \phi$. For those vertices which are marked as heavy, we iterate over them according to their chronological order of being saturated and assign at this moment all the remaining unassigned demand from their closed neighbors to them. A high-level description of this algorithm is given in Fig. 4.4.

Let $f^* : \mathbf{V} \times \mathbf{V} \to R^+ \cup \{0\}$ denote the resulting demand assignment function, and $x^* : \mathbf{V} \to Z^+ \cup \{0\}$ denotes the corresponding multiplicity function. The following lemma bounds the cost of the solution produced by our algorithm.

Lemma 4.13. *We can distribute the total cost of the demand assignment function f^*, which is $w(f^*) = \sum_{u \in \mathbf{V}} w(u) \cdot x^*(u)$, to each unit of the demand in $\mathbf{G}$ such that each unit demand, say, from vertex u, gets a cost of at most $deg[u] \cdot y_u$. In other words, we have*

$$w(f^*) \leq \sum_{u \in \mathbf{V}} d(u) \cdot deg[u] y_u.$$

Proof. Let $u \in \mathbf{V}$ be a vertex with $x^*(u) > 0$. We consider two cases.

(1) If u has been marked as heavy, then by our scheme, we have $g_{u,v} = 0$ and $y_v = z_u$ for all $v \in \mathbf{N}[u]$. Therefore $w(u) = c(u) \cdot z_u = c(u) \cdot y_v$, and for each multiplicity of u, we need $c(u)$

units of demand from $\mathbf{N}[u]$. If $x^*(u) > 1$, then at least $c(u) \cdot (x^*(u) - 1)$ units of demand are assigned to u, and by distributing the cost to them, each unit of demand gets charged at most twice. If $x^*(u) = 1$, then we charge the cost to any $c(u)$ units of demand that are counted in $d^\phi(u)$ when u is saturated. Since u is a heavily-loaded vertex, $d^\phi(u) > c(u)$ and there will be sufficient amount of demand to charge.

(2) If u is lightly-loaded, then $x^*(u) = 1$ and we have two subcases.

- **Case 2a.** If $\sum_{v \in \mathbf{N}[u]} d(v) \leq c(u)$, then u is lightly-loaded in the beginning and we have $z_u = 0$ and $y_v = g_{u,v}$ for each $v \in \mathbf{N}[u]$, which implies $w(u) = \sum_{v \in \mathbf{N}[u]} d(v) \cdot g_{u,v} = \sum_{v \in \mathbf{N}[u]} d(v) \cdot y_v$. The cost $w(u)$ of u can be distributed to all the demand from its closed neighbors, each unit demand, say from vertex $v \in \mathbf{N}[u]$, gets a charge of y_v.

- **Case 2b.** If $\sum_{v \in \mathbf{N}[u]} d(v) > c(u)$, then u is heavily-loaded in the beginning and at some point turned into lightly-loaded. Let $\mathbf{U}_0 \subseteq \mathbf{D}^*(u)$ be the set of vertices whose removal from $\mathbf{V}^\phi$ makes this change. By our scheme, z_u is raised in the beginning and at some point when $d^\phi(u)$ is about to fall under $c(u)$, we fixed z_u and start raising $g_{u,v}$ for $v \in \mathbf{D}^*(u) \backslash \mathbf{U}_0$. Note that, we have $y_{u_0} = z_u$ for all $u_0 \in \mathbf{U}_0$, $y_v = z_u + g_{u,v}$ for each $v \in \mathbf{D}^*(u) \backslash \mathbf{U}_0$, and $g_{u,v} = 0$ for $v \in \mathbf{N}[u] \backslash \mathbf{D}^*(u) \cup \mathbf{U}_0$. Let $d^*_{\mathbf{U}_0} = c(u) - \sum_{v \in \mathbf{D}^*(u) \backslash \mathbf{U}_0} d(v)$. We have $w(u) = c(u) \cdot z_u + \sum_{v \in \mathbf{N}[u]} d(v) \cdot g_{u,v} = \left(d^*_{\mathbf{U}_0} + \sum_{v \in \mathbf{D}^*(u) \backslash \mathbf{U}_0} d(v) \right) \cdot z_u + \sum_{v \in \mathbf{D}^*(u)} d(v) \cdot g_{u,v} = d^*_{\mathbf{U}_0} \cdot z_u + \sum_{v \in \mathbf{D}^*(u) \backslash \mathbf{U}_0} d(v) \cdot (z_u + g_{u,v}) \leq \sum_{v \in \mathbf{U}_0} d_v \cdot y_v + \sum_{v \in \mathbf{D}^*(u) \backslash \mathbf{U}_0} d(v) \cdot y_v$.

For both cases, the cost $w(u)$ of the single multiplicity can be distributed to the demand of vertices in $\mathbf{D}^*(u)$.

Finally, for each unit demand, say demand d from vertex u, consider the set of vertices $\mathbf{V}_d \subseteq \mathbf{N}[u]$ that has charged d. First, by our assigning scheme, $\mathbf{V}_d$ consists of at most one heavily-loaded vertex. If d is assigned to a heavily-loaded vertex, then, by our charging scheme, we have $|\mathbf{V}_d| = 1$, and d is charged at most twice. Otherwise, if d is assigned to a lightly-loaded vertex, then, by our charging scheme, each vertex in $\mathbf{V}_d$ charges d at most once, disregarding heavily-loaded or lightly-loaded vertices. This shows that d gets a charge of at most $deg[u] \cdot y_u$. □

Theorem 4.14. *Given any graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, we can compute a Δ^*-approximation for the capacitated domination problem on $\mathbf{G}$ in polynomial time, where Δ^* is the maximum closed degree of $\mathbf{G}$.*

Proof. By Lemma 4.13 and the linear program duality, we have

$$w(f^*) \leq \sum_{u \in \mathbf{V}} d(u) \cdot deg[u] y_u \leq \Delta^* \sum_{u \in \mathbf{V}} d(u) y_u \leq \Delta^* \cdot \mathcal{O}pt_{frac}(\mathbf{G}) \leq \Delta^* \cdot \mathcal{O}pt(\mathbf{G}).$$

□

Chapter 5

Graphs of Bounded Treewidth

In the chapter, we present results for graphs of bounded treewidth. In particular, in section §5.1, we show that this problem is *W[1]*-hard when parameterized by treewidth, regardless of demand assigning models. Then we present a fixed-parameter tractable algorithm for inseparable demand model, taking both treewidth and maximum capacity as the parameters, in section §5.2. This algorithm is further extended to separable demand model. At the end of this chapter, we present in §5.3 a constant factor approximation for separable demand model on outerplanar graphs, which is also the graphs of treewidth two, based on a novel hierarchical perspective on the structure of outerplanar graphs followed by a primal-dual analysis.

5.1 $W[1]$-Hardness w.r.t. Treewidth

We show that the capacitated domination problem is *W[1]*-hard when parameterized by treewidth. The reduction is made from the *k-Multicolored Clique*, which is a restriction of *k-Clique* problem.

Definition 5.1 (Multicolored Clique)**.** Given an integer k and a connected undirected graph $\mathcal{G} = \left(\bigcup_{i=1}^{k} \mathcal{V}[i], \mathcal{E}\right)$ such that $\mathcal{V}[i]$ induces an independent set for each i, the MULTICOLORED CLIQUE problem asks whether or not there exists a clique of size k in $\mathcal{G}$.

Given an instance $(\mathcal{G}, k)$ of multicolored clique, we show how an instance $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ of treewidth $O(k^2)$ for the capacitated domination problem can be built such that $\mathcal{G}$ has a clique of size k if and only if $\mathbf{G}$ has a capacitated dominating multi-set of cardinality at most $k' = (3k^2 - k)/2$. For convenience, we distinguish the vertices of $\mathbf{G}$ by referring to them as *nodes*.

While the technical detail is subtle, the general idea behind is conceivable. For each independent set $\mathcal{V}[i]$, $1 \leq i \leq k$, we construct a star rooted at a node x_i containing a node $\overline{u}$ for each vertex $u \in \mathcal{V}[i]$. From the construction we guarantee that exactly one node will be picked in the optimal capacitated dominating multi-set, and this will correspond to the selection of the vertices to form a clique in $\mathcal{G}$. Similarly, we construct a rooted star $y_{i,j}$ for each $1 \leq i < j \leq k$ containing nodes corresponding to the set of edges between $\mathcal{V}[i]$ and $\mathcal{V}[j]$. This will represent the set of edges that form a clique together with the chosen vertices.

To ensure that the set of vertices and the set of edges we pick will exactly form a clique, additional bridge nodes as well as propagation nodes are created to link the node corresponding to each edge in $\mathcal{E}$ and the nodes corresponding to its two end-vertices. See also Figure 5.1 for an illustration.

Let $\mathcal{N} = \sum_{1 \leq i \leq k} |\mathcal{V}[i]|$ be the number of vertices. Without loss of generality, we label the vertices of G by integers between 1 and $\mathcal{N}$, for which we denote by $label(v)$ for each $v \in V$. For each $i \neq j$, let $\mathcal{E}[i, j]$ denote the set of edges between $\mathcal{V}[i]$ and $\mathcal{V}[j]$. The graph $\mathbf{G}$ is defined as follows. For each i, $1 \leq i \leq k$, we create a node x_i with $w(x_i) = k' + 1$, $c(x_i) = 0$, and $d(x_i) = 1$.

For each $u \in \mathcal{V}[i]$, we have a node $\overline{u}$ with $w(\overline{u}) = 1$, $c(\overline{u}) = 1 + (k-1)\mathcal{N}$, and $d(\overline{u}) = 0$. We also connect $\overline{u}$ to x_i. For convenience, we refer to the star rooted at x_i as vertex star T_i.

Similarly, for each $1 \leq i < j \leq k$, we create a node y_{ij} with $w(y_{ij}) = k' + 1$, $c(x_i) = 0$, and $d(x_i) = 1$. For each $e \in \mathcal{E}[i,j]$ we have a node $\overline{e}$ with $w(\overline{e}) = 1$, $c(\overline{e}) = 1 + 2\mathcal{N}$, and $d(\overline{e}) = 0$. We connect $\overline{e}$ to y_{ij}. We refer to the star rooted at y_{ij} as edge star $\mathbf{T}_{ij}$. The selection of nodes in $\mathbf{T}_i$ and $\mathbf{T}_{ij}$ in the capacitated dominating multi-set will correspond to the decision of selecting the vertices to form a clique in $\mathcal{G}$.

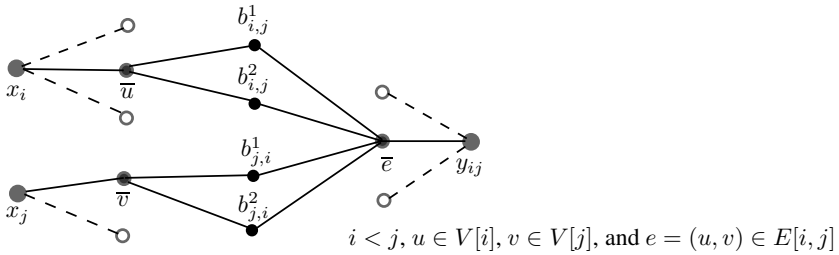


Figure 5.1: The connections between stars and bridge nodes.

In addition, for each $i \neq j$, $1 \leq i, j \leq k$, we create two bridge nodes $b^1_{i,j}$, $b^2_{i,j}$ with $w(b^1_{i,j}) = w(b^2_{i,j}) = 1$ and $d(b^1_{i,j}) = d(b^2_{i,j}) = 1$. The capacities of the bridge nodes are to be defined later.

Now we describe how the leaf nodes of $\mathbf{T}_i$ and $\mathbf{T}_{ij}$ are connected to bridge nodes such that the result we claimed holds. For each vertex star $\mathbf{T}_i$, $1 \leq i \leq k$, each j with $1 \leq j \leq k$, $i \neq j$, and each $v \in \mathcal{V}[i]$, we create two propagation nodes $p^1_{v,i,j}$, $p^2_{v,i,j}$ and connect them to $\overline{v}$. Besides, we connect $p^1_{v,i,j}$ to $b^1_{i,j}$ and $p^2_{v,i,j}$ to $b^2_{i,j}$. We set $w(p^1_{v,i,j}) = w(p^2_{v,i,j}) = k' + 1$ and $c(p^1_{v,i,j}) = c(p^2_{v,i,j}) = 0$. The demands of $p^1_{v,i,j}$ and $p^2_{v,i,j}$ are set to be $d(p^1_{v,i,j}) = \mathit{label}(v)$ and $d(p^2_{v,i,j}) = \mathcal{N} - \mathit{label}(v)$.

For each edge star $\mathbf{T}_{i,j}$, $1 \leq i < j \leq k$ and each $e = (u, v) \in \mathcal{E}[i,j]$ such that $u \in \mathcal{V}[i]$ and $v \in \mathcal{V}[j]$, we create four propagation nodes $p^1_{e,i,j}$, $p^2_{e,i,j}$, $p^1_{e,j,i}$, and $p^2_{e,j,i}$, which are connected to $\overline{e}$, with zero capacity and $k' + 1$ cost. In addition, we also connect $p^1_{e,i,j}$, $p^2_{e,i,j}$, $p^1_{e,j,i}$, $p^2_{e,j,i}$ to $p^1_{i,j}$, $p^2_{i,j}$, $p^1_{j,i}$, $p^2_{j,i}$, respectively. The demands of the four nodes are set as the following: $d(p^1_{e,i,j}) = \mathcal{N} - \mathit{label}(u)$, $d(p^2_{e,i,j}) = \mathit{label}(u)$, $d(p^1_{e,j,i}) = \mathcal{N} - \mathit{label}(v)$, and $d(p^2_{e,j,i}) = \mathit{label}(v)$.

Finally, for each bridge node b, we set $c(b) = \sum_{u \in \mathbf{N}[b]} d(u) - \mathcal{N}$.

Lemma 5.2. *The treewidth of* $\mathbf{G}$ *is* $O(k^2)$.

Proof. Consider the set of bridge nodes, $\mathbf{B} = \bigcup_{i \neq j} \left\{ b^1_{i,j} \cup b^2_{i,j} \right\}$. Since $\mathbf{G} \backslash \mathbf{B}$ is a forest, which is of treewidth 1, and the removal of a vertex from a graph decreases the treewidth by at most one, the treewidth of $\mathbf{G}$ is upper bounded by the number of bridge nodes plus 1, which is $O(k^2)$. □

Lemma 5.3. $\mathcal{G}$ *admits a clique of size* k *if and only if* $\mathbf{G}$ *admits a capacitated dominating set of cost at most* $k' = (3k^2 - k)/2$.

Proof. Let $\mathbf{C} \subseteq \mathcal{G}$ be a clique of size k in $\mathcal{G}$. By choosing the bridge nodes, $b^1_{i,j}$ and $b^2_{i,j}$ for each $i \neq j$, $\overline{u}$ for each $u \in \mathbf{C}$, and $\overline{e}$ for each $e \in C$ exactly once, we have a vertex subset of cost exactly $(3k^2 - k)/2$. One can easily verify that this is also a feasible capacitated dominating multi-set for $\mathbf{G}$.

On the other hand, let $\mathbf{D}$ be a capacitated donimating multi-set of cost at most k' in $\mathbf{G}$. We will argue that there exists a clique of size k in $\mathcal{G}$. First observe that none of the propagation nodes are chosen in $\mathbf{D}$, otherwise the cost would exceed k'. This implies $b^1_{i,j} \in \mathbf{D}$ and $b^2_{i,j} \in \mathbf{D}$, for

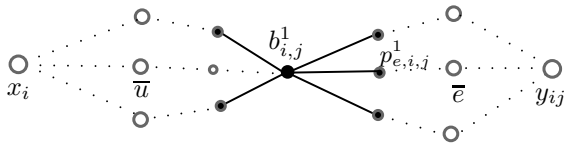


Figure 5.2: Local connections around the bridge node $b^1_{i,j}$. The solid circles represent the set $\mathbf{S} = \mathbf{N}[b^1_{i,j}]\backslash\mathbf{N}[\overline{u}]$.

each $i \neq j$, as they are adjacent only to propagation nodes. Note that this already contributes $k(k-1)$ nodes to $\mathbf{D}$ with cost at least $k(k-1)$ and the rest of the nodes in $\mathbf{D}$ together contributes at most $k(k+1)/2$ cost.

Similarly, we conclude that $x_i \notin \mathbf{D}$ and $y_{ij} \notin \mathbf{D}$ for each $i \neq j$. Therefore, for each $1 \leq i \leq k$, $\exists u \in \mathcal{V}[i]$ such that $\overline{u} \in \mathbf{D}$, and for each $i \neq j$, $\exists e \in \mathcal{E}[i,j]$ such that $\overline{e} \in \mathbf{D}$. Since we have

$$\frac{k(k-1)}{2} + k = \frac{k(k+1)}{2}$$

such stars, exactly one node from each star is chosen to be included in $\mathbf{D}$ and therefore the multiplicity of each node in $\mathbf{D}$ is exactly one.

Next we argue that the nodes chosen in each star will correspond to a clique of size k in $\mathcal{G}$. For each $1 \leq i < j \leq k$, let $\overline{u} \in \mathbf{T}_i$ and $\overline{v} \in \mathbf{T}_j$ be the nodes chosen in $\mathbf{D}$. Let $\overline{e} \in \mathbf{T}_{ij}$ be the node chosen in $\mathbf{D}$. In the following, we argue that the two end-vertices of e are exactly u and v, meaning that $e = (u, v)$. Note that, this will imply the existence of a clique of size k in $\mathcal{G}$, formed by the vertices corresponding to the nodes chosen in each $\mathbf{T}_i$, $1 \leq i \leq k$.

Since the capacity of $\overline{u}$ equals the sum of the demands over $\mathbf{N}[\overline{u}]$, without loss of generality we can assume that the demands of nodes from $\mathbf{N}[\overline{u}]$ are served merely by $\overline{u}$. Consider the bridge vertex b^1_{ij} and the set $\mathbf{S} = \mathbf{N}[b^1_{ij}]\backslash\mathbf{N}[\overline{u}]$. The demand of vertices in $\mathbf{S}$ can only be served by either b^1_{ij} or $\overline{e}$, as they are the only two vertices in $\mathbf{N}[\mathbf{S}]$ that are chosen in dominating multi-set $\mathbf{D}$. See also Figure 5.2 for an illustration. In particular, vertices in $\mathbf{S}\backslash\{p^1_{e,i,j}\}$ can only be served by b^1_{ij}. Therefore, we have

$$c(b^1_{ij}) \geq \sum_{u \in \mathbf{S}\backslash\{p^1_{e,i,j}\}} d(u) \ .$$

Since $c(b^1_{ij}) = \sum_{u \in \mathbf{N}[b^1_{ij}]} d(u) - \mathcal{N}$ by our setting, the above inequality implies $d(p^1_{e,i,j}) \geq \mathcal{N} - \mathit{label}(u)$, which in turn implies $d(p^2_{e,i,j}) \leq \mathit{label}(u)$ as we have $d(p^1_{e,i,j}) + d(p^2_{e,i,j}) = \mathcal{N}$ by our construction. By a symmetric argument on $b^2_{i,j}$, we obtain $d(p^2_{e,i,j}) \geq \mathit{label}(u)$. Hence $d(p^2_{e,i,j}) = \mathit{label}(u)$.

By another symmetric argument on b^1_{ji} and b^2_{ji}, we obtain $d(p^2_{e,j,i}) = \mathit{label}(v)$. Therefore $e = (u, v)$ and the lemma follows. □

Note that this proof holds for both separable and inseparable demand models. We have the following theorem.

Theorem 5.4. *The capacitated domination problem is W[1]-hard when parameterized by treewidth, regardless of demand assigning model.*

Proof. Clearly the reduction instance $\mathbf{G}$ can be computed in time polynomial in both k and $\mathcal{N}$. By Lemma 5.2, $\mathbf{G}$ has treewidth $O(k^2)$. This theorem follows directly from Lemma 5.3 and the *W[1]*-hardness of multicolored clique. □

5.2 Fixed-Parameter Tractability w.r.t. Treewidth and Maximum Capacity

Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a graph of treewidth k. We show that, by taking the maximum capacity of the vertices, denoted by $\mathcal{M}$, as one extra parameter, this problem is fixed-parameter tractable when the demand is inseparable and can be solved in $O\left(2^{2k(\log \mathcal{M}+1)+\log k)} \cdot n\right)$ time, where $n = |\mathbf{V}|$ is the number of vertices.

To this end, we give a dynamic programming algorithm on a *nice tree decomposition* [59], which is a special tree decomposition and for which we give the formal definition below, of the input graph $\mathbf{G}$.

Definition 5.5 (Nice Tree Decomposition [59])**.** A tree decomposition $(\mathbf{X}, \mathbf{T})$ is a nice tree decomposition if one can root $\mathbf{T}$ in a way such that each node $i \in \mathbf{I}$ is of one of the following four types.

- **Leaf**: node i is a leaf of $\mathbf{T}$, and $|\mathbf{X}_i| = 1$.
- **Join**: node i has exactly two children, say j_1 and j_2, and $\mathbf{X}_i = \mathbf{X}_{j_1} = \mathbf{X}_{j_2}$.
- **Introduce**: node i has exactly one child, say j, and there is a vertex $v \in \mathbf{V}$ such that $\mathbf{X}_i = \mathbf{X}_j \cup \{v\}$.
- **Forget**: node i has exactly one child, say j, and there is a vertex $v \in \mathbf{V}$ such that $\mathbf{X}_j = \mathbf{X}_i \cup \{v\}$.

Given a tree decomposition of width k, a nice tree decomposition of the same width can be found in linear time [59]. The advantage of this decomposition lies in the fact that it provides the structural information of the given graph in a well-organized fashion which supports bottom-up traversals to reconstruct the whole graph.

Let $(\mathbf{X}, \mathbf{T})$ be a nice tree decomposition of $\mathbf{G}$. For each node $i \in \mathbf{T}$, let $\mathbf{T}_i$ be the subtree of $\mathbf{T}$ rooted at i and $\mathbf{Y}_i := \bigcup_{j \in \mathbf{T}_i} \mathbf{X}_j$. Literally, $\mathbf{Y}_i$ denotes the set of vertices that are contained in the bags of the nodes in $\mathbf{T}_i$.

Starting from the leaf nodes of $\mathbf{T}$, our algorithm proceeds in a bottom-up manner and maintains for each node $i \in \mathbf{T}$ a table $\mathcal{A}_i$ whose columns consist of the following information.

- A subset $\mathbf{P}$ of $\mathbf{X}_i$ indicating the set of vertices in $\mathbf{X}_i$ that have already been dominated, and
- for each $u \in \mathbf{X}_i$, the amount of residue capacity of u, denoted $rc(u)$, where $0 \leq rc(u) < c(u)$.

For each possible configuration of the columns described above, the algorithm will maintain a row in $\mathcal{A}_i$ and computes the cost of the optimal demand assignment function for the subgraph induced by $\mathbf{Y}_i$ under the condition that the set of vertices that have been dominated by this demand assignment and also the residue capacity of each vertex meet exactly the values specified by the row.

In the following, we describe the computation of the table $\mathcal{A}_i$ for each node i in the tree $\mathbf{T}$. For the ease of presentation, we use the terms "insert a new row" and "replace the value of an old row by the new one" interchangeably. Whenever the algorithm attempts to insert a new row into a table while another row with identical configuration already exists, the one with the smaller cost will be kept. According to different types of vertices we encounter during the procedure, we have the following four cases.

(1) i is a leaf node. Let $X_i = \{v\}$. We add two rows to the table $\mathcal{A}_i$ which correspond to cases whether or not v is served.

1: let $r_1 = (\{\phi\}, \{rc(v) = 0\})$ be a new row with $cost(r_1) \longleftarrow 0$
2: let $r_2 = (\{v\}, \{rc(v) \equiv d(v) \bmod c(v)\})$ be a new row with $cost(r_2) \longleftarrow w(v) \cdot \left\lceil \frac{d(v)}{c(v)} \right\rceil$
3: add r_1 and r_2 to $\mathcal{A}_i$

---

(2) i is an introduce node. Let j be the child of i, and let $\mathbf{X}_i = \mathbf{X}_j \cup \{v\}$. The data in $\mathcal{A}_j$ is inherited by $\mathcal{A}_i$. We extend $\mathcal{A}_i$ by considering, for each existing row r in $\mathcal{A}_j$, all $2^{|\mathbf{X}_j \setminus \mathbf{P}_r|}$ possible ways of choosing vertices in $\mathbf{X}_j \setminus \mathbf{P}_r$ to be assigned to v. In addition, v can be either unassigned or assigned to any vertex in X_i. In either case, the cost and the residue capacity are modified accordingly.

1: **for all** row $r_0 = (P, R) \in \mathcal{A}_j$ **do**
2: **for all** possible U such that $U \subseteq (X_j \setminus P) \cap N_G(v)$ **do**
3: let $R' = R \cup \{rc(v) = \sum_{u \in U} d(u) \bmod c(v)\}$, and
let $r = (P \cup U, R')$ be a new row with
$cost(r) = cost(r_0) + w(v) \cdot \left\lceil \frac{\sum_{u \in U} d(u)}{c(v)} \right\rceil$
4: add r to $\mathcal{A}_i$
5: **for all** $u \in X_i$ **do**
6: let $r' = (P \cup U \cup \{v\}, R' \cup \{rc(u) = (rc(u) - d(v)) \bmod c(u)\})$ be a new row with
$cost(r') = cost(r)+$ the cost required by this assignment
7: add r' to $\mathcal{A}_i$
8: **end for**
9: **end for**
10: **end for**

---

(3) i is a forget node. Let j be the child of i, and let $\mathbf{X}_i = \mathbf{X}_j \setminus \{v\}$. In this case, for each row $r \in \mathcal{A}_j$ such that $v \in \mathbf{P}_r$, we insert a row r' to $\mathcal{A}_i$ identical to r except for the absence of v in $\mathbf{P}_{r'}$. The remaining rows in $\mathcal{A}_j$, which correspond to situations where v is not served, are ignored without being considered.

1: **for all** row $r_0 = (P, R) \in \mathcal{A}_j$ such that $v \in P$ **do**
2: let $r = (P \setminus \{v\}, R \setminus \{rc(v)\})$ be a new row with $cost(r) = cost(r_0)$
3: add r to $\mathcal{A}_i$
4: **end for**

---

(4) i is a join node. Let j_1 and j_2 be the two children of i in $\mathbf{T}$. We consider every pair of rows r_1, r_2 where $r_1 \in \mathcal{A}_{j_1}$ and $r_2 \in \mathcal{A}_{j_2}$. We say that two rows r_1 and r_2 are *compatible* if $\mathbf{P}_{r_1} \cap \mathbf{P}_{r_2} = \phi$. For each compatible pair of rows (r_1, r_2), we insert a new row r to $\mathcal{A}_i$ with $\mathbf{P}_r = \mathbf{P}_{r_1} \cup \mathbf{P}_{r_2}$, $rc_r(u) = (rc_{r_1}(u) + rc_{r_2}(u)) \bmod c(u)$, for each $u \in \mathbf{X}_i$, and $cost(r) = cost(r_1) + cost(r_2) - \sum_{u \in \mathbf{X}_i} \left\lfloor \frac{rc_{r_1}(u) + rc_{r_2}(u)}{c(u)} \right\rfloor$.

1: **for all** compatible pairs $r_1 = (P_1, R_1) \in \mathcal{A}_{j_1}$ and $r_2 = (P_2, R_2) \in \mathcal{A}_{j_2}$ **do**
2: let $r = (P_1 \cup P_2, R)$ be a new row.
3: $cost(r) \longleftarrow cost(r_1) + cost(r_2) - \sum_{u \in X_i} \left\lfloor \frac{rc_{R_1}(u) + rc_{R_2}(u)}{c(u)} \right\rfloor$, and
4: $R \longleftarrow \{rc_{R_1}(u) + rc_{R_2}(u) \bmod c(u) : u \in X_i\}$
5: add r to $\mathcal{A}_i$
6: **end for**

Theorem 5.6. *The capacitated domination problem with inseparable demand on graphs of bounded treewidth can be solved in time $2^{2k(\log \mathcal{M}+1)+\log k+O(1)} \cdot n$, where k is the treewidth and $\mathcal{M}$ is the maximum capacity of the graph.*

Proof. The correctness of the algorithm follows from an inductive argument along with the description given above. Regarding the running time of this algorithm, first notice that the size of the table we maintained for each node of the tree decomposition is bounded by $2^k \cdot \mathcal{M}^k$. The computation for leaf nodes takes $O(1)$ time, while the computation for introduce nodes and forget nodes take $O(k2^{2k} \cdot \mathcal{M}^k)$ and $O(2^k \cdot \mathcal{M}^k)$, respectively. For join nodes, however, we consider each compatible pair of rows from two tables. Therefore the computation requires $O(k2^{2k} \cdot \mathcal{M}^{2k})$ time. The size of a nice tree decomposition is linear in the number of vertices of the graph. Therefore the overall running time of the algorithm is $O(k2^{2k} \cdot \mathcal{M}^{2k} \cdot n) = 2^{2k(\log \mathcal{M}+1)+\log k+O(1)} \cdot n$. □

We state without going into details that, by suitably replacing the set $\mathbf{P}_i$ we maintained for each row of the table $\mathcal{A}_i$ with the residue demand of each vertex contained in the bag $\mathbf{X}_i$, the algorithm can be modified to handle separable demand model as well. We have the following corollary.

Theorem 5.7. *The capacitated domination problem with separable demand on graphs of bounded treewidth can be solved in time $2^{(2\mathcal{M}+2\mathcal{N}+1)\log k+O(1)} \cdot n$, where k is the treewidth, $\mathcal{M}$ is the maximum capacity, and $\mathcal{N}$ is the maximum demand.*

5.3 A Constant Factor Approximation for Outerplanar Graphs with Separable Demand

In the following, we first classify outer-planar graphs into a class of graphs called general-ladders and show how the corresponding general-ladder representation can be extracted in $O(n \log^3 n)$ time in §5.3.1. Then we consider in §5.3.2 and §5.3.3 both the primal and the dual linear programs of the relaxation of (4.2) to further reduce a given general-ladder and obtain a constant factor approximation. We give an overall analysis on the algorithm in §5.3.4.

Throughout this section we will assume $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ to be an outerplanar graph. In addition to the graph $\mathbf{G}$, we assume that the corresponding outerplanar embedding of $\mathbf{G}$ in the plane is given as well. Otherwise we apply the $O(n \log^3 n)$ algorithm provided by Bose [16] to compute

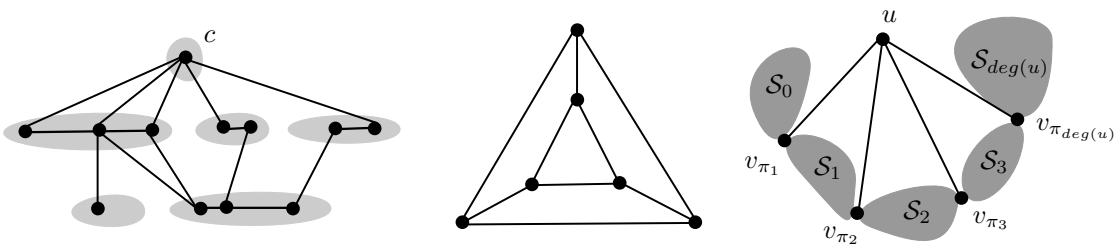


Figure 5.3: (a) A general-ladder with anchor c. (b) A 2-outerplanar graph which fails to be a general-ladder. (c) The subdivision formed by a vertex u in an outer-planar embedding.

such an embedding. We denote this embedding by $\mathcal{D}$. Furthermore, we will adopt the notations from §4.4 and use $\mathcal{O}pt(G)$ and $\mathcal{O}pt_{frac}(G)$ to denote the optimal objective value for the integer linear program (4.2) and its relaxation with respect to graph G.

5.3.1 The Structure - General Ladders

First we define the notation which we will use later on. By a total order of a set we mean that each pair of elements in the set can be compared, and therefore an ascending order of the elements is well-defined. Let $P = (v_1, v_2, \ldots, v_k)$ be a path. We say that P is an *ordered path* if a total order $v_1 \prec v_2 \prec \ldots \prec v_k$ or $v_k \prec v_{k-1} \prec \ldots \prec v_1$ is defined on the set of vertices.

Definition 5.8 (General-Ladder)**.** A graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ is said to be a general-ladder if a total order on the set of vertices is defined, and $\mathbf{G}$ is composed of a set of layers $\{\mathbf{L}_1, \mathbf{L}_2, \ldots, \mathbf{L}_k\}$, where each layer is a collection of subpaths of an ordered path such that the following holds. The top layer, $\mathbf{L}_1$, consists of a single vertex, which is referred to as the anchor, and for each $1 < j < k$ and $u, v \in \mathbf{L}_j$, we have

1. $\mathbf{N}[u] \subseteq \mathbf{L}_{j-1} \cup \mathbf{L}_j \cup \mathbf{L}_{j+1}$, and
2. $u \prec v$ implies $\max_{p \in \mathbf{N}[u] \cap \mathbf{L}_{j+1}} p \preccurlyeq \min_{q \in \mathbf{N}[v] \cap \mathbf{L}_{j+1}} q$.

Note that each layer in a general-ladder consists of a set of ordered paths which are possibly connected only to vertices in the neighboring layers. See Fig. 5.3 (a). Although the definition of general-ladders captures the essence and simplicity of an ordered hierarchical structure, there are planar graphs which fall outside this framework. See also Fig. 5.3 (b).

In the following, we state and argue that every outerplanar graph meets the requirements of a general-ladder. Let $u \in \mathbf{V}$ be an arbitrary vertex of $\mathbf{G}$. We fix u to be the smallest element and define a total order on the vertices of $\mathbf{G}$ according to their orders of appearances on the outer face of $\mathcal{D}$ in a counter-clockwise order. For convenience, we label the vertices such that $u = v_1$ and $v_1 \prec v_2 \prec v_3 \prec \ldots \prec v_n$.

Let $\mathbf{N}(u) = \left\{v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_{deg(u)}}\right\}$ denote the neighbors of u such that $v_{\pi_1} \prec v_{\pi_2} \prec \ldots \prec v_{\pi_{deg(u)}}$. According to the total order we defined on $\mathbf{V}$, $\mathbf{N}(u)$ divides the set of vertices except u into $deg(u) + 1$ subsets, namely,

$$\begin{cases} \mathbf{S}_0 = \{v_2, v_3, \ldots, v_{\pi_1}\}, \\ \mathbf{S}_i = \left\{v_{\pi_i}, v_{\pi_i+1}, \ldots, v_{\pi_{i+1}}\right\}, & \text{for } 1 \leq i < deg(u), \text{ and} \\ \mathbf{S}_{deg(u)} = \left\{v_{\pi_{deg(u)}}, v_{\pi_{deg(u)}+1}, \ldots, v_n\right\}. \end{cases}$$

See also Fig. 5.3 (c) for an illustration.

For any $0 < i < j < deg(u)$, $v_{\pi_{i-1}} \prec p \prec v_{\pi_i}$, and $v_{\pi_{j-1}} \prec q \prec v_{\pi_j}$, there is no edge connecting p and q. Otherwise it will result in a crossing with the edge (u, v_{π_i}), contradicting to the fact that $\mathcal{D}$ is a planar embedding.

For $1 \leq i < deg(u)$, we partition $\mathbf{S}_i$ into two subsets L_i and R_i as follows. Let $d_{\mathbf{S}_i}$ denote the distance function defined on the induced subgraph of $\mathbf{S}_i$. Let

$$L_i = \left\{ v : v \in \mathbf{S}_i, d_{\mathbf{S}_i}\left(v_{\pi_i}, v\right) \leq d_{\mathbf{S}_i}\left(v, v_{\pi_{i+1}}\right) \right\}$$

and $R_i = \mathbf{S}_i \backslash L_i$.

Lemma 5.9. *We have* $\max_{a \in L_i} a \prec \min_{b \in R_i} b$ *for all* $1 \leq i < deg(u)$.

Proof. For convenience, let $a = \max_{a \in L_i} a$ and $b = \min_{b \in R_i} b$. Since $L_i \cap R_i = \phi$, we have $a \neq b$. Assume that $b \prec a$. Recall that in an outer-planar embedding, the vertices are placed on a circle and the edges are drawn as straight lines. Since $\mathbf{E}$ is an outer-planar embedding, the shortest path from a to v_{π_i} must intersect with the shortest path from b to $v_{\pi_{i+1}}$. Let c be the vertex for which the two paths meet. Since L_i and R_i form a partition of $\mathbf{S}_i$, either $c \in L_i$ or $c \in R_i$.

If $c \in L_i$, then $d_{\mathbf{S}_i}\left(c, v_{\pi_i}\right) \leq d_{\mathbf{S}_i}\left(c, v_{\pi_{i+1}}\right)$ by definition, which implies that $d_{\mathbf{S}_i}\left(b, v_{\pi_i}\right) \leq d_{\mathbf{S}_i}\left(b, v_{\pi_{i+1}}\right)$, a contradiction to the fact that $b \in R_i$. On the other hand, if $c \in R_i$, then $d_{\mathbf{S}_i}\left(c, v_{\pi_i}\right) > d_{\mathbf{S}_i}\left(c, v_{\pi_{i+1}}\right)$, and we have $d_{\mathbf{S}_i}\left(a, v_{\pi_i}\right) > d_{\mathbf{S}_i}\left(a, v_{\pi_{i+1}}\right)$, a contradiction to the fact $a \in L_i$. In both cases, we have a contradiction. Therefore we have $a \prec b$. □

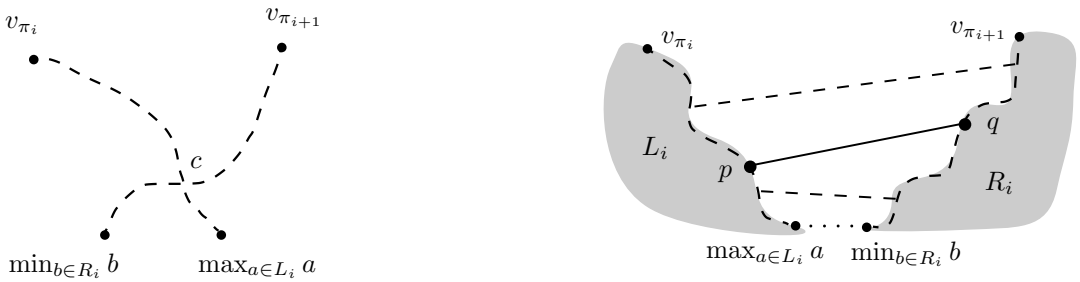


Figure 5.4: (a) A contradiction led by $\min_{b \in R_i} b \prec \max_{a \in L_i} a$. (b) Partition of $\mathbf{S}_i$ into L_i and R_i.

Let $\ell(v) \equiv d_{\mathbf{G}}(u, v)$ and $\ell_i(v) \equiv \min\left\{ d_{\mathbf{S}_i}(v_{\pi_{(i)}}, v), d_{\mathbf{S}_i}(v_{\pi_{(i+1)}}, v) \right\}$, for any $1 \leq i < deg(u)$ and $v \in \mathbf{S}_i$. Observe that $\ell(v) = \ell_i(v) + 1$, for any $1 \leq i < deg(u)$ and $v \in \mathbf{S}_i$. Now consider the set of the edges connecting L_i and R_i. Note that, this is exactly the set of edges connecting vertices on the shortest path between v_{π_i} and $\max_{a \in L_i} a$ and vertices on the shortest path between $v_{\pi_{i+1}}$ and $\min_{b \in R_i} b$. We have the following lemma, which states that, when the vertices are classified by their distances to u, these edges can only connect vertices between neighboring sets and do not form any crossing. See also Fig. 5.4.

Lemma 5.10. *For any edge* (p, q), $p \in L_i$, $q \in R_i$, *connecting* L_i *and* R_i, *we have*

- $|\ell(p) - \ell(q)| \leq 1$, *and*
- $\nexists$ *edge* (r, s), $(r, s) \neq (p, q)$, $r \in L_i$, $s \in R_i$, *such that* $\ell(r) = \ell(q)$ *and* $\ell(p) = \ell(s)$.

Proof. The first half of the lemma follows from the definition of ℓ. If $|\ell(p) - \ell(q)| > 1$, without loss of generality, suppose that $\ell(p) > \ell(q) + 1$, by going through (p, q) then following the shortest path from q to u, we find a shorter path for p, which is a contradiction. The second half follows from the fact that $\mathcal{D}$ is a planar embedding. □

Now we are ready to present our structural statement.

Lemma 5.11. *Any outer-planar graph* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *together with an arbitrary vertex* $u \in \mathbf{V}$ *is a general-ladder anchored at* u*, where the set of vertices in each layer are classified by their distances to the anchor* u*.*

Proof. We prove by induction on the number of vertices of $\mathbf{G}$. First, an isolated vertex is a single-layer general-ladder. For non-trivial graphs, let $\mathbf{S}_0, \mathbf{S}_1, \ldots, \mathbf{S}_{deg(u)}$ be the subsets defined as above. By assumption, the induced subgraphs of $\mathbf{S}_0$ and $\mathbf{S}_{deg(u)}$ are general-ladders with anchors v_{π_1} and $v_{\pi_{deg(u)}}$, respectively. Furthermore, the layers are classified by $\ell - 1$. That is, vertex v belongs to layer $\ell(v) - 1$. Similarly, the induced subgraphs of L_i and R_i are also general-ladders with anchors v_{π_i} and $v_{\pi_{i+1}}$ whose layers are classified by ℓ_i.

Now we argue that these general-ladders can be arranged properly to form a single general-ladder with anchor u and layers classified by ℓ. Since there is no edge connecting p and q for any p, q with $v_{\pi_{i-1}} \prec p \prec v_{\pi_i}$ and $v_{\pi_{j-1}} \prec q \prec v_{\pi_j}$, $0 < i < j < deg(u)$, we only need to consider the edges connecting vertices between L_i and R_i. By Lemma 5.10, when the general-ladders L_i and R_i are hung over v_{π_i} and $v_{\pi_{i+1}}$, respectively, the edges between them connect exactly only vertices from adjacent layers and do not form any crossing. Therefore, it constitute as a single general-ladder together with u and the lemma follows. □

Extracting the General-ladder For the outerplanar graph $\mathbf{G}$, we describe how the corresponding general-ladder can be extracted.

Theorem 5.12. *Given an outer-planar graph* $\mathbf{G}$ *and its outer-planar embedding, we can compute in linear time a general-ladder representation for* $\mathbf{G}$*.*

Proof. First we compute the shortest distance of each vertex $v \in \mathbf{V}$ to u, denoted by $\ell(v)$. Let $\mathcal{M} = \max_{v \in \mathbf{V}} \ell(v)$. We create $\mathcal{M} + 1$ empty queues, denoted $layer(0)$, $layer(1)$, …, $layer(\mathcal{M})$, which will be used to maintain the set of layers. Retrieve an outer-planar embedding of $\mathcal{G}$ and traverse the outer face, starting from u, in a counter-clockwise order. For each vertex v visited, we attach v to the end of $layer(\ell(v))$.

Since the number of edges in a planar graph is linear in the number of vertices, the shortest-path tree computation takes linear time. The traversal of the outer face also takes linear time. □

For the rest of this paper we will denote the layers of this particular general-ladder representation by $\mathbf{L}_0, \mathbf{L}_1, \ldots, \mathbf{L}_{\mathcal{M}}$. The following additional structural property comes from the outer-planarity of $\mathbf{G}$ and our construction scheme.

Lemma 5.13. *For any* $0 < i \leq M$ *and* $v \in \mathbf{L}_i$*, we have*

$$|\mathbf{N}(v) \cap \mathbf{L}_{i-1}| \leq 2.$$

Moreover, if v *has two neighbors in* $\mathbf{L}_i$*, say,* v_1 *and* v_2 *with* $v_1 \prec v \prec v_2$*, then there is an edge joining* v_1 *(and* v_2*, respectively) and each neighboring vertex of* v *in* $\mathbf{L}_{i-1}$ *that is smaller (larger) than* v*.*

Proof. First, since the layers are classified by the distances to the anchor u, if $|\mathbf{N}(v) \cap \mathbf{L}_{i-1}| \geq 3$, then consider the shortest paths from vertices in $\mathbf{N}(v) \cap \mathbf{L}_{i-1}$ to u. At least one vertex would be surrounded by other two paths, contradicting the fact that $\mathcal{G}$ is an outer-planar graph.

The second part is obtained from a similar argument. Let $v' \in \mathbf{N}(v) \cap \mathbf{L}_{i-1}$ be a neighbor of v in $\mathbf{L}_{i-1}$. If v' is not joined to either v_1 or v_2, then consider the shortest paths from u to v_1, v', and v_2, respectively. v' would be a vertex in the interior, which is a contradiction. □

The Decomposition The idea behind this decomposition is to help reduce the dependency between vertices of large degrees and their neighbors such that further techniques can be applied. To this end, we tackle the demands of vertices from every three layers separately.

For each $0 \leq i < 3$, let $\mathbf{R}_i = \bigcup_{j \geq 0} \mathbf{L}_{3j+i}$. Let $\mathbf{G}_i = (\mathbf{V}_i, \mathbf{E}_i)$ consist of the induced subgraph of $\mathbf{R}_i$ and the set of edges connecting vertices in $\mathbf{R}_i$ to their neighbors. Formally,

$$\mathbf{V}_i = \bigcup_{v \in \mathbf{R}_i} \mathbf{N}[v] \quad \text{and} \quad \mathbf{E}_i = \bigcup_{v \in \mathbf{R}_i} \bigcup_{u \in \mathbf{N}[v]} e(u, v).$$

In addition, we set $d(v) = 0$ for all $v \in \mathbf{G}_i \backslash \mathbf{R}_i$. Other parameters remain unchanged.

Lemma 5.14. *Let f_i, $0 \leq i < 3$, be an optimal demand assignment function for $\mathbf{G}_i$. The assignment function*

$$f = \sum_{0 \leq i < 3} f_i$$

is a 3-approximation of $\mathbf{G}$.

Proof. First, for any vertex $v \in \mathbf{V}$, the demand of v is considered in $\mathbf{G}_i$ for some $0 \leq i < 3$ and therefore is assigned by the assignment function f_i. Since we take the union of the three assignments, it is a feasible assignment to the entire graph $\mathbf{G}$.

Since the demand of each vertex in $\mathbf{G}_i$, $0 \leq i < 3$, is no more than that of in the original graph $\mathbf{G}$, any feasible solution to $\mathbf{G}$ will also serve as a feasible solution to $\mathbf{G}_i$. Therefore we have $\mathcal{O}pt(\mathbf{G}_j) \leq \mathcal{O}pt(\mathbf{G})$, for $0 \leq j < 3$, and the lemma follows. □

5.3.2 Removing More Edges

We describe an approach to further simplifying the graphs $\mathbf{G}_i$, for $0 \leq i < 3$. Given any feasible demand assignment for $\mathbf{G}_i$, we can properly reassign the demand of a vertex to a constant number of neighbors while the increase in terms of fractional cost remains bounded.

For each $v \in \mathbf{R}_i$, we sort the closed neighbors of v according to their cost in ascending order such that $w(\pi_v(1)) \leq w(\pi_v(2)) \leq \ldots \leq w(\pi_v(deg[v]))$, where $\pi_v : \{1, 2, \ldots, deg[v]\} \to \mathbf{N}[v]$ is an injective function. For convenience, we set $\pi_v(deg[v] + 1) = \phi$. Suppose that $v \in \mathbf{L}_\ell$. We identify the following four vertices.

- Let j_v, $1 \leq j_v \leq deg[v]$, be the smallest integer such that $c(\pi_v(j_v)) > d(v)$. If $c(\pi_v(j_v)) \leq d(v)$ for all $1 \leq j \leq deg[v]$, then we let $j_v = deg[v] + 1$. Intuitively, $\pi_v(j_v)$ is the first vertex in the sorted list whose capacity is greater than $d(v)$.

- Let k_v, $1 \leq k_v < j_v$, be the integer such that

  $$\frac{w(\pi_v(k_v))}{c(\pi_v(k_v))}$$

  is minimized. k_v is defined only when $j_v > 1$. Literally, $\pi_v(k_v)$ is the vertex with best cost-capacity ratio among the first $j_v - 1$ vertices.

- Let

  $$p_v = \max_{u \in \mathbf{N}[v] \cap \mathbf{L}_{\ell-1}} u \quad \text{and} \quad q_v = \max_{u \in \mathbf{N}[v] \cap \mathbf{L}_{\ell+1}} u.$$

  p_v and q_v correspond to the rightmost neighbor of v in layer $\mathbf{L}_{\ell-1}$ and $\mathbf{L}_{\ell+1}$, respectively.

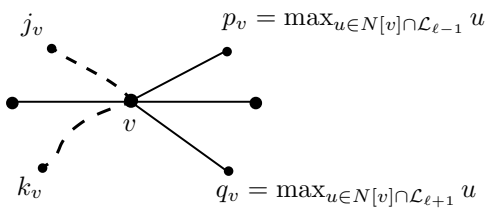


Figure 5.5: At most 6 incident edges for a vertex $v \in \mathbf{L}_\ell$ are to be kept in $\mathbf{H}_i$.

We will omit the function π_v and use j_v, k_v to denote $\pi_v(j_v)$, $\pi_v(k_v)$ when there is no confusion in the context. The reduced graph $\mathbf{H}_i$ is defined as follows. Denote the set of neighbors to be disconnected from v by $R(v) = \mathbf{N}[v] \setminus (\mathbf{L}_\ell \cup \{j_v \cup k_v \cup p_v \cup q_v\})$, and let $\mathbf{H}_i = \mathbf{G}_i \setminus \bigcup_{v \in \mathbf{R}_i} \bigcup_{u \in R(v)} \{e(u,v)\}$. Roughly speaking, in graph $\mathbf{H}_i$ we remove the edges which connect vertices in $\mathbf{R}_i$, say v, to vertices not in $\mathbf{R}_i$, except possibly for j_v, k_v, p_v, and q_v. See also Fig. 5.5. Note that, although our reassigning argument applies to arbitrary graphs, only when two vertices are unimportant to each other can we remove the edge between them.

Lemma 5.15. *In the reduced subgraph $\mathbf{H}_i$, we have*

1. *For each $v \notin \mathbf{R}_i$, at most one incident edge of v which was previously in $\mathbf{G}_i$ will be removed.*
2. *For each $v \in \mathbf{R}_i$, the degree of v in $\mathbf{H}_i$ is upper-bounded by 6.*
3. $\mathcal{O}pt_{frac}(\mathbf{H}_i) \leq 2 \cdot \mathcal{O}pt_{frac}(\mathbf{G}_i)$

Proof. For the first part, let $v \notin \mathbf{R}_i$ be a vertex and denote $\mathbf{S} = \mathbf{N}[v] \cap \mathbf{R}_i$ the set of neighbors of v that are in $\mathbf{R}_i$. By the definition of general-ladders, for any $u \in \mathbf{S}$, $u \neq \max_{a \in \mathbf{S}} a$, we have either $p_u = v$ or $q_u = v$, since v serves as the rightmost neighbor of u. Therefore, by our approach, only the edge between v and $\max_{a \in \mathbf{S}} a$ will possibly be removed.

For the second part, for any $v \in \mathbf{R}_i$, v has at most two neighbors in the same layer, since each layer is a subgraph of an ordered path. We have removed all the edges connecting v to vertices not in the same layer, except for at most four vertices, j_v, k_v, p_v, and q_v. Therefore $deg(v) \leq 6$.

Now we prove the third part of this lemma. Let $f_{\mathbf{G}_i}$ be an optimal demand assignment for $\mathbf{G}_i$, and $x_{\mathbf{G}_i}$ be the corresponding multiplicity function. Note that, from the second and the third inequalities of (4.2), for each $v \in \mathbf{V}$ and $u \in \mathbf{N}[v]$, we have

$$x_{\mathbf{G}_i}(u) \geq \max\left\{ \frac{f_{\mathbf{G}_i}(v,u)}{d(v)}, \frac{f_{\mathbf{G}_i}(v,u)}{c(u)} \right\}. \tag{5.1}$$

For each $v \in \mathbf{R}_i$ and $u \in R(v)$ such that $f_{\mathbf{G}_i}(v,u) \neq 0$, we modify this assignment as follows. If $\pi_v^{-1}(u) \geq j_v$, then we assign it to j_v instead of to u. Otherwise, we assign it to k_v. That is, depending on whether $\pi_v^{-1}(u) \geq j_v$, we raise either $f_{\mathbf{G}_i}(v, j_v)$ or $f_{\mathbf{G}_i}(v, k_v)$ by the amount of $f_{\mathbf{G}_i}(v,u)$ and then set $f_{\mathbf{G}_i}(v,u)$ to be zero. Note that, after this reassignment, the modified assignment function $f_{\mathbf{G}_i}$ will be a feasible assignment for $\mathbf{H}_i$ as well.

In order to cope with this change, $x_{\mathbf{G}_i}(j_v)$ or $x_{\mathbf{G}_i}(k_v)$ might have to be raised as well until both the second and the third inequalities are valid again. If $\pi_v^{-1}(u) \geq j_v$, then $x_{\mathbf{G}_i}(j_v)$ is raised by at most

$$\max\left\{ \frac{f_{\mathbf{G}_i}(v,u)}{d(v)}, \frac{f_{\mathbf{G}_i}(v,u)}{c(j_v)} \right\},$$

which is equal to

$$\frac{f_{\mathbf{G}_i}(v,u)}{d(v)},$$

since $c(j_v) > d(v)$. Hence the total cost will be raised by at most

$$w(j_v) \cdot \frac{f_{\mathbf{G}_i}(v,u)}{d(v)} \leq w(j_v) \cdot \max\left\{\frac{f_{\mathbf{G}_i}(v,u)}{d(v)}, \frac{f_{\mathbf{G}_i}(v,u)}{c(u)}\right\} \leq w(u) \cdot x_{\mathbf{G}_i}(u),$$

by equation (5.1) and the fact that $w(j_v) \leq w(u)$. Similarly, if $\pi_v^{-1}(u) < j_v$, the cost is raised by at most

$$\begin{aligned} w(k_v) \cdot \max\left\{\frac{f_{\mathbf{G}_i}(v,u)}{d(v)}, \frac{f_{\mathbf{G}_i}(v,u)}{c(k_v)}\right\} &= w(k_v) \cdot \frac{f_{\mathbf{G}_i}(v,u)}{c(k_v)} \\ &\leq w(u) \cdot \frac{f_{\mathbf{G}_i}(v,u)}{c(u)} \leq w(u) \cdot x_{\mathbf{G}_i}(u), \end{aligned}$$

since we have

$$\frac{w(k_v)}{c(k_v)} \leq \frac{w(u)}{c(u)}$$

by definition of k_v and Eq. (5.1).

In both cases, the extra cost required by this specific demand reassignment between v and u is bounded by $w(u) \cdot x_{\mathbf{G}_i}(u)$. By the first part of this lemma, we have at most one such pair for each $u \notin \mathbf{R}_i$, the overall cost is at most doubled and this lemma follows. □

We also remark that, although $\mathcal{O}pt_{frac}(\mathbf{H}_i)$ is bounded in terms of $\mathcal{O}pt_{frac}(\mathbf{G}_i)$, an α-approximation for $\mathbf{H}_i$ is not necessarily a 2α-approximation for $\mathbf{G}_i$. That is, having a demand assignment function f' with $w(f') \leq \alpha \cdot \mathcal{O}pt(\mathbf{H}_i)$ does not give $w(f') \leq 2\alpha \cdot \mathcal{O}pt(\mathbf{G}_i)$, for $\mathcal{O}pt(\mathbf{H}_i)$ could be strictly larger than $\mathcal{O}pt_{frac}(\mathbf{H}_i)$. Instead, to obtain our claimed result, an approximation with a stronger bound, in terms of $\mathcal{O}pt_{frac}(\mathbf{H}_i)$, is desired.

5.3.3 Refined Charging Scheme

We show how we can further obtain the optimal solution for the reduced graph $\mathbf{H}_i$. By Lemma 5.15, we know that for each vertex u that is still demanding in $\mathbf{H}_i$, that is, for each $u \in \mathbf{R}_i$, the closed degree of u is upper-bounded by 6. Together with the primal-dual algorithm we presented in §4.4 and the charging argument provided in Lemma 4.13, this already gives a demand assignment function whose cost is upper-bounded by $7 \cdot \mathcal{O}pt_{frac}(\mathbf{H}_i)$.

Thanks to the structural property provided in Lemma 5.13, given the fact that the input graph is outer-planar, we can modify the algorithm slightly and further tighten the bound given in the previous lemma. To this end, we consider the situations when a unit demand from a vertex u with $deg[u] = 7$ and argue that, either it is not fully-charged by all its closed neighbors, or we can modify the demand assignment, without raising the cost, to make it so.

Lemma 5.16. *Given the fact that $\mathbf{H}_i$ is extracted from an outerplanar graph, we can modify the algorithm to obtain a demand assignment function f^* such that $w(f^*) \leq 6 \cdot \mathcal{O}pt_{frac}(\mathbf{H}_i)$.*

Proof. Consider any unit demand, say demand d from vertex u in $\mathbf{L}_j$, and let $\mathbf{V}_d \subseteq \mathbf{N}[u]$ be the set of vertices that has charged d by our original charging scheme.

First, we have $|\mathbf{N}[u]| \leq 7$ by Lemma 5.15. By our charging scheme, $|\mathbf{N}[u]| < 7$ implies $|\mathbf{V}_d| < 7$. In the following, we assume $|\mathbf{N}[u]| = 7$ and argue that either we have $|\mathbf{V}_d| < 7$, or we can modify the solution in a way such that $|\mathbf{V}_d| < 7$. By assumption, k_u, p_u, q_u are well-defined. Let $u_1, u_2 \in \mathbf{N}(u) \cap \mathbf{L}_j$ denote the set of neighbors of u in $\mathbf{L}_j$ such that $u_1 \prec u \prec u_2$. By Lemma 5.13, depending on the layer to which j_u and k_u belong, we have the following two cases.

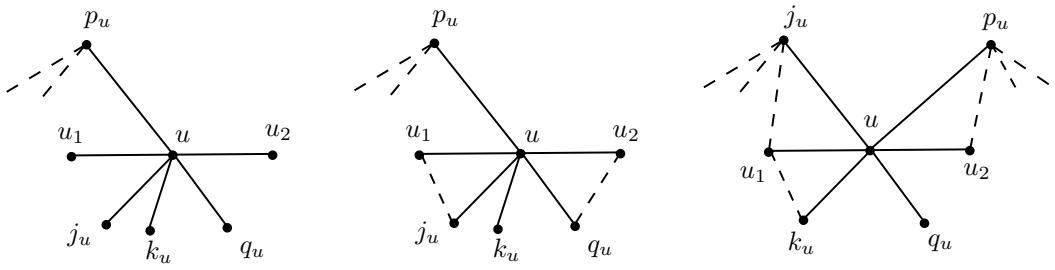


Figure 5.6: Situations when a unit demand of u is fully-charged.

(1) Both j_u and k_u belong to $\mathbf{L}_{j+1}$. If two of $\{j_u, k_u, q_u\}$, say, j_u and k_u, are not joined to u_1 and u_2 by any edge, then at most one of j_u and k_u can charge d, since u is the only vertex with possibly non-zero demand in their closed neighborhoods. When the first closed neighbor of u is saturated and u is removed from V^ϕ, both j_u and k_u will be removed from V^* and will not be picked in later iterations. Therefore, at most one of j_u and k_u can charge d.

On the other hand, if two of $\{j_u, k_u, q_u\}$, say j_u and q_u, are joined to u_1 and u_2, respectively, then we argue that at most two out of $\{j_u, u, q_u\}$ can charge d. Indeed, u_1, u, and u_2 are the only vertices with non-zero demands in the closed neighborhoods of $\{j_u, u, q_u\}$. After two of $\{j_u, u, q_u\}$ is saturated, u_1, u, and u_2 will be removed from V^ϕ. Therefore, at most two out of $\{j_u, u, q_u\}$ can charge d. See also Fig. 5.6 (a) and (b).

(2) Only one of $\{j_u, k_u\}$ belongs to $\mathbf{L}_{j+1}$ and the other belongs to $\mathbf{L}_{j-1}$. Without loss of generality, we assume that $j_u \in \mathbf{L}_{j-1}$ and $k_u \in \mathbf{L}_{j+1}$. By Lemma 5.13, both j_u and p_u are joined either to u_1 or u_2 separately. Since $\mathbf{H}_i$ is outerplanar, we have $j_u \prec u \prec p_u$, otherwise j_u will be contained inside the face surrounded by p_u, u_1, and u, which is a contradiction. See also Fig. 5.6 (c).

If both k_u and q_u are not joined to either u_1 or u_2, then by a similar argument we used in previous case, at most one of k_u and q_u can charge d. Now, suppose that, one of $\{k_u, q_u\}$, say, k_u, is joined to u_1 by an edge. We argue that, if both u_1 and k_u have charged d after d has been assigned in a feasible solution returned by our algorithm, then we can cancel the multiplicity placed on k_u and reassign to u_1 the demand which was previously assigned to k_u without increasing the cost spent on u_1.

If u_1 is lightly-loaded in the beginning, then the above operation can be done without extra cost. Otherwise, observe that, in this case, u_1 must have been lightly-loaded when d is assigned so that it can charge d later. Moreover, u_1 is also in the set V^ϕ and not yet served, for otherwise k_u will be removed from V^* and will not be picked later. In other words, at this moment when u_1 becomes lightly-loaded, we have $u_1 \in V^\phi$, meaning that it is possible to assign the demand of u_1 to itself later without extra cost.

On the other hand, if u_1 or k_u is the first one to charge d, consider the relation between q_u and u_2. If there is no edge between q_u and u_2, then q_u will not be picked and will not charge d. Otherwise, if (q_u, u_2) exists in $\mathbf{H}_i$, then it is a symmetric situation described in the last sequel.

In both cases, either we have $|\mathbf{V}_d| \leq 6$ or we can modify the solution returned by the algorithm to make $|\mathbf{V}_d| \leq 6$ to hold. Therefore we have $w(f^*) \leq 6 \cdot \mathcal{O}pt_{frac}(\mathbf{H}_i)$ as claimed. □

5.3.4 Overall Analysis

In the following, we summarize the entire algorithm and analysis followed by stating our main theorem. For the given outer-planar graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, we use the algorithm described in

§5.3.1 to compute a general-ladder representation of $\mathbf{G}$, followed by applying the decomposition to obtain three subproblems, $\mathbf{G}_0$, $\mathbf{G}_1$, and $\mathbf{G}_2$. For each $\mathbf{G}_i$, we use the approach described in §5.3.2 to further remove more edges and obtain the reduced subgraph $\mathbf{H}_i$, for which we apply the modified algorithm described in §5.3.3 to obtain an approximation, which is a demand assignment function f_i for $\mathbf{H}_i$. The overall approximation, e.g., the demand assignment function f, for $\mathbf{G}$ is defined as $f = \sum_{0 \le i < 3} f_i$.

Theorem 5.17. *Given an outerplanar graph $\mathbf{G}$ as an instance of capacitated domination, we can compute a constant factor approximation for $\mathbf{G}$ in $O(n^2)$ time, where n is the number of vertices.*

Proof. First, we argue that the procedures we describe can be done in $O(n^2)$ time. It takes $O(n \log^3 n)$ time to compute an outer-planar representation of $\mathbf{G}$ [16]. By Theorem 5.12, computing a general-ladder representation takes linear time. The construction of $\mathbf{G}_i$ takes time linear in the number of edges, which is linear in the number of vertices since $\mathbf{G}$ is planar.

In the construction of the reduced graphs $\mathbf{H}_i$, for each vertex v, although we use a sorted list of the closed neighborhood of v to define j_v, k_v, p_v, and q_v, the sorted lists are not necessary and $\mathbf{H}_i$ can be constructed in $O(n)$ time by a careful implementation. This is done by a two-passes traversal on the set of edges of $\mathbf{G}_i$ as follows.

- (a) In the first pass, we iterate over the set of edges to locate j_v, p_v, and q_v, for each vertex $v \in V$. Specifically, we keep a current candidate for each vertex and for each edge $(u, v) \in E$ iterated, we make an update on u and v if necessary.
- (b) In the second pass, based on the j_v computed for each $v \in V$, we iterate over the set of edges again to locate k_v.

The whole process takes time linear in the number of edges, which is $O(n)$ since $\mathbf{G}$ is a planar graph.

In the following, we explain how the primal-dual algorithm, i.e., the algorithm presented in Fig. 4.4, can be implemented to compute a feasible solution in $O(n^2)$ time. First, we traverse the set of edges in linear time to compute the value $d^\phi(v)$ for each vertex. In each iteration, the next vertex to be saturated, which is the one with minimum $w^\phi(v)/\min\{c(v), d^\phi(v)\}$, can be found in linear time. The update of $w^\phi(v)$ for each $v \in V^*$ described in line 8 can be done in linear time. When a vertex $v \in \mathbf{N}[u]$ with non-zero demand is removed from $\mathbf{V}^\phi$, we have to update the value $d^\phi(v')$ for all $v' \in \mathbf{N}[v]$. By Lemma 5.15, the closed degree of such vertices is bounded by 7. This update can be done in $O(1)$ time. The construction of $\mathbf{S}_u$ can be done in linear time. Since $d^\phi(v)$ can only decrease, each vertex can turn into lightly-loaded at most once. Therefore the process time for these vertices is bounded in linear time. The outer-loop iterates at most $O(n)$ times. Therefore the whole algorithm runs in $O(n^2)$ time.

The feasibility of the demand assignment function f is guaranteed by Lemma 5.14 and the fact that $\mathbf{H}_i$ is a subgraph of $\mathbf{G}_i$. Since $\mathbf{H}_i \subseteq \mathbf{G}_i$, the demand assignment we obtained for $\mathbf{H}_i$ is also a feasible demand assignment for $\mathbf{G}_i$. Therefore, f is feasible for G.

By the definition of f, Lemma 5.16, Lemma 5.15, and Lemma 5.14, we have

$$\begin{aligned} w(f) &\le \sum_{0 \le i < 3} w(f_i) \\ &\le 6 \cdot \sum_{0 \le i < 3} \mathcal{O}pt_{frac}(\mathbf{H}_i) \quad \le 12 \cdot \sum_{0 \le i < 3} \mathcal{O}pt_{frac}(\mathbf{G}_i) \\ &\le 12 \cdot \sum_{0 \le i < 3} \mathcal{O}pt(\mathbf{G}_i) \le 36 \cdot \mathcal{O}pt(G). \end{aligned}$$

□

Chapter 6

Planar Graphs

In this chapter, we discuss the problem complexity on planar graphs. Specifically, we show how the algorithms presented in preceding chapters can be extended to obtain approximations for planar graphs under a general framework due to [9]. On the other hand, for the negative side, we show that it is impossible to approximate this problem with inseparable demand to the factor of $\left(\frac{3}{2}-\epsilon\right)$ for any $\epsilon > 0$, unless $\mathbf{P} = \mathbf{NP}$.

6.1 A Well-Known Framework - from Bounded Treewidth to Planar

Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a planar graph. We generate a planar embedding and retrieve the vertices of each level using the linear-time algorithm of Hopcroft and Tarjan [46]. Let m be the number of levels in this embedding.

Let $\mathcal{O}pt(\mathbf{G})$ be the cost of the optimal demand assignment function of $\mathbf{G}$, and $\mathcal{O}pt_j(\mathbf{G})$ be the cost contributed by vertices at level j. For the ease of presentation, in the following, for $j \leq 0$ or $j > m$, we refer the vertices in level j to an empty set and the corresponding cost $\mathcal{O}pt_j(\mathbf{G})$ is defined to be zero.

Let $k \geq 3$ be a constant to be determined. For $0 \leq i < k$, we define $\mathcal{C}_i$ as

$$\mathcal{C}_i = \sum_{0 \leq j \leq \frac{m}{k}} \left(\mathcal{O}pt_{k \cdot j - i}(\mathbf{G}) + \mathcal{O}pt_{k \cdot (j+1) - i - 1}(\mathbf{G})\right).$$

Note that, we have $\sum_{0 \leq i < k} \mathcal{C}_i \leq 2 \cdot OPT$. Hence there exist an i_0 with $0 \leq i_0 < k$ such that

$$\mathcal{C}_{i_0} \leq \frac{2}{k} \cdot OPT.$$

For each $0 \leq j \leq \frac{m}{k}$, define the graph $\mathbf{G}_j$ to be the graph induced by vertices between level $k \cdot j - i_0$ and level $k \cdot (j+1) - i_0 - 1$. The parameters of the vertices in $\mathbf{G}_j$ are set as follows. For those vertices which are from level $k \cdot j - i_0$ and level $k \cdot (j+1) - i_0 - 1$, their demands are set to be zero. The remaining parameters remain unchanged.

Clearly, $\mathbf{G}_j$ is a k-outerplanar graph, which has treewidth at most $k+1$ [9], and we have

$$\sum_{0 \leq j \leq \frac{m}{k}} \mathcal{O}pt(\mathbf{G}_j) \leq \left(1 + \frac{2}{k}\right) \cdot \mathcal{O}pt(\mathbf{G}),$$

where $\mathcal{O}pt(\mathbf{G}_j)$ is the cost of the optimal demand assignment of $\mathbf{G}_j$. The following theorem follows directly from Theorem 5.6, Theorem 5.7, Lemma 4.8, and the above discussion.

Theorem 6.1. *Given a planar graph* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *and an integer* $k \geq 3$*, we can compute the following approximations for the capacitated domination problem:*

- $\left(1 + \frac{2}{k}\right)$*-approximation in* $O\left(2^{2k(\log \mathcal{M}+1)+2\log k} \cdot n\right)$ *time for inseparable demand model, where* n *is the number of vertices and* $\mathcal{M}$ *is the maximum capacity of the vertex set.*
- $\left(1 + \frac{2}{k}\right)$*-approximation in time* $O\left(2^{(2\mathcal{M}+2\mathcal{N}+1)\log k} \cdot n\right)$ *for separable demand model, where* $\mathcal{N}$ *is the maximum demand of the vertex set.*
- $\left(2 + \frac{2}{k}\right)$*-approximation in time* $O\left(2^{(4\mathcal{M}+1)\log k} \cdot n\right)$ *for separable demand model.*

Proof. The first two follows directly from Theorem 5.6 and Theorem 5.7. For the last one, we apply the framework provided in §4.3, whose cost is at most $\mathcal{O}pt(\mathbf{G})$, and we have $d(u) < \mathcal{M}$ for all $u \in \mathbf{V}$ afterwards. This proves the theorem. □

6.2 A Constant Factor Approximations for Separable Demand

We show how the approximation algorithm presented in §5.3 for outerplanar graphs can be adapted to fit into the proposed framework in §6.1. As the algorithm is designed mainly for outerplanar graphs, to meet the minimum requirement of the framework, which is the ability to deal with planar graphs of at least three levels, we have to modify our algorithm to undertake this change.

Let $\mathbf{G}_j$, $0 \leq j \leq \frac{m}{3}$, be the 3-outerplanar graphs obtained from the framework. It suffices to show how each $\mathbf{G}_j$ can be handled separately. Consider a specific component, say, $\mathbf{G}_{j'}$, $0 \leq j' \leq \frac{m}{3}$. We describe how our approximation algorithm for outerplanar graphs can be modified accordingly and applied to $\mathbf{G}_{j'}$ to obtain a constant approximation.

For the ease of presentation, we denote the set of vertices from the three levels of $\mathbf{G}_{j'}$ by $\mathbf{L}_0$, $\mathbf{L}_1$, and $\mathbf{L}_2$, respectively. See also Fig. 6.1 (a) for an illustration. Note that, from our construction scheme, only vertices in $\mathbf{L}_1$ have non-zero demand.

(a) Obtaining the General Ladder. We extract the general ladder from $\mathbf{L}_1$ as described in §5.3.1. When decomposing the ladder, for each vertex of the ladder, its incident edges to vertices in $\mathbf{L}_0$ and $\mathbf{L}_2$ are also included in addition to the ladder itself. Edges connecting vertices in $\mathbf{L}_2$ and edges connecting vertices in $\mathbf{L}_0$ are discarded. As vertices in $\mathbf{L}_2$ and $\mathbf{L}_0$ are non-demanding, discarding these edges does not alter the optimal demand assignment.

Figure 6.1: (a) A 3-outerplanar graph with levels $\mathbf{L}_0$, $\mathbf{L}_1$, and $\mathbf{L}_2$. (b) Local connections w.r.t. a vertex v. Bold edges represent edges to be kept in the edge reduction process, while dashed edges represent edges to be removed.

(b) Removing Edges. In addition to the four neighboring vertices we identified for each vertex v with non-zero demand, we identify two more vertices, which literally corresponds to the rightmost neighbors of v in $\mathbf{L}_0$ and $\mathbf{L}_2$, respectively. See also Fig. 6.1 (b).

As a result, the first part and the third part of Lemma 5.15 still hold, and the degree upper-bound provided in the second part is increased by 2. The remaining part of our algorithm, which computes a constant factor approximation for the reduced ladder, remains unchanged.

The assignment function computed remains feasible as the algorithm is independent to the ladder structure while the approximation factor provided in Lemma 5.16 increases by 2. We conclude our result in the following theorem.

Theorem 6.2. *Given a planar graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, we can compute a constant factor approximation for the capacitated domination problem with separable demand for $\mathbf{G}$ in polynomial time.*

Proof. Let $\mathbf{G}_j$, $0 \leq j \leq \frac{m}{3}$ be the 3-outerplanar graphs obtained from the framework of § 6.1, and denote by $\mathcal{G}_{j,i}$ and $\mathcal{H}_{j,i}$, $0 \leq i < 3$, the general ladders and the corresponding reduced ladders obtained from the above discussion. Let $f_{j,i}$ be the demand assignment function computed by the modified algorithm proposed in §5.3.3 for the reduced ladder $\mathcal{H}_{j,i}$ and $f = \sum_{0 \leq j \leq \frac{m}{3}} \sum_{0 \leq i < 3} f_{j,i}$ be the demand assignment function for the entire graph.

We have

$$\begin{aligned}
w(f) &\leq \sum_{0 \leq j \leq \frac{m}{3}} \sum_{0 \leq i < 3} w(f_{j,i}) \\
&\leq 9 \cdot \sum_{0 \leq j \leq \frac{m}{3}} \sum_{0 \leq i < 3} \mathcal{O}pt_{frac}(\mathcal{H}_{j,i}) \\
&\leq 18 \cdot \sum_{0 \leq j \leq \frac{m}{3}} \sum_{0 \leq i < 3} \mathcal{O}pt_{frac}(\mathcal{G}_{j,i}) \\
&\leq 18 \cdot \sum_{0 \leq j \leq \frac{m}{3}} \sum_{0 \leq i < 3} \mathcal{O}pt(\mathcal{G}_{j,i}) \\
&\leq 54 \cdot \sum_{0 \leq j \leq \frac{m}{3}} \mathcal{O}pt(\mathbf{G}_j) \\
&\leq 90 \cdot \mathcal{O}pt(\mathbf{G}),
\end{aligned}$$

where the last inequality follows from the framework of § 6.1. □

6.3 $\left(\frac{3}{2} - \epsilon\right)$-Approximation Threshold for Inseparable Demand

Below we show that, when the demand is inseparable, it is $\mathcal{NP}$-hard to approximate this problem within a factor of $\left(\frac{3}{2} - \epsilon\right)$, for any $\epsilon > 0$. The reduction is made from *Partition*, which is a well-known combinatorial **NP**-hard problem.

Definition 6.3 (Partition Problem)**.** Given a sequence of positive integers $a_1, a_2, \ldots, a_n$, the partition problem is to decide whether or not there exists a subset $\mathbf{A} \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in \mathbf{A}} a_i = \sum_{1 \leq i \leq n, i \notin \mathbf{A}} a_i$.

Given a problem instance $\mathbf{I} = \{a_1, a_2, \ldots, a_n\}$ of *Partition*, we construct a planar graph $\mathbf{G}$ with $n + 4$ vertices as follows. For each $1 \leq i \leq n$, we create a vertex v_i with demand a_i and capacity 1. We create two vertices v_ℓ and v_r with demand 1 and capacity $\sum_{1 \leq i \leq n} a_i + 1$, and

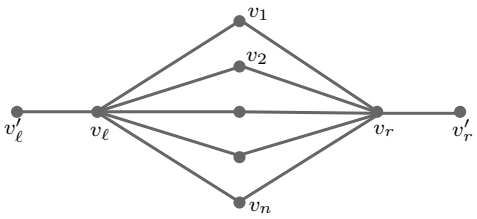


Figure 6.2: The construction of the reduction from *Partition*.

connect v_ℓ and v_r to each v_i, $1 \leq i \leq n$. We create two additional vertices v'_ℓ and v'_r, which are connected to v_ℓ and v_r, respectively, with demand $\frac{1}{2}\sum_{1\leq i\leq n} a_i$ and capacity 1. The weight for each vertex is set to be 1. See also Fig. 6.2 for an illustration.

Theorem 6.4. *For any $\epsilon > 0$, there exists no approximation algorithm which approximate the capacitated domination problem with inseparable demand on planar graphs to the factor of $\left(\frac{3}{2} - \epsilon\right)$, unless* $\mathbf{P} = \mathbf{NP}$.

Proof. In the following, we argue that, there exists a subset $\mathbf{A}$ satisfying the criterion of partition problem if and only if the cost of the optimal demand assignment of $\mathbf{G}$ is at most 2. As a consequence, any algorithm that approximates $\mathbf{G}$ within the factor of $\frac{3}{2} - \epsilon$ would imply the correct decision for $\mathbf{I}$.

If there exists a subset $\mathbf{A} \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i\in\mathbf{A}} a_i = \sum_{1\leq i\leq n, i\notin\mathbf{A}} a_i$, then we assign the demand of v'_ℓ and v_i, for each $i \in \mathbf{A}$ to v_ℓ, and assign the demand of v'_r and v_i, for each $1 \leq i \leq n$, $i \notin \mathbf{A}$, to v_r. The total cost required is 2.

On the other hand, if the cost of the optimal demand assignment of $\mathbf{G}$ is 2, then we argue that there will exist a subset $\mathbf{A}$ satisfying the criterion. First, it is easy to see that the demand of v'_ℓ and the demand of v'_r must have been assigned to v_ℓ and v_r, respectively. This leaves $\frac{1}{2}\sum_{1\leq i\leq n} a_i$ residue capacity at both v_ℓ and v_r. Let $\mathbf{A} = \{i : \text{the demand of } v_i \text{ is assigned to } v_\ell\}$. Then we have

$$\sum_{i\in\mathbf{A}} a_i = \sum_{1\leq i\leq n, i\notin\mathbf{A}} a_i.$$

□

Chapter 7

Other Algorithmic Results for Trees

This chapter presents both complexity and algorithmic results for the capacitated domination problem on trees. In particular, for the inseparable demand model, we present a linear time algorithm in §7.1. This matches the complexity of the dominating set problem on trees. In contrast, we show in §7.2 that when the demand is separable, this problem is **NP**-complete in trees. Based on the reduction in the hardness proof, we further characterize and tackle the difficulty of this problem and provide a polynomial time approximation scheme for separable demand in §7.3.

7.1 A Linear Time Algorithm for Inseparable Demand *

7.2 NP-Completeness for Separable Demand

Unlike the dominating set problem, which is linear time solvable in trees by a standard bottom-up traversing approach, capacitated domination is **NP**-hard on trees. The main reason comes from the problem nature when we have to assign the demands of a set of vertices which share a common parent in a way such that the utilization of the capacity on the parent is optimal. The following reduction demonstrates this observation and completes the **NP**-hardness proof on trees. We start with the definition of the *subset sum* problem, which is well-known as a **NP**-complete problem [36].

Definition 7.1 (Subset Sum)**.** Given a sequence of positive integers $a_1, a_2, \ldots, a_n$ and a target integer W, the subset sum problem is to determine the existence of a subset A of $\{1, 2, \ldots, n\}$ such that $\sum_{i \in A} a_i = W$.

Given an instance $\mathbf{I}$ of subset sum, we build an instance $\mathbf{T}(\mathbf{I})$ for the capacitated domination as follows. Let $\mathcal{M} = \left(\sum_{1 \le i \le n} a_i\right) + 1$, $\mathcal{W}' = \mathcal{M} \cdot W$, and $a'_i = \mathcal{M} \cdot a_i$, for $1 \le i \le n$. $\mathbf{T}(\mathbf{I})$ is a tree consisting of $n + 2$ vertices, $v_0, v_1, \ldots, v_{n+1}$. v_{n+1} is the root vertex with capacity 1 and demand W. v_0 is the only child vertex of v_{n+1} with capacity $\mathcal{M}$ and demand $\sum_{1 \le i \le n} a'_i - \mathcal{W}'$. $v_1, v_2, \ldots, v_n$ are children vertices of v_0 and have capacity $\mathcal{M} + a'_i - a_i$ and demand $\mathcal{M} - a_i$. Finally, each vertex has unit cost, 1.

Lemma 7.2. *Let* $\mathbf{D}$ *be a feasible capacitated dominating multi-set of* $\mathbf{T}(\mathbf{I})$*, and let*

$$\mathbf{A} = \{i : 1 \le i \le n, v_i \notin \mathbf{D}\}.$$

Then we have $x_{\mathbf{D}}(v_0) \ge |\mathbf{A}|$.

Proof. Suppose that $x_{\mathbf{D}}(v_0) < |\mathbf{A}|$. By the capacity constraint of v_0 we have

$$x_{\mathbf{D}}(v_0) \cdot \mathcal{M} \ge \sum_{i \in \mathbf{A}} (\mathcal{M} - a_i),$$

which leads to

$$\sum_{i \in \mathbf{A}} a_i - (|\mathbf{A}| - x_{\mathbf{D}}(v_0))\,\mathcal{M} \geq 0,$$

a contradiction since $|\mathbf{A}| > x_{\mathbf{D}}(v_0)$ and $\mathcal{M} > \sum_{i \in \mathbf{A}} a_i$. □

Lemma 7.3. *The cardinality of any feasible capacitated dominating multi-set of* $\mathbf{T}(\mathbf{I})$ *is no less than* n.

Proof. Consider the vertex v_i for any $1 \leq i \leq n$. Any feasible demand assignment can only assign the demand of v_i either to v_0 or to v_i itself. By Lemma 7.2, in both cases it requires at least one copy in order to serve the demand of v_i. □

Lemma 7.4. *Provided* $\sum_{1 \leq i \leq n} a_i \geq W$, *there exists a subset* $\mathbf{A}$ *of* $\{1, 2, \ldots, n\}$ *such that* $\sum_{i \in A} a_i = W$ *if and only if there exists a feasible capacitated dominating multi-set of cardinality* n *for* $\mathbf{T}(\mathbf{I})$.

Proof. If there is a subset $\mathbf{A}$ of $\{1, 2, \ldots, n\}$ satisfying $\sum_{i \in \mathbf{A}} a_i = W$, then we can construct a capacitated dominating multi-set $\mathbf{D}$ as follows. For each $i \in \mathbf{A}$, we have a multiplicity of v_0 and assign the demand of v_i to v_0. Each such assignment leaves a residue capacity of a_i at v_0. We also assign the demand of v_{n+1} to v_0 since $\sum_{i \in \mathbf{A}} a_i = W$. For $i \notin \mathbf{A}$, we have a copy of v_i and assign the demand of v_i to itself. Each such assignment leaves a residue capacity of a'_i at v_i. Since the demand of v_0 is

$$\sum_{1 \leq i \leq n} a'_i - \mathcal{W}' = \sum_{i \notin \mathbf{A}} a'_i,$$

we can assign the demand of v_0 to vertices whose indexes belong to $\{1, 2, \ldots, n\} \backslash \mathbf{A}$. Thus, we have a feasible capacitated dominating multi-set $\mathbf{D}$ of cardinality n.

On the other hand, if there is a feasible dominating multi-set $\mathbf{D}$ of cardinality n for $\mathbf{T}(\mathbf{I})$, we define $\mathbf{A}$ as $\{i : 1 \leq i \leq n, v_i \notin \mathbf{D}\}$. By Lemma 7.2, we have at least $|\mathbf{A}|$ copies of v_0 and one copy for each vertex whose index belongs to $\{1, 2, \ldots, n\} \backslash \mathbf{A}$. Therefore, $v_{n+1} \notin \mathbf{D}$, which implies that the demand of v_{n+1} is assigned to v_0 and $\sum_{i \in A} v_i \geq W$. The demand of v_0 is served by the residue capacities of v_0 and v_i for $i \notin \mathbf{A}$, which is

$$\left(\sum_{i \notin \mathbf{A}} a'_i\right) + \left(\sum_{i \in \mathbf{A}} a_i - W\right).$$

Therefore

$$\sum_{1 \leq i \leq n} a'_i - \mathcal{W}' \leq \sum_{i \notin \mathbf{A}} a'_i + \sum_{i \in \mathbf{A}} a_i - W,$$

which leads to $\sum_{i \in \mathbf{A}} a_i \leq W$. Hence we have $\sum_{i \in \mathbf{A}} a_i = W$. This completes the proof. □

Theorem 7.5. *The capacitated domination problem with separable demand model is* **NP***-complete even on trees.*

Proof. We argue in the following that this problem is in **NP**. Given a capacitated dominating multi-set, we can verify the feasibility by a standard bottom-up tree traversal, greedily assigning the demand of a vertex, say v, in child-first, parent-last manner. That is, whenever there is residue capacity in the children vertex of v, we greedily assign the demand of v to its children. If there is still residue demand, we assign the demand to v itself. Finally, we assign the demand, if there is still any, to the parent of v. The given set is infeasible if the demand of any vertex fails to be assigned.

For any instance $\mathbf{I}$ of subset sum problem, if $\sum_{1 \leq i \leq n} a_i < W$, we know immediately that there is no subset of $\{1, 2, \ldots, n\}$ can fulfil the requirement of subset sum and the answer is no. Otherwise, by Lemma 7.4, the answer is yes if and only if the cardinality of the optimal dominating multi-set for $\mathbf{T}(\mathbf{I})$ is at most n. This completes the proof. □

7.3 A Polynomial Time Approximation Scheme for Separable Demand

The reduction we provided in §7.2 gives a precise hint on where the main difficulty of this problem is when we want to compute an optimal demand assignment function for trees. In this section, we characterize this bottleneck and formulate it as a combinatorial optimization problem named *Relaxed Knapsack Problem*, for which we provide both a pseudo-polynomial time algorithm and a fully-polynomial time approximation scheme. Then, we show that the capacitated domination problem with separable demand model on trees is pseudo-polynomial time solvable and adopts a polynomial time approximation scheme.

7.3.1 Relaxed Knapsack Problem

Below we provide the formal definition to the relaxed knapsack problem. Then we present a dynamic program which computes an optimal solution for this problem in pseudo-polynomial time.

Definition 7.6 (Relaxed Knapsack Problem)**.** Given m pairs of non-negative integers (a_i, b_i), $1 \leq i \leq m$, and a non-negative integer W, where a_i and b_i denote the size and the profit of the i-th item and W is the packet size, the relaxed knapsack problem asks for a subset $\mathbf{A} \subseteq \{1, 2, \ldots, m\}$ such that

$$\sum_{i \in \mathbf{A}} b_i - \max\left\{0, \sum_{i \in \mathbf{A}} a_i - W\right\}$$

is maximized.

Intuitively, this problem extends the concept of the well-known *Knapsack Problem* in a sense that we are packing a set of items to maximize the total profit of the items we packed, except that, in this problem we are given a soft limit on the packet size and we allow the size of the items we packed to exceed this soft limit at the cost of a certain amount of penalty in the profit.

In the following, we present a dynamic program which solves this problem optimally. The central idea is similar to the well-known dynamic program for the knapsack problem except that we have more cases to consider in this problem.

Let $\mathcal{Q}(k, p)$ denote the minimum total size among all possible combinations of the first k items which exactly achieve a total profit p. If no such combination exists, $\mathcal{Q}(k, p)$ is defined to be ∞. As an initial condition, we let $\mathcal{Q}(0, p) = \infty$ for all $0 \leq p \leq m\mathcal{M}$, where $\mathcal{M} = \max_{1 \leq i \leq m} b_i$.

For each (k, p) with $1 \leq k \leq m$ and $0 \leq p \leq m\mathcal{M}$, we compute $\mathcal{Q}(k, p)$ based on $\mathcal{Q}(k-1, q)$ for each $0 \leq q \leq m\mathcal{M}$. Depending on all possible configurations on the k^{th} item and the total size, we have the following cases:

(1) the k-th item is not picked. In this case, we have

$$\mathcal{Q}(k, p) = \mathcal{Q}(k-1, p).$$

(2) the k-th item is picked, and the total size does not exceed W. In this case,

$$\mathcal{Q}(k, p) = \mathcal{Q}_1(k, p), \text{ where}$$

$$\mathcal{Q}_1(k, p) = \begin{cases} \mathcal{Q}(k-1, p-b_k) + a_k, & \text{if } p \geq b_k \text{ and } \mathcal{Q}(k-1, p-b_k) + a_k \leq W \\ \infty, & \text{otherwise.} \end{cases}$$

(3) the k-th item is picked and the total size exceeds W. For this case,

$$\mathcal{Q}(k,p) = \mathcal{Q}_2(k,p), \text{ where}$$

$$\mathcal{Q}_2(k,p) = \min \begin{cases} \infty, \\ \mathcal{Q}(k-1,q), & \text{for each } q \text{ with } 0 \le q \le m\mathcal{M}, \text{ such that} \\ & \mathcal{Q}(k-1,q) + a_k > W \quad \text{and} \\ & p = (q+b_k) - (\mathcal{Q}(k-1,q) + a_k - W) \end{cases}$$

The recurrence relation of $\mathcal{Q}(k,p)$ is defined as

$$\mathcal{Q}(k,p) = \min\left\{\mathcal{Q}(k-1,p), \mathcal{Q}_1(k,p), \mathcal{Q}_2(k,p)\right\},$$

where $\mathcal{Q}_1(k,p)$ and $\mathcal{Q}_2(k,p)$ are defined as above in cases (2) and (3), respectively.

The algorithm iteratively computes $\mathcal{Q}(k,p)$ for $1 \le k \le m$. After $\mathcal{Q}(m,p)$ is computed for each $0 \le p \le m\mathcal{M}$, the algorithm outputs the maximum p such that $0 \le p \le m\mathcal{M}$ and $\mathcal{Q}(m,p) < \infty$. By maintaining another table of the same dimension to record the recursive decision we made during the computation of table $\mathcal{Q}$, we can also output the corresponding subset $\mathbf{A}$ that maximizes the total profit as well.

Theorem 7.7. *We can compute the optimal solution for the relaxed knapsack problem in pseudo-polynomial time $O(m^2\mathcal{M})$ time, where m is the number of items and $\mathcal{M}$ is the maximum profit of the items.*

Proof. The correctness is obvious by an inductive argument. The base case when $k = 0$ is correct for all p with $0 \le p \le m\mathcal{M}$. For $k > 1$, there are three possibilities depending on the choices of the k-th item and the total size, which we have considered in the recurrence formula. Therefore, this approach correctly computes an optimal solution for relaxed knapsack problem.

Regarding the time complexity, a naive approach would lead to an $O\left(m^3\mathcal{M}^2\right)$ running time. We can, however, reduce the time by pre-considering the third case in the recurrence formula. To be precise, when $\mathcal{Q}(k,p)$ is computed, we check whether $\mathcal{Q}(k,p) + a_{k+1}$ exceeds W or not. If it does, we update $\mathcal{Q}_2\left(k+1, p+b_{k+1} - (\mathcal{Q}(k,p) + a_{k+1} - W)\right)$ in advance. Otherwise, we do nothing. This reduces the time cost of the third case to amortized constant time for computing each entry. Therefore the overall time complexity is $O\left(m \cdot m\mathcal{M}\right) = O\left(m^2\mathcal{M}\right)$. □

7.3.2 A Fully-Polynomial Time Approximation Scheme for the Relaxed Knapsack Problem

Let $\mathbf{I} = \left\{\bigcup_{1\le i\le m}(a_i, b_i), \mathcal{W}\right\}$ be an instance of the relaxed knapsack problem, where a_i and b_i are the size and the profit of the i-th item, $1 \le i \le n$, and $W > 0$ is the packet size. Below we show how we can obtain an $(1-\epsilon)$-approximation for this problem, for any $\epsilon > 0$. The idea is to properly rescale the input followed by applying the dynamic programming algorithm presented in §7.3.1.

Let $\mathcal{M} = \max_{1\le i\le n} b_i$ be the maximum profit and

$$k = \frac{\epsilon M}{2n+1}.$$

We create a new problem instance $\mathbf{I}'$ for relaxed knapsack with the same number of items as follows. Let

$$a'_i = \left\lceil \frac{a_i}{k} \right\rceil, \quad b'_i = \left\lfloor \frac{b_i}{k} \right\rfloor, \quad \text{and} \quad W' = \left\lfloor \frac{W}{k} \right\rfloor.$$

Then we apply the dynamic program proposed in §7.3.1 on the new instance $\mathbf{I}'$ and return the solution for $\mathbf{I}'$ as the output. Note that, without loss of generality, we may assume that $k > 1$ since if it is not the case, we simply apply the dynamic program without rescaling. The solution we obtained will be optimal and the time required will be no more than that required by this approach. We start with the following proposition, which follows from the elementary arithmetic.

Proposition 7.8. For any $r, s \in \mathbb{R}$ with $s > 0$, we have

$$r \leq s \cdot \left\lceil \frac{r}{s} \right\rceil \leq r + s \quad \text{and} \quad r - s \leq s \cdot \left\lfloor \frac{r}{s} \right\rfloor \leq r.$$

For any subset $\mathbf{A}$ of $\{1, 2, \ldots, m\}$, we denote the total profit of $\mathbf{A}$ with respect to $\mathbf{I}$ by $\mathcal{P}(\mathbf{A})$ and the total profit of $\mathbf{A}$ with respect to $\mathbf{I}'$ by $\mathcal{P}'(\mathbf{A})$. Formally,

$$\mathcal{P}(\mathbf{A}) = \sum_{i \in \mathbf{A}} b_i - \max\left\{0, \sum_{i \in \mathbf{A}} a_i - W\right\}$$

and

$$\mathcal{P}'(\mathbf{A}) = \sum_{i \in \mathbf{A}} b'_i - \max\left\{0, \sum_{i \in \mathbf{A}} a'_i - W'\right\}.$$

Let $\mathbf{O}pt$ and $\mathbf{O}pt^*$ be the optimal subsets with respects to instance $\mathbf{I}$ and instance $\mathbf{I}'$, respectively. We have the following lemma.

Lemma 7.9.

$$\mathcal{P}(\mathbf{O}pt^*) \geq (1 - \epsilon) \cdot \mathcal{P}(\mathcal{O}pt).$$

Proof. Consider an arbitrary subset $\mathbf{A}$ of $\{1, 2, \ldots, m\}$. By Proposition 7.8 and the definition of $\mathcal{P}$, we know that

$$\sum_{i \in \mathbf{A}} k \left\lceil \frac{a_i}{k} \right\rceil - k \left\lfloor \frac{W}{k} \right\rfloor \geq \sum_{i \in \mathbf{A}} a_i - W,$$

and

$$\begin{aligned}
0 &\leq \mathcal{P}(\mathbf{A}) - k \cdot \mathcal{P}'(\mathbf{A}) \\
&= \sum_{i \in \mathbf{A}} \left(b_i - k \left\lfloor \frac{b_i}{k} \right\rfloor \right) + \left(\max\left\{0, \sum_{i \in \mathbf{A}} k \left\lceil \frac{a_i}{k} \right\rceil - k \left\lfloor \frac{W}{k} \right\rfloor \right\} - \max\left\{0, \sum_{i \in \mathbf{A}} a_i - W \right\} \right) \\
&\leq nk + \left(\max\left\{0, \sum_{i \in \mathbf{A}} k \left\lceil \frac{a_i}{k} \right\rceil - k \left\lfloor \frac{W}{k} \right\rfloor \right\} - \max\left\{0, \sum_{i \in \mathbf{A}} a_i - W \right\} \right).
\end{aligned}$$

Consider the last item in the above inequality. Since

$$\sum_{i \in \mathbf{A}} k \left\lceil \frac{a_i}{k} \right\rceil - k \left\lfloor \frac{W}{k} \right\rfloor \geq \sum_{i \in \mathbf{A}} a_i - W,$$

$\sum_{i\in A} k\left\lceil \frac{a_i}{k}\right\rceil - k\left\lfloor \frac{W}{k}\right\rfloor \le 0$ would imply $\sum_{i\in \mathbf{A}} a_i - W \le 0$. Therefore, we have

$$\begin{aligned}
& \max\left\{0, \sum_{i\in\mathbf{A}} k\left\lceil\frac{a_i}{k}\right\rceil - k\left\lfloor\frac{W}{k}\right\rfloor\right\} - \max\left\{0, \sum_{i\in\mathbf{A}} a_i - W\right\} \\
\le\ & \left(\sum_{i\in\mathbf{A}} k\left\lceil\frac{a_i}{k}\right\rceil - k\left\lfloor\frac{W}{k}\right\rfloor\right) - \left(\sum_{i\in\mathbf{A}} a_i - W\right) \\
\le\ & (n+1)k.
\end{aligned}$$

Hence, we obtain

$$0 \le \mathcal{P}(\mathbf{A}) - k\cdot\mathcal{P}'(\mathbf{A}) \le nk + (n+1)k = k(2n+1). \tag{7.1}$$

From Eq. 7.1 and the definition of $\mathbf{O}pt^*$, we have

$$\begin{aligned}
\mathcal{P}(\mathbf{O}pt^*) &\ge k\cdot\mathcal{P}'(\mathbf{O}pt^*) \ge k\cdot\mathcal{P}'(\mathbf{O}pt) \\
&\ge \mathcal{P}(\mathbf{O}pt) - k\cdot(2n+1) \\
&= \mathcal{P}(\mathbf{O}pt) - \epsilon\cdot M \\
&\ge (1-\epsilon)\mathcal{P}(\mathbf{O}pt).
\end{aligned}$$

□

Theorem 7.10. *For any $\epsilon > 0$, we can compute an $(1-\epsilon)$-approximation for the relaxed knapsack problem in $O\left(n^2\left\lfloor\frac{n}{\epsilon}\right\rfloor\right)$ time.*

Proof. By Lemma 7.9, the profit of the solution obtained is at least $\mathcal{P}(\mathbf{O}pt)$. By Theorem 7.7, the running time of the algorithm on instance $\mathbf{I}'$ is $O\left(n^2\left\lfloor\frac{\mathcal{M}}{k}\right\rfloor\right) = O\left(n^2\left\lfloor\frac{n}{\epsilon}\right\rfloor\right)$. □

7.3.3 A Pseudo-Polynomial Time Algorithm **

We present below an algorithm which optimally solves the capacitated domination problem for trees in pseudo-polynomial time. The algorithm is built upon a standard bottom-up tree traversal, the results we provided in §7.3.1, and case analysis.

Suppose we are given a tree $T = (V, E)$ with a post-order traversal on the set of vertices $\{v_1, v_2, \ldots, v_n\}$. For any $v \in V$, let T_v denote the subtree rooted at v. During the process, the algorithm maintains for each vertex, say v_i, its residue capacity and residue demand, denoted by $RC[v_i]$ and $RD[v_i]$, respectively. Initially, $RC[v_i] = 0$ and $RD[v_i] = d(v_i)$ for each $1 \le i \le n$. The algorithm iterates on i, $1 \le i \le n$, processing one vertex at a time. At the end of iteration i, $1 \le i \le n$, the following conditions hold:

- The demand of the children of v_i are fully-served and assigned optimally.
- Either $RC[v_i]$ or $RD[v_i]$ is zero. If $RD[v_i]$ is zero, then the demand of v_i is also assigned optimally.
- If $RD[v_i] \neq 0$, then v_i is not root, i.e., $i < n$, and $RD[v_i] \le c(p(v_i)) < c(v_i)$, where $p(v_i)$ is the parent vertex of v_i.

At iteration i, vertex v_i is considered. Let $u_1, u_2, \ldots, u_k$ be the children of v_i. Let $\mathcal{P} \subseteq \{u_1, u_2, \ldots, u_k\}$ be the set of children whose residue demand is non-zero. Note that, by the induction hypothesis, we have $RD[v] \le c(v_i) < c(v)$ for all $v \in \mathcal{P}$. The algorithm first determines the demand assignments of vertices in $\mathcal{P}$ in an optimal way such that either $RD[v_i]$ is fully-served and $RC[v_i]$ is maximized, or the largest portion of $RD[v_i]$ is served. Depending on whether $RD[v_i] \le \sum_{v\in\mathcal{P}}(c(v) - RD[v])$, we have two cases.

1. If $RD[v_i] > \sum_{v \in \mathcal{P}} (c(v) - RD[v])$, then it is impossible to fully-serve $RD[v_i]$. In this case, we assign $RD[v]$ to the vertex v itself, for each $v \in \mathcal{P}$, and assign $c(v) - RD[v]$ units of demand from v_i to v.

2. If $RD[v_i] \leq \sum_{v \in \mathcal{P}} (c(v) - RD[v])$, then we have to compute an assignment of vertices in $\mathcal{P}$ such that $RD[v_i]$ can be served while the residue capacity of v_i is maximized. We model this scenario as an instance of *Relaxed Knapsack Problem* as follows. Let $W = \sum_{v \in \mathcal{P}} (c(v) - RD(v)) - RD[v_i]$ be the packet size, and $a_v = c(v) - RD[v]$, $b_v = c(v_i) - RD[v]$ be the size and profit of the item corresponding to v, for each $v \in \mathcal{P}$. Intuitively, by assigning all the vertices in $\mathcal{P}$ to themselves, we can serve $RD[v_i]$. If we switch the demand assignment of a particular vertex, say $v \in \mathcal{P}$, $RC[v_i]$ is increased by $c(v_i) - RD[v]$ while the capacity available in $\mathcal{P}$ is decreased by $c(v) - RD[v]$. This is exactly the relaxed knapsack problem.

After the the above procedure, all the children of v_i is fully-served. Below we describe how the demand of v_i can be assigned such that the invariant conditions are met. If $RD[v_i] = 0$ and v_i is not the root, then we assign as much demand from the parent of v_i, $p(v_i)$, to v_i as possible until either $RC[v_i]$ is exhausted or the demand of $p(v_i)$ is served.

Lemma 7.11. *If $RD[v_i] \neq 0$, then we can determine the optimal assignment of the residue demand of v_i, except for the case when v_i is not the root and $RD[v_i] \leq c(p(v_i)) < c(v_i)$.*

Theorem 7.12. *Given a tree $T = (V, E)$, we can optimally solve the capacitated domination problem on T in $O\left(n^2 C\right)$ time, where n is the number of vertices and $C = \max_{v \in V} c(v)$ is the maximum capacity.*

Proof. First we bound the time required for each iteration, say iteration i. The transformation can be done in time linear to the size of the children of v_i, which is $O\left(deg(v_i)\right)$. By Theorem 7.7, the time required to solve the relaxed knapsack problem is bounded by $O\left(deg(v_i)^2 \cdot C\right)$. The rest update can be done in constant time. Therefore, the overall time complexity is $\sum_{1 \leq i \leq n} O\left(deg(v_i)^2 \cdot C\right) = O\left(n^2 C\right)$. □

7.3.4 Extension to Polynomial-Time Approximation Scheme

In the following, we show that the pseudo-polynomial time algorithm presented in §7.3.3 can be further extended to obtain a polynomial time approximation scheme. To this end, we first present a fully-polynomial time approximation scheme for the *Relaxed Knapsack Problem*, for which we will use as an alternative subroutine in the original pseudo-polynomial time algorithm to obtain a 2-approximation. Then we make an observation and further improve the algorithm to obtain a polynomial time approximation scheme.

A simple 2-approximation on Trees To this end, we modify the algorithm proposed in §7.3.3 as follows. Let $\epsilon = \frac{1}{\Delta}$, where Δ is the maximum degree of the input tree. Instead of directly applying the dynamic program for the relaxed knapsack problem, we use the approach proposed above to obtain an $(1 - \epsilon)$-approximation and place one extra copy on that vertex. Note that, this additional copy is placed only when the computation of relaxed knapsack is required.

Lemma 7.13. *The modified algorithm based on rescaling yields a 2-approximation for the capacitated domination on trees in $O\left(n^3 \Delta\right)$ time, where n is the number of vertices and Δ is the maximum degree of the tree.*

Proof. For any vertex $v_i \in V$, the maximum residue capacity of v_i resulted by the original approach is at most $deg(v_i) \cdot c(v_i)$ after arranging the demand assignment of its children. By the

choice of ϵ, the deficit of the resulting residue capacity of v_i to that of the original approach is upper-bounded by $\epsilon \cdot deg(v_i) \cdot c(v_i) \leq c(v_i)$, which can be supplemented by the additional copy. The approximation ratio is 2, since it requires at least one copy of v_i in the original scenario. The overall time complexity is $\sum_{v_i \in V} O\left(deg(v_i)^3 \frac{1}{\epsilon}\right) = O\left(n^3 \Delta\right)$. □

A PTAS on Trees The basic idea of the above 2-approximation extends to a polynomial time approximation scheme for the capacitated domination on trees. Let $k > 0$ be a positive integer. The idea is to invoke the above FPTAS for knapsack only when it requires at least k copies of v_i to satisfy the arrangement of the demand assignment of the children of v_i.

This process is done as follows. For any $v_i \in V$ and j with $1 \leq j \leq k$, we verify that whether j copies of v_i are sufficient for the demand assignment of the children set of v_i by enumerating all possible assignments, whose amount are bounded by $O\left(deg(v_i)^j\right)$. If it does, then the optimal assignment corresponds to smallest such j, which we will find during the process. Otherwise we apply the above FPTAS as usual.

Theorem 7.14. *For any positive integer k, we can compute a $\frac{k+1}{k}$-approximation for the capacitated domination problem on trees in $O\left(n^k + n^3 \Delta\right)$ time, where n is the number of vertices and Δ is the maximum degree of the tree.*

Proof. The approximation ratio is clearly $\frac{k+1}{k}$, since the additional copy for any vertex v_i is placed only when it requires at least k copies of v_i to satisfy the children of v_i. The overall time is $\sum_{v_i \in V} \left(deg(v_i)^k + deg(v_i)^3 \Delta\right) = O\left(n^k + n^3 \Delta\right)$. □

Part III

Quality Backbone Design and Maintenance

Chapter 8

Building Acyclic Backbones with Low DWA-Stretch

In this chapter, we consider the Tree Metric Embeddings of Low DWA-Stretch problem. First, a point set cutting lemma which relates two seemingly unrelated quantities, i.e., the sum of pairwise distances and the diameter of the set, is presented. Based on this lemma, we show how an arbitrary metric can be embedded into a tree metric of low DWA-stretch. As a further step to explore the structure of Euclidean metrics, we show that any Euclidean point adopts a spanning tree of low DWA-stretch by recursively applying the cutting lemma to decompose the point set.

8.1 A Point-Set Cutting Lemma

This section presents the aforementioned 1-dimensional point set cutting lemma which guarantees a good cut for any point set such that the sum of pairwise distances between the points separated by the cut is upper-bounded in terms of the diameter of the given point set.

Lemma 8.1 (1-Dimensional Point Sets Cutting Lemma)**.** *Given a set of real numbers* $\mathbf{Q} = \{a_1, a_2, \ldots, a_n\}$*,* $a_1 \leq a_2 \leq \ldots \leq a_n$*, there exists a cutting point* $z \in R$ *with* $a_1 < z < a_n$ *such that the following holds.*

$$\mathcal{L}_{\mathbf{Q}}(z) \cdot (n - \mathcal{L}_{\mathbf{Q}}(z)) \cdot \Delta \leq \delta_0 \cdot \sum_{1 \leq i \leq \mathcal{L}_{\mathbf{Q}}(z)} \sum_{\mathcal{L}_{\mathbf{Q}}(z) < j \leq n} (a_j - a_i),$$

where $\mathcal{L}_{\mathbf{Q}}(z) = |\{a \in \mathbf{Q} : a < z\}|$ *is the number of elements in* $\mathbf{Q}$ *that are smaller than* z*,* $\Delta = a_n - a_1$ *is the diameter of* $\mathbf{Q}$*, and*

$$\delta_0 \leq \frac{210}{59} \approx 3.5594$$

is a constant.

For the ease of presentation, for a given set of real numbers $\mathbf{Q} = \{a_1, a_2, \ldots, a_n\}$, where $a_1 \leq a_2 \leq \ldots \leq a_n$, for each i with $1 \leq i \leq n$, we define $\mathcal{RC}(i)$ to be

$$\mathcal{RC}(i) = \sum_{1 \leq j \leq i} \sum_{i < k \leq n} a_k - a_j.$$

Literally, $\mathcal{RC}(i)$ corresponds to the sum of pairwise distances, or, *interaction*, between $\{a_1, a_2, \ldots, a_i\}$ and $\{a_{i+1}, a_{i+2}, \ldots, a_n\}$. For brevity, we will denote $a_k - a_j$ by $d(a_j, a_k)$ in the remaining content. We also denote by $\Delta(\mathbf{Q})$ the diameter of the set $\mathbf{Q}$, which is exactly $d(a_1, a_n) = a_n - a_1$.

8.1.1 Proof of the Cutting Lemma

Consider the following random distribution:

$$Pr\left[\beta = i\right] = \frac{\mathcal{RC}(i)}{\sum_{\frac{n}{4} \le i \le \frac{3n}{4}} \mathcal{RC}(i)} \ , \qquad \text{where } \beta \in \left\{ \left\lceil \frac{n}{4} \right\rceil, \left\lceil \frac{n}{4} \right\rceil + 1, \ldots, \left\lfloor \frac{3n}{4} \right\rfloor \right\}.$$

To prove our cutting lemma, it suffices to prove the following lemma.

Lemma 8.2. *We have*

$$\min \left\{ E\left[\frac{\beta \cdot (n-\beta) \cdot \Delta(\mathbf{Q})}{\mathcal{RC}(\beta)} \right], \min_{1 \le \gamma \le \frac{n}{3}} \left\{ \frac{\gamma \cdot (q-\gamma) \cdot \Delta(\mathbf{Q})}{\mathcal{RC}(\gamma)}, \frac{\gamma \cdot (q-\gamma) \cdot \Delta(\mathbf{Q})}{\mathcal{RC}(q-\gamma)} \right\} \right\} \le \frac{210}{59}.$$

The rest of this subsection will be devoted to prove Lemma 8.2. Let us derive a lower bound on the overall interaction $\sum_{1 \le i < n} \mathcal{RC}(i)$. Recall that,

$$\mathcal{RC}(i) = \sum_{1 \le j \le i} \sum_{i < k \le n} d(a_j, a_k).$$

For brevity, we will denote by ℓ_k the quantity $a_{k+1} - a_k$, for each $1 \le k < n$. First, observe that, for each j, k with $1 < j < k < n$, we have exactly $(k-j)$ duplications of the item $d(a_j, a_k)$ in the summation $\sum_{1 \le i < n} \mathcal{RC}(i)$, i.e., it appears exactly once in $\mathcal{RC}(i)$ for each $j \le i < k$. Therefore, after re-arranging the items we have

$$\sum_{1 \le i < n} \mathcal{RC}(i) = \sum_{1 \le k < n} k \cdot \sum_{1 \le i \le n-k} d(a_i, a_{i+k}).$$

Define the function $f(n)$ as follows.

$$f(n) = \begin{cases} \frac{n}{2} \sum_{1 \le i \le \frac{n}{2}} d(a_i, a_{i+\frac{n}{2}}), & \text{if } n \text{ is even, and} \\ 0, & \text{otherwise.} \end{cases}$$

Literally, $f(n)$ corresponds to the unique central item in the above summation, if there is one. Then we have

$$\begin{aligned} & \sum_{1 \le k < n} k \cdot \sum_{1 \le i \le n-k} d(a_i, a_{i+k}) \\ = & \sum_{1 \le k < \frac{n}{2}} k \cdot \sum_{1 \le i \le n-k} d(a_i, a_{i+k}) + \sum_{\frac{n}{2} < k < n} k \cdot \sum_{1 \le i \le n-k} d(a_i, a_{i+k}) + f(n) \\ = & \sum_{1 \le k < \frac{n}{2}} k \cdot \sum_{1 \le i \le n-k} d(a_i, a_{i+k}) + \sum_{1 \le k < \frac{n}{2}} (n-k) \sum_{1 \le i \le k} d(a_i, a_{i+n-k}) + f(n), \end{aligned}$$

where in the last inequality we substitute the variable k by $n-k$. By re-organizing and aligning the items from the above summation (see also Fig. 8.1), we have the following lemma.

Lemma 8.3. *For $1 \le k \le \left\lfloor \frac{n}{2} \right\rfloor$, we have*

$$\sum_{1 \le i \le n-k} d(a_i, a_{i+k}) = k \cdot \Delta(\mathbf{Q}) - \sum_{1 \le i < k} (k-i) \cdot (\ell_i + \ell_{n-i}) = \sum_{1 \le i \le k} d(a_i, a_{i+n-k})$$

Proof. We prove the first half of this lemma,

$$\sum_{1\le i\le n-k} d_v(a_{i+k}, a_i) = k\cdot\Delta(\mathbf{Q}) - \sum_{1\le i<k}(k-i)\cdot(\ell_i+\ell_{n-i}).$$

The second half, $\sum_{1\le i\le k} d_v(a_{i+n-k}, a_i) = k\cdot\Delta(\mathbf{Q}) - \sum_{1\le i<k}(k-i)\cdot(\ell_i+\ell_{n-i})$, follows by a similar argument. Consider the alignments of the set of intervals which spans exactly k consecutive elements, that is, intervals $[a_i, a_{i+k}]$, for $1 \le k \le \lfloor \frac{n}{2} \rfloor$. We have exactly k alignments, each starting with a_i for $1 \le i \le k$. See also Fig. 8.1. This sums up to $k\cdot\Delta(\mathbf{Q})$, except for exactly $k-i$ times over-counting of ℓ_i and ℓ_{n-i}. □

The following lemma provides an estimation to the overall interaction, $\sum_{1\le i<n}\mathcal{RC}(i)$.

Lemma 8.4.

$$\sum_{1\le i<n}\mathcal{RC}(i) \ge \sum_{1\le k<\frac{n}{2}} n\cdot \sum_{\frac{n}{2}-k<i<\frac{n}{2}} i\cdot(\ell_k+\ell_{n-k}) + g(n),$$

where

$$g(n) = \begin{cases} n\cdot\sum_{1\le i<\frac{n}{2}} i\cdot\ell_{\frac{n}{2}}, & \text{if } n \text{ is even, and} \\ 0, & \text{otherwise.} \end{cases}$$

Proof. By the above discussion and Lemma 8.3, we have

$$\begin{aligned} &\sum_{1\le i<n}\mathcal{RC}(i) \\ &= \sum_{1\le k<\frac{n}{2}} k\cdot \sum_{1\le i\le n-k} d(a_i, a_{i+k}) + \sum_{1\le k<\frac{n}{2}}(n-k)\sum_{1\le i\le k} d(a_i, a_{i+p-k}) + f(n) \\ &= \sum_{1\le k\le\frac{n}{2}} n\cdot\left(k\cdot\Delta(\mathbf{Q}) - \sum_{1\le i<k}(k-i)(\ell_i+\ell_{n-i})\right) \end{aligned}$$

For $1 \le i < \frac{n}{2}$, the coefficient of ℓ_i and ℓ_{n-i} in the above summation is

$$n\cdot\sum_{i<k<\frac{n}{2}}(k-i),$$

which equals

$$n\cdot\sum_{1\le k<\frac{n}{2}-i} k$$

by substituting the variable k by $k-i$. Therefore, we have

$$\sum_{1\le i<n}\mathcal{RC}(i) \ge \sum_{1\le k<\frac{n}{2}} n\cdot k\cdot\Delta(\mathbf{Q}) - \sum_{1\le k<\frac{n}{2}} n\cdot\sum_{1\le i<\frac{n}{2}-k} i\cdot(\ell_k+\ell_{n-k}).$$

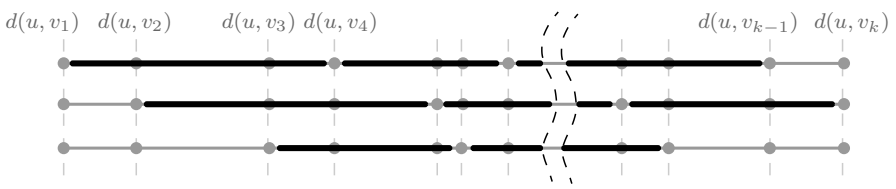


Figure 8.1: Alignment of the intervals when $k = 3$. The first group starts with $d(a_1, a_1)$ while the second and the third start with $d(a_1, a_2)$ and $d(a_1, a_3)$, respectively.

Since $\Delta(\mathbf{Q}) = \sum_{1 \le i < n} \ell_i$, by further expanding $\Delta(\mathbf{Q})$, we obtain

$$\sum_{1 \le i < n} \mathcal{RC}(i) \ge \sum_{1 \le k < \frac{n}{2}} n \cdot \sum_{\frac{n}{2} - k < i < \frac{n}{2}} i \cdot (\ell_k + \ell_{n-k}) + g(n),$$

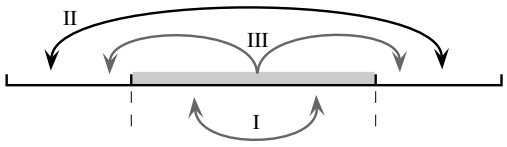
as claimed. □

Provided the lower bound stated in Lemma 8.4, we can derive a lower bound on the total amount of interaction between the numbers in the central-half of the sequence.

Lemma 8.5. *We have*

$$\sum_{\frac{n}{4} \le i \le \frac{3}{4}n} \mathcal{RC}(i) \ge \left(\frac{3}{32} n^3 + \frac{n}{2} \cdot \sum_{\frac{n}{6} n \le i \le \frac{n}{4}} i \right) \cdot \sum_{\frac{n}{3} \le k \le \frac{2n}{3}} \ell_k$$

Proof. We divide the total interaction to be lower-bounded, $\sum_{\frac{n}{4} \le i \le \frac{3}{4}n} \mathcal{RC}(i)$, into three parts which we discuss below.



I. the interaction between points from $\left\{ a_{\lceil \frac{n}{4} \rceil}, a_{\lceil \frac{n}{4} \rceil + 1}, \ldots, a_{\lfloor \frac{3n}{4} \rfloor} \right\}$.

The situation is equivalent to computing the overall interaction for a point set of $\frac{n}{2}$ points. By Lemma 8.4 with index replacement, the interaction is lower-bounded by

$$\sum_{1 \le k < \frac{n}{4}} \frac{n}{2} \cdot \sum_{\frac{n}{4} - k < i < \frac{n}{4}} i \cdot (\ell_{\frac{n}{4} + k} + \ell_{\frac{3n}{4} - k}) + g'(n),$$

where

$$g'(n) = \begin{cases} \frac{n}{2} \cdot \sum_{1 \le i < \frac{n}{4}} i \cdot \ell_{\frac{n}{2}} & \text{if } \frac{n}{2} \text{ is even, and} \\ 0, & \text{otherwise.} \end{cases}$$

Dropping the items corresponding to $k < \frac{n}{12}$ from the first summation, we obtain

$$\frac{n}{2} \cdot \sum_{\frac{n}{6} n \le i \le \frac{n}{4}} i \cdot \sum_{\frac{n}{3} \le k \le \frac{2n}{3}} \ell_k.$$

For the remaining two cases, we consider the number of times each of the items from $\sum_{\frac{n}{3} \le k \le \frac{2n}{3}} \ell_k$ contributes to $\sum_{\frac{n}{4} \le i \le \frac{3n}{4}} \mathcal{RC}(i)$.

II. the interaction between $\left\{ a_1, a_2, \ldots, a_{\lceil \frac{n}{4} \rceil} \right\}$ and $\left\{ a_{\lfloor \frac{3n}{4} \rfloor}, a_{\lfloor \frac{3n}{4} \rfloor + 1}, \ldots, a_n \right\}$.

For each j, k such that $1 \le j \le \frac{n}{4}$, $\frac{3n}{4} \le k < n$, the pair $d(a_j, a_k)$ contributes exactly once to the term $\mathcal{RC}(i)$ for each i with $\frac{n}{4} \le i \le \frac{3n}{4}$. There are $\frac{1}{16} n^2$ such pairs, while there are $\frac{n}{2}$ different terms in the final summation $\sum_{\frac{n}{4} \le i \le \frac{3}{4}n} \mathcal{RC}(i)$. Therefore, we obtain a lower bound of

$$\frac{1}{32} n^3 \cdot \sum_{\frac{n}{3} \le k \le \frac{2n}{3}} \ell_k$$

for this part.

III. the interaction between $\left\{a_{\lceil \frac{n}{4} \rceil}, a_{\lceil \frac{n}{4} \rceil+1}, \ldots, a_{\lfloor \frac{3n}{4} \rfloor}\right\}$ and other points.

For any specific interval ℓ_p with $\frac{n}{4} \leq p \leq \frac{3n}{4}$, we consider the number of pairs between $\left\{a_{\lceil \frac{n}{4} \rceil}, a_{\lceil \frac{n}{4} \rceil+1}, \ldots, a_{\lfloor \frac{3n}{4} \rfloor}\right\}$ and other points that contain this specific interval ℓ_p. There are $p - \frac{n}{4}$ points, $\left\{a_{\lceil \frac{n}{4} \rceil}, a_{\lceil \frac{n}{4} \rceil+1}, \ldots, a_p\right\}$, which lie to the left of a_p and form pairs with points from $\left\{a_{\lfloor \frac{3n}{4} \rfloor}, a_{\lfloor \frac{3n}{4} \rfloor+1}, \ldots, a_n\right\}$ that contain ℓ_p. Similarly, the $\frac{3n}{4} - p$ points that lie to the right of a_p also form pairs with points from $\left\{a_1, a_2, \ldots, a_{\lceil \frac{n}{4} \rceil}\right\}$ that contain ℓ_p. Therefore there are

$$\frac{n}{4} \cdot \left(p - \frac{n}{4} + \frac{3n}{4} - p\right) = \frac{n}{4} \cdot \frac{n}{2}$$

such pairs. This is true for all $\mathcal{RC}(i)$ with $\frac{n}{4} \leq i \leq \frac{3n}{4}$. Therefore ℓ_p contributes $\frac{n}{4} \cdot \frac{n}{2} \cdot \frac{n}{2}$ times in the summation and we obtain a lower bound of

$$\frac{1}{16} n^3 \cdot \sum_{\frac{n}{3} \leq k \leq \frac{2n}{3}} \ell_k.$$

Summing up the bounds we obtained in the three parts and we have this lemma. □

Now we are ready to prove the main lemma of this subsection.

Proof of Lemma 8.2. This lemma holds trivially when $n \leq 3$. For $n \geq 4$, by the definition of expected values, we have

$$\begin{aligned} E\left[\frac{\beta \cdot (n-\beta) \cdot \Delta(\mathbf{Q})}{\mathcal{RC}(\beta)}\right] &= \sum_{\frac{n}{4} \leq i \leq \frac{3n}{4}} Pr\left[\beta = i\right] \cdot \frac{\beta \cdot (n-\beta) \cdot \Delta(\mathbf{Q})}{\mathcal{RC}(\beta)} \\ &= \frac{\sum_{\frac{n}{4} \leq i \leq \frac{3n}{4}} i \cdot (n-i) \cdot \Delta(\mathbf{Q})}{\sum_{\frac{n}{4} \leq i \leq \frac{3n}{4}} \mathcal{RC}(i)}. \end{aligned}$$

First, we have

$$\begin{aligned} \sum_{\frac{n}{4} \leq i \leq \frac{3n}{4}} i(n-i) \cdot \Delta(\mathbf{Q}) &= \left(n \cdot \sum_{\frac{n}{4} \leq i \leq \frac{3n}{4}} i - \sum_{\frac{n}{4} \leq i \leq \frac{3n}{4}} i^2\right) \cdot \Delta(\mathbf{Q}) \\ &\leq \frac{11}{96} n^3 \Delta(\mathbf{Q}). \end{aligned}$$

Depending on whether or not

$$\sum_{\frac{n}{3} \leq k \leq \frac{2n}{3}} \ell_i \geq \frac{11}{35} \Delta(\mathbf{Q}),$$

we consider between two cases.

(1) $\sum_{\frac{n}{3}\le k\le\frac{2n}{3}}\ell_i \ge \frac{11}{35}\Delta(\mathbf{Q})$**.** Then, by Lemma 8.5, we have

$$\sum_{\frac{n}{4}\le i\le\frac{3n}{4}}\mathcal{RC}(i) \ge \sum_{\frac{n}{3}\le k\le\frac{2n}{3}}\ell_k\cdot\left(\frac{3}{32}n^3+\frac{n}{2}\cdot\sum_{\frac{n}{6}\le i\le\frac{n}{4}} i\right) \ge \frac{11}{35}\Delta(\mathbf{Q})\cdot\frac{59}{96\cdot 6}n^3,$$

which implies

$$E\left[\frac{\beta\cdot(n-\beta)\cdot\Delta(\mathbf{Q})}{\mathcal{RC}(\beta)}\right] \le \frac{\frac{11}{96}n^3\Delta(\mathbf{Q})}{\frac{11}{35}\Delta(\mathbf{Q})\cdot\frac{59}{96\cdot 6}n^3} \le \frac{210}{59}.$$

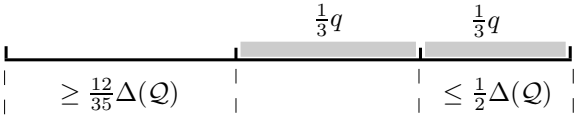
(2) $\sum_{1\le i\le\frac{n}{3}}(\ell_i+\ell_{n-i}) \ge \frac{11}{35}\Delta(\mathbf{Q})$**.** Then we have either

$$\sum_{1\le i\le\frac{n}{3}}\ell_i \ge \frac{12}{35}\Delta(\mathbf{Q}),$$

or

$$\sum_{1\le i\le\frac{n}{3}}\ell_{n-i} \ge \frac{12}{35}\Delta(\mathbf{Q}).$$

Without loss of generality, assume that $\sum_{1\le i\le\frac{n}{3}}\ell_i \ge \sum_{1\le i\le\frac{n}{3}}\ell_{n-i} \ge \frac{12}{35}\Delta(\mathbf{Q})$.



In this case, we have

$$\sum_{1\le i\le\frac{n}{3}}\ell_i + \sum_{\frac{n}{3}<i<\frac{2n}{3}}\ell_i \ge \sum_{\frac{2n}{3}\le i<n}\ell_i.$$

Therefore

$$\sum_{\frac{2n}{3}\le i<n}\ell_i \le \frac{\Delta(\mathbf{Q})}{2}.$$

Let p be the smallest integer such that $\ell_p > 0$. Counting the interaction between $\{a_1, a_2, \ldots, a_p\}$ and $\{a_{p+1}, a_{p+2}, \ldots, a_n\}$, we have

$$\mathcal{RC}(p) \ge p\cdot\frac{n}{3}\cdot\frac{12}{35}\Delta(\mathbf{Q}) + p\cdot\frac{n}{3}\cdot\frac{1}{2}\Delta(\mathbf{Q}).$$

Therefore,

$$\frac{p\cdot(n-p)\cdot\Delta(\mathbf{Q})}{\mathcal{RC}(p)} \le \frac{p\cdot n\cdot\Delta(\mathbf{Q})}{p\cdot n\cdot\Delta(\mathbf{Q})\cdot\left(\frac{1}{3}\cdot\frac{12}{35}+\frac{1}{3}\cdot\frac{1}{2}\right)} = \frac{210}{59}.$$

The argument for the case $\sum_{1\le i\le\frac{n}{3}}\ell_{n-i} \ge \sum_{1\le i\le\frac{n}{3}}\ell_i$ is analogous. This proves the lemma. □

8.1.2 Computing the Optimal Cut in Linear Time

In this following, we show how the best cut can be computed efficiently in linear time for any given set $\mathbf{A} = \{a_1, a_2, \ldots, a_n\}$ of real numbers. For each k with $1 \leq k < n$, let $\mathcal{RC}_\ell(k) = \sum_{1 \leq i < k} (a_k - a_i)$ and $\mathcal{RC}_r(k) = \sum_{k < i \leq n} (a_i - a_k)$ be the sum of the distances between a_k and the points to the left of a_k and the sum of distances between a_k and the points to the right of a_k, respectively. The first observation is that, for $i \leq i < n$,

$$\mathcal{RC}(i) = (n - i) \cdot \mathcal{RC}_\ell(i) + i \cdot \mathcal{RC}_r(i). \tag{8.1}$$

The following lemma shows how these quantities can be computed recursively.

Lemma 8.6. *For $1 \leq k < n - 1$, We have*

- $\mathcal{RC}_\ell(k+1) = \mathcal{RC}_\ell(k) + \sum_{1 \leq i \leq k} \ell_k$, *and*
- $\mathcal{RC}_r(k+1) = \mathcal{RC}_r(k) - \sum_{k < i \leq n} \ell_k$.

Proof of Lemma 8.6. By the definition of function $\mathcal{RC}$, we have

$$\mathcal{RC}_\ell(k+1) = \sum_{1 \leq i < k+1} (\ell_k + a_k - a_i),$$

which equeals to $\mathcal{RC}_\ell(k) + \sum_{1 \leq i \leq k} \ell_k$. Similarly,

$$\mathcal{RC}_r(k+1) = \sum_{k+1 < i \leq n} (a_i - a_k - \ell_k) = \mathcal{RC}_r(k) - \sum_{k < i \leq n} \ell_k.$$

□

By Eq. 8.1 and Lemma 8.6, we can compute in linear time the values $\mathcal{RC}_\ell(k), \mathcal{RC}_r(k)$, and therefore $\mathcal{RC}(k)$ for all $1 \leq k < n$, and the optimal cut. For any given interval $\mathcal{I} \subseteq [a_1, a_n]$, we can also compute the optimal cut inside $\mathcal{I}$ by the same approach.

8.2 Approximating Arbitrary Metrics

Given an arbitrary metric $\mathbf{M} = (\mathbf{V}, d)$, we describe an algorithm which computes a dominating tree metric of $\mathbf{M}$ with small constant distance-weighted average stretch.

This is done by decomposing $\mathbf{V}$ recursively to define a hierarchical net decomposition, which in turn defines the resulting tree metric. The algorithm runs in $\delta = \lceil \log_2 \Delta(V) \rceil$ iterations. Initially, the algorithm sets a variable $i = \delta$ and maintain the trivial root partition $\mathbf{P}_\delta = \{\mathbf{V}\}$. In each of the following iteration, the algorithm decrease the value of i by one and computes $\mathbf{P}_i$ from $\mathbf{P}_{i+1}$ as follows.

For each non-singleton cluster in $\mathbf{P}_{i+1}$, say $\mathbf{C}$, we compute a 2^i-cut decomposition $\mathcal{C}(\mathcal{P})$ of $\mathcal{P}$ by repeatedly decomposing $\mathcal{P}$ by the process described below until the diameter of each clusters in the refinement falls under 2^i.

Let $\mathbf{Q}$ be a cluster in the refinement of $\mathcal{P}$ such that $\Delta(\mathcal{Q}) \geq 2^i$. We pick a vertex $u \in \mathcal{Q}$ such that $\Delta_u(\mathcal{Q}) = \Delta(\mathcal{Q})$. Then we consider the centripetal metric of $\mathcal{Q}$ with respect to u. Let $v_1, v_2, \ldots, v_q$ be the set of vertices of $\mathcal{Q}$ such that $d(u, v_1) \leq d(u, v_2) \leq \ldots \leq d(u, v_q)$. For $1 \leq i \leq q-1$, we denote $\sum_{1 \leq j \leq i} \sum_{i < k \leq q} d(v_j, v_k)$ by $\mathcal{RC}(i)$. Literally, $\mathcal{RC}(i)$ corresponds to the sum of pairwise distances, or, the interaction, between $\{v_1, v_2, \ldots, v_i\}$ and $\{v_{i+1}, v_{i+2}, \ldots, v_q\}$. Let p, $1 \leq p < q$, be the index such that $\frac{p \cdot (q-p) \cdot \Delta(\mathcal{Q})}{\mathcal{RC}(p)}$ is minimized. We create a new cluster in the refinement of $\mathcal{P}$ containing the vertices $\{v_1, v_2, \ldots, v_p\}$ and let $\mathcal{Q} \leftarrow \mathcal{Q} \backslash \{v_1, v_2, \ldots, v_p\}$.

ALGORITHM *Hierarchical-Net-Decomposition*$(\mathbf{V}, d)$

$D_\delta \leftarrow \{V\}$, $i \leftarrow \delta - 1$.
while $i \geq 0$ and D_{i+1} has non-singleton clusters **do**
  for all non-singleton cluster $\mathcal{P}$ in D_{i+1} **do**
    $\mathcal{C}(\mathcal{P}) \leftarrow \{\phi\}$, $\mathcal{S} \leftarrow \{\mathcal{P}\}$.
    while $\mathcal{S} \neq \phi$ **do**
      Let $\mathbf{Q}$ be an arbitrary cluster in $\mathcal{S}$.
      if $\Delta(\mathbf{Q}) < 2^i$ **then**
        Add $\mathbf{Q}$ to $\mathcal{C}(\mathcal{P})$ and remove $\mathbf{Q}$ from $\mathcal{S}$.
      else
        Let $u \in \mathbf{Q}$ be a vertex such that $\Delta_u(\mathbf{Q}) = \Delta(\mathbf{Q})$.
        Let $v_1, v_2, \ldots, v_q$ be the set of vertices in $\mathbf{Q}$ such that $d(u, v_1) \leq d(u, v_2) \leq \ldots \leq d(u, v_q)$.
        Let p, $1 \leq p < q$, be the index such that $\frac{p \cdot (q-p) \cdot \Delta(\mathbf{Q})}{\mathcal{RC}(p)}$ is minimized.
        Let $\mathbf{Q}' \leftarrow \{v_1, v_2, \ldots, v_p\}$, $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbf{Q}'\}$, and $\mathbf{Q} \leftarrow \mathbf{Q} \backslash \mathbf{Q}'$.
      end if
    end while
    Let $\mathcal{C}(\mathcal{P})$ be the refinement clusters of $\mathcal{P}$ in D_i.
  end for
  $i \leftarrow i - 1$.
end while
Return the tree metric corresponding to $D_0, D_1, \ldots, D_\delta$.

Figure 8.2: A high-level description of the hierarchical decomposition algorithm.

This process is repeated until all the clusters in the refinement of $\mathcal{P}$ have diameter less than 2^i. D_i is defined to be the union of the refinements of non-singleton clusters of D_{i+1}. A high-level description of this algorithm is presented in Fig. 8.2.

First we argue that the algorithm computes a dominating tree metric. Let T be the tree corresponding to the hierarchical net decomposition constructed by our algorithm and d_T be the distance function induced by T. For any non-singleton cluster $\mathcal{P}$ in D_i and $u, v \in \mathcal{P}$, we have $d(u, v) \leq \Delta(\mathcal{P}) < 2^i$ by the definition of hierarchical net decomposition, and $d_T(u, v) \leq 2 \cdot \sum_{0 \leq j < i} 2^j < 2^{i+1}$ by the construction of the tree metric. Therefore, (T, d_T) is a dominating tree metric of M.

In the following, we will show that $\mathcal{R}(T) \leq 4 \cdot \frac{210}{59} \cdot \mathcal{R}(M)$. To this end, we prove that, for any partition of a cluster $\mathcal{Q}$ into, say $\mathcal{Q}_1$ and $\mathcal{Q}_2$ such that $u \in \mathcal{Q}_1$, we performed in our algorithm,

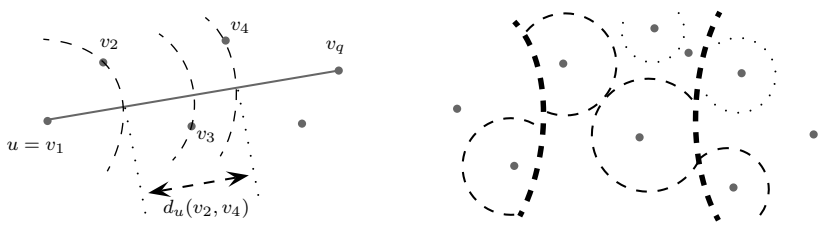


Figure 8.3: (a) An illustration of the centripetal metric with respect to a vertex u. (b) A hierarchical decomposition of the points.

we have

$$|\mathcal{Q}_1| \cdot |\mathcal{Q}_2| \cdot \Delta(\mathcal{Q}) \leq \frac{210}{59} \cdot \mathcal{R}(\mathcal{Q}_1, \mathcal{Q}_2). \tag{8.2}$$

Let $T[\mathcal{Q}]$, $T[\mathcal{Q}_1]$, and $T[\mathcal{Q}_2]$ denote the subtree of T corresponding to $\mathcal{Q}$, $\mathcal{Q}_1$, and $\mathcal{Q}_2$, respectively. As a consequence to (8.2), we have $\mathcal{R}(T_{\mathcal{Q}_1}, T_{\mathcal{Q}_2}) \leq |\mathcal{Q}_1| \cdot |\mathcal{Q}_2| \cdot 2^{i+1} \leq 4 \cdot |\mathcal{Q}_1| \cdot |\mathcal{Q}_2| \cdot \Delta(\mathcal{Q}) \leq 4 \cdot \frac{210}{59} \cdot \mathcal{R}(\mathcal{Q}_1, \mathcal{Q}_2)$. Since $\max\{|\mathcal{Q}_1|, |\mathcal{Q}_2|\} < |\mathcal{Q}|$, by an inductive argument we have $\mathcal{R}(T_{\mathcal{Q}}) = \mathcal{R}(T_{\mathcal{Q}_1}) + \mathcal{R}(T_{\mathcal{Q}_2}) + \mathcal{R}(T_{\mathcal{Q}_1}, T_{\mathcal{Q}_2}) \leq 4 \cdot \frac{210}{59} \cdot (\mathcal{R}(\mathcal{Q}_1) + \mathcal{R}(\mathcal{Q}_2) + \mathcal{R}(\mathcal{Q}_1, \mathcal{Q}_2)) = 4 \cdot \frac{210}{59} \cdot \mathcal{R}(\mathcal{Q})$. This holds for all cluster $\mathcal{Q}$, including the trivial cluster in D_δ. Therefore $\mathcal{R}(T) \leq 4 \cdot \frac{210}{59} \cdot \mathcal{R}(M)$.

ALGORITHM *Euclidean-Spanning-Tree*($\mathbf{V}$)

Input: A set $\mathbf{V}$ of n points in $\mathcal{R}^d$.
Output: A pair $(\mathbf{T}, r)$, which is a spanning tree $\mathbf{T}$ of $\mathbf{V}$ with root r.

1: **if** $\mathbf{V}$ is a singleton point set containing point p **then**
2: Return $(\mathbf{V}, p)$.
3: **end if**
4: Let $\alpha = \frac{1}{4}$ be a constant.
5: Let k be the index of dimension such that $\mathcal{L}_k(\mathcal{B}(\mathbf{V})) = \mathcal{L}_{max}(\mathcal{B}(\mathbf{V}))$.
6: Let $a_1 \leq a_2 \leq \ldots \leq a_n$ be the coordinates of the projection of $\mathbf{V}$ into k^{th} dimension, labelled in sorted order.
7: $p = \alpha \cdot (a_1 + a_n)$, $q = (1 - \alpha) \cdot (a_1 + a_n)$.
8: $(\mathbf{V}_1, \mathbf{V}_2) \longleftarrow$ *1d-cut*$(\{a_1, a_2, \ldots, a_n\}, [p, q])$.
9: $(T_1, r_1) \longleftarrow$ *Euclidean-Spanning-Tree*$(\mathbf{V}_1)$, $(T_2, r_2) \longleftarrow$ *Euclidean-Spanning-Tree*$(\mathbf{V}_2)$.
10: Let $T \longleftarrow T_1 \cup T_2 \cup \{(r_1, r_2)\}$.
11: Return (T, r_1).

Figure 8.4: A high-level description of the algorithm which computes a spanning tree of low DWA-stretch for Euclidean graphs.

8.3 Approximating Euclidean Metrics by Their Spanning Trees

This section shows how a spanning tree of small constant distance-weighted average stretch for a Euclidean graph can be computed in polynomial time. The basic idea is to extend the previous point-set decomposition. In order to guarantee a constant blow-up in the diameter of the resulting spanning tree, we cannot allow the cut to be made at arbitrary positions. Instead, each cut is restricted to be made within the central $(1 - 2\alpha)$ portion along the longest side of its bounding box, where α is a constant chosen to be $\frac{1}{4}$. This guarantees a balanced partition, an exponentially decreasing size of the bounding boxes, and a constant blow-up of the diameter of the resulting spanning tree. This is crucial in the analysis, as we need a tight diameter in order to provide a good upper-bound on the interaction between pairs separated by our cuts. On the other hand, we also have to guarantee the existence of good cuts in the central $(1 - 2\alpha)$ portion so that the overall interaction stays bounded.

Given a set of points $\mathbf{V}$ in the Euclidean space $\mathcal{R}^d$ of finite dimension d, the algorithm recursively computes a rooted tree $\mathbf{T}$ with root r as follows. Let $\mathcal{B}(\mathbf{V})$ be the bounding box of $\mathbf{V}$, and k be the specific index of dimension such that $\mathcal{L}_k(\mathcal{B}(\mathbf{V})) = \mathcal{L}_{max}(\mathcal{B}(\mathbf{V}))$. Consider the projection of the points onto the k^{th}-axis, and let $a_1, a_2, \ldots, a_n$, $a_1 \leq a_2 \leq \ldots \leq a_n$, be the corresponding coordinates. The algorithm first applies the linear time algorithm provided in §8.1.2 as a subroutine to compute a decomposition of the point set $\mathbf{V}$ for which the cut

is restricted to be made inside the central $(1-2\alpha)$ portion of the sequence $\{a_1, a_2, \ldots, a_n\}$, which is $\{\alpha \cdot (a_1 + a_n), (1-\alpha) \cdot (a_1 + a_n)\}$. Also refer to Fig. 8.5 (a) for an illustration. Let $\mathbf{V}_1$ and $\mathbf{V}_2$ be the corresponding partitioned subsets of points. Then we recursively computes two rooted trees for $\mathbf{V}_1$ and $\mathbf{V}_2$, further denoted by $\mathbf{T}_1$ with root r_1 and $\mathbf{T}_2$ with root r_2. The overall spanning tree $\mathbf{T}$ for the point set $\mathbf{V}$ is constructed by joining r_1 and r_2, and the root of $\mathbf{T}$ is chosen to be r_1. A high-level description of the algorithm is provided in Fig. 8.4.

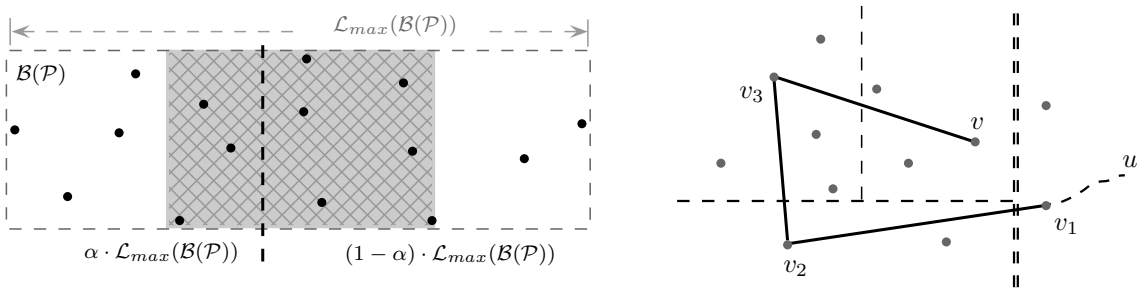


Figure 8.5: (a) The vertical cut is restricted to be placed in the central $(1-2\alpha)$ portion along the longest side of the bounding box, which corresponds to the shaded region. (b) A possible decomposition and the $u-v$ path in the resulting tree. The boldly-dashed line corresponds to the first cut and v_1v_2 corresponds to the edge connecting the two rooted trees returned by the recursion.

For the ease of presentation, let $\mathbf{F}$ be the family collection of subsets of $\mathbf{V}$ which occurs during the recursions made by the algorithm. For each set $\mathbf{S} \in \mathbf{F}$, we denote by $\mathbf{T}[\mathbf{S}]$ the induced subgraph of $\mathbf{T}$ on $\mathbf{S}$. Note that, according to our construction, $\mathbf{T}[\mathbf{S}]$ will be a spanning tree of $\mathbf{S}$. Let $e(\mathbf{S})$ the edge connecting the two rooted trees corresponding to the two further partitions of $\mathbf{S}$. $e(\mathbf{S})$ is defined to be a dummy self-loop with length zero if $\mathbf{S}$ is a singleton set.

For each point $p \in \mathbf{V}$, let $\mathbf{S}(p,1), \mathbf{S}(p,2), \ldots, \mathbf{S}(p, depth(p))$, $\mathbf{S}(p,1) \supset \mathbf{S}(p,2) \supset \ldots \supset \mathbf{S}(p, depth(p))$, $\mathbf{S}(p,i) \in \mathbf{F}$ for $1 \leq i \leq depth(p)$, be the subsets of $\mathbf{V}$ occurred during the recursions to which p belongs, where $depth(p)$ denotes the depth of the set $\{p\}$ in the recursion tree. Note that, by the above definitions we have $\mathbf{S}(p)_1 = \mathbf{V}$ and $\mathbf{S}(p)_{depth(p)} = \{p\}$.

For any two points $p, q \in \mathbf{V}$, the distance of p and q in the tree $\mathbf{T}$ is determined by the set of bridge edges The following lemma provides an upper-bound on the pairwise distances.

Lemma 8.7. *For any $p, q \in \mathbf{V}$, we have $d_{\mathbf{T}}(p,q) \leq \frac{2}{\alpha} d\sqrt{d} \cdot \mathcal{L}_{max}(\mathcal{B}(\mathbf{V}))$.*

Proof. Let Similarly, let $\mathbf{S}(q)_1 \supset \mathbf{S}(q)_2 \supset \ldots \supset \mathbf{S}(q)_{depth(q)}$ be the subsets of $\mathbf{V}$ to which q belongs. Note that, we have $\mathbf{S}(p)_1 = \mathbf{S}(q)_1 = \mathbf{V}$, $\mathbf{S}(p)_{depth(p)} = \{p\}$, and $\mathbf{S}(q)_{depth(q)} = \{q\}$.

From the construction of $\mathbf{T}$, we have

$$d_{\mathbf{T}}(p,q) \leq d_{\mathbf{T}[\mathbf{S}(p)_1]}(p, r_1) + |e(\mathbf{V})| + d_{\mathbf{T}[\mathbf{S}(q)_1]}(r_2, q) \leq \sum_{1 \leq i \leq a} |e(A_i)| + |e(\mathbf{V})| + \sum_{1 \leq j \leq b} |e(B_j)|\,,$$

where r_1 and r_2 are the roots of $\mathbf{T}[A_1]$ and $\mathbf{T}[B_1]$. Since the longest straight-line distance inside a hyper-rectangle is bounded by its longest diagonal, we have $|e(Q)| \leq \sqrt{d}\mathcal{L}_{max}(\mathcal{B}(Q))$ for any subset $Q \in \mathcal{F}$. Furthermore, since we always cut along the longest side of the bounding box, we have $\mathcal{L}_{max}(\mathcal{B}(A_{i+d})) \leq (1-\alpha)\mathcal{L}_{max}(\mathcal{B}(A_i))$ and $\mathcal{L}_{max}(\mathcal{B}(B_{j+d})) \leq (1-\alpha)\mathcal{L}_{max}(\mathcal{B}(B_j))$ for

all $1 \le i \le a-d$ and $1 \le j \le b-d$. Therefore, it follows that

$$\begin{aligned} d_{\mathbf{T}}(p,q) &\le \sum_{1\le i\le a} \sqrt{d}\mathcal{L}_{max}(\mathcal{B}(A_i)) + \sqrt{d}\mathcal{L}_{max}(\mathcal{B}(\mathbf{V})) + \sum_{1\le j\le b} \sqrt{d}\mathcal{L}_{max}(\mathcal{B}(B_j)) \\ &\le 2d \cdot \sum_{i\ge 1} \sqrt{d}(1-\alpha)^i \mathcal{L}_{max}(\mathcal{B}(\mathbf{V})) + \sqrt{d}\mathcal{L}_{max}(\mathcal{B}(\mathbf{V})) \\ &\le \frac{2}{\alpha} d\sqrt{d} \cdot \mathcal{L}_{max}(\mathcal{B}(\mathbf{V})), \end{aligned}$$

where in the second last inequality we collect every d items from the summation of the first inequality and then combine them together into a geometric series. □

In the following lemma, we show that, in exchange of certain penalty in the performance factor that is inverse proportional to the length of the interval to which the cut is restricted, we can always guarantee a good and balanced decomposition.

Lemma 8.8 (Constrained Point Set Cutting Lemma)**.** *Given a set of real numbers $A = \{a_1, a_2, \ldots, a_n\}$, $a_1 \le a_2 \le \ldots \le a_n$ and an interval $\mathcal{I} = [\ell, r]$ such that $\mathcal{I} \subseteq [a_1, a_n]$, there exists a cutting point $z \in \mathcal{I}$ such that the following holds.*

$$L_A(z) \cdot (n - L_A(z)) \cdot |\mathcal{I}| \le \delta_0 \cdot \sum_{1\le i\le L_A(z)} \sum_{L_A(z)<j\le n} (a_j - a_i),$$

where $L_A(z) = |\{a \in A : a < z\}|$ is the number of elements in A that are smaller than z and $\delta_0 \le \frac{210}{59}$ is a constant.

Proof of Lemma 8.8. We say that an interval degenerates if it has length zero. First we argue that, if there are degenerating intervals at a_1, then it is always worse to cut at those degenerating intervals. Let k, $1 \le k \le n$, be the largest index such that $a_1 = a_2 = \ldots = a_k$. Observe that, for any i, j with $1 \le i, j \le k$, we have $\mathcal{RC}(i) = \frac{i}{j} \cdot \mathcal{RC}(j)$. On the other hand, for $1 \le i < k$ and $1 \le j \le k - i$, we have

$$\frac{(i+j)(n-i-j)}{i(n-i)} = \frac{i(n-i) + j(n-2i-j)}{i(n-i)} \le \frac{i+j}{i} = \frac{\mathcal{RC}(i+j)}{\mathcal{RC}(i)},$$

which implies that $\frac{(i+j)(n-i-j)}{\mathcal{RC}(i+j)} \le \frac{i(n-i)}{\mathcal{RC}(i)}$ and therefore cutting at $(a_k, a_{k+1}]$ is always better than cutting at degenerating intervals at a_1. Similarly, we can argue that, it is always worse to cut at the degenerating intervals at a_n, if there is any.

Now we argue that there will be a feasible cut satisfying the criterion. According to the given interval $\mathcal{I} = [a, b]$ and the point set A, we create a new point set $B = \{b_1, b_2, \ldots, b_n\}$ as follows.

$$\text{For } 1 \le i \le n, \quad b_i = \begin{cases} \ell & \text{if } a_i < \ell, \\ a_i & \text{if } \ell \le a_i \le r, \\ r & \text{otherwise.} \end{cases}$$

Let z be the best cut of B in $\mathcal{I}$. By the above argument, we have $\ell < z < r$ and therefore $L_A(z) = L_B(z)$. By Lemma **??**, we have $L_B(z) \cdot (n - L_B(z)) \cdot |\mathcal{I}| \le \frac{210}{59} \sum_{b_i<z\le b_j} (b_j - b_i)$. According to our setting, we have $(b_j - b_i) \le (a_j - a_i)$ for all $1 \le i < j \le n$. Therefore $L_A(z) \cdot (n - L_A(z)) \cdot |\mathcal{I}| \le \frac{210}{59} \sum_{1\le i\le L_A(z)} \sum_{L_A(z)<j\le n} (a_j - a_i)$ as claimed. □

In the following, we state the theorem and leave the rest detail in the appendix for further reference.

Theorem 8.9. *Given a set of points* $\mathbf{V}$ *in* $\mathcal{R}^d$*, we can compute in polynomial time a spanning tree* $\mathbf{T}$ *of* $\mathbf{V}$ *such that the distance-weighted average stretch of* $\mathbf{T}$ *with respect to* $\mathbf{V}$ *is at most* $16\delta_0 \cdot d\sqrt{d}$*, where* $\delta_0 \leq \frac{210}{59}$ *is the constant in our point set cutting lemma.*

Proof of Theorem 8.9. If $|\mathbf{V}| = 1$, then this theorem holds trivially. Otherwise, by Lemma 8.7, Lemma 8.8, and the fact that the length of the restricted interval is $(1 - 2\alpha) \cdot \mathcal{L}_{max}(\mathcal{B}(\mathbf{V}))$, we have

$$\mathcal{R}_{\mathbf{T}}(\mathbf{V}_1, \mathbf{V}_2) \leq |\mathbf{V}_1| \cdot |\mathbf{V}_2| \cdot \frac{2}{\alpha} d\sqrt{d} \cdot \mathcal{L}_{max}(\mathcal{B}(\mathbf{V})) \leq \frac{2\delta_0}{\alpha(1-2\alpha)} d\sqrt{d}\mathcal{R}(\mathbf{V}_1, \mathbf{V}_2).$$

This holds for all recursions. Choose α to be $\frac{1}{4}$ and this theorem follows directly by induction on the depth of recursion. □

Chapter 9

Maintaining Acyclic Backbones under Link Failures

In this chapter, we consider the maintenance problem of low DWA-Stretch acyclic backbones. We present an asymptotically optimal quadratic-time algorithm for the general case and show that the problem can be solved more efficiently for the Euclidean metric, when vertices are mapped to points in the plane, as well as for compactly representable graph metrics.

9.1 An Optimal Algorithm for an Arbitrary Underlying Graph

In this section, we consider general distance functions on the vertex set. We show that the problem can be solved in $\Theta(n_1 \cdot n_2)$ time, which is optimal. For ease of notation we write $C_1 = c(V_1)$ and $C_2 = c(V_2)$ for the total demand in T_1 and T_2, respectively. Given two vertices $u \in V_1$ and $v \in V_2$, the routing cost of the tree T_{uv} resulting from joining T_1 and T_2 by the edge uv is given by

$$\begin{aligned} rc(T_{uv}) = rc(T_1) + rc(T_2) \\ + C_2 \cdot \sum_{u' \in V_1} c(u') \cdot d_{T_1}(u', u) \\ + C_1 \cdot \sum_{v' \in V_2} c(v') \cdot d_{T_2}(v, v') \\ + C_1 \cdot C_2 \cdot d(u, v) \; . \end{aligned} \tag{9.1}$$

It is composed of the routing cost inside the subtrees T_1 and T_2 of T_{uv}, respectively, and the routing cost effected by the shortest paths using the edge uv between the two trees. Since the total sum of demands for these paths equals $C_1 \cdot C_2$, the edge uv contributes a total amount of $C_1 \cdot C_2 \cdot d(u, v)$ to the routing cost. Furthermore, each shortest path starting at u' in T_1 and ending at u can be extended to a shortest path ending at some vertex v' in T_2. Hence, each shortest path of this kind contributes its length, weighted by its demand $c(u')$ and the total sum of the demands C_2 in T_2, to the routing cost. The situation is symmetrical for the paths starting in T_2 and ending at v.

Since the routing costs of T_1 and T_2 do not depend on the choice of the link between the two trees, our problem is equivalent to minimizing the remaining summands in equation (9.1).

We define the weight of a vertex $u \in V_1$, denoted by $w(u)$, as the sum of lengths of all shortest paths starting at $u' \in V_1$ and ending at u, weighted by the demand of u', i.e.,

$$w(u) = \sum_{u' \in V_1} c(u') \cdot d_{T_1}(u', u) \; .$$

We define the weight of a vertex $v \in V_2$, denoted by $w(v)$, analogously. Hence, we seek to minimize the term

$$rc'(T_{uv}) = C_2 w(u) + C_1 w(v) + C_1 \cdot C_2 \cdot d(u, v) \tag{9.2}$$

over all possible combinations of $u \in V_1$ and $v \in V_2$.

The weights of the trees can be computed in linear time as follows. First we compute the total demands in T_1 and T_2, respectively. We compute the weights in T_1 by rooting the tree in some vertex r and performing one bottom-up pass over the tree, followed by a top-down pass. For a vertex u in T_1 we denote the subtree rooted in u by T_u.

In the bottom-up pass, we compute two values for each vertex $u \in V_1$: the total demand $\gamma(u)$ of the vertices in T_u, and the sum $\lambda(u)$ of the shortest paths starting at some vertex u' in T_u and ending at u, weighted by the demand of u', i.e.,

$$\gamma(u) = \sum_{u' \in V(T_u)} c(u')$$

and

$$\lambda(u) = \sum_{u' \in V(T_u)} c(u') d(u', u) \ .$$

For a vertex u with children $u_1, \ldots, u_k$ these values can be computed in linear time as

$$\gamma(u) = c(u) + \sum_{i=1}^{k} \gamma(u_i)$$

and

$$\lambda(u) = \sum_{i=1}^{k} \big(\lambda(u_i) + \gamma(u_i) \cdot d(u_i, u)\big) \ ,$$

respectively. In the top-down pass, we compute the weight for each vertex $v \in V_1$. For the root r this weight is equal to $\lambda(r)$. For a vertex v with father $u \in V_1$ the weight can be computed by

$$w(v) = w(u) + (C_1 - 2\gamma(v)) d(u, v) \ .$$

This equation is due to the fact that the weight of v is obtained from the weight of u by removing the demand $\gamma(v)$ in the subtree of v from the edge uv and adding the remaining demand $C_1 - \gamma(v)$ to the edge uv. For T_2 we proceed analogously.

Having this, we can compute the best and second-best connection between the two trees by enumerating all possible pairs uv such that $u \in V_1$ and $v \in V_2$, which yields a total running time of $O(n_1 \cdot n_2)$. Note, that the described algorithm only finds the best or second-best solution, but does not compute the routing cost of this solution. If we have no restriction on the distance between the vertices, however, the algorithm is optimal.

Theorem 9.1. *The optimal routing cost augmentation problem and the optimal routing cost replacement problem can be solved in $O(n_1 \cdot n_2)$ time for general distance function. This is optimal in the algebraic decision tree model.*

Proof. We have already outlined the algorithm and argued why it runs within the stated time complexity. It remains to show the lower bound on the running time. For this, we assume that we are given a set of integers $a_1, \ldots, a_N$. We construct an instance of the optimal routing cost augmentation problem such that finding the minimum routing cost connection between the two

trees is equivalent to the minimum of the numbers $a_1, \ldots, a_N$. For this problem, we need at least $N-1$ comparisons in the algebraic decision tree model of computation.

Let $N = n_1 n_2$ be any factorization of N and let V be a set of $n_1 + n_2$ vertices. Further, let $V_1, V_2 \subseteq V$ be a partition of V such that $|V_1| = n_1$ and $|V_2| = n_2$ and let T_1 and T_2 be two arbitrary trees on V_1 and V_2, respectively. We set the distance between two vertices in the same tree equal to one. Let $x : V_1 \times V_2 \to \{a_1, \ldots, a_N\}$ be a bijective mapping between the pairs of vertices in V_1 and V_2 and the numbers a_i. Then we choose the remaining distances as follows. Let W_1 and W_2 be the maximum weights of the vertices in T_1 and T_2, respectively. For $u \in V_1$ and $v \in V_2$ we define

$$d_0(u, v) = C_2 W_1 + C_1 W_2 - C_2 w(u) - C_1 w(v) \; .$$

Further, we set

$$d(u, v) = \frac{d_0(u, v) + x(u, v)}{C_1 C_2} \; .$$

Then $rc'(T_{uv}) = C_2 W_1 + C_1 W_2 + x(u, v)$. For both the augmentation and the replacement problem we need to compute the minimum routing cost solution. However, minimizing the routing cost for the given instance is equivalent to computing the minimum over the values $x(u, v)$ for $u \in V_1$ and $v \in V_2$. Hence, in the algebraic decision tree model of computation, we need at least $n_1 \cdot n_2 - 1$ comparisons, which completes the proof. □

9.2 An Efficient Algorithm for the Euclidean Metric

The proof for the lower bound in the previous section crucially exploits the fact that we can choose distances between the vertices in an arbitrary fashion. If this is not the case, we can come up with more efficient algorithms.

In this section we consider the case that vertices are points in the plane and that the considered metric d is the Euclidean metric. In this case, we can compute the best connection between two trees in $O((n_1 + n_2) \log \min\{n_1, n_2\})$ time. Throughout the section, we do not distinguish between vertices and points.

Theorem 9.2. *The optimal augmentation problem for the Euclidean metric can be solved in* $O((n_1 + n_2) \log \min\{n_1, n_2\})$ *time.*

Proof. Without loss of generality we may assume that $n_2 \leq n_1$. Let $\sigma : \mathcal{R}^2 \to \mathcal{R}^2$ be an isotropic scaling with scale factor $s = C_1 \cdot C_2$, i.e., σ scales distances by a factor s and we thus have

$$d(\sigma u, \sigma v) = C_1 \cdot C_2 \cdot d(u, v) \; . \tag{9.3}$$

Let σV_1 and σV_2 denote the scaled sets of points.

For $x \in \mathcal{R}^2$ and $\widetilde{v} \in \sigma V_2$ we define a new distance function, defined by $d_+(x, \widetilde{v}) := d(x, \widetilde{v}) + C_1 \cdot w(\widetilde{v})$, where w is defined as in the previous section. The *additively weighted Voronoi cell of* $\widetilde{v}$ is the locus of points

$$\{x \in \mathcal{R}^2 \mid \forall \widetilde{u} \in \sigma V_2 \setminus \{\widetilde{v}\} : d_+(x, \widetilde{v}) < d_+(x, \widetilde{u})\} \tag{9.4}$$

The additively weighted Voronoi diagram $\mathcal{V}$ defined by d_+ consists of the additively weighted Voronoi cells of the points in σV_2 and can be computed in $O(n_2 \log n_2)$ time [**?**].

For each point $u \in V_1$, we locate the nearest neighbor σv of σu in $\mathcal{V}$ using an algorithm with $O(\log n_2)$ query time described by Kirkpatrick [**?**]. Then σv satisfies

$$d_+(\sigma u, \sigma v) = \min_{v' \in V_2} d_+(\sigma u, \sigma v') \tag{9.5}$$

and we have

$$d_+(\sigma u, \sigma v) = d(\sigma u, \sigma v) + C_1 \cdot w(v) \tag{9.6}$$
$$= C_1 \cdot C_2 \cdot d(u, v) + C_1 \cdot w(v) \; . \tag{9.7}$$

Hence, $v \in V_2$ is the best endpoint of an edge starting at $u \in V_1$ with respect to routing cost. Minimizing $C_2 \cdot w(u) + d_+(\sigma u, \sigma v)$ over all vertices $\sigma u \in V_1$ and their respective nearest neighbor $\sigma v \in V_2$ will thus minimize the overall routing cost. The resulting overall running time is $O(n_1 \log n_2 + n_2 \log n_2)$. □

In order to solve the replacement problem, we also need to compute the second-best solution. We can do this as follows. Let $u^* \in V_1$ and $v^* \in V_2$ be the best solution computed by the algorithm above. This algorithm can trivially be modified to simultaneously compute

$$\min_{u \in V_1 \setminus \{u^*\}, v \in V_2} rc'(T_{uv})$$

in the same time complexity. By additionally computing the Voronoi diagram only for the points in $V_2 \setminus \{v^*\}$ and repeating the algorithm on this instance, we can also compute

$$\min_{u \in V_1, v \in V_2 \setminus \{v^*\}} rc'(T_{uv}) \; .$$

Clearly, the second-best solution is either of the two. Hence, we have the following corollary.

Corollary 9.3. *The optimal routing cost replacement problem for the Euclidean metric can be solved in time $O((n_1 + n_2) \log n_2)$.*

Note that the same approach can also be used in a planar setting, i.e., when the newly introduced edge connecting the two trees may not intersect any other edge of the two trees. In this case we compute an additively weighted constrained Voronoi diagram, which can be done by adapting Fortune's sweepline algorithm [**?**] with $O(n \log n)$ running time. In a constrained Voronoi diagram, we are given an additional set of line segments representing obstacles. Whenever the straight line connecting two points intersects one of the obstacles, the distance between the two points is assumed to be infinity, otherwise, it is equal to the (weighted) Euclidean distance between the points. In our application each edge defined by one of the trees is one such obstacle. Seidel shows how to adapt Fortune's algorithm to compute the constrained Voronoi diagram [**?**]. The adaption to additively weighted sites has been sketched in Fortune's original paper [**?**].

Corollary 9.4. *The planar augmentation problem for the Euclidean metric can be solved in $O((n_1 + n_2) \log n_2)$ time.*

9.3 General Metrics

Every finite metric d can be encoded by a finite graph $M = (V, D)$ where each edge $e \in D$ has some length $\ell(e)$ and the distance d between two vertices in V is equal to the sum of the lengths of the shortest path between the vertices in the graph in terms of the edge lengths. We can directly translate our idea from the previous section to this setting by computing the additively weighted Voronoi diagram in M instead. Although the computation of various Voronoi diagrams on graphs has been considered by Hurtado et al. [**?**], among them a multiplicatively weighted Voronoi diagram, we are not aware of any investigation of the additively weighted Voronoi diagram on graphs. The following theorem is similar to the results by Hurtado et al. [**?**]. We assume that the additively weighted Voronoi diagram of a a set of sites $S \subseteq V$ on a metric graph $G = (V, E)$ is completely known if every vertex $v \in V \setminus S$ knows its nearest neighbor in S and we know the bisector point for each edge, if it exists.

Theorem 9.5. *The additively weighted Voronoi diagram of a set of sites $S \subseteq V$ on a graph $G = (V, E)$ has complexity $\Theta(m)$ and can be computed in time $O(m + n \log n)$.*

Proof. Each edge of the graph contains at most one bisector point, since moving along the edge will alter the additively weighted distances by the same amount—either increasing or decreasing—for all distances. Hence we have at most m bisector points. On the other hand, we can have exactly m bisectors by setting $V' = V$. Hence, the complexity of the additively weighted Voronoi diagram is $\Theta(m)$.

To compute the additively weighted Voronoi diagram in G we use the parallel Dijkstra algorithm proposed by Erwig [**?**] with running time $O(m + n \log n)$. To compute the diagram, we run Dijkstra's algorithm in parallel using the vertices in S as starting points. For a vertex $v \in V \setminus S$ and some vertex $s \in S$ the distance between v and s is $d_s(v, s) = d_G(v, s) + w(s)$. Whenever a vertex $v \in V \setminus S$ is settled, we update its closest neighbor in S. The bisector points can be computed in $O(m)$ time from this information. □

Using this result, we can almost directly translate the technique for the Euclidean case to the general metric case studied in this section.

Theorem 9.6. *The optimal routing cost augmentation problem for general metrics can be solved in time $O(m + n \log n)$ if the metric is given by a graph $M = (V, D)$ with edge length function ℓ.*

Proof. Instead of scaling the point set as in the Euclidean case, we scale the lengths of the edges in G by a factor C_1C_2, i.e., instead of using ℓ to assess the distance between two vertices in M, we use $C_1C_2\ell$. The rest of the proof is completely analogous. We compute the additively weighted Voronoi diagram on M for the set of sites V_2. Then we locate the vertex $u \in V_1$ that minimizes $C_2 \cdot w(u) + d_+(u, v)$ where $d_+(u, v)$ is the scaled and additively weighted distance between u and its closest neighbor v. The resulting time complexity is $O(m + n \log n)$. □

Again we can proceed as in the Euclidean case in order to compute the second-best connection between the two trees.

Corollary 9.7. *The optimal routing cost replacement problem for general metrics can be solved in time $O(m + n \log n)$ if the metric is given by a graph $M = (V, D)$ with edge length function ℓ.*

Although this result does not provide an asymptotic improvement in the worst-case, it does show that we can efficiently solve the augmentation problem for compactly representable metrics. If the graph representing the metric is sparse, then the above theorem states that we can solve the augmentation problem in $O(n \log n)$ as in the Euclidean case.

Chapter 10

Cost-Efficient Bi-Constrained Backbone Construction

In this chapter, we consider the Bi-constrained Maximum Cost-Efficiency Pattern problem. Given an instance $\mathcal{I} = (\mathbf{G}, w, \ell, \mathcal{W}, \mathcal{L})$ of bi-constrained maximum cost-efficiency pattern, we wish to find a connected $(\mathcal{W}, \mathcal{L})$-viable pattern of $\mathbf{G}$ with maximum cost-efficiency. Since this problem can be solved in time $O(n^2)$ when the host is a tree and the pattern is a path by enumerating all possible paths, it is natural to ask if it can be solved efficiently on more general hosts and patterns. However, we show that it is **NP**-hard to find a maximum cost-efficiency path, even if the host is only slightly more complicated than a tree.

Theorem 10.1. *The Bi-constrained Maximum Cost-Efficiency Pattern problem is* **NP***-hard, even if the host is an outerplanar graph of treewidth 2, the pattern is a path, and we drop the upper bound on the length of the pattern.*

Proof. The proof is by reduction from the Partition problem, which is a well-known **NP**-hard problem. Assume we are given an instance of the partition problem, that is, a set of positive integers $C = \{c_1, c_2, \ldots, c_m\}$ and we want to decide whether there is a subset $\mathbf{S}$ of $\{1, \ldots, m\}$ such that

$$\sum_{i \in \mathbf{S}} c_i = \frac{1}{2}\mathcal{M},$$

where $\mathcal{M} = \sum_{i=1}^{m} c_i$.

We transform this into an instance of bi-constrained maximum cost-efficiency pattern problem as follows. First, we create a path $v_0, v_1, \ldots, v_{2m}$ with $w_e = \ell_e = 1$ for each edge e on this path. Besides, we create additional m vertices, $p_1, p_2, \ldots, p_m$, and connect p_i to both v_{2i-2} and v_{2i} with $w_e = \ell_e = c_i + 1$ for $e \in \{p_i v_{2i-2}, p_i v_{2i}\}$. Then we create additional vertices q_0 and q_1, which we connect to v_0 and v_{2m}, respectively, such that $w_{q_0 v_0} = w_{q_1 v_{2m}} = 4\mathcal{M}$

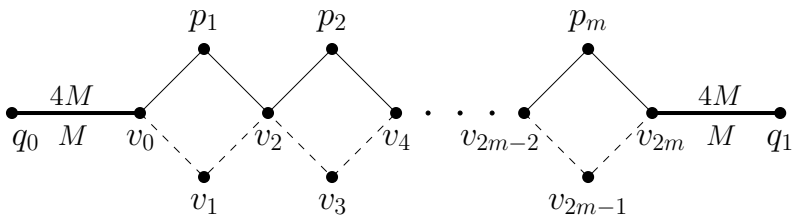


Figure 10.1: Graph used in the reduction from partition. Bold edges have cost-efficiency 4, all other edges have cost-efficiency 1. Dashed edges have weight and length 1, solid non-bold edges incident to p_i have weight and length $c_i + 1$.

and $\ell_{q_0 v_0} = \ell_{q_1 v_{2m}} = \mathcal{M}$. Furthermore, we set $\mathcal{W} = 9\mathcal{M} + 2m$ and $\mathcal{L} = \infty$. Since the graph is outerplanar, its treewidth is bounded by 2. This reduction is also illustrated in Fig. 10.1.

We claim that there is a path with length at least $\mathcal{W}$ and cost-efficiency at least

$$d := \frac{\mathcal{W}}{3\mathcal{M} + 2m}$$

if and only if there exists a subset $\mathbf{S}$ of $\{1, 2, \ldots, m\}$ such that

$$\sum_{i \in \mathbf{S}} c_i = \frac{1}{2}\mathcal{M}.$$

Clearly, any partition $\mathbf{S}$ of C can be transformed into a path with cost-efficiency d and length $\mathcal{W}$ as follows: Let $\mathbf{P}$ be the path from q_0 to q_1 that visits all vertices v_i with $i \in \mathbf{S}$ and that contains none of the vertices v_j with $j \notin \mathbf{S}$. This path has weight $8\mathcal{M} + 2m + 2\sum_{i \in \mathbf{S}} c_i = 9\mathcal{M} + 2m$ and length $2\mathcal{M} + 2m + \sum_{i \in \mathbf{S}} c_i = 3\mathcal{M} + 2m$. Hence, it has cost-efficiency d.

Conversely, assume that $\mathbf{P}$ is a path with cost-efficiency at least d. Since $\mathcal{W} \geq 9\mathcal{M}$, the path must have end at q_0 and q_1, respectively. Let $\mathbf{S}$ be the set of indices such that p_i is on the path if and only if $i \in S$. Then the cost-efficiency of this path can be expressed as

$$\frac{8\mathcal{M} + 2m + 2\sum_{i \in \mathbf{S}} c_i}{2\mathcal{M} + 2m + 2\sum_{i \in \mathbf{S}} c_i},$$

which is strictly decreasing as $2\sum_{i \in \mathbf{S}} c_i$ increases. Hence we have $2\sum_{i \in \mathbf{S}} c_i \leq \mathcal{M}$. On the other hand the weight of the path must be at least $\mathcal{W}$, which implies $2\sum_{i \in \mathbf{S}} c_i \geq \mathcal{M}$. Thus, $2\sum_{i \in \mathbf{S}} c_i = \mathcal{M}$. □

Notice that, we ignored the length upper-bound in the above reduction and the resulting instance is already **NP**-hard. When both the lower-bound and the upper-bound are imposed, the problem becomes much harder. By setting $\mathcal{L}$ to be $3M + 2m$, we can show that it is **NP**-hard even to compute a feasible solution. Hence, the problem is not likely to be approximable in polynomial time unless $\mathbf{P} = \mathbf{NP}$.

10.1 Cost-Efficiency Maximization for Trees and Almost-Trees

In the previous section, we have shown that it is **NP**-hard to compute a maximum cost-efficiency path even if the host graph has treewidth 2. Hence, the problem is unlikely to be *fixed-parameter tractable* with respect to the parameter treewidth.

In this section, we show, however, that the problem of computing a maximum cost-efficiency path is fixed-parameter tractable with respect to

$$k := |\mathbf{E}| - |\mathbf{V}|,$$

which is also the number of edges that must be deleted from a graph in order to obtain a tree. Note that, the treewidth of such a graph is bounded by $k + 1$. We prove this result in two steps. First, we show how to compute a maximum cost-efficiency path when the host is a tree. The problem can trivially be solved in $O(n^2)$ time by enumerating all possible paths. We show how it can be computed efficiently in $O(n \log^3 n)$ time. Our basic approach is similar to one described by Wu [93] and Lau et al. [60] with respect to decomposing the problem into smaller sub-problems. However, we use completely different techniques for the sub-problems to obtain our results, since the results by Wu and Lau et al. are not applicable in our setting. Second, we use this result to show that finding a maximum cost-efficiency path in a general graph is fixed-parameter tractable with respect to k.

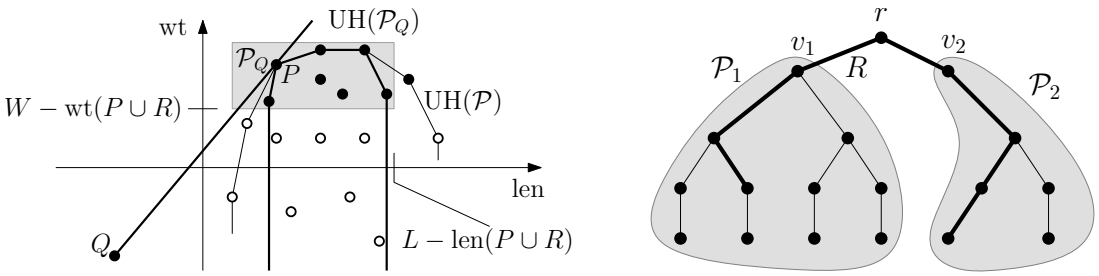


Figure 10.2: Tangent query to find the best candidate for Q (a) and combination of two paths using vertex r (b).

Throughout the section we use the key idea that the combined cost-efficiency of two disjoint subpaths P and Q is equal to the slope between two points $u_P = (\ell(P), w(P))$ and $-u_Q = (-\ell(Q), -w(Q))$ in the Euclidean plane. This path is viable if and only if

$$w(P) \geq \mathcal{W} - w(Q), \quad \text{and} \quad \ell(P) \leq \mathcal{L} - \ell(Q).$$

For a given query path Q this slope is maximized on the convex hull of the set of points $\mathcal{P}_Q$ representing the candidate subpaths for Q in the range $(-\infty, \mathcal{L} - \ell(Q)] \times [\mathcal{W} - w(Q), \infty)$ as illustrated in Figure 10.1. Since we wish to maximize the cost-efficiency, it suffices to perform tangent queries to the upper chain of the convex hull of $\mathcal{P}_Q$, denoted by $\mathrm{UH}(\mathcal{P}_Q)$, which is more efficient than trying all possible combinations.

We use a dynamic data structure for the maintenance of the upper chain of the convex hull of a set of points P [72] which allows point insertions in time $O(\log^2 n)$. It maintains the upper chain of the convex hull by a dynamically maintained ordered binary tree. Each leaf of this tree corresponds to a point in the plane and each inner node v corresponds to the segment of the *upper hull* of the set of points P_v that does not contribute to the upper hull of the set of points in the subtree of the father of v. Each inner node additionally stores the number of points on its upper hull that it inherits from its father and, whether these points are at its left or right boundary. The segments of the upper hulls are represented by concatenable queues which allow insertion, deletion, concatenation and split in $O(\log n)$ time. Lemma 10.2 shows how to compute $\mathrm{UH}(\mathcal{P}_Q)$ from a given set of candidate paths $\mathcal{P}$ in time $O(\log^2 n)$ given the dynamic data structure.

Lemma 10.2. *Given a point Q, a set of points $\mathcal{P}$ and a dynamic data structure for the maintenance of* $\mathrm{UH}(\mathcal{P})$ *as described in [72] the upper convex hull* $\mathrm{UH}(\mathcal{P}_Q)$ *can be computed in time $O(\log^2 n)$.*

Proof. If $\mathcal{P}_Q$ is empty, there is nothing to do. Otherwise, let $p_{\min}$ be the leftmost point in $\mathcal{P}_Q$ and let $p_{\max}$ be the rightmost point in $\mathcal{P}_Q$. They can be computed in $O(\log n)$ time using a dynamic priority search tree [69]. Let $x_{\min}$ and $x_{\max}$ be the respective x-coordinates of these points. Let $\mathcal{P}' := \{(x, y) \in \mathcal{P} \mid x_{\min} \leq x \leq x_{\max}\}$. First, we prove that $\mathrm{UH}(\mathcal{P}_Q) \subseteq \mathrm{UH}(\mathcal{P}')$. Clearly, $\mathcal{P}' = \mathcal{P}_Q \uplus \mathcal{P}''$ where $\mathcal{P}''$ contains all the points $(x, y) \in \mathcal{P}$ with $x_{\min} \leq x \leq x_{\max}$ and $y < W$. Thus, all points in $\mathcal{P}''$ are either in the interior of the convex hull of $\mathcal{P}_Q$ or on a vertical line through $x_{\min}$ or $x_{\max}$, respectively. Hence, the claim holds and we can reduce the problem of computing $\mathrm{UH}(\mathcal{P}_Q)$ to computing the upper hull of a set of points $\mathcal{P}'$ in a vertical strip of the plane, which is supported by the dynamic data structure. Let $\mathcal{T}$ denote the tree used by the dynamic data structure for the maintenance of the upper hull. In order to compute $\mathrm{UH}(\mathcal{P}')$ we traverse the paths from the root of $\mathcal{T}$ to $p_{\min}$ and $p_{\max}$, respectively, in parallel. In each step we reconstruct the upper hull using the concatenation and split operation of the concatenable queues stored in the nodes of the tree. We split off branches of the tree

that are to the left of the path from the root to $p_{\min}$ and to the right of the path from the root to $p_{\max}$. These branches contain only points whose x-coordinates are either greater than $x_{\max}$ or smaller than $x_{\min}$. Since $\mathcal{T}$ is balanced we have reconstructed $\mathrm{UH}(\mathcal{P}')$ after at most $\log n$ steps using time $O(\log n)$ per step and $O(\log^2 n)$ time in total. Clearly, we can reconstruct the original data structure with the same complexity. □

Theorem 10.3. *Given an instance $(T, w, \ell, \mathcal{W}, \mathcal{L})$ of the Bi-constrained Maximum Cost-Efficiency Pattern problem, where $T = (V, E)$ is a tree, a $(\mathcal{W}, \mathcal{L})$-viable maximum cost-efficiency path can be computed in $O(n \log^3 n)$ time.*

Proof. Without loss of generality we may assume that T is a binary tree. Otherwise we can make it binary by adding auxiliary vertices and edges with weight and length 0 in linear time such that the resulting tree has linear size. A *centroid* of a binary tree is a vertex whose removal disconnects T into at most three subtrees with at most half of the vertices of the original tree in each of the subtrees. We root T in one of its centroids r. Clearly, a centroid can be computed in linear time by aggregating weights of the tree starting in the leaves. Let v_1, v_2 be two children of r and let R be the path between v_1 and v_2 via r as illustrated in Figure 10.1. Then we can compute the maximum cost-efficiency path including R using tangent queries in time $O(n \log^2 n)$ as follows. First, we compute the set $\mathcal{P}_1$ of paths starting in v_1 and compute their cost-efficiency in linear time. Each of those paths $P \in \mathcal{P}_1$ is mapped to a point $u_P := (\ell(P \cup R), w(P \cup R))$ in the plane and inserted into the dynamic datastructure for the maintenance of the upper hull. This can be done in $O(n \log^2 n)$ time. Then we compute the set of paths $\mathcal{P}_2$ starting in v_2. For each of these paths $Q \in \mathcal{P}_2$ we want to compute the best path $P \in \mathcal{P}_1$, that is, a path P such that the concatenation of Q and $P \cup R$ has maximum cost-efficiency.

To this end, we map each $Q \in \mathcal{P}_2$ to a point $-u_Q := (-\ell(Q), -w(Q))$. Since we have bounds on both the weight and the length of a feasible solution, not all paths in $\mathcal{P}_1$ will be feasible partners for a given $Q \in \mathcal{P}_2$. We require that the length of $P \in \mathcal{P}_1$ is bounded by $w(P) \geq \mathcal{W} - w(Q) - w(R)$ and $\ell(P) \leq \mathcal{L} - \ell(Q) - \ell(R)$. Using Lemma 10.2 we can compute the maximum cost-efficiency partner for $Q \in \mathcal{P}_2$ in time $O(\log^2 n)$. During the computation of the upper convex hull we can simultaneously perform the necessary tangent queries using binary search on the constructed hull. Then we can compute the maximum cost-efficiency path P^* through r in time $O(n \log^2 n)$. We do this for all (at most 6) combinations of children of r and store the path of maximum cost-efficiency. Next, we recursively compute the best path through each of the children of r in the subtrees rooted in the children. Let $\hat{P}$ be the maximum cost-efficiency path over all the paths computed this way. Then the maximum cost-efficiency path in the tree rooted in r is the maximum cost-efficiency path over P^* and $\hat{P}$. The recurrence relation for the computation is given by $T(n) = \sum_{i=1}^{3} T(n_i) + O(n \log^2 n)$, where n_i is the number of vertices in the tree rooted in v_i. Hence, the running time of this approach is $O(n \log^3 n)$. □

Next, we show that we can obtain a similar result if the host is a graph that can be turned into a tree by deleting a fixed number of k edges. Roughly, the key idea consists of enumerating all possible subsets of the k edges and computing, for each of those subsets, the maximum cost-efficiency path containing all these edges. The following lemma can be used to enumerate these paths efficiently.

Lemma 10.4. *Given a graph $G = (V, E \cup F)$ such that $T = (V, E)$ is a tree and $F \cap E = \emptyset$ as well as $F' \subseteq F$ and vertices $s, t \in V$ incident to the edges in F', then there is at most one s-t-path in G containing all edges in F'. There is a linear-time algorithm that either computes such a path or concludes that no path exists.*

Proof. We prove the existence of at most one path by contradiction. Suppose that there are two different paths P_1 and P_2 both containing all edges in F' and ending with s and t, respectively. Since the paths are different and both contain all edges in F' the symmetric difference Δ of $E(P_1)$ and $E(P_2)$ is non-empty and contained in E. Since both paths end at s and t, all vertices of Δ have even degree. Hence, Δ contains a cycle contradicting the fact that Δ is a subgraph of T.

We proceed by showing that the uniquely determined feasible path can be computed in linear time, if it exists. We root T in some vertex $r \in V(T)$. By T_v we denote the tree rooted in $v \in V(T)$. For a given $F' \subseteq F$ we call $v \in V(T) \setminus \{s, t\}$ a *loose end* if it is incident to exactly one edge in F'. To compute P we traverse T in a bottom-up fashion constructing P by iteratively matching loose ends. For each vertex v we store a reference to the unmatched loose end, if it exists. Let v be a vertex with children $w_1, \ldots, w_\ell$. Clearly, there can only be a valid path if at most two children, say, w_1 and w_2, contain an unmatched loose end in their subtrees. Otherwise there is no feasible path. If none of the children contains an unmatched loose end, then there is nothing to do. If exactly one child contains an unmatched loose end in its subtree, we store a reference to this vertex in v. If exactly two children of v contain unmatched loose ends ℓ_1 and ℓ_2 in their subtrees, then we update the path by matching these loose ends and adding the unique path in T that connects ℓ_1 and ℓ_2. We accept the resulting graph if it is a path, which can be checked in linear time. □

Theorem 10.5. *Given an instance $(G, w, \ell, \mathcal{W}, \mathcal{L})$ of the Bi-constrained Maximum Cost-Efficiency Pattern problem such that G is a tree with k additional edges, we can compute a maximum cost-efficiency $(\mathcal{W}, \mathcal{L})$-viable path in time $O(2^k k^2 n \log^2 n + n \log^3 n)$.*

Proof. Given a tree with k additional edges $G = (V, E)$, we first compute an arbitrary spanning tree $T = (V, E')$ of G. This leaves exactly k edges, denoted by $F := E \setminus E'$, which may or may not be used by the optimal path. For each $F' \subseteq F$ we compute a maximum cost-efficiency path containing all edges in F' and we return the maximum cost-efficiency path over all $F' \subseteq F$. First, we compute the maximum cost-efficiency path in T in $O(n \log^3 n)$ time using Theorem 10.3. Any path containing some non-empty subset of edges $F' \subseteq F$ can be decomposed into three subpaths P, Q and R such that R starts and ends with edges in F' and contains all edges in F'. By Lemma 10.4 the possible paths R are uniquely determined by choosing a set $F' \subseteq F$ as well as two vertices incident to F'. Moreover, R can be computed in linear time from this information.

Hence, we iterate over all possible $F' \subseteq F$ and all $s, t \in V$ incident to F'. In each of the $2^k k^2$ iterations, we first compute both weight and length of R in linear time, resulting in $O(2^k k^2 n)$ time. Then we find paths P and Q starting at s and t, respectively, such that the cost-efficiency of the concatenation of P, R and Q has maximum cost-efficiency among all paths including R in time $O(\log^2 n)$. For the remainder of the proof we show how this can be accomplished and we thus assume that R, s and t are fixed. Our approach is similar to the proof of Theorem 10.3. However, we must take care of the disjointness of the paths.

Let $s = v_0, \ldots, v_\ell = t$ be the sequence of vertices on the path from s to t in T. For each of these vertices $v_i \neq s, t$, we define $W_s(v)$ as the set of vertices in T_{v_i} that are reachable from s in T without crossing the path R. Each of the vertices $w \in W_s(v_i)$ defines a path $P_s(w)$. Analogously, we define the set of vertices $W_t(v_i)$ in T_{v_i} that are reachable from t in T without crossing R. Each of those vertices $w \in W_t(v_i)$ defines a path $P_t(w)$ from t to w. Two paths in $P_s(v_i)$ and $P_t(v_j)$, respectively, are disjoint, whenever v_i is encountered before v_j on the path from s to t, that is, if $i < j$, otherwise they will have at least one vertex in common as illustrated in Figure 10.3.

Now we describe how we insert the paths into the dynamic data structure for the maintenance of the upper hull. As pointed out, paths may not be disjoint, hence, we must insert the paths in a specific order. First, we insert all paths starting in s that do not include any vertex on the

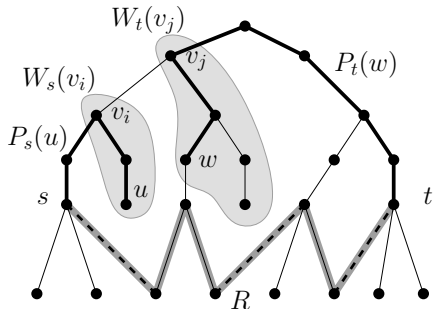


Figure 10.3: Illustration for the proof of Theorem 10.5. Extension of the unique path R using all edges in F (dashed) with end vertices s and t.

path from s to t. Then, for each $i = 1, \ldots, \ell - 1$ we insert all paths $P_s(w)$ for all $w \in W_s(v_i)$. After inserting the paths for a specific $i < \ell - 2$ we make tangent queries for all paths $P_t(w)$ for $w \in W_t(v_{i+1})$. Note that at that point, we have included all paths starting in s except those that would not be disjoint to the paths in $P_s(w)$ for $w \in W_t(v_{i+1})$. After we have inserted all paths $P_s(w)$ for all $w \in W_s(v_{\ell-1})$ we have inserted all paths starting in s, which do not cross R. Then we make tangent queries for all paths starting in t that do not use any vertex on the path from s to t.

In order to compute the best path for $F = \emptyset$ we proposed an algorithm with $O(n \log^3 n)$ running time. For each specific non-empty choice of $F' \subseteq F$ and vertices s and t incident to F' we thus insert at most n points into the data structure with a total running time of $O(n \log^2 n)$ and we perform at most n tangent queries, each with a running time of at most $O(\log^2 n)$. Hence, the overall running time is $O(2^k k^2 n \log^2 n + n \log^3 n)$. □

Note, that the relaxed maximum cost-efficiency pattern problem can be solved within the same asymptotic bounds by similar means if the pattern is a path and the host is a tree or a tree with k additional edges, respectively.

10.2 Graphs of Bounded Treewidth

In this section we show that a large class of problems can be solved in pseudo-polynomial **FPT** time when parameterized by the treewidth k of the host, that is, in time $O(f(k)p(\mathcal{L}, n))$ where f is a function depending only on k and p is a polynomial depending on the maximum length $\mathcal{L}$ of any feasible pattern and the number of vertices n of the graph. In the light of the results on the hardness of the problem this seems to be the best we can hope for. Given a graph G with treewidth k and a finite set of graphs $\mathcal{F}$, we wish to find a *connected* $(\mathcal{W}, \mathcal{L})$-viable pattern H with maximum cost-efficiency that does not contain any graph in $\mathcal{F}$ as a minor. Such a graph is called $\mathcal{F}$-minor-free. This includes trees, (outer-)planar graphs as well as graphs from various other minor-closed families of graphs. We give an algorithm for the general case but note that the running time can be improved by considering special classes of graphs. We assume that we are given a tree decomposition of the host as an input; otherwise it can be computed in FPT time [15, 58] with respect to the treewidth of the graph. We note that the size of the forbidden obstructions is small for many interesting examples, such as trees and (outer-)planar graphs whose sets of forbidden minors include graphs with ≤ 6 vertices.

Our algorithm is based on dynamic programming on the tree decomposition of the graph and is inspired by Eppstein's work on subgraph isomorphism in planar graphs [28]. Based on Eppstein's idea of enumerating partial isomorphisms for the bags of the tree decomposition, we enumerate partial minors of the graphs induced by the bags. In the following we present the key ideas in more detail. Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $(\mathcal{X}, \mathcal{T})$

where $\mathcal{X} = \{X_i \mid i \in I\}$ is a collection of subsets of V which are called *bags* and $\mathcal{T} = (I, E_\mathcal{T})$ is a tree with the following properties.

1. The union of all bags $\bigcup_{i \in I} X_i$ is equal to V.
2. For all edges $e \in E$ there is an index $i \in I$ such that $e \subseteq X_i$.
3. For all vertices $v \in V$, the tree induced by the set of nodes $\mathcal{X}_v = \{i \in I \mid v \in X_i\}$ induces a connected subtree of $\mathcal{T}$.

We will refer to the elements in I as nodes—as opposed to vertices in the original graph. The *treewidth* of a tree decomposition equals $\max_{i \in I} |X_i| - 1$. The treewidth of a graph $G = (V, E)$ is equal to the minimum treewidth of a tree decomposition of G.

Theorem 10.6. *Let $(G, w, \ell, \mathcal{W}, \mathcal{L})$ be an instance of the Bi-constrained Maximum Cost-Efficiency Pattern problem such that G has treewidth at most k, let $\mathcal{F}$ be a non-empty finite set of graphs and let $N := \max_{F \in \mathcal{F}} |V(F)|$. Then a maximum cost-efficiency connected $\mathcal{F}$-minor-free $(\mathcal{W}, \mathcal{L})$-viable pattern can be computed in $2^{O(k^2 + k \log N + N)} |\mathcal{F}| Ln$ time.*

Proof. We describe the algorithm for the case that $\mathcal{F}$ consists of only one forbidden obstruction F. The extension to a larger family of forbidden obstructions is straightforward. Our algorithm is by dynamic programming on the tree decomposition of the graph. Robertson and Seymour have proven the existence of an $O(n^3)$ graph minor test for any fixed minor F [78]. However, the proof is non-constructive and involves huge constants. Therefore, we describe an explicit algorithm for the graph minor test which relies on the enumeration of subgraphs instead. We note, however, that we need some explicit representation of the minor mappings for the dynamic programming anyway, thus, this does not change the asymptotic complexity of our approach.

For the proof we assume that we are given a nice tree decomposition. A tree decomposition is called *nice* if $\mathcal{T}$ is a rooted binary tree where each node is of one of the following types: A *leaf node* X contains only one vertex. An *introduce node* X has only one child Y such that $X = Y \cup \{v\}$ for some $v \in V$. We say X *introduces* v. A *forget node* X has only one child Y such that $X = Y \setminus \{v\}$ for some $v \in V$. We say X *forgets* v. Finally, a *join node* X has two children Y_1 and Y_2 such that $X = Y_1 = Y_2$. Given a graph with treewidth k, we can always find a nice tree decomposition with $O(n)$ nodes in linear time [58].

Throughout the proof we assume that $G = (V, E)$ is a graph with treewidth at most k and we let $(\mathcal{X}, \mathcal{T})$ be a nice tree decomposition of G with treewidth at most k, rooted in a node $r \in I$. For $i \in I$ we denote the graph induced by the union of the bags of all descendants of i (including i) by G_i. Using standard notation, we denote the graph induced by X_i by $G[X_i]$. Let $\mathcal{C} := \{V_1, \ldots, V_q\}$ be a disjoint collection of connected subsets of V. We denote the graph obtained by contracting the vertices in each of the sets V_i into a single vertex by $G/\mathcal{C}$ and that we refer to the sets V_i as *branch sets* and to $\mathcal{C}$ as a *contraction set*. Then F is a minor of G iff there is a subgraph H and a contraction set $\mathcal{C}$ such that $H/\mathcal{C}$ is isomorphic to F.

Let $\mathcal{C}$ be a contraction set and let H be some subgraph in $G[X_i]$ for some $i \in I$. A *partial minor embedding of F into H with respect to $\mathcal{C}$* is a mapping $\varphi : V(F) \to V(H/\mathcal{C}) \cup \{\bot, \top\}$ such that $uv \in E(F) \Rightarrow \varphi(u)\varphi(v) \in E(H/\mathcal{C})$ for all $uv \in E(F)$ with $u, v \notin \varphi^{-1}(\bot) \cup \varphi^{-1}(\top)$, hence, φ maps a subgraph of F to a minor of H. The image $\bot$ represents vertices in $G_i - X_i$ and the image $\top$ represents vertices in G which have not been considered yet, that is, vertices in $G - G_i$. A partial minor embedding φ is called *proper* if and only if $\varphi^{-1}(\top) \neq \emptyset$. Otherwise it represents a minor embedding of F into some subgraph of G_i, and hence, H must be disregarded as a partial solution.

For the algorithm we identify the vertices of F with the numbers $1, \ldots, |V(F)|$. The images of these vertices under φ that are not contained in $\varphi^{-1}(\bot) \cup \varphi^{-1}(\top)$ correspond to branch-sets

of H, that is, a partition of the vertices of H. By considering all $|V(F)+1|^{k+1}$ labelings of the vertex set of H where each vertex is labeled with some number in $0,\ldots,V(F)$, we obtain a partition of the vertices induced by the labeling. Such a partition is valid only if each set of vertices forms a connected set. Further, it defines an implicit mapping f of a subset of the vertices of F to the partitions induced by the labeling. Vertices labeled 0 are considered not to be images under f. We further encode for each vertex in F that does not have an image under f whether it is mapped to $\perp$ or $\top$. A mapping to $\perp$ means that the vertex can be mapped to some branch-set in the subgraph induced by the descendants of node i whereas a mapping to $\top$ means that we will try to map the vertex to some branch-set we have not encountered, yet. We can check in $O(k^2)$ time if such an encoding represents a valid partial minor embedding of F into H. We check connectedness of the partitions in time $O(k)$. Further, we check if each edge in F is represented by some edge between the corresponding branch-sets in H. This can be done in time $O(k^2)$ by iterating over all pairs of vertices in H, and hence, over all pairs of labels and checking corresponding edges in both F and H. Using these conventions, we can encode a partial minor embedding φ. It is not hard to see that there are at most $|V(F)|^{k+1}2^{|V(F)|}$ many partial minor embeddings using this kind of encoding.

By $W(i,H,\Phi,\ell)$ we denote the maximum weight of a subgraph G' of G_i with length ℓ, such that $G'[X_i]=H\subseteq G[X_i]$ and Φ represents all partial minor embeddings φ of F into G'. We call the quadruple (i,H,Φ,ℓ) an *interface for* i. An interface is called *proper* if and only if $\varphi(\top)\neq\emptyset$ for all $\varphi\in\Phi$.

By G_i^* we denote the graph obtained by adding new vertices $\top$ and $\perp$ to $G[X_i]$ which are each connected to all vertices in X_i. Both weight and length of the additional edges is equal to zero. We then consider connected subgraphs in G_i^*. Note that any *connected* subgraph G' can be mapped to a *connected* subgraph in G_i^*.

The solution we are looking for will be the maximum over all interfaces (r,H,Φ,ℓ) where r is the root of the tree decomposition, such that H is connected and does not contain $\top$, Φ is proper and ℓ is at most $\mathcal{L}$. If the maximum weight is at least $\mathcal{W}$, then we return this weight, otherwise there is no feasible solution. We now describe how $W(i,H,\Phi,\ell)$ can be computed in $\mathcal{T}$ in a bottom-up fashion by dynamic programming starting at the leaves of $\mathcal{T}$.

Leaf node i with $X_i=\{v\}$: For each subgraph H in G_i^* that does not include $\perp$ we compute the set Φ of partial minor embeddings of F into H and we set $W(i,H,\Phi,0)=0$, since both the weight and length of any subgraph of G_i^* are equal to 0 by construction. Note that any vertex in F which is not mapped to v must be mapped to $\top$. Hence, the time complexity is asymptotically bounded by

$$\underbrace{O(1)}_{\text{subgraphs } H}\cdot\underbrace{|V(F)|^2\cdot|V(F)|}_{|\Phi|}\cdot\underbrace{O(1)}_{\text{check mapping}}\cdot\mathcal{L}\,.$$

Introduce node i introducing v: Let j be the only child of i and let (j,H',Φ',ℓ') be an interface for j such that $W_j:=W(j,H',\Phi',\ell')$. We consider all connected subgraphs H of G_i^* which can be obtained from H' by adding v and some set of edges E^+ incident to both v and some set of vertices in H'. For each fixed H obtained this way, we further consider the set Φ of all partial minor embeddings φ of F into H that can be obtained from some $\varphi'\in\Phi'$ by choosing some vertex in $\varphi'^{-1}(\top)$ to be mapped to v by φ. If all partial minor embeddings φ constructed this way are proper and $\ell:=\ell'+\ell(E^+)\leq\mathcal{L}$, the interface (i,H,Φ,ℓ) is proper, and we compute $W(i,H,\Phi,\ell)=W_j+w(E^+)$ and set $W(i,H',\Phi',\ell')=W_j$. Hence, the complexity for an introduce node is asymptotically bounded by

$$\underbrace{2^{\binom{k}{2}}}_{\text{subgraphs } H}\cdot\underbrace{|V(F)|^{k+1}\cdot 2^{|V(F)|}}_{|\Phi'|}\cdot\underbrace{2^k}_{|E^+|}\cdot\underbrace{|V(F)|}_{\text{new mappings to } v}\cdot\underbrace{k^2}_{\text{check mapping}}\cdot\mathcal{L}$$

Forget node i forgetting v: Let j be the only child of i and let (j, H', Φ', ℓ') be an interface for j such that $W_j := W(j, H', \Phi', \ell')$. If H' does not contain v, then there is nothing to do and we simply set $W(i, H', \Phi', \ell') = W_j$. Otherwise, we consider the set Φ of all mappings φ that can be obtained from mappings φ' by removing v from its partition in the branch set. If v is the only vertex in its partition, then the corresponding vertex in F must additionally be mapped to $\perp$. We set $W(i, H, \Phi, \ell') = W_j$ where H is obtained from H' by removing v and mapping all edges from v to any vertex in X_i by a corresponding edge with the end-vertex corresponding to v in $\perp$. The resulting complexity of a forget node is asymptotically bounded by

$$\underbrace{2^{\binom{k}{2}}}_{\text{subgraphs } H} \cdot \underbrace{|V(F)|^{k+1} \cdot 2^{|V(F)|}}_{|\Phi'|} \cdot \underbrace{O(1)}_{\text{remapping}} \cdot \mathcal{L}$$

Join node i joining j_1 and j_2: Let j_1 and j_2 be the two children of i and consider two interfaces of j_1 and j_2, respectively, given by (j_1, H, Φ_1, ℓ_1) and (j_2, H, Φ_2, ℓ_2). Two partial minor-embeddings $\varphi_1 \in \Phi_1$ and $\varphi_2 \in \Phi_2$ are compatible if all vertices $v \in X_{j_1} \cap X_{j_2}$ satisfy $\varphi_1^{-1}(v) = \varphi_2^{-1}(v)$. If φ_1 and φ_2 are compatible, we can obtain a new partial minor embedding by combining the two partial embeddings into a new partial minor embedding φ_{12}.

Let Φ_{12} be the set of partial minor embeddings combined in this manner from all pairs of compatible partial minor embeddings in $\Phi_1 \times \Phi_2$. Let $W_1 := W(j, H, \Phi_1, \ell_1)$ and $W_2 := W(j', H, \Phi_2, \ell_2)$. If $\ell := \ell_1 = \ell_2$ and $\Phi := \Phi_1 = \Phi_2$ we set $W(i, H, \Phi_1, \ell) = \max\{W_1, W_2\}$. Otherwise, we set $W(i, H, \Phi_1, \ell_1) = W_1$ and $W(i, H, \Phi_2, \ell) = W_2$. Additionally, we set $W(i, H, \Phi_1 \cup \Phi_2 \cup \Phi_{12}, \ell_1 + \ell_2 - \ell(H)) := W_1 + W_2 - w(H)$. Clearly, the weight and length of H must be subtracted, since otherwise, these values would be counted twice.

Since $|\Phi_1 \times \Phi_2|$ is bounded by $|V(F)|^{k+1} \times |V(F)|^{k+1}$, the resulting complexity in total is asymptotically bounded by

$$2^{\binom{k}{2}} \cdot |V(F)|^{2k+2} \cdot 2^{2|V(F)|} \cdot 2^k \cdot |V(F)| \cdot k^2 \cdot M \cdot n = 2^{O(k^2 + k \log |V(F)| + |V(F)|)} Mn \,.$$

If $\mathcal{F}$ contains more than one obstruction, the running time can be bounded by

$$2^{O(k^2 + k \log N + N)} |\mathcal{F}| \mathcal{L} n;\,,$$

where N denotes the maximum number of vertices of any graph in $\mathcal{F}$. □

The following result can be obtained by a straightforward modification of the approach sketched in this section. With the technique developed in the next section, this result will yield an FPTAS for the relaxed maximum cost-efficiency pattern problem.

Corollary 10.7. *Let $(G, w, \ell, \mathcal{W})$ be an instance of the Relaxed Maximum Cost-Efficiency Pattern problem and let G and $\mathcal{F}$ be as in Theorem 10.6. Then for any $\lambda \in \mathbb{R}$ a maximum penalized cost-efficiency $\mathcal{F}$-minor-free $(\mathcal{W}, \lambda)$-viable pattern can be computed in time $2^{O(k^2 + k \log N + N)} |\mathcal{F}| \lambda n$ where $N = \max_{F \in \mathcal{F}} |V(F)|$.*

10.3 An FPTAS for the Relaxed Cost-Efficiency Maximization

In this section we consider the Relaxed Maximum Cost-Efficiency Pattern problem, where the upper bound on the length may be violated at the cost of an additional penalty term. We assume that the weight function is strictly positive. While it is **NP**-hard to decide whether a feasible solution exists for the original problem, we show that this slight relaxation allows us

ALGORITHM *FPTAS-RMDS*

1: $k \longleftarrow \lceil \frac{2}{\epsilon} \rceil$.
2: **for** $i = 0 \to \lfloor \log_k B \rfloor$ **do**
3: $\quad H_i \leftarrow$ result of $\mathcal{A}$ on instance I_i with $\lambda = k^2 m$.
4: **end for**
5: return $\max_{0 \leq i \leq \lfloor \log_k B \rfloor} \widetilde{\varrho}_i(H_i)$ as the approximating solution.

Figure 10.4: The high-level description of the $(\ln n)$-approximation for the inseparable demand model.

to give an FPTAS for penalized cost-efficiency. This FPTAS can be applied to any problem that allows a quasi-polynomial-time algorithm that computes an optimal solution with respect to the penalized cost-efficiency. Note that the relaxed cost-efficiency maximization problem remains NP-hard as we can choose L very small such that every subgraph is penalized and we have that $\widetilde{\varrho}(H) \approx \varrho(H)/2$ for any subgraph H. Then the NP-hardness result of Theorem 10.1 naturally applies to this problem. For simplicity, we will assume that the scaling constant c for the penalized cost-efficiency equals 1.

Let Π be a relaxed cost-efficiency maximization problem that admits an algorithm $\mathcal{A}$ that takes as input an instance I of the relaxed cost-efficiency maximization problem and $\lambda \in \mathbb{N}$ and computes an optimal $(\mathcal{W}, \lambda)$-viable pattern H with respect to penalized cost-efficiency, $\widetilde{\varrho}$, in $O(p(\lambda, n))$ time, where $p(\lambda, n)$ is a function that is polynomial in λ and n. We show how to construct an FPTAS for Π that uses $\mathcal{A}$ as a subroutine. We present our algorithm within the terminology introduced by Schuurman and Woeginger for approximation schemes [81]. We first structure the output of our algorithm to form exponentially growing buckets based on the length of the solutions. In order to compute approximately optimal solutions in each of the buckets efficiently, we structure the input of algorithm $\mathcal{A}$ by exponentially compressing the lengths and weights in such a way that the error resulting from the compression is proportional to the size of the solutions in each bucket.

Assume that we are given a graph T. Let k be a suitably chosen integer depending on ε, which will be defined later. We structure the output in $\lfloor \log_k B \rfloor - 1$ buckets, where $B = \ell(G)$, such that bucket i with $0 \leq i \leq \lfloor \log_k B \rfloor - 2$ contains solutions with total length at most $k^{i+2}m$, where m is the number of edges. For each bucket we compute an approximately optimal solution and return the overall best solution as output of our algorithm. To compute an approximately optimal solution for bucket i, we structure the input by considering instances $I_i = (G, \ell_i, w_i, W_i, L_i)$, where $\ell_i(e) = \lceil \ell(e)/k^i \rceil$, $w_i(e) = w(e)/k^i$ for $e \in E(G)$ and $W_i = W/k^i$ as well as $L_i = L/k^i$. We can think of these instances as being *compressed*. We apply algorithm $\mathcal{A}$ on the compressed instances I_i with $\lambda = k^2 m$. A high-level description of this algorithm is listed as Algorithm 10.4.

When considering the i-th bucket, we refer to the deviation of $H \subseteq G$ with respect to ℓ_i and L_i as the *compressed deviation* $\Delta_i(H) = \max\{0, \ell_i(H) - L_i\}$. Similarly, the penalized cost-efficiency of $H \subseteq G$ is defined as the *compressed penalized cost-efficiency* $\widetilde{\varrho}_i(H) = w_i(H)/(\ell_i(H) + \Delta_i(H))$.

In order to show that Algorithm 10.4 is an FPTAS we proceed in several steps. First, we bound the compressed penalized cost-efficiency used in the i-th iteration of the algorithm in terms of the ordinary penalized cost-efficiency. In Lemma 10.10 we use this bound to derive an approximation ratio for the penalized cost-efficiency. Finally, we show that the algorithm is an FPTAS in Theorem 10.11.

Lemma 10.8. *For any pattern H and each $1 \leq i < \lfloor \log_k B \rfloor$, the following holds*

$$\begin{aligned} \ell(H) &\leq k^i \cdot \ell_i(H) &&\leq \ell(H) + |E(H)| \cdot k^i, && (10.1)\\ \Delta(H) &\leq k^i \cdot \Delta_i(H) &&\leq \Delta(H) + |E(H)| \cdot k^i, && (10.2)\\ \widetilde{\varrho}_i(H) &\leq \widetilde{\varrho}_{i-1}(H) &&\leq \widetilde{\varrho}(H), && (10.3)\\ \ell_i(H) \leq k \cdot m \quad &\textit{implies that} \quad &&\ell_{i-1}(H) \leq k^2 \cdot m\,. && (10.4) \end{aligned}$$

Proof. We use the following equation, which holds for any real positive numbers $r, s \in \mathbb{R}$.

$$r \leq s \cdot \left\lceil \frac{r}{s} \right\rceil \leq r + s \qquad (\star)$$

We start out by proving Equation (10.1), which relates the length of a graph to the length of the corresponding subgraph in the compressed instance of iteration i. By the definition of ℓ and Equation $(\star)$ we have

$$\begin{aligned} \ell(H) &= \sum_{e \in E(H)} \ell_e \\ &\leq k^i \cdot \sum_{e \in E(H)} \left\lceil \frac{\ell_e}{k^i} \right\rceil && \text{by Equation } (\star) \\ &= k^i \ell_i(H) \end{aligned}$$

On the other hand,

$$\begin{aligned} k^i \ell_i(H) &= k^i \cdot \sum_{e \in E(H)} \left\lceil \frac{\ell_e}{k^i} \right\rceil \\ &\leq \sum_{e \in E(H)} (\ell_e + k^i) && \text{by Equation } (\star) \\ &= \ell(H) + |E(H)| \cdot k^i\,. \end{aligned}$$

Next, we consider Equation (10.2). Note that the first inequality trivially holds if $\Delta(H) = 0$. So, we may assume that $\Delta(H) > 0$. By applying Equation (10.1) and the definition of the compressed deviation we obtain

$$\Delta(H) = \ell(H) - \mathcal{L} \leq k^i \cdot \ell_i(H) - \mathcal{L} \leq k^i \cdot \Delta_i(H)\,.$$

The second inequality of Equation (10.2) again trivially holds if $\Delta_i(H) = 0$. For $\Delta_i(H) > 0$, we obtain

$$k^i \Delta_i(H) = k^i \ell_i(H) - \mathcal{L} \leq \ell(H) + |E(H)| \cdot k^i - \mathcal{L} = \Delta(H) + |E(H)| \cdot k^i$$

using Equation (10.1) and the definition of the compressed deviation.

Finally, we consider Equations (10.3) and (10.4). Note that the first inequality of Equation (10.3) implies the second inequality since $\widetilde{\varrho}_0(H) = \widetilde{\varrho}(H)$ holds for any subgraph H. From Inequality $(\star)$ we obtain that

$$\ell_{i-1}(H) = \sum_{e \in E(H)} \left\lceil \frac{\ell_e}{k^{i-1}} \right\rceil \leq \sum_{e \in E(H)} k \cdot \left\lceil \frac{\ell_e}{k \cdot k^{i-1}} \right\rceil = k \cdot \ell_i(H)$$

for any subgraph H, which immediately implies Equation (10.4). Similarly, we also obtain

$$k \cdot \Delta_i(H) = k \cdot \max\{0, \ell_i(H) - L\} \geq k \cdot \max\{0, \ell(H) - L\} = \Delta_{i-1}(H)$$

Therefore, we obtain

$$\widetilde{\varrho}_i(H) = \frac{w_{i-1}(H)}{k \cdot (\ell_i(H) + \Delta_i(H))} \leq \frac{w_{i-1}(H)}{\ell_{i-1}(H) + \Delta_{i-1}(H)} = \widetilde{\varrho}_{i-1}(H)$$

using $w_i(H) = w_{i-1}(H)$ and $\ell_i(H)+\Delta_i(H) = k\cdot(\ell_i(H)+\Delta_i(H))$, which concludes the proof. □

Let $\Omega(H)$ be the smallest integer such that $\ell_{\Omega(H)}(H) \leq k^2 m$. In other words, $\Omega(H)$ denotes the smallest bucket for which H will be considered by algorithm $\mathcal{A}$. Equation (10.4) immediately implies a lower bound on the length of H in this bucket.

Corollary 10.9. *For any pattern H, $\Omega(H) > 0$ implies $\ell_{\Omega(H)}(H) > km$.*

Now we are ready to bound the penalized cost-efficiency of an instance H in bucket $\Omega(H)$ in terms of k and its true penalized cost-efficiency $\widetilde{\varrho}(H)$.

Lemma 10.10. *For any pattern H, we have $\widetilde{\varrho}_{\Omega(H)}(H) \geq \frac{k-1}{k+1} \cdot \widetilde{\varrho}(H)$.*

Proof. Clearly, this inequality holds if $\Omega(H) = 0$. For $\Omega(H) \geq 1$, we have

$$\begin{aligned}
\ell(H) &\geq k^{\Omega(H)} \cdot \ell_{\Omega(H)}(H) - |E(H)| \cdot k^{\Omega(H)} && \text{by Equation (10.1)} \\
&\geq k^{\Omega(H)} \cdot (\ell_{\Omega(H)}(H) - m) && \text{since } |E(H)| \leq m. \\
&\geq k^{\Omega(H)} \cdot (km - m) && \text{due to Corollary 10.9} \\
&= (k-1) \cdot k^{\Omega(H)} m \,. && (10.5)
\end{aligned}$$

This implies

$$k^{\Omega(H)} m \leq \frac{\ell(H)}{k-1} \leq \frac{\ell(H) + \Delta(H)}{k-1} \,. \tag{10.6}$$

Then the penalized cost-efficiency satisfies

$$\begin{aligned}
\widetilde{\varrho}_{\Omega(H)}(H) &= \frac{w_{\Omega(H)}(H)}{\ell_{\Omega(H)}(H) + \Delta_{\Omega(H)}(H)} && \text{by definition of } \widetilde{\varrho} \\
&= \frac{w(H)}{k^{\Omega(H)} \cdot (\ell_{\Omega(H)}(H) + \Delta_{\Omega(H)}(H))} && \text{by definition of } w_{\Omega(H)} \\
&\geq \frac{w(H)}{\ell(H) + \Delta(H) + 2|E| \cdot k^{\Omega(H)}} && \text{by Equations (10.1) and (10.2)} \\
&\geq \frac{w(H)}{\ell(H) + \Delta(H) + 2m \cdot k^{\Omega(H)}} && \text{since } |E| \leq m \\
&\geq \frac{w(H)}{\left(1 + \frac{2}{k-1}\right)(\ell(H) + \Delta(H))} && \text{by Equation (10.6)} \\
&= \frac{k-1}{k+1} \cdot \widetilde{\varrho}(H) \,.
\end{aligned}$$

This concludes the proof. □

Theorem 10.11. *Given $0 < \varepsilon < 1$, we can compute a $(1-\varepsilon)$-approximation for the relaxed cost-efficiency maximization problem in $O(p(m/\varepsilon^2, n) \log B)$ time, where G is the input graph and B is the maximum total length of the edges, provided that an $O(p(\lambda, n))$ time algorithm for the penalized cost-efficiency maximization as described above exists.*

Proof. Clearly, the algorithm computes a $\mathcal{W}$-viable solution if one exists due to the correctness of algorithm $\mathcal{A}$, the fact that we do not introduce any errors when scaling the weights, and since the union of the buckets covers all feasible solutions.

Next we show that the algorithm indeed produces a $(1-\varepsilon)$ approximation of the optimal penalized cost-efficiency. Let opt be an optimal solution and H^* be the solution returned by our algorithm. By the above lemmas and choosing $k = \lceil \frac{2}{\varepsilon} \rceil$, we have

$$\begin{aligned}
\widetilde{\varrho}(H^*) &\geq \max_{i \geq \Omega(H^*)} \widetilde{\varrho}_i(H^*) && \text{by Algorithm 10.4} \\
&\geq \max_{i \geq \Omega(\text{opt})} \widetilde{\varrho}_i(\text{opt}) && \text{by Algorithm 10.4} \\
&\geq \widetilde{\varrho}_{\Omega(\text{opt})}(\text{opt}) && \text{by Equation (10.3)} \\
&\geq \left(\frac{k-1}{k+1}\right) \cdot \widetilde{\varrho}(\text{opt}) && \text{by Lemma 10.10} \\
&= \left(1 - \frac{2}{k+1}\right) \cdot \widetilde{\varrho}(\text{opt}) \\
&\geq (1-\varepsilon) \cdot \widetilde{\varrho}(\text{opt}) && \text{by definition of } k.
\end{aligned}$$

The running time of this approach clearly is $O(p(m/\varepsilon^2, n) \log B)$ since $k = \lceil \frac{2}{\varepsilon} \rceil \geq 2$, and $\log_k B \leq \log_2 B = O(\log B)$. □

For reasons of simplicity, we assumed a scaling factor $c = 1$. By choosing $k = \lceil c + 1/\varepsilon \rceil$ we can accomplish the same result for any scaling factor $c \neq 1$. In our analysis, we further assumed that we are given an algorithm $\mathcal{A}$ that computes a $(\mathcal{W}, \lambda)$-viable pattern for a given value of λ. However, our approach still works if $\mathcal{A}$ only computes a $\mathcal{W}$-viable pattern with maximum penalized cost-efficiency. In each iteration we pre-process the instance I_i by removing edges that are longer than $k^2 m$ from G. Then the maximum length of any W-viable pattern considered by $\mathcal{A}$ is naturally bounded by $k^2 m^2$. The running time of the resulting FPTAS is bounded by $O(p(m^2/\varepsilon^2, n) \log B)$, assuming that $\mathcal{A}$ has a running time bounded by $O(p(\ell(G), n))$. Finally, with the results from Corollary 10.7 we immediately obtain the following result as an application of the FPTAS to the problem of maximizing the penalized cost-efficiency objective function.

Corollary 10.12. *Let $(G, w, \ell, \mathcal{W})$ be an instance of the Relaxed Maximum Cost-Efficiency Pattern problem such that G has treewidth at most k and let $\mathcal{F}$ be a finite set of graphs. Let $B := \ell(G)$, $0 < \varepsilon < 1$ and let* opt *be the optimal penalized cost-efficiency of an $\mathcal{F}$-minor-free $\mathcal{W}$-viable pattern. Then a $\mathcal{W}$-viable $\mathcal{F}$-minor-free pattern with penalized cost-efficiency at least $(1-\varepsilon) \cdot$ opt can be computed in time $O(2^{O(k^2 + k \log N + N)} |\mathcal{F}| m/\varepsilon^2 \log B)$.*

Chapter 11

Maximizing Cost-Efficiency under Structural Constraints

In this section, we drop the lower bound on the weight as well as the upper bound on the length. Instead, we impose structural constraints on the set of vertices by requiring a subset of the vertices to be contained in any feasible solution. This models a scenario in which we would like to inter-connect a subset of the given vertices in a certain way. For instance, we may wish to connect the sites to form a spanning tree or a perfect matching or we may wish to inter-connect a (small) subset of the vertices.

11.1 A Parametric Searching Approach and its Application

One of the most natural constraints on the vertex set is to require the pattern to span the whole set of vertices. Chandrasekaran [17] shows that a spanning tree with maximum cost-efficiency can be computed in polynomial time. We provide an adapted version of the main theorem that makes this possible, along with a proof of its correctness, and show how this can be used as a generic tool in order to obtain efficient algorithms for the maximum cost-efficiency pattern problem.

Theorem 11.1 (Chandrasekaran [17])**.** *Let $G = (V, E)$ be a graph with edge weights $a_e \in \mathbb{Z}$, $b_e \in \mathbb{N}$ for $e \in E$ such that $b_e > 0$ for all $e \in E$ and let $\mathcal{S} \in 2^E$ be an arbitrary collection of subsets of the edges. Let*

$$\theta^* = \max_{X \in \mathcal{S}} \left\{ \frac{\sum_{e \in X} a_e}{\sum_{e \in X} b_e} \right\} \quad \textit{and} \quad \varphi(\theta) = \max_{X \in \mathcal{S}} \left\{ \sum_{e \in X} (a_e - \theta b_e) \right\} ,$$

then

$$\varphi(\theta) = \begin{cases} > 0 & \Leftrightarrow \quad \theta < \theta^* \\ = 0 & \Leftrightarrow \quad \theta = \theta^* \\ < 0 & \Leftrightarrow \quad \theta > \theta^* \end{cases} .$$

ALGORITHM *Parametric Search*

Input: Graph $G = (V, E)$, $a_e \in \mathbb{Z},\ b_e \in \mathbb{N}$ for $e \in E$, set of feasible solutions $\mathcal{S} \subseteq 2^E$ **Output:** $S^* \subseteq \mathcal{S}$ with maximum cost-efficiency

$[\alpha, \beta] \leftarrow \left[\min_{e\in E}\left\{\frac{a_e}{b_e}\right\}, \max_{e\in E}\left\{\frac{a_e}{b_e}\right\}\right]$.
while $\beta - \alpha \geq \left(\sum_{e\in E} b_2\right)^{-2}$ **do**
  $k \leftarrow \frac{\alpha+\beta}{2}$.
  $d \leftarrow \max_{X\in\mathcal{S}}\left\{\sum_{e\in X}(a_e - k \cdot b_e)\right\}$.
  if $d > 0$ **then**
    $[\alpha, \beta] \leftarrow [k, \beta]$.
  else if $d < 0$ **then**
    $[\alpha, \beta] \leftarrow [\alpha, k]$.
  else
    $[\alpha, \beta] \leftarrow [k, k]$.
  end if
end while
$S_\alpha \leftarrow \operatorname{argmax}_{X\in\mathcal{S}}\left\{\sum_{e\in X}(a_e - \alpha \cdot b_e)\right\}$.
$S_\beta \leftarrow \operatorname{argmax}_{X\in\mathcal{S}}\left\{\sum_{e\in X}(a_e - \beta \cdot b_e)\right\}$.
if $\varrho(S_\alpha) > \varrho(S_\beta)$ **then**
  return S_α.
else
  return S_β.
end if

Figure 11.1: A high-level description of the parametric search.

Proof. Let $X^* \in \mathcal{S}$ be such that $\varphi(\theta) = \sum_{e\in X^*}(a_e - \theta \cdot b_e)$:

$$\begin{aligned}
& & \varphi(\theta) = \sum_{e\in X^*}(a_e - \theta \cdot b_e) < 0 & & \\
\Leftrightarrow & & \sum_{e\in X}(a_e - \theta \cdot b_e) < 0 & & \forall X \in \mathcal{S} \\
\Leftrightarrow & & \frac{\sum_{e\in X} a_e}{\sum_{e\in X} b_e} < \theta & & \forall X \in \mathcal{S} \\
\Leftrightarrow & & \theta^* < \theta & &
\end{aligned}$$

On the other hand, assume $\theta^* > \theta$. Then and only then there is some $X' \in \mathcal{S}$ such that

$$\begin{aligned}
& & \frac{\sum_{e\in X'} a_e}{\sum_{e\in X'} b_e} > \theta & \\
\Leftrightarrow & & \exists X' \in \mathcal{S} \ : \ \sum_{e\in X'}(a_e - \theta \cdot b_e) > 0 & \\
\Leftrightarrow & & \varphi(\theta) = \sum_{e\in X^*}(a_e - \theta \cdot b_e) > 0\ . &
\end{aligned}$$

Equality for the case $\varphi(\theta) = 0$ follows from the previous observations. □

In order to solve maximum cost-efficiency pattern problems we adapt the optimization algorithm suggested by Chandrasekaran to our setting. Essentially, the algorithm performs binary

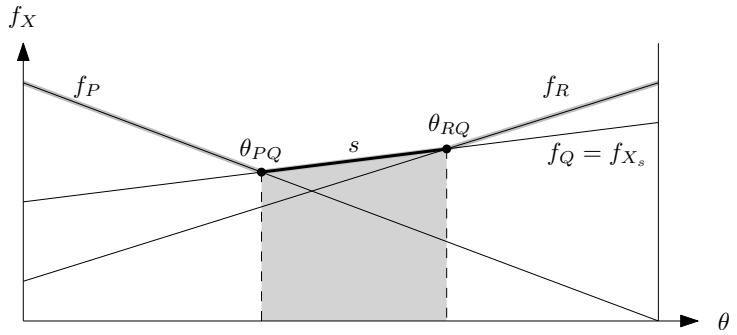


Figure 11.2: Upper boundary of the set of functions $\mathcal{F}_\mathcal{S}$ (gray line), segment s on the upper boundary (bold black segment) and corresponding linear function f_{X_s} associated with $X_s \in \mathcal{S}$ and interval that is dominated by s (gray area).

search on the cost-efficiency space using Theorem 11.1 and is listed as Algorithm 11.1. In Lemma 11.2 we show that this algorithm finds an optimal solution after a polynomial number of steps, when the interval containing the optimal cost-efficiency is smaller than some value depending on the input numbers. In this case, we return the maximum cost-efficiency of the two solutions corresponding to the interval boundaries.

Lemma 11.2. *Let $G = (V, E)$ be a graph with edge weights $a_e \in \mathbb{Z}, b_e \in \mathbb{N}$ for $e \in E$ and let $\mathcal{S} \subseteq 2^E$ be a set of feasible solutions. If we can compute*

$$\mathrm{opt}(\theta) = \mathrm{argmax}_{X \in \mathcal{S}} \left\{ \sum_{e \in X} (a_e - \theta b_e) \right\}$$

in time $f(n)$, then Algorithm 11.1 computes a solution $X^* \in \mathcal{S}$ with maximum cost-efficiency in $O(f(n) \log(nM))$ time where M denotes the largest input number.

Proof. First, we show that, if an interval I containing the optimal cost-efficiency is smaller than some threshold value depending on the input numbers, then the optimal solution corresponds to a solution that can be associated with the densities at the boundaries of the interval. Note that every $X \in \mathcal{S}$ corresponds to a linear function $f_X : \theta \mapsto \sum_{e \in X} a_e - \theta \cdot \sum_{e \in X} b_e$. Essentially we are interested in examining the upper envelope of the set of functions $\mathcal{F}_\mathcal{S} := \{f_X \mid X \in \mathcal{S}\}$. This upper boundary is composed of linear segments, such that each segment s corresponds to some $X_s \in \mathcal{S}$ and to some interval I_s on the θ-axis. This interval contains all θ for which X_s maximizes $\sum_{e \in X}(a_e - \theta \cdot b_e)$ over all $X \in \mathcal{S}$. We say that X_s is *dominating* on I_s. See Figure 11.2 for an illustration.

Let $f_P, f_Q, f_R, f_S \in \mathcal{F}_\mathcal{S}$ and let θ_{PQ} and θ_{RS} be the intersections of f_P, f_Q and f_R, f_S, respectively, such that $\theta_{PQ} < \theta_{RS}$. Then the length of the interval $[\theta_{PQ}, \theta_{RS}]$ can be bounded from below as follows. Let $A_X := \sum_{e \in X} a_e$ and let $B_X := \sum_{e \in X} b_e$. Then we have

$$f_P(\theta_{PQ}) = A_P + \theta_{PQ} B_P = A_Q + \theta_{PQ} B_Q = f_Q(\theta_{PQ})$$
$$f_R(\theta_{RS}) = A_R + \theta_{RS} B_R = A_S + \theta_{RS} B_S = f_S(\theta_{RS})$$

which, by a simple transformation, is equivalent to

$$\theta_{PQ} = \frac{A_P - A_Q}{B_Q - B_P} \quad \text{and} \quad \theta_{RS} = \frac{A_R - A_S}{B_S - B_R}\,.$$

It follows that, if $|\theta_{PQ} - \theta_{RS}| > 0$, we have

$$\begin{aligned}|\theta_{PQ} - \theta_{RS}| &= \left|\frac{A_P - A_Q}{B_Q - B_P} - \frac{A_R - A_S}{B_S - B_R}\right| \\ &= \left|\frac{(A_P - A_Q)(B_S - B_R) - (A_R - A_S)(B_Q - B_P)}{(B_Q - B_P)(B_S - B_R)}\right| \\ &\geq \left|\frac{1}{(B_Q - B_P)(B_S - B_R)}\right|\end{aligned}$$

since the numerator must be an integer

$$\geq \frac{1}{\left(\sum_{e\in E} b_e\right)^2}$$

since both B_Q and B_S are bounded by $\sum_{e\in E} b_e$. Hence, whenever $|\theta_{RS} - \theta_{PQ}| > 0$, we have

$$|\theta_{RS} - \theta_{PQ}| > \left(\sum_{e\in E} b_e\right)^{-2}.$$

As a consequence, let $[\alpha, \beta]$ be an arbitrary non-degenerate interval of length less than this value containing the optimal cost-efficiency θ^*. Then this interval may contain at most one intersection point of all the pairs of linear functions $f_P, f_Q \in \mathcal{F}_S$, that is, it intersects at most two segments on the upper boundary of $\mathcal{F}_S$. Then, clearly, either the solution $\text{opt}(\alpha)$ corresponding to α or the solution $\text{opt}(\beta)$ corresponding to β must be optimal. Note, that the optimal cost-efficiency will, in general, match neither α nor β in this case.

Next, we prove the bound on the running time. It is clear that $\delta_{\min} \leq \theta^* \leq \delta_{\max}$, where $\delta_{\min}$ and $\delta_{\max}$ denote the minimum and maximum cost-efficiency of an edge of G, respectively. Hence we only need to search the optimal value in the interval $[\delta_{\min}, \delta_{\max}]$. The size of this interval is at most $|2a_{\max}|$, where $a_{\max}$ denotes the maximum weight of an edge. since we assumed $b_e \geq 1$ for all $e \in E$. Algorithm 11.1 performs binary parameter search on this interval using Theorem 11.1. In each step we bisect the previous interval, that is, after t steps the interval has size at most $2|a_{\max}|/2^t$. Thus, after $t > \log|a_{\max}| + 2 \cdot \log\sum_{e\in E} b_e + 1$ steps the size of the interval is smaller than $\left(\sum_{e\in E} b_e\right)^{-2}$. By previous arguments either the solution corresponding to the left boundary of the interval or the solution corresponding to the right boundary of the interval is an optimal solution. Hence, it suffices to compute two optimal solutions for α and β, respectively, and to compare their densities. The running time of the algorithm is in $O(f(n) \cdot \log(nM))$ where M is the largest absolute value of the input numbers. Hence, the running time is polynomial in the input size. □

The algorithm provides a generic tool that can be applied to various problems, whenever the corresponding single-objective optimization problem can be solved efficiently in the presence of both positive and negative numbers. For instance, since perfect weighted matchings can be computed in time $O((m + n\log n)n)$ [33] the lemma immediately implies the following.

Corollary 11.3. *A perfect maximum cost-efficiency matching can be computed in $O((m + n\log n)n\log(nM))$ time.*

Next, we show that a maximum cost-efficiency subtree with k leaves can be computed in a tree in polynomial time, again using the fact that we can solve the underlying single-objective optimization problem efficiently.

Theorem 11.4. *Given a tree $T = (V, E)$, a maximum cost-efficiency subtree with exactly k leaves can be computed in time $O(k^2 n\log(nM))$ where M denotes the largest input number.*

Proof. The proof exploits a combination of the parametric search technique and dynamic programming. By applying the parametric search we reduce the problem to finding—for various values of $\theta \in \mathcal{R}$—a longest subtree of T with k leaves, where the new length of each edge e is given by $a_e - \theta b_e$. For an edge $e = \{u, v\}$ we denote this new length by $\lambda(u, v) = a_e - \theta b_e$.

In order to do compute the longest subtree of T with k leaves, we root the tree in some vertex $r \in V$. For a vertex $v \in V$ we denote the number of children of v by $n(v)$ and we denote the children by $u_1^v \ldots, v_{n(v)}^v$.

Suppose we are given an optimal solution T^* for this problem. Let v^* be the topmost vertex in T^* with respect to the rooting. Then either v^* is a leaf in T^* and there are only $k-1$ leaves of T^* in the subtrees rooted in the children of v or v^* is an internal vertex with k leaves in the subtrees rooted in the children of v^*. Let $u \in V$ be a vertex of T and let T_u denote the subtree of T rooted in u. Let T'_u be a subtree of T_u containing u. Further, let T^*_u denote the subtree of T^* that is contained in T'_u and let k' denote the number of leaves of T^* contained in T'_u. Then, clearly T^*_u is the longest subtree in T'_u rooted in u with k' leaves. We use this observation to decompose the problem into smaller sub-problems.

Let $v \in V$, $1 \leq i \leq k-1$ and $1 \leq j \leq n(v)$. Then we denote by $\Lambda(v, i, j)$ the length of a longest tree $\widehat{T}$ with i leaves, where v does not count as a leaf, such that $\widehat{T}$ is contained in the tree induced by v and the subtrees rooted in its children $u_1^v, \ldots, u_j^v$. We can compute these values from the following equation

$$\Lambda(v, i, j+1) = \max \left\{ \begin{array}{l} \Lambda(v, i, j), \\ \Lambda(v, i-1, j) + \lambda(v, u_{j+1}), \\ \max_{1 \leq t \leq i} \{\Lambda(v, i-t, j) + \Lambda(u_{j+1}, t, n(u_{j+1})\} + \lambda(v, u_{j+1}) \end{array} \right\}$$

in a bottom-up manner on the tree, using $\Lambda(v, 0, 0) = 0$. This can be done in $O(k^2 n)$ time.

Furthermore, we denote by $\overline{\Lambda}(v, i, j)$ the length of a longest tree $\widehat{T}$ with i leaves, where v does not count as a leaf, such that $\widehat{T}$ is contained in the tree induced by v and exactly one subtree rooted in some child u_r^v of v, where $j \leq r \leq n(v)$. These values can be computed in $O(kn)$ time from the following equation using the values computed in the previous step

$$\overline{\Lambda}(v, i, j) = \max_{j \leq r \leq n(v)} \{\Lambda(u_r, i, n(u_r)) + \lambda(v, u_r)\} .$$

Clearly, a tree with k leaves can only be found in a subtree of T with at least $k-1$ leaves. For each vertex v such that the tree T_v rooted in v contains at least $k-1$ leaves, we compute the longest subtree with k leaves, denoted by $\Lambda^*(v)$, as follows

$$\Lambda^*(v) = \max \left\{ \begin{array}{l} \overline{\Lambda}(v, k-1, 1), \\ \max_{1 \leq t \leq k-1, 1 \leq r \leq n(v)} \{\Lambda(v, k-t, r) + \overline{\Lambda}(v, t, r+1)\} \end{array} \right\} .$$

As mentioned earlier, this equation reflects the fact that v is either a leaf itself or an internal vertex, in which case v must have at least two children in T_v. We achieve this by "guessing" an index r such that the computed tree contains at least on child in $u_1^v, \ldots, u_r^v$ and one child in $u_{r+1}^v, \ldots, u_{n(v)}^v$. Again, these values can be computed in $O(kn)$ time. Finally, we return the solution corresponding to the maximum value $\Lambda * (v)$ over all $v \in V$ with at least $k-1$ leaves in the subtree T_v. Using Lemma 11.2 the total time complexity is $O(k^2 n \log(nM))$. □

11.2 General Steiner Constraints

In this section we consider maximum cost-efficiency pattern problems under Steiner constraints. Given a graph $G = (V, E)$ and a set of *terminals* $S \subseteq V$ the maximum cost-efficiency Steiner

pattern problem asks for a maximum cost-efficiency subgraph H containing all vertices in S. First we show that this problem NP-hard and inapproximable unless $\mathbf{P} = \mathbf{NP}$, even if the pattern is a path and there is only one terminal.

Theorem 11.5. *The Maximum Cost-Efficiency Steiner Pattern problem is* $\mathbf{NP}$*-hard, even if all weights are positive, all numbers are chosen from two distinct values, there is only one terminal and the pattern is a path. Furthermore, unless* $\mathbf{P} = \mathbf{NP}$*, this problem can not be approximated within a constant factor in polynomial time under the same conditions.*

We prove this theorem by a two-step reduction from the Longest Path problem. Given a graph $G = (V, E)$ of the longest path problem is to compute a path with maximum length, where the length is given by the number of edges on this path. This problem is NP-hard [37] and cannot be approximated within a constant factor unless $\mathbf{P} = \mathbf{NP}$ due to Karger et al. [56]. First, we show that this problem remains NP-hard and inapproximable if we require that the path starts in a predefined vertex $r \in V$. We refer to this problem as the Rooted Longest Path problem and we refer to r as the root.

Lemma 11.6. *The Rooted Longest Path problem problem is* $\mathbf{NP}$*-hard and cannot be approximated within a constant factor unless* $\mathbf{P} = \mathbf{NP}$*.*

Proof. Let $\mathcal{A}$ be an algorithm that approximates the rooted longest path problem within a factor r. Then we immediately obtain an algorithm $\mathcal{A}'$ approximating the longest path by running $\mathcal{A}$ with root v once for each $v \in V$ and returning the maximum of these values. Clearly, $\mathcal{A}'$ approximates the longest path within a factor of r. The claim then follows from the results of Karger et al. [56]. □

Now, we prove Theorem 11.5.

Proof of Theorem 11.5. We make a reduction from the rooted longest path problem. Assume we are given an instance $I = (G, r)$ of the rooted longest path problem, where $G = (V, E)$ is a graph and $r \in V$ is the root. We construct a new instance $I' = (G', S)$ of the maximum cost-efficiency Steiner pattern problem as follows. We let $G' = (V', E')$ such that $V' = V \cup \{x\}$ for some new vertex $x \notin V$ and we set $E' = E \cup \{\{x, r\}\}$ and $S = \{x\}$. Further, let $M := n^2 + 1$. We set $w_e = 1$ and $\ell_e = M$ for $e = \{x, r\}$ and we set $w_e = M$ and $\ell_e = 1$ for all $e \in E$.

We claim that there is a path in G' rooted in x with cost-efficiency at least θ if and only if there is a path in G with length at least $\lceil \theta - 1 \rceil$ rooted in r. Note that a path in G' of length $i+1$ rooted in x has cost-efficiency $\theta_i = (M + iM)/(M + i)$ for $i \geq 0$. Since θ_i is monotonically increasing in i and we have $i < \theta_i < i + 1$ for $i \leq n$ and $M \geq n^2$ it follows that $\lceil \theta_i - 1 \rceil = i$ and, hence, the claim holds.

Suppose that $\mathcal{A}$ is an approximation algorithm that approximates the maximum cost-efficiency Steiner pattern problem for path patterns within a factor $r_{\mathcal{A}}$. We show that we can use $\mathcal{A}$ to approximate the rooted longest path problem within a factor $3r_{\mathcal{A}}$. If r is isolated in G, we return r as a longest path, which is an optimal solution in this case. Otherwise, let e be an arbitrary edge incident to r in G. For a given instance $I = (G, r)$ let $P_{\mathcal{A}}$ be the path computed by $\mathcal{A}$ for the instance $I' = (G', S)$ constructed from I as described above and let $\theta_{\mathcal{A}}$ be its cost-efficiency. If $\theta_{\mathcal{A}} \geq 2M/(M + 1)$ we return $P_{\mathcal{A}}$, otherwise we return the single edge e, which together with the edge $\{x, r\}$, forms a path of length two with cost-efficiency $2M/(M+1)$ in G'.

In the following, let OPT_I denote the longest path in G rooted in r and let $OPT_{I'}$ denote the maximum cost-efficiency Steiner path in G'. Further, let $APX_{I'}$ denote the cost-efficiency of the approximation as described above. Note that since $2M/(M + 1) > 3/2$ for $n > 1$ we

have $APX_{I'} > 3/2$. It follows that

$$\begin{aligned}\frac{OPT_I}{APX_{I'}} &= \frac{\lceil OPT_{I'} - 1\rceil}{\lceil APX_{I'} - 1\rceil} \\ &\leq \frac{OPT_{I'}}{APX_{I'} - 1} \\ &\leq \frac{1}{1 - 1/APX_{I'}} \cdot \frac{OPT_{I'}}{APX_{I'}} \\ &< \frac{1}{1 - 2/3} r_{\mathcal{A}} \\ &= 3 \cdot r_{\mathcal{A}} \,.\end{aligned}$$

The claim then follows from Lemma 11.6. □

Although the maximum cost-efficiency Steiner pattern problem is **NP**-hard and unlikely to be approximable if the pattern is a path, we may still be able to obtain fixed-parameter tractable algorithms. First, we show that it is unlikely that the general problem is FPT when parameterized by the number of vertices in the solution, when we have no constraint on the feasible patterns.

Theorem 11.7. *Maximum Cost-Efficiency Steiner Pattern problem is $W[1]$-hard when parameterized by the number of vertices of the Steiner subgraph, even if S contains only one vertex.*

Proof. We prove the theorem by reduction from the $W[1]$-hard problem k-Clique problem [24]. Given an instance of k-clique, that is, a graph $G = (V, E)$, we wish to decide if G has a clique of size at least k. We transform this into an instance of maximum cost-efficiency Steiner pattern as follows. We construct a graph G' by adding a new vertex x to G that is connected to all vertices in V. We set $w_{vw} = \ell_{vw} = 1$ for all $vw \in E$ and we set $w_{xv} = 0$ and $\ell_{xy} = 1$ for all $v \in V$. Further, we set $S = \{x\}$. Clearly, there is a clique of size k with $m = \binom{k}{2}$ edges in G if and only if G' has a subgraph H with $x \in H$ and cost-efficiency at least $m/(m+1)$. □

While the Maximum Cost-Efficiency Steiner Pattern problem problem is W[1]-hard on general graphs, it turns out to be FPT on planar graphs.

Theorem 11.8. *Maximum Cost-Efficiency Steiner Pattern problem is FPT on planar graphs when parameterized by the number of vertices of the subgraph.*

Proof. Let G be a planar graph and let $S \subseteq V$ be a non-empty set of terminals. Since we are looking for a connected subgraph and since $S \neq \emptyset$, it suffices to consider the subgraph G' consisting of the $(k-1)$-neighborhood of some vertex $s \in S$. This graph is $(k-1)$-outerplanar and has radius at most $k-1$. Hence, by a result of Robertson and Seymour [79], G' has treewidth bounded by $3k - 2$. Note, that a path is a simple, connected graph that does not contain a triangle as a minor. Hence, using a modification of the algorithm proposed in Theorem 10.6 in combination with Theorem 11.2, we can compute a subgraph of G' with maximum cost-efficiency in FPT time.

□

In contrast to the W[1]-hardness of the maximum cost-efficiency Steiner pattern problem without constraints on the pattern, we show that the problem is FPT when parameterized by the number of vertices if the pattern is a path. In order to obtain this result we use the parametric search technique introduced in Section 11.1 in combination with Color Coding [8]. Color Coding was introduced by Alon et al. [8] as a method for finding subgraphs of bounded size in arbitrary graphs in FPT time. It can be used to find paths and cycles in expected time $2^{O(k)}m$. The algorithms are randomized by construction, but can be derandomized. We

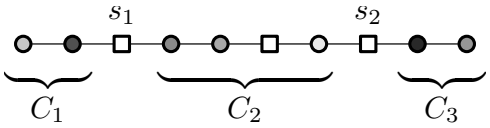


Figure 11.3: Decomposition of an optimal colorful Steiner path. Terminals are depicted as boxes.

can directly apply Color Coding to find a maximum cost-efficiency Steiner path with at most k vertices with high probability (WHP) in time $2^{O(k)}m\log(nM)$. We can slightly improve on the standard algorithm by coloring the terminals in a deterministic way, thus, using only $k-s$ random colors and thereby improving the probability of obtaining an optimal solution. In the following we further improve on this result when the number of Steiner vertices is large compared to k.

Theorem 11.9. *Given a set S of s terminals a maximum cost-efficiency Steiner path with k vertices can be computed WHP in time $O((2^{k-s}m + 3^{k-s})s^2\log(nM))$.*

Proof. Since we apply the parametric search technique, we need to find, for various values of θ, a longest path with respect to the edge-lengths $\lambda(e) := a_e - \theta b_e$ for all $e \in E$. Suppose we are given a random coloring of the vertices in $V \setminus S$ with $k-s$ colors using some color set C and we wish to find a longest *colorful* Steiner path for S. A Steiner path is called colorful, if it contains each color exactly once. Note, that this implies that the path is simple, that is, it does not contain multiple copies of any vertex. Our approach is based on a decomposition of any optimal colorful Steiner path P for S into three optimal subpaths P_1, P_2 and P_3 as illustrated in Figure 11.3 such that the following holds:

1. P_2 starts and ends at terminals s_1 and s_2, respectively, and contains all terminals
2. P_1 and P_2 end at terminals s_1 and s_2, respectively, and do not contain any other terminals.

In order to compute P_2, we further decompose this path into fragments each starting and ending with a terminal and containing no other terminals apart from the end-vertices. Let $c(v)$ denote the color of vertex v. For each vertex $v \in V \setminus S$, $x \in S$ and each set of colors $C' \subseteq C$ with $c(v) \in C'$ we compute the maximum length $L(x, v, C')$ of any colorful path from x to v using all colors in C'. Similarly, we compute for each $z \in V$ the length $\widehat{L}(z, C')$ of the longest path ending in z using all colors in C'. We compute these values by dynamic programming in $O(2^{k-s} \cdot s \cdot m)$ time using the equations

$$L(x, v, C') = \max_{w \in N(v) \setminus S} \{L(x, w, C' \setminus \{c(v)\}) + \lambda(v, w)\}$$
$$\widehat{L}(z, C') = \max_{w \in N(z) \setminus S} \{\widehat{L}(w, C' \setminus \{c(v)\}) + \lambda(w, z)\} .$$

For each $x, y \in S$ and each set of colors $C' \subseteq C$ we subsequently compute the maximum length $L(x, y, C')$ of any path from x to y using all colors in C' as well as for all $z \in V$ the maximum length $\widehat{L}(z, C')$ of any path ending in z using all colors in C' by

$$L(x, y, C') = \max_{w \in N(y) \setminus S} \{L(x, w, C') + \lambda(w, y)\} .$$

in time $O(2^{k-s} \cdot s^2 \cdot m)$. Implicitly, we obtain a combinatorial description of the longest paths starting and ending at terminals as a complete multigraph $G_S = (S, M)$ on the set of terminals in which each edge is annotated with its length and the set of colors used to attain this length. The weighted multiset of edges in our multigraph $G_S = (S, M)$ is defined by

$$M = \left\{ \left(xy, C', L(x, y, C')\right) \mid x, y \in S, C' \subseteq C \right\} .$$

Then we compute for each vertex $x \in S$ and each subset $X \subseteq S$ with $x \in X$ and each set of colors $C' \subseteq C$ the length $\widetilde{L}(x, X, C')$ of the longest path in G_S ending in x using all vertices in X and all colors in C' using the equation

$$\widetilde{L}(x, X, C') = \max_{y \in S, C'' \subseteq C'} \{\widetilde{L}(y, X \setminus \{x\}, C' \setminus C'') + L(x, y, C'')\}$$

where $L(x, \{x\}, C') = \widehat{L}(x, C')$. In order to do this efficiently, we consider all partitions of C into three sets $C \setminus C'$, C'' and $C' \setminus C''$. The number of these partitions is at most 3^{k-s}. Hence, the computation can be performed in time $O(3^{k-s})s^2$. The optimal solution can then be computed as

$$\max_{x \in S, C' \subseteq C} \{\widetilde{L}(x, S, C') + \widehat{L}(x, C \setminus C')\} \tag{11.1}$$

in time $O(2^{k-s}s)$. Hence, the dynamic programming for fixed θ takes $O(2^{k-s}s^2m + 3^{k-s}s^2)$ time and the problem can be solved in time $O((2^{k-s}m + 3^{k-s})s^2 \log(nM))$.

The probability that a path of length at most k interconnecting a set of s vertices is colorful is given by $p(k, s) = (k-s)!/(k-s)^{(k-s)} > \sqrt{2\pi(k-s)}e^{-(k-s)}$. Hence, a colorful path can be found with probability $\leq \varepsilon$ if the number of trials is at least $t_\varepsilon(k, s) = \frac{\ln \varepsilon}{\ln(1-p(k,s))} = |\ln \varepsilon| \cdot O(e^{k-s})$. □

We conclude this section by showing that we cannot hope to extend this result to more general patterns.

Theorem 11.10. *It is* **NP***-hard to decide whether there is a Steiner tree with at most k vertices and cost-efficiency at least θ, even if we allow only one terminal.*

Proof. We show **NP**-hardness by reduction from the NP-hard k-MST problem. Given an edge-weighted graph G, a non-negative integer k and a weight W, the k-MST problem is to decide whether there is a tree spanning at least k vertices with weight at least W. This problem has been shown to be NP-hard by Ravi et al. [76] and it obviously remains NP-hard if we require the solution to contain exactly k vertices.

We observe that deciding whether there is a tree with k vertices and cost-efficiency at least θ is equivalent to deciding whether there is a tree with k vertices and length *at most* 0 where the length of each edge is given by $\theta b_e - a_e$. To see this, note that

$$\frac{\sum_{e \in E'} a_e}{\sum_{e \in E'} b_e} \geq \theta \Leftrightarrow \sum_{e \in E'} (\theta b_e - a_e) \leq 0$$

by simple equivalent transformations.

Assume we are given an instance of k-MST, that is, a graph $G = (V, E)$ and we wish to decide if there is a tree with k vertices and length at most θ where the edge-lengths are integral. We transform this into a set of $|V|$ k-Maximum Cost-Efficiency Steiner Pattern problem G_v such that G is solvable if and only if at least one of the new instances G_v is solvable. We choose $G_v = (V \cup \{x\}, E \cup \{xv\})$ for some new vertex x. Further, we choose the $b_e \equiv 1$ and $a_e := \ell_e - \theta$ and $a_{xv} = 2\theta$ and $S = \{x\}$. Hence, $\theta b_e - a_e = \ell_e$, $\theta b_{xv} - a_{xv} = -\theta$ and each solution contains the edge sx.

By the above observations there is a tree with k vertices and length at most θ if and only if there is a tree with $k+1$ vertices including x with length at most 0, where edge lengths are defined by $\theta b_e - a_e$. The latter is equivalent to deciding whether there is a tree with k vertices and cost-efficiency at least θ. □

Part IV

Conclusion

We consider in this dissertation the intricate resource allocation problem from an algorithmic perspective. Starting from resource allocations with locality constraints, in which the resource assignments are valid only between adjacent objects, to global resource planning for which the resources are packed and delivered via intermediate stops to demanding targets, we present algorithmic results with solid theoretical guarantees as well as hardness proofs to jointly compose a comprehensive study for the entire problem complexity.

In terms of local demand supplying, we consider a generalization of *dominating set problem* under the concept of *capacitataion* with *soft capacity*, i.e., *capacitated domination problem* with *soft capacity*. On one hand, approximation algorithms matching the classical results for dominating set which are asymptotically optimal are presented for general graphs. On the other hand, a series of hardness results for trees, graphs of bounded treewidths, and planar graphs shows that the considered problem is fundamentally more difficult than the dominating set problem, whereas the corresponding approximation algorithms are also provided.

Open Problems and Future Research Topics

Bibliography

[1] I. Abraham, Y. Bartal, T-H. Chan, K. Dhamdhere, A. Gupta, J. Kleinberg, O. Neiman, and A. Slivkins. Metric embeddings with relaxed guarantees. In *FOCS'05*, pages 83–100, Washington, DC, USA, 2005.

[2] I. Abraham, Y. Bartal, and O. Neiman. Embedding metrics into ultrametrics and graphs into spanning trees with constant average distortion. In *SODA'07*, pages 502–511, Philadelphia, PA, USA, 2007.

[3] Ittai Abraham, Yair Bartal, and Ofer Neiman. On embedding of finite metric spaces into hilbert space. manuscript, 2005.

[4] Ittai Abraham, Yair Bartal, and Ofer Neiman. Advances in metric embedding theory. In *STOC '06*, pages 271–286, New York, NY, USA, 2006. ACM.

[5] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002.

[6] Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, 2004.

[7] N. Alon, R. Karp, D.d Peleg, and D. West. A graph-theoretic game and its application to the k-server problem. *SIAM J. Comput.*, 24:78–100, February 1995.

[8] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[9] Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

[10] Vojtech Bálint. The non-approximability of bicriteria network design problems. *J. of Discrete Algorithms*, 1:339–355, June 2003.

[11] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *FOCS'96*, pages 184–, Washington, DC, USA, 1996.

[12] Yair Bartal. On approximating arbitrary metrices by tree metrics. In *STOC'98*, pages 161–168, New York, NY, USA, 1998. ACM.

[13] Yair Bartal. Graph decomposition lemmas and their role in metric embedding methods. In *ESA'04*, pages 89–97, 2004.

[14] Yair Bartal, Nathan Linial, Manor Mendel, and Assaf Naor. On metric ramsey-type phenomena. In *STOC'03*, pages 463–472, New York, NY, USA, 2003. ACM.

[15] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *STOC '93: Proceedings of the 25th annual ACM symposium on Theory of computing*, pages 226–234, New York, NY, USA, 1993. ACM.

[16] Prosenjit Bose. On embedding an outer-planar graph in a point set. *CGTA: Computational Geometry: Theory and Applications*, 23:2002, 1997.

[17] R. Chandrasekaran. Minimal ratio spanning trees. *Networks*, 7(4):335–342, 1977.

[18] M. Charikar, C. Chekuri, A. Goel, S. Guha, and S. Plotkin. Approximating a finite metric by a small number of tree metrics. In *FOCS'98*, pages 379–, 1998.

[19] Altannar Chinchuluun and Panos Pardalos. A survey of recent developments in multiobjective optimization. *Annals of Operations Research*, 154:29–50, 2007.

[20] Fabián A. Chudak and David P. Williamson. Improved approximation algorithms for capacitated facility location problems. *Math. Program.*, 102:207–222, March 2005.

[21] Kai-min Chung and Hsueh-I Lu. An optimal algorithm for the maximum-density segment problem. *SIAM J. Comput.*, 34(2):373–387, 2005.

[22] Julia Chuzhoy. Covering problems with hard capacities. *SIAM J. Comput.*, 36(2):498–515, 2006.

[23] Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and h-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

[24] Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141(1-2):109 – 131, 1995.

[25] Rod G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer, 1999.

[26] S. Dreyfus and R. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

[27] M. Elkin, Y. Emek, D. Spielman, and S.-H. Teng. Lower-stretch spanning trees. In *STOC'05*, pages 494–503, New York, NY, USA, 2005. ACM.

[28] David Eppstein. Subgraph isomorphism in planar graphs and related problems. In *Proc. 6th Ann. ACM-SIAM Sympos. Disc. Alg.*, pages 632–640. SIAM, 1995.

[29] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC'03*, pages 448–455, New York, NY, 2003.

[30] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[31] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[32] Fedor V. Fomin and Dimitrios M. Thilikos. Dominating sets in planar graphs: Branch-width and exponential speed-up. *SIAM J. Comput.*, 36(2):281–309, 2006.

[33] Harold N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, SODA '90, pages 434–443, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.

[34] Rajiv Gandhi, Eran Halperin, Samir Khuller, Guy Kortsarz, and Aravind Srinivasan. An improved approximation algorithm for vertex cover with hard capacities. *J. Comput. Syst. Sci.*, 72:16–33, February 2006.

[35] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding in bipartite graphs. In *FOCS 2002*, pages 323–332, Washington, DC, USA, 2002. IEEE Computer Society.

[36] M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman & Co., New York, 1979.

[37] Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[38] Michael H. Goldwasser, Ming-Yang Kao, and Hsueh-I Lu. Linear-time algorithms for computing maximum-density sequence segments with bioinformatics applications. *J. Comput. Syst. Sci.*, 70(2):128–144, 2005.

[39] Sudipto Guha, Refael Hassin, Samir Khuller, and Einat Or. Capacitated vertex covering. *J. Algorithms*, 48(1):257–270, 2003.

[40] Jiong Guo and Rolf Niedermeier. Linear problem kernels for np-hard problems on planar graphs. In *ICALP*, pages 375–386, 2007.

[41] Teresa W. Haynes, Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Michael A. Henning. Domination in graphs applied to electric power networks. *SIAM J. Discret. Math.*, 15(4):519–529, 2002.

[42] Teresa W. Haynes, Stephen Hedetniemi, and Peter Slater. *Fundamentals of Domination in Graphs (Pure and Applied Mathematics).* Marcel Dekker, 1998.

[43] J.-M. Ho. *Optimal trees in network design.* PhD thesis, Evanston, IL, USA, 1989. UMI Order No: GAX90-01807.

[44] Jan-Ming Ho, D. T. Lee, Chia-Hsiang Chang, and C. K. Wong. Minimum diameter spanning trees and related problems. *SIAM J. Comput.*, 20:987–997, October 1991.

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

[46] John Hopcroft and Robert Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

[47] Sun-Yuan Hsieh and Chih-Sheng Cheng. Finding a maximum-density path in a tree under the weight and length constraints. *Information Processing Letters*, 105(5):202 – 205, 2008.

[48] Sun-Yuan Hsieh and Ting-Yu Chou. *Algorithms and Computation*, volume 3827 of *LNCS*, chapter Finding a Weight-Constrained Maximum-Density Subtree in a Tree, pages 944–953. Springer Berlin / Heidelberg, 2005.

[49] Ross B Inman. A denaturation map of the lambda phage dna molecule determined by electron microscopy. *Journal of Molecular Biology*, 18(3):464–476, 1966.

[50] Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, March 2001.

[51] D. S. Johnson, J. K. Lenstra, and A. H. G. Rinnooy Kan. The complexity of the network design problem. *Networks*, 8:279–285, 1978.

[52] David S. Johnson. Approximation algorithms for combinatorial problems. In *the 5th annual ACM symposium on Theory of computing*, pages 38–49, New York, NY, USA, 1973. ACM.

[53] Mong-Jen Kao and Han-Lin Chen. Approximation algorithms for the capacitated domination problem. In *FAW 2010*, pages 185–196, Berlin, Heidelberg, 2010. Springer-Verlag.

[54] Mong-Jen Kao and D. T. Lee. Capacitated domination: Constant factor approximations for planar graphs. In *ISAAC*, pages 494–503, 2011.

[55] Mong-Jen Kao, Chung-Shou Liao, and D. T. Lee. Capacitated domination problem. *Algorithmica*, 60:274–300, 2011. 10.1007/s00453-009-9336-x.

[56] D. Karger, R. Motwani, and G. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18:82–98, 1997. 10.1007/BF02523689.

[57] Jon Kleinberg, Aleksandrs Slivkins, and Tom Wexler. Triangulation and embedding using small sets of beacons. *J. ACM*, 56:32:1–32:37, September 2009.

[58] T. Kloks. *Treewidth, Computations and Approximations.* LNCS. Springer, 1994.

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

[60] Hoong Chuin Lau, Trung Hieu Ngo, and Bao Nguyen Nguyen. Finding a length-constrained maximum-sum or maximum-density subtree and its application to logistics. *Discrete Optimization*, 3(4):385 – 391, 2006.

[61] D. T. Lee, Tien-Ching Lin, and Hsueh-I Lu. Fast algorithms for the density finding problem. *Algorithmica*, 53(3):298–313, 2009.

[62] Chung-Shou Liao and Gerard J. Chang. k-tuple domination in graphs. *Inf. Process. Lett.*, 87(1):45–50, July 2003.

[63] Chung-Shou Liao and Der-Tsai Lee. Power domination problem in graphs. In *COCOON*, pages 818–828, 2005.

[64] Yaw-Ling Lin, Tao Jiang, and Kun-Mao Chao. Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. *J. Comput. Syst. Sci.*, 65(3):570–586, 2002.

[65] Hsiao-Fei Liu and Kun-Mao Chao. Algorithms for finding the weight-constrained k longest paths in a tree and the length-constrained k maximum-sum segments of a sequence. *Theor. Comput. Sci.*, 407(1-3):349–358, 2008.

[66] Daniel Lokshtanov. *New Methods in Parameterized Algorithms and Complexity.* PhD thesis, University of Bergen Norway, 2009.

[67] G. Macaya, J.-P. Thiery, and G. Bernardi. An approach to the organization of eukaryotic genomes at a macromolecular level. *Journal of Molecular Biology*, 108(1):237 – 254, 1976.

[68] Madhav V. Marathe, R. Ravi, Ravi Sundaram, S. S. Ravi, Daniel J. Rosenkrantz, and Harry B. Hunt. Bicriteria network design problems,. *Journal of Algorithms*, 28(1):142 – 171, 1998.

[69] Edward M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, 1985.

[70] Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks.* Cambridge University Press, New York, NY, USA, 2007.

[71] Rolf Niedermeier. *Invitation to Fixed Parameter Algorithms.* Oxford University Press, 2006.

[72] Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23(2):166 – 204, 1981.

[73] M. Pál, É. Tardos, and T. Wexler. Facility location with nonuniform hard capacities. In *FOCS 2001*, pages 329–, Washington, DC, USA, 2001. IEEE Computer Society.

[74] Yuri Rabinovich. On average distortion of embedding metrics into the line. In *STOC'03*, pages 456–462, 2003.

[75] Yuri Rabinovich and Ran Raz. Lower bounds on the distortion of embedding finite metric spaces in graphs. *Discrete & Computational Geometry*, 19, 1996.

[76] R. Ravi, R. Sundaram, M. V. Marathe, D. J. Rosenkrantz, and S. S. Ravi. Spanning trees—short or small. *SIAM J. Discret. Math.*, 9:178–200, May 1996.

[77] Fred S. Roberts. *Graph Theory and Its Applications to Problems of Society*. 1978.

[78] N. Robertson and P. D. Seymour. Graph minors .XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65 – 110, 1995.

[79] Neil Robertson and P. D. Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49 – 64, 1984.

[80] Gabriel Robins and Alexander Zelikovsky. Improved steiner tree approximation in graphs. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '00, pages 770–779, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[81] P. Schuurman and G. Woeginger. Approximation schemes – a tutorial, 2011. preliminary version of a chapter in the book Lectures on Scheduling, to appear in 2011.

[82] Dae Young Seo. *On the complexity of bicriteria spanning tree problems for a set of points in the plane*. PhD thesis, Evanston, IL, USA, 1999. AAI9953371.

[83] Asaf Shapira, Raphael Yuster, and Uri Zwick. All-pairs bottleneck paths in vertex weighted graphs. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 978–985, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[84] David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems (extended abstract). In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 265–274, New York, NY, USA, 1997. ACM.

[85] Michiel Smid. Spanning trees with o(1) average stretch factor. manuscript, 2009.

[86] Virginia Vassilevska, Ryan Williams, and Raphael Yuster. All-pairs bottleneck paths for general graphs in truly sub-cubic time. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, STOC '07, pages 585–589, New York, NY, USA, 2007. ACM.

[87] P.-J. Wan, K. M. Alzoubi, and O. Frieder. A simple heuristic for minimum connected dominating set in graphs. *International Journal of Foundations of Computer Science*, 14(2):323–333, 2003.

[88] Richard Wong. Worst-case analysis of network design problem heuristics. *SIAM. J. Alg. Disc. Meth.*, 1, 1980.

[89] B.-Y. Wu. Approximation algorithms for the optimal p-source communication spanning tree. *Discrete Appl. Math.*, 143:31–42, September 2004.

[90] B.-Y. Wu, K.-M. Chao, and C.-Y. Tang. Light graphs with small routing cost. *Networks*, 39:2002.

[91] B.-Y. Wu, K.-M. Chao, and C.-Y. Tang. A polynomial time approximation scheme for optimal product-requirement communication spanning trees. *J. Algorithms*, 36:182–204, August 2000.

[92] B.-Y. Wu, G. Lancia, V. Bafna, K.-M. Chao, R. Ravi, and C.-Y. Tang. A polynomial-time approximation scheme for minimum routing cost spanning trees. *SIAM J. Comput.*, 29:761–778, December 1999.

[93] Bang Ye Wu. An optimal algorithm for the maximum-density path in a tree. *Inf. Process. Lett.*, 109(17):975–979, 2009.

[94] Bang Ye Wu, Kun-Mao Chao, and Chuan Yi Tang. An efficient algorithm for the length-constrained heaviest path problem on a tree. *Inf. Process. Lett.*, 69(2):63–67, 1999.

Index