ABSTRACT

Title of dissertation: **ALLOCATION ALGORITHMS FOR NETWORKS WITH SCARCE RESOURCES**

Kanthi K. Sarpatwar, Doctor of Philosophy, 2015

Dissertation directed by: Professor Samir Khuller
Department of Computer Science

Many fundamental algorithmic techniques have roots in applications to computer networks. We consider several problems that crop up in wireless ad hoc networks, sensor networks, P2P networks, and cluster networks. The common challenge here is to deal with certain *bottleneck resources* that are crucial for the performance of underlying system. Broadly, we deal with the following issues.

Data: The primary goal in *resource replication* problems is to replicate data objects on server nodes with limited storage capacities, so that the latency of client nodes needing these objects is minimized. Previous work in this area is heuristic and without guarantees. We develop tight (or nearly) approximation algorithms for several problems including *basic resource replication* - where clients need all objects and server can store at most one object, *subset resource replication*- where clients require different subsets of objects and servers have limited non-uniform capacity, and related variants.

Computational resources: To facilitate packing of jobs needing disparate amounts

of computational resources in *cluster networks*, an important auxiliary problem to solve is that of *container selection*. The idea is to select a limited number of "containers" that represent a given pool of jobs while minimizing "wastage" of resources. Subsequently, containers representing jobs can be packed instead of jobs themselves. We study this problem in two settings: *continuous*- where there are no additional restrictions on chosen containers, and *discrete* - where we must choose containers from a given set. We show that the continuous variant is NP-hard and admits a *polynomial time approximation scheme*. Contrastingly, the discrete variant is shown to be NP-hard to approximate. Therefore, we seek bi-approximation algorithms for this case.

Energy resources: *Wireless ad hoc* networks contain nodes with limited battery life and it is crucial to design energy efficient algorithms. We obtain tight approximation (up to constant factors) guarantees for partial and budgeted versions of the *connected dominating set* problem, which is regarded as a good model for a virtual backbone of a wireless ad hoc network. Further, we will discuss approximation algorithms for some problems involving target monitoring in sensor networks and message propagation in radio networks.

We will end with a discussion on future work.

ALLOCATION ALGORITHMS FOR NETWORKS WITH SCARCE RESOURCES

by

Kanthi Kiran Sarpatwar

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2015

Dedication

To *Anvika*, *Arvin* and *Ananya*

who shall not be held responsible for the contents of this dissertation.

Acknowledgments

It is indeed a blessing to receive an opportunity to thank people who have made personal sacrifices to advance your cause. I shall remain eternally indebted to Prof. Samir Khuller for his amazing support and guidance through out the course of my Ph.D. This phase of my life included several important decisions, including my marriage, and Samir's faith in me has never waivered. Another mentor who has guided me through the initial stages of my research career is Prof. N.S. Narayanaswamy - I cannot thank you enough for encouraging me to pursue doctoral studies. I would also like to thank Prof. Aravind Srinivasan, Prof. Mark Shayman, Prof. Mohammad Hajiaghayi, Prof. Peter Keleher, Prof. V. S. Subrahmanian, for graciously agreeing to be a part of my dissertation and proposal committees. I have immensely gained from their guidance and feedback during these final stages of my Ph.D. A special thanks to Prof. Jeff Foster for helping me prepare for my job interviews - your spirit of helping out students in the most innovative ways is truly inspiring. My warm regards to the entire staff at the Computer Science department for always being readily available to help. In particular, I would like to thank - Jennifer Story, Fatima Bangura, and Adelaide Findlay.

Over the past few years, I have had the privilege of working with some amazing teams. My deepest gratitude to the team at IBM T. J. Watson - Baruch Schieber, Viswanath Nagarajan, Joel Wolf, Kun-Lung Wu, Krzysztof Onak, and also Hadas Shachnai who visited IBM during that period. I would also like to thank my mentors at other internships - Guy Kortsarz at Rutgers, Randeep Bhatia at Bell Labs, Amit at Yahoo. A good fraction of this thesis is joint work with my colleague and dear friend Manish Purohit. The countless number of hours we spent together, coming up with algorithms and proofs, are some of the most productive periods in past few years. My warm gratitude to a number of labmates, friends, roommates and colleagues - Tom, Ioana, Brian, Jayanta, Rajesh, Vahid, Reza, Ejaz, Aishwarya, Sravanthi, Abdul, Kaushik, Mili, Jagadeesh, Bhuwan, Chanakya, Vijay, Malli, Philip, Pan, Bhaskar, Amit, Karthik(s), Anshul, Bhargav, Jessica, Barna, Koyel, Saeed, Greg, Udayan, Eunhui, and perhaps many others I might have overlooked.

No part of this thesis would have been possible without the love and support of my family. One person has suffered through it as much as I have - my dear wife, Niharika. Nothing but true love would lead someone through this tough path. You have earned this as much as I have, if not more. The hardwork and dedication of my dear parents has always been a true inspiration for me. All the sacrifices, both known and unknown to me, you have made shall remain in the depths of my heart. My brother, Karthik, has been a true mentor through all the important phases of my life. He guided me through the application process, and encouraged me through various "mid-career crises" that are inevitable in any Ph.D student's life. My sister-in-law, Sindhu, is one of the sweetest people I have ever known. I will always cherish our delightful chats which have always been a ready break from the tedium of the doctoral process.

Finally, I would like to gratefully acknowledge that a part of my funding came from NSFs grant CCF-1217890 and CCF-0728839.

Table of Contents

List of Figures vii

1 **Introduction** 1

1.1 **Data replication in P2P networks** 4

1.2 **Container selection in cluster networks** 6

1.3 **Virtual backbone in wireless ad hoc networks** 9

1.4 **Coverage in sensor and radio networks** 14

1.5 **Oracles in massive networks** 18

1.6 **Contributions** 21

2 **Resource Replication Problems** 26

2.1 **Road map to the chapter** 26

2.2 **Satisfying everyone: Resource replication** 28

2.2.1 **Basic resource replication problem** 28

2.2.2 **Subset resource replication problem** 36

2.2.3 **Hardness results for resource replication problems** . . . 42

2.3 **Dealing with outliers: Robust resource replication** 44

2.3.1 **Robust basic resource replication** 45

2.3.2 **Budgeted robust basic resource replication** 47

2.3.3 **Robust subset resource replication** 51

2.4 **Serving fairly: Capacitated basic resource replication** 55

3 **The Container Selection Problem** 63

3.1 **Road map to the chapter** 63

3.2 **The continuous container selection problem** 65

3.2.1 **Two dimensions** 66

3.2.2 **Higher dimensions** 69

3.2.3 **Dynamic program for the restricted problem** 72

3.3 **The discrete container selection problem** 75

3.3.1 **Two dimensions** 77

3.3.2 **Higher dimensions** 85

3.4 **Hardness results** 89

4 **Partial and Budgeted Connected Dominating Set.** 94
4.1 **Roadmap to the chapter** . . . 94
4.2 **Partial connected dominating set** . . . 97
4.2.1 **Algorithm** . . . 97
4.2.2 **Analysis** . . . 99
4.3 **Budgeted connected dominating set** . . . 103
4.3.1 **Algorithm** . . . 104
4.3.2 **Analysis** . . . 105
4.3.2.1 **Obtaining a true approximation** . . . 108
4.3.2.2 **Finding the best subtree** . . . 110
4.4 **Budgeted generalized CDS** . . . 111
4.4.1 **Algorithm.** . . . 112
4.4.2 **Analysis.** . . . 113
4.5 **Partial generalized connected domination** . . . 116
4.5.1 **Analysis** . . . 117

5 **Covering Problems in Sensor and Radio Networks** 119
5.1 **Roadmap to the chapter** . . . 119
5.2 **The max-min k-cover problem** . . . 120
5.3 **The minimum conflict-free coloring problem** . . . 124
5.4 **Two relaxations of the MCFC problem** . . . 128
5.4.1 **The R1-MCFC problem** . . . 129
5.4.2 **The R2-MCFC problem** . . . 133

6 **Steiner Tree and Cheapest Tour in Oracle Model** 136
6.1 **Road map to the chapter** . . . 136
6.2 **A brief introduction to distance oracles** . . . 137
6.3 **Steiner tree and cheapest tour oracles** . . . 140
6.3.1 **Preprocessing algorithm** . . . 142
6.3.2 **Query algorithms** . . . 143

7 **Future Work** 153

Bibliography 156

List of Figures

1.1 The framework of resource replication. 5

1.2 Container selection problem. 8

1.3 Illustration of CDS, PCDS and BCDS. 11

1.4 The *max-min k-cover* and *minimum conflict-free coloring* problems. . 16

2.1 An illustrative example for the BRR algorithm. 31

2.2 Illustrative example for the SRR algorithm. 39

2.3 A case for the outlier version. 45

2.4 Reduction of maximum k-clique instance to an RSRR instance: (a) given instance of the maximum k-clique problem (for this example, $k = 3$); (b) a RSRR instance: the vertex set comprises of 3 parts: V_1 with $k = 3$ vertices, V_2 with vertices corresponding to the edges of G; each $v \in V_2$ has a m^2 clique G_v associated with it as shown; these gadgets form the set V_3; each edge represents distance 1 and the remaining distances are computed using shortest path metric. . . 52

3.1 The continuous container selection problem. 67

3.2 Description of the cells. 77

3.3 LP relaxation. 85

4.1 A bad example for the c-step look-ahead greedy algorithm 96

4.2 Pictorial representation of different layers. (a) L_1 is an optimal solution (b) L_2 is set of the vertices adjacent to L_1 (c) L_3 is the subsequent layer (d) R is the set of all vertices other than $L_1 \cup L_2$ and (e) $L'_i = L_i \cap D$. 100

4.3 Decomposition of trees into 13 subtrees 108

4.4 Decomposition of trees into 13 subtrees 109

Chapter 1

Introduction

Computer networks play a pivotal role in various communication aspects of our lives as is evident from their wide applicability. Perhaps less obvious is the fact that optimization problems arising from various applications in such networks have led to much of the development in the field of algorithms over several decades. Most of the problems in this domain are NP-hard [1] and therefore it is considered highly unlikely that we can design efficient optimal algorithms for these problems. Consequently, we allow algorithms to come up with sub-optimal solutions that are not much worse compared to optimal solutions. Formally, an approximation algorithm (for a minimization problem) can be defined as follows.

Definition 1.0.1 (α-approximation algorithm) *An α-approximation algorithm for a minimization problems runs in polynomial time and yields a solution whose cost is within α times the optimal cost.*

Our main focus in this thesis is to design efficient approximation algorithms for problems arising in wireless ad hoc networks, P2P networks, sensor networks, and cluster networks. The common goal in all these problems is to handle *bottleneck resources*, i.e., resources available in limited quantities. Any reasonable algorithm must use such resources effectively. Depending on the type of bottleneck resource, we broadly classify our problems into following groups.

Data. Problems involving data replication are of fundamental importance in P2P networks (such as content distribution networks). Video content providers (e.g. Netflix) need to distribute content types on their servers such that client latency is minimized. As the amount of content keeps growing, where as the infrastructure essentially remains static, the storage capacity of nodes is the primary bottleneck on the performance of such systems. In this part, we study several variants of resource replication problems that were originally proposed by Ko and Rubenstein [2, 3].

Computation. The need to consolidate computational resources in cluster network frameworks has resulted in the development of several cross platform schedulers like *dominant resource fairness* (DRF) [4] and X-Flex [5]. Data sets and procedures, abstracted as "jobs", emerging from disparate sources, referred to as "platforms", must be allocated their *fair* share of the available computational resources. To simplify this task, X-Flex solves an important auxilliary problem called the container selection problem. The key goal in the container selection problem is to determine a specified (small) number of "container jobs" that can represent the given set of jobs while minimizing the "wastage" of resources. The main idea here is to pack fewer different kinds of container jobs instead of jobs themselves.

Energy. Wireless ad hoc networks are typically characterized by nodes with limited battery life. It is therefore essential to design systems that use the available energy effectively. In the third part, we tackle some interesting covering problems that arise in wireless ad hoc networks.

Routing is one of the most important challenges in wireless ad hoc networks, primarily due to the non-existence of a concrete communication backbone. Ephremides et. al. [6] introduced the concept of a *virtual backbone* that simulates the function of a network backbone. Bharghavan and Das [7] suggest using the well studied problem of *connected dominating set* [8–10] as a good model for a virtual backbone. We consider partial and budgeted versions of the connected dominating set problem as a more robust model for the virtual backbone.

Target monitoring is another important issue in wireless ad hoc networks such as sensor networks (imagine a swarm of robots trying to sense an unknown environment). The goal here is to monitor the targets as long as possible while having access to limited energy resources. The idea is to partition sensor nodes into as many disjoint groups as possible such that each group can monitor all or most of the targets. Another interesting problem that we consider has applications in message propagation in radio networks. Along with limited energy resources, these networks are characterized by interference issues. The objective is to minimize the number of *rounds* in which we can propogate a message to all or most the *receiver nodes*, while making sure that each transmitter is used as few times as possible (typically once).

Finally, we study the classic network design problems of *Steiner tree* and *cheapest tour* in the oracle model. Thorup and Zwick [11] introduced the basic oracle framework in a seminal work on so called *distance oracles*. The framework involves two algorithms: *preprocessing* - that works on an input graph to generate a data structure, *query processing* - that uses the data structure to answer queries quickly.

1.1 Data replication in P2P networks

In Chapter 2, we consider several problems related to data placement and replication. Such problems are of fundamental interest both in the areas of large scale distributed networking systems as well as centralized storage systems. The performance of distributed systems such as P2P file sharing systems, wireless ad hoc networks, sensor networks, where resources are shared among clients, can be significantly impacted by placement of the replicated resources [2, 3, 12]. On the other hand, centralized storage systems, such as Netflix, might have data distributed across different data centers to keep data closer to the demand to prevent over loading the network. Demand patterns for data can also vary widely, especially in the context of video on demand distribution.

A lot of research exists on centralized storage systems [13] that addresses the problem of data layout when all the storage units are centrally located in a single location and thus the "distance" of each client from any storage unit is the same. However, this assumption is not valid in modern storage management systems. Internet content providers rent storage space all over the world from different data centers in different locations. As said earlier, most interesting objective functions are NP-hard and it is of interest to consider efficient approximation algorithms.

The general framework is the following: given a collection of data items, with clients needing a subset of these items and servers having a limited capacity to hold data items. Servers and clients are modeled by a graph embedded in a metric space and we wish to distribute data items on to the servers. The goal is to minimize the

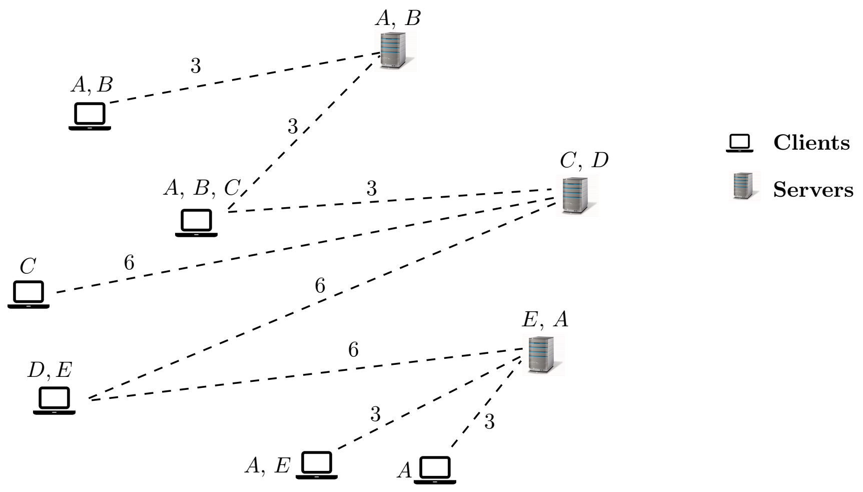


Figure 1.1: The framework of resource replication.

maximum distance any node has to travel to access all its required data items.

Formally, in the most general setting we are given: a set of vertices V, a metric $d : V \times V \rightarrow \mathbb{R}^+ \cup \{0\}$, and a set of resources or colors $\mathcal{C}$. Every vertex $v \in V$ has a subset $\mathcal{C}_v \subseteq \mathcal{C}$ of "required" colors and a non-negative integer s_v as the storage capacity, i.e., we can assign s_v colors to vertex v. A valid assignment of colors to vertices allocates a list of colors $\phi(v) \subseteq \mathcal{C}$ to each vertex v such that $|\phi(v)| \leq s_v$. The objective is find a valid assignment such that the following quantity is minimized: $\delta = \underset{\phi}{\mathsf{Min}} \underset{\substack{v \in V \\ r \in \mathcal{C}_v}}{\mathsf{Max}} d_r(v)$, where $d_r(v)$ is the shortest distance from v to some vertex u such that color $r \in \mathcal{C}_u$. We illustrate this by an example.

Example 1.1.1 *We refer to Figure 1.1 for an example of the resource replication framework. The set of data items is $\{A, B, C, D, E\}$, subsets of data items required by various clients are depicted next to them and each server has capacity to store two data items. The figure shows one way of distributing data items on servers, while respecting the capacity constraints. Clearly, for this distribution, the maximum* *distance a client has to travel to obtain its required data items is* 6 *units.*

We will formally define several variants in this general framework and study them in Chapter 2.

1.2 Container selection in cluster networks

In Chapter 3, we deal with the issue of limited computational resources such as memory, processing cores, bandwidth etc. These issues lead to an interesting special case of the non-metric k-median problem called the *container selection* problem. This is a geometric resource allocation problem that occurs naturally in any distributed computer environment. The goal here is to maximize resource utilization. This environment may, for example, consist of a private cloud [14], or it may consist of a collection of in-house, physical computers processors employing a cluster manager such as Mesos [15] or YARN [16].

We describe and motivate the container selection problem as follows. The input points correspond to *tasks*, each of which can be described in terms of multiple resource requirements. These dimensions typically include both CPU and memory, sometimes also network and I/O bandwidth. The tasks are then placed and executed in *virtual containers*, and of course each task must "fit" into its assigned container. For a variety of reasons, including ease of selection, maintenance and testing, it is important to create only a modest number k of container sizes. Amazon's EC2 cloud offering [17], for example, allows its customers to choose from $k = 13$ standard "instance types". The goal is to select k container sizes so that the aggregate resource usage (when each task is assigned its "smallest" dominating container) is minimized. We use the (normalized) sum of resources as the aggregate resource usage of a container. In these applications, the container sizes are usually determined in advance, before the actual tasks arrive: so suitably massaged historical task data is used as input. We refer the reader to work by Wolf et. al. [5] for more details.

Formally, an instance of the *continuous container selection* problem consists of a set of input points $\mathcal{C}$ in a d-dimensional space $\mathbb{R}^d$, and a budget k. We say that a point $c(c_1, c_2, \ldots, c_d)$ *dominates* (alternatively *contains*) a point $p(x_1, x_2, \ldots, x_d)$ if $x_i \leq c_i$, for all $i \in [d]$. The cost of assigning any input point p to a container point $c(c_1, c_2, \ldots, c_d)$ is the ℓ_1-norm of the container point, i.e, $c_1 + c_2 + \ldots + c_d$, if c dominates p; else, the assignment cost is ∞. The goal is to choose a set $S \subseteq \mathbb{R}^d$ of k container points, such that each input point is assigned to a container point in S, and the total assignment cost is minimized. In the *discrete* version of the problem, we have a further restriction that $S \subseteq \mathscr{F}$, where $\mathscr{F} \subseteq \mathbb{R}^d$ is an arbitrary, but finite, subset of points in the space. This problem variant is motivated by the fact that each container must itself "fit" into at least one physical processing node, or by the fact that certain resources (memory, for instance) are only allocated in fixed increments.

Example 1.2.1 *Figure 1.2 shows an instance of the continuous container selection problem. Here, we are allowed to choose $k = 3$ container points. We note that an input point can be assigned to a container point only if the former lies on the rectangle formed by the latter and the origin as shown. In this example, the input*

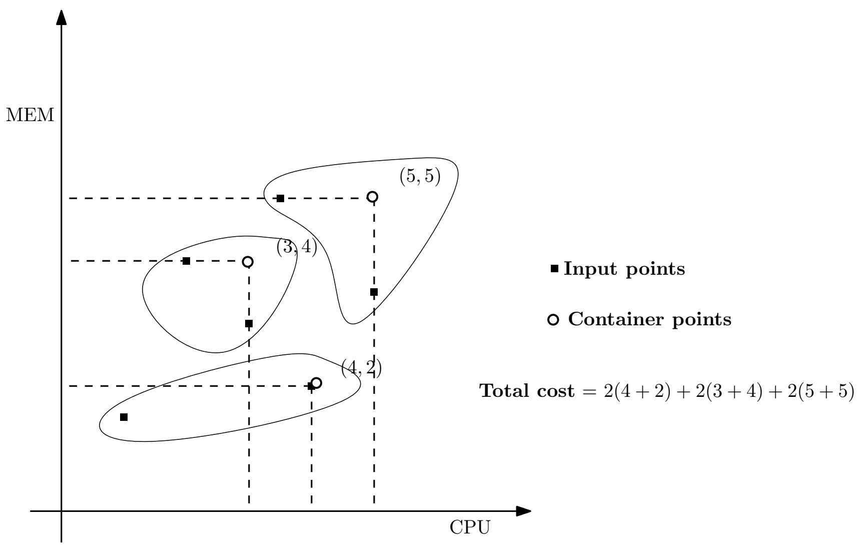


Figure 1.2: Container selection problem.

points in a given closed region is assigned to the (only) container point in that region.

Clustering problems such as k-median, k-center, and k-means have received considerable attention in recent decades. (See, for example, [18–20] and the references therein). Below, we only discuss the highlights directly relevant to our work. Our problem is a special case of the *non-metric k-median* problem, and it also bears some similarity to the ℓ_1-norm *Euclidean k-median* problem. There is a $(1+\epsilon,(1+\frac{1}{\epsilon})\ln n)$ bi-approximation algorithm for non-metric k-median [21], which finds a solution whose cost is within a $(1+\epsilon)$ factor of optimal, for any constant $\epsilon > 0$, while using at most $k(1+\frac{1}{\epsilon})\ln n$ centers. The paper [21] also shows, using a reduction from the set cover problem, that these guarantees are the best one can hope for. On the other hand, the metric variant of the k-median problem is known to have small constant-factor approximation algorithms, with no violation of k. The best known ratio $2.611+\epsilon$ is due to Byrka et. al. [22]. For the Euclidean k-median problem, which is a special case of metric k-median, there is a *polynomial* *time approximation scheme* (PTAS) [23].

We note that our problems, due to their "non-metric" characteristics, are fundamentally different from the Euclidean k-median problem.

1.3 Virtual backbone in wireless ad hoc networks

A *connected dominating set* (CDS) in a graph is a dominating set that induces a connected subgraph. The CDS problem, which seeks to find the minimum such set, has been widely studied [8,9,24–31] starting from the work of Guha and Khuller [8]. The CDS problem is NP-hard and thus the literature has focused on the development of fast polynomial time approximation algorithms. For general graphs, Guha and Khuller [8] propose an algorithm with a $\ln \Delta + 3$ approximation factor, where Δ is the maximum degree of any vertex. Better approximation algorithms are known in special classes of graphs. For the case of planar [32] or geometric unit disk graphs [10] polynomial time approximation schemes (PTAS) are known. This problem has also been extensively studied in the distributed setting [24, 25]. Not surprisingly, CDS problem in general graphs is at least as hard to approximate as the set cover problem for which a hardness result of $(1-\epsilon)\log n$ (unless $NP \subseteq DTIME(n^{O(\log\log n)})$) follows by the work of Feige [33].

CDS has become an extremely popular topic, for example, the recent book by Du and Wan [28] focuses on the study of ad hoc wireless networks as CDSs provide a platform for routing on such networks. In these ad hoc wireless networks, a CDS can act as a virtual backbone so that only nodes belonging to the CDS are responsible for packet forwarding and routing. Minimizing the number of nodes in the virtual backbone leads to increased network lifetime, and lesser bandwidth usage due to control packets, and hence the CDS problem has been extensively studied and applied to create such virtual backbones.

One shortcoming of using a CDS as a virtual backbone is that a few distant clients (outliers) can have the undesirable effect of increasing the size of the CDS without improving the quality of service to a majority of the clients. In such scenarios, it is often desirable to obtain a much smaller backbone that provides services to, say, (at least) 90% of the clients. Liu and Liang [29] study this problem of finding a minimum *partial connected dominating set* in wireless sensor networks (geometric disk graphs) and provide heuristics (without guarantees) for the same.

A complementary problem is the budgeted CDS problem where we have a budget of k nodes, and we wish to find a connected subset of k nodes which dominate as many vertices as possible. Budgeted domination has been studied in sensor networks where bandwidth constraints limit the number of sensors we can choose and the objective is to maximize the number of targets covered [30, 31].

Example 1.3.1 *Figure 1.3a shows a connected dominating set. Figure 1.3b depicts a partial connected dominating set with quota* 19 (15 *shaded and* 4 *darkened nodes*), *which is also a budgeted connected dominating set with budget* 4 (*darkened nodes*).

Another application of BCDS arises in the context of social networks. Consider a social network where vertices of the network correspond to people and an edge joins two vertices if the corresponding people influence each other. Avrachenkov et

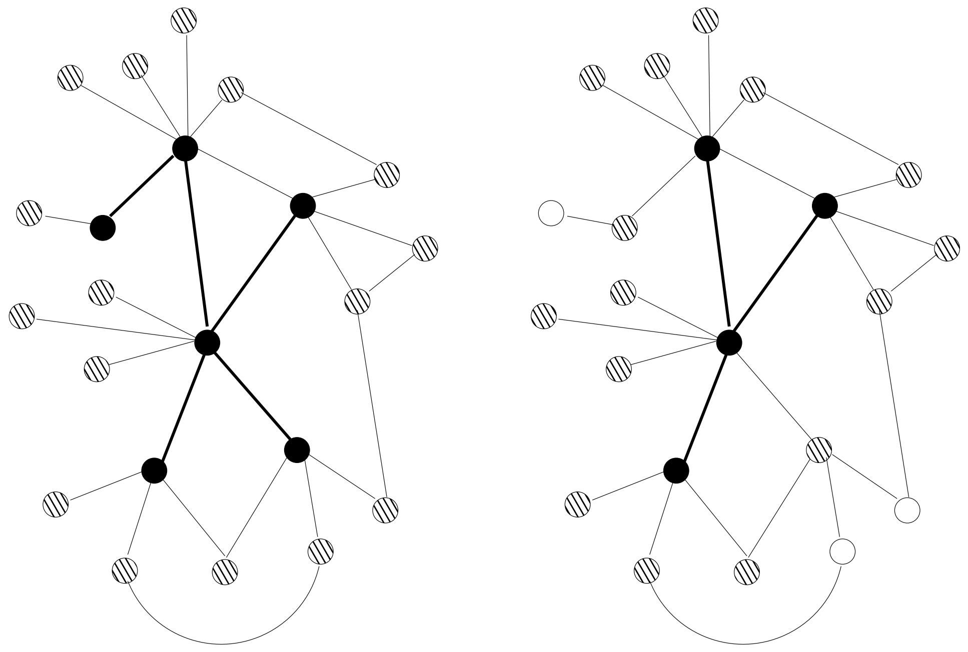

(a) A connected dominating set (b) A partial/budgeted CDS

Figure 1.3: Illustration of CDS, PCDS and BCDS.

al. [34] consider the problem of choosing k connected vertices having maximum total influence in a social network using local information only (i.e., the neighborhood of a vertex is revealed only after the vertex is "bought") and provide heuristics (without guarantees) for the same. Borgs et al. [35] show that no local algorithm for the partial dominating set problem can provide an approximation guarantee better than $O(\sqrt{n})$. As the influence of a set of vertices is simply the number of dominated vertices, these problems are exactly the budgeted and partial connected dominating set problems with the additional restriction of local only information.

Budgeted versions of set cover (known as max-coverage)[1] are well understood and the standard greedy algorithm is known to give the optimal $1 - \frac{1}{e}$ approxima-

[1]Here instead of finding the smallest sub-collection of sets to cover a given set of elements, we fix a budget on number of sets we wish to pick with the objective of maximizing the number of covered elements.

tion [36]. Khuller et al. [37] give a $(1-\frac{1}{e})$ approximation algorithm for a generalized version with costs on sets. In addition, we may consider a partial version of the set cover problem, also known as partial covering, in which we wish to pick the minimum number of sets to cover a pre-specified number of elements. Kearns [38] first showed that greedy gives a $2H_n + 3$ approximation guarantee (where n is the ground set cardinality and H_n is the n^{th} harmonic number), which was improved by Slavík [39] to obtain a guarantee of $min(H_{n'}, H_\Delta)$(where n' is the minimum coverage required and Δ is the maximum size of any set). Wolsey [40] considered the more general *submodular cover* problem and showed that the simple greedy delivers a best possible $\ln n$ approximation.

For the case where each element belongs to at most f (called the *frequency*) different sets, Gandhi et al. [41], using a primal-dual algorithm, and Bar-Yehuda [42], using the local-ratio technique, achieve an f-approximation guarantee.

Unfortunately, for both the budgeted and partial versions of the CDS problem, greedy approaches based on prior methods fail. The fundamental reason is that while the greedy algorithm works well as a method for rapidly "covering" nodes, the cost to connect different chosen nodes can be extremely high if the chosen nodes are far apart. On the other hand if we try to maintain a connected subset, then we cannot necessarily select nodes from dense regions of the graph. In fact, none of the approaches in the work by Guha and Khuller [8] appear to extend to these versions. We will discuss this further in Chapter 4

Partial and budgeted optimization problems have been extensively studied in the literature. Most of these problems, with the exception of partial and budgeted set cover, required significantly different techniques and ideas from the corresponding "complete" versions. We will now cite several such problems.

The best example is the minimum spanning tree problem, which is well known to be polynomial time solvable. However, the partial version of this problem where we look for a minimum cost tree which spans at least k vertices is NP-hard [43]. A series of approximation techniques [44–47] finally resulted in a 2-approximation [48] for the problem.

Partial versions of several classic location problems such as k-center and k-median have required new techniques as well. The partial k-center problem, which is also called the outlier k-center problem or the robust k-center problem, requires us to minimize the maximum distance to the "best" n' nodes (while the complete version requires us to consider all the nodes) to the centers. Charikar et al. [49] gave a 3-approximation algorithm whose analysis was significantly different from the classic k-center 2-approximation algorithm [50, 51]. Chen [52] gives a constant approximation for outlier k-median problem, while Charikar et al. [49] gave a 4-approximation for the outlier uncapacitated facility location problem.

Several other optimization problems need special techniques to tackle the corresponding partial versions. Notable examples of such problems, include *partial vertex cover* [41, 53–57], quota Steiner tree problems [58], budgeted and partial node weighted Steiner tree problems [59, 60], and scheduling with outliers [61, 62]. We end this subsection by noting that partial versions of some optimization problems are completely inapproximable even though, the corresponding complete version has a small constant approximation algorithm. The best example of this is the *robust* *subset resource replication* problem studied in Chapter 2.

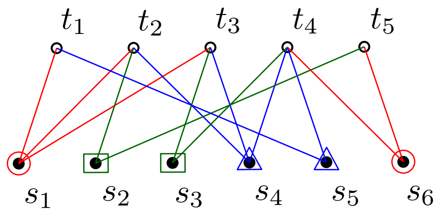
1.4 Coverage in sensor and radio networks

One of the key challenges in designing algorithms and protocols for sensor and radio networks is efficient energy utilization. For instance, in sensor networks a trivial solution to monitor all targets is to keep all the sensors active. However, such a scheme is bound to have a very limited lifetime because the battery life of the sensors is limited. Slijepcevic et al. [63] propose partitioning the sensors into disjoint set covers and keeping only one set active at a time in order to extend the network lifetime. Similarly in radio networks where we have a set of transmitters and receivers, it is beneficial to use one transmitter only once in sending a particular message. However, in such a radio network we need to deal with the additional issue of network interference. One of the theoretical frameworks proposed to handle interference in radio networks is called *unique coverage*. Here, a receiver gets the message only if exactly one of the transmitters in its range transmits the message at a time.

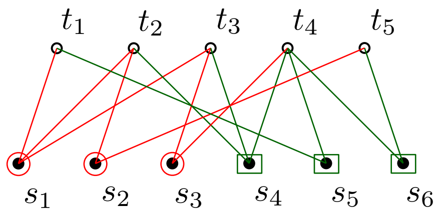
In Chapter 5, we consider two important problems as described below that arise in these scenarios. In both these problem, we are given a bipartite graph with parts S and T. For $A \subseteq S$ (respectively $\subseteq T$), we represent the set of neighbors of A in T (resp. in S) by $\mathcal{N}(A)$. For $A \subseteq S$, we define the coverage of A, denoted by $\mathcal{C}(A)$, as the size of its neighborhood, i.e. $\mathcal{C}(A) = |\mathcal{N}(A)|$. We say that a partition $\mathcal{P}$ of S uniquely covers a vertex $t \in T$, if there is at least one subset of $S_i \in \mathcal{P}$ such that S_i uniquely covers t. Further let $|S| = n$ and $|T| = m$.

Scenario 1. The primary goal of a sensor network is to monitor targets for as long as possible. Due to energy constraints, it is infeasible to keep all the sensors active for long. One approach to prolong the network lifetime is to partition the sensors into k disjoint sets such that each set covers all the targets in T. This is exactly the well studied domatic partition problem that is known to have a tight $O(\log n)$ approximation algorithm [64, 65]. However, requiring each set to be a full set cover is often too strict a requirement in practice. Wang and Kulkarni [66] show that empirically the network lifetime can be improved significantly if we allow each set to only cover most (and not all) targets. Indeed, Abrams et al. [67] consider the *set k-cover* problem of partitioning the sensors into exactly k sets with the objective of maximizing the total number of targets covered by each set, i.e., maximize $\sum_{i=1}^{k} \mathcal{C}(S_i)$. One potential drawback of this formulation is that, while the sum of targets covered is high, a particular set of sensors might have very low coverage. To tackle this problem, we study the following *max-min k-cover* problem: *partition the sensors into k sets such that the minimum coverage of any set is maximized*, i.e., maximize $\mathsf{Min}_{i\in[k]}\ \mathcal{C}(S_i)$.

Scenario 2. Our second scenario is motivated by interference issues in radio networks. The interference effect is modeled in the following way: a receiver gets a message if and only if exactly one of the transmitters in its neighborhood is active at a time. In such a setting, the *unique coverage problem* [68] asks for a set of transmitters that maximizes the number of receivers who acquire the message. We consider the situtation where *all* receivers need to get the message. This can be achieved by grouping the transmitters into sets and activating only one set at a time. As long as every receiver is covered uniquely in *at least one* set, all receivers will obtain the message at the end. Thus, the number of sets we create will equal the time it takes for all receivers to get the message. Further as the transmitters have limited energy resources, it is desirable to use them only once, i.e. the sets are disjoint. Even et al. [69] introduce the *minimum conflict-free coloring* problem that seeks to minimize the number of such sets[2]. As this problem is NP-hard to approximate with a factor of $\max(n^{1-\epsilon}, m^{\frac{1}{2}-\epsilon})$, we consider two of its natural relaxations and obtain polylogarithmic approximation algorithms.



(a) Scenario 1



(b) Scenario 2

Figure 1.4: The *max-min k-cover* and *minimum conflict-free coloring* problems.

Example 1.4.1 *Figure 1.4a depicts an optimal solution for a max-min 3-cover instance, i.e., $S_1 = \{s_1, s_6\}$, $S_2 = \{s_2, s_3\}$, $S_3 = \{s_4, s_5\}$. The respective coverages of S_1, S_2 and S_3 are 5, 5 and 4. Hence, the max-min coverage is 4. Figure 1.4b shows an optimal solution for an ASDF instance: $S_1 = \{s_1, s_2, s_3\}, S_2 = \{s_4, s_5, s_6\}$. When S_1 is activated to transmit the message, in the first round, t_1, t_4 and t_5 uniquely receive it. Subsequently, in the second round, when S_2 is activated t_1, t_2, t_3 and t_5 receive it. Thus, in two rounds all the receivers get the message.*

[2]A trivial solution is to have each transmitter as a singleton set. Clearly all receivers will be uniquely covered but the number of sets is too large

Abrams et al. [67] study the *set k-cover* problem and show that a simple randomized scheme provides a $(1-\frac{1}{e})$ approximation. Deshpande et al. [70] introduce the *max-min k-cover* problem that we address in this paper and provide an LP-based heuristic for the same. While Deshpande et al. [70] provide an approximation ratio of $(1 - \frac{1}{e})$, in their work the sensor sets are not guaranteed to be disjoint and only the expected number of sets to which a sensor belongs is one. We resolve this open problem by guaranteeing disjoint sets of sensors and obtaining a $\frac{1}{3}(1 - \frac{1}{e})$ approximation. Cheng et al. [71] also consider the *max-min k-cover* problem but do not provide any approximation guarantees for this problem. The *max-min k-cover* problem and its variants have been extensively studied in geometric graphs as they model wireless networks well. A number of authors [72, 73] provide efficient heuristics for the same. Survey articles such as [74, 75] succinctly summarize the vast body of work on energy efficient monitoring in sensor networks.

Related to Scenario 2, is the *unique coverage* problem where the goal is to choose a set of transmitters such that the number of receivers covered *uniquely* is maximized. Demaine et al. [68] obtain an approximation algorithm with an $\Omega(\frac{1}{\log n})$ guarantee and also prove a semi-logarithmic $(O(1/\log^{1/3-\epsilon} n))$ hardness of approximation. Even et al. [69] introduce the *minimum conflict-free coloring* (MCFC) problem and study it in the special case of set systems induced by geometric regions in the plane. In this setting, the receivers are simply points in the plane and each transmitter defines a region such that it can send messages to all receivers in that region. The MCFC problem is now to color all regions using minimum number of colors so that for each point there is an uniquely colored region that contains it. They show that for a number of geometric structures such as disks, axis parallel rectangles, and regular hexagons, $O(\log n)$ colors (i.e. partitions) are sufficient. Pach and Tardos [76] show that $O(\sqrt{m})$ colors are enough to uniquely cover all elements in general set systems.

Closely related to the conflict-free coloring is the concept of an *ad-hoc selective family* introduced by Clementi et al. [77]. Motivated by broadcasting in radio networks, they describe a procedure to find a small family of sets of transmitters (called *ad-hoc selective family*), with size $O(\log n \log m)$, such that every receiver is uniquely covered by at least one member set. But the key property that this solution lacks is disjointness among the member sets of the family, which is critical to ensure network longevity.

1.5 Oracles in massive networks

With the proliferation of social networks, it is not uncommon to have networks with billions of nodes, e.g., Facebook. The use of classic algorithms on such networks for processing even simple distance queries may not be acceptable in a realistic setting. For example, using the Dijkstra's shortest path algorithm to answer distance queries on a billion node graph might take hours (if not days) to complete, as the running time depends on the entire graph size. A natural approach to handle this problem is to preprocess and generate a data structure that allows the query algorithm running time to depend only on the query size. In the case of distance queries, a naive way to do this is to preprocess and store distances between all the pairs of vertices in the graph so as to obtain a constant query time. Unfortunately, in this approach we require space quadratic in the network size and therefore is infeasible. Ideally, it is desirable to obtain a data structure whose size is nearly linear in the input size and which enables query processing algorithms with running times that depend only on the query size instead of the entire graph size.

A significant amount of research [7, 11, 78–84] has been dedicated to the problem of constructing data structures, called the *distance oracles*, that require space sub-quadratic in the network size while answering distance queries approximately in constant query time. Thorup and Zwick (referred to as TZ, from now on) [11] design a preprocessing algorithm that runs in time $O(\ell m n^{1/\ell})$ on a graph with n nodes, m edges, and a parameter ℓ, and constructs a data structure of size $O(\ell m n^{1+1/\ell})$. Using this data structure they then design a query algorithm that can answer distance queries in time $O(\ell)$ with an approximation guarantee of $2\ell - 1$. Following this work, there is has been a flurry of research work [7, 78–81] dealing the problem under special conditions. For example, improved distance oracles have been obtained when the graph is sparse [78], planar [81] or other special graphs like power law graphs [84].

We study the problems of *Steiner tree* and *cheapest tour* in the *oracle* setting, in the same vein as the seminal work of Thorup and Zwick [11] on distance oracles. Besides their theoretical importance, these classic problems have a number of applications in a variety of domains including social networks [85], computer vision [86], very large scale integration [87], relational databases [85], evolutionary biology [88], planning [89], and vehicle routing [90].

Cygan et. al. [91] consider the problem of Steiner tree (along with other problems) in the oracle model, under the special case when the metric is a *doubling metric*, and obtain near optimal approximation guarantees. Unfortunately, their approach depends heavily on the *doubling metric* properties and do not seem to extend to the general metric case. Gubichev and Neumann [85] study the (general metric) Steiner tree problem in the oracle model and obtain heuristic based algorithms but with no approximation guarantees.

A classic result due to Kou, Markowsky and Berman [92] shows that if we construct the shortest path metric over the terminal set and then compute a minimum spanning tree on the metric graph we obtain a 2-approximation guarantee. An immediate corollary of this result is that if we have access to an α approximate distance oracle we obtain a Steiner tree with a 2α approximation guarantee. Therefore, this result along with TZ's distance oracle gives a $2(2\ell - 1) = 4\ell - 2$ guarantee for the Steiner tree query problem. A similar result for the cheapest tour problem due to Christofides [93] yields a $3/2$ approximation guarantee. Again, this result along with TZ's result yields a guarnatee of $3/2(2\ell - 1) = 3\ell - 1.5$ on the approximation ratio for the cheapest tour problem. In Chapter 6, we discuss improved algorithms for these both these problems.

We note that both these problems have been studied extensively in the classical setting. Kou, Markowsky and Berman [92] gave the first non-trivial 2-approximation algorithm for the Steiner tree problem. This was followed by a series of improvements [92, 94–99] finally resulting in the current best of 1.39 [100]. From a hardness point of view, it has been shown that, unless $P \neq NP$, we cannot obtain a $1 + \epsilon$

approximation guarantee, for an arbitrarily small ϵ [101]. Focus has also been on obtaining better approximation guarantees for special graphs. For example polynomial time approximation schemes (PTAS) are known for the Steiner tree problem in planar graphs [102] and geometric graphs [103].

The cheapest tour is a special case of the metric travelling salesman problem when the underlying metric is the shortest path. Christofides [93] gave a famous 1.5 approximation algorithm for this problem. Gharan et al. [104] improved this guarantee to $1.5 - \epsilon$, for some small constant $\epsilon > 0$. Arora [103] and Mitchell [105] simultaneously and independently came up with a PTAS for this problem when the underlying metric is Euclidean. The problem also admits a PTAS on planar graphs [106]. It is known to be MAX SNP-hard [107] and therefore no polynomial time approximation scheme is possible assuming $P \neq NP$.

1.6 Contributions

In this section, we briefly describe various results that form the content of this thesis.

Resource replication problems. In Chapter 2, we obtain approximation results for several resource replication problems. These range from small constant factor approximation guarantees to proving non-existence of any non-trivial approximation guarantees.

- In Section 2.2, we consider the basic replication problem where each client needs all k data items (*basic resource replication*) and its generalization where each client might need a subset of data items (*subset resource replication*).

For the first problem, we give a distributed polynomial time 3-approximation algorithm and show that there does not exist any polynomial time algorithm achieving a $2 - \epsilon$ (for any $\epsilon > 0$) approximation (Theorem 2.2.3 and Theorem 2.2.12). For the later, we give the first polynomial time 3-approximation algorithm (in a centralized setting) along with matching hardness (Theorem 2.2.11 and Theorem 2.2.12).

- In Section 2.3, we consider the outlier version of the basic as well as subset resource replication problem. For the former, we give a polynomial time 3-approximation algorithm while for the latter, somewhat surprisingly, we show that there does not exist any non-trivial approximation guarantee (in polynomial time). We also consider the case where each resource can be replicated at most K times and give polynomial time 5-approximation algorithm for it.

- In Section 2.4, we consider another natural generalization of the basic resource replication problem where each node has an upper bound (load) on the number of clients it can serve. We give a polynomial time 4-approximation algorithm for this version when load $L \geq 2k - 1$ (k is the number of resources). A simple counting argument shows that this problem is infeasible if $L < k$. This implies our 4-approximation algorithm is a bicriteria approximation algorithm and the load capacity is not violated by more than a factor of 2.

This is joint work with Khuller and Saha [108, 109].

Container selection problems. As noted before, the container selection problem is a special case of non-metric k-median, which is inapproximable unless we violate k

significantly [21]. However, this problem still has sufficient geometric structure. This structure allows us to obtain near optimal algorithms that, in the case of continuous container selection, do not violate k, and in the discrete case violate k mildly. In particular, in Chapter 3, we discuss the following results.

- We show that the *continuous container selection problem* admits a PTAS, for any fixed dimension d. On the negative side, we show that the problem is NP-hard for $d \geq 3$.

- We show that the *discrete* variant (for $d \geq 3$) is NP-hard to approximate within *any* guarantee if the budget k is not violated. On a positive note, we obtain constant factor bi-approximation algorithms for this variant. For any constant $\epsilon > 0$, the guarantees are $(1+\epsilon, 3)$, for $d = 2$, and $(1+\epsilon, O(\frac{d}{\epsilon} \log dk))$, for any $d \geq 3$.

Section 3.2 discusses the continuous variant, while Section 3.3 deals with the discrete problem. Section 3.4 describes the hardness results for these problems.

This is joint work with Nagarajan, Schieber, Shachnai and Wolf [110]. The corresponding systems version of this work is joint work with Wolf, Nabi, Nagarajan, Saccone, Wagle, Hildrum and Pring [5].

Connected dominating set problems. Chapter 4 will focus on some variants and generalizations of the connected dominating set problem.

- In Section 4.2, we obtain the first $O(\ln \Delta)$ approximation algorithm for the PCDS problem. To be precise, our approximation guarantee is $4 \ln \Delta + 2 + o(1)$, where Δ is the maximum degree.

- In Section 4.3, we obtain a $\frac{1}{13}(1-\frac{1}{e})$-approximation algorithm for the BCDS problem. This is the first constant approximation known for BCDS.

- We generalize the above problems to a special kind of submodular optimization problem (to be defined later) which has the *weighted profit connected dominating set* problem as a special case. Again we obtain $O(\ln q)$ and $\frac{1}{13}(1-\frac{1}{e})$ approximation algorithms for the partial and budgeted version of this problem respectively where q denotes the quota for the partial version. These results form the content of Sections 4.4, and 4.5.

This is joint work with Khuller and Purohit [111].

Covering problems in radio and sensor networks. We now briefly outline the main results of Chapter 5.

- In Section 5.2, we obtain a polynomial time algorithm with an approximation guarantee of $\frac{1}{3}(1-\frac{1}{e})$ for the *max-min k-cover* problem. We extend this result to a more general *submodular max-min k-cover* problem and obtain the same approximation ratio.

- As the *minimum conflict-free coloring* problem is hard to approximate, we relax it in two ways, in Section 5.4

  1. We allow a small fraction of vertices in T to be left uncovered. We show that we can obtain a partition of expected size $O(\frac{\log m \log n}{\epsilon})$ such that at least $(1-\epsilon)m$ vertices in T are uniquely covered in expectation.

  2. We relax the disjointness requirement by allowing a vertex $s \in S$ to ap-

pear in at most $O(\log n)$ sets. We now obtain a family of size $O(\log n \log m)$ that uniquely covers every $t \in T$.

- Matching the guarantee obtained by Pach and Tardos [76], in Section 5.3, we obtain an alternate $2\sqrt{m}$ approximation algorithm for the *minimum conflict-free coloring* problem in general networks.

This is joint work with Khuller and Purohit [112].

Steiner and cheapest tour oracles. Thorup and Zwick's work [11] directly yields oracles for *Steiner tree* and *cheapest tour* problems that can answer these queries with an approximation guarantee of $4\ell - 2$ and $3\ell - 1.5$ respectively, for any given parameter $\ell \geq 1$. The preprocessing time for these data structures is $O(\ell m n^{1/\ell})$ time and the data structure size itself is $O(\ell n^{1+1/\ell})$ - same as the distance oracle. In Chapter 6, we obtain improved guarantees of $3\ell + 2$ and $2.5\ell + 1.5$ respectively for Steiner tree and cheapest tour oracles, while maintaining the same time space time complexity.

This is joint work with Bhatia and Gupta [113].

Chapter 2

Resource Replication Problems

2.1 Road map to the chapter

We consider several variants of problems in the following framework: given a collection of k data items, we wish to distribute them to a collection of n nodes modeled by a graph, where the vertices are embedded in a metric space. In the basic model, each node wishes to access each of the k data items and the goal is to minimize the *maximum* distance any node has to travel to access all k items. For this problem, Ko and Rubenstein [3] give a distributed algorithm based on a local search idea and also show that this algorithm delivers a solution with a worst case approximation guarantee of 3. Although the algorithm is claimed to converge reasonably quickly in practice, it is unknown if it does so in polynomial time. In a followup piece of work [2], Ko and Rubenstein introduced a generalization of the basic problem in which each node only required a *subset* of the items. For this problem, they develop a heuristic; however, for this heuristic, unlike the other case, there is no approximation guarantee any more.

In Section 2.2, we study both these problems, i.e., the *basic resource replication*, where each client needs all k data items and its generalization, the *subset resource replication*, where each client might need a subset of data items. For the first problem, we give a distributed polynomial time 3-approximation algorithm and show that there does not exist any polynomial time algorithm achieving a $2-\epsilon$ (for any $\epsilon > 0$) approximation (Theorem 2.2.3 and Theorem 2.2.12). For the latter, we give the first polynomial time 3-approximation algorithm (in a centralized setting) along with matching hardness (Theorem 2.2.11 and Theorem 2.2.12).

In Section 2.3, we consider the outlier version of the basic as well as subset resource replication problem. In an outlier version of a resource replication problem, the goal is satisfy a specified fraction (say 90%) of nodes, instead of all nodes. The motivation to consider these problems, is that a few "outlier" nodes might have an adverse effect on optimal solution distance and therefore it might be useful to optimize the solution that ignores some such nodes. For the outlier version of the BRR problem, we give a polynomial time 3-approximation algorithm while for the general problem, somewhat surprisingly, we show that there does not exist any non-trivial approximation guarantee (in polynomial time). We also consider the case where each resource can be replicated at most K times and give polynomial time 5-approximation algorithm for it.

In Section 2.4, we consider another natural generalization of the basic resource replication problem where each node has an upper bound (load) on the number of clients it can serve. We give a polynomial time 4-approximation algorithm for this version when load $L \geq 2k-1$ (k is the number of resources). A simple counting argument shows that this problem is infeasible if $L < k$. This implies our 4-approximation algorithm is a bicriteria approximation algorithm and the load capacity is not violated by more than a factor of 2.

2.2 Satisfying everyone: Resource replication

We start with the basic and subset resource replication problems in this section, and subsequently look at more general variants. Subsection 2.2.1 will consider the basic version, while the Subsection 2.2.2 is dedicated to the subset resource replication problem.

2.2.1 Basic resource replication problem

Formally, the *basic resource replication* problem can be defined as follows.

Definition 2.2.1 (The basic resource replication (BRR) problem) *In an instance of the problem, we are given, a set of nodes or vertices $V = \{v_1, v_2, \dots, v_n\}$, a metric space defined by the function $d : V \times V \to \mathbb{R}^+ \cup \{0\}$, a set of resources (or colors) $\mathcal{C} = \{C_1, C_2, C_3, \dots, C_k\}$. We seek to find an* optimal *mapping $\phi : V \to \mathcal{C}$ of colors to vertices, with respect to the following objective: $\underset{\phi}{\mathsf{Min}} \underset{\substack{v \in V \\ C_r \in \mathcal{C}}}{\mathsf{Max}} d_r(v)$, where $d_r(v)$ to be the shortest distance between a vertex assigned the color C_r[1] and the vertex v.*

This is the central problem of the work of Ko and Rubenstein [3] who give a distributed algorithm with a 3-approximation guarantee. Unfortunately, their algorithm has no proven polynomial running time bound. We give a simple distributed polynomial time 3-approximation algorithm for this problem.

All the algorithms in this work use a technique called *threshold graph construction* introduced by Edmonds and Fulkerson [114] and used extensively for k-center type

[1]We may abuse the notation and use same expression, $d_r(v)$, when r represents a color.

problems [50,115–117]. We observe that the use of this approach enables the design of very simple and efficient algorithms for several resource replication problems. Given $\delta \in \mathbb{R}^+ \cup \{0\}$, the *threshold graph*, denoted by G_δ, is constructed by adding edges between every pair of vertices u, v which are at distance at most δ.

Our distributed algorithm (Algorithm 1) for BRR works in the following way. In the first step, each vertex v determines the distance of the $(k-1)^{th}$ closest neighbor - call this $l_{k-1}(v)$. Now in a distributed fashion each vertex obtains the maximum value $\delta_L = \max_v l_{k-1}(v)$. We observe that the threshold graph G_{δ_L} has minimum degree at least $k-1$. Let δ_{OPT} be the minimum value of δ for which a feasible solution exists. δ_L must be a lower bound on this optimal δ value (δ_{OPT}) - because δ_L is the least value such that the threshold graph has degree at least $k-1$ and $G_{\delta_{OPT}}$ has minimum degree at least $k-1$. We set $\delta = \delta_L$, and construct the graph G^2_δ which is the graph formed by squaring G_δ. In other words, each vertex v maintains a list of all vertices within two hops in G_δ as its neighbors. Using standard distributed algorithms (see for e.g., [118]), we compute a maximal independent set $\mathcal{I}$ in G^2_δ. Finally, each vertex in $\mathcal{I}$ colors itself with C_1 and picks $k-1$ vertices from its list of neighbors in G_δ ($N_{G_\delta}(v)$) and assigns them a distinct color from the remaining $k-1$ colors.

Algorithm 1 Distributed 3-approximation algorithm for BRR

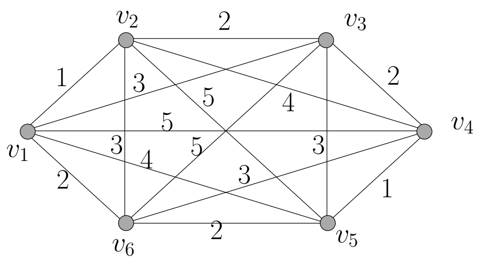
```
1: Choose $\delta = \delta_L$, where $\delta_L$ is the smallest value such that $G_{\delta_L}$ has minimum degree $\geq k-1$.
2: Each node maintains a list of its neighbors in $G_\delta^2$.
3: Compute a maximal independent set $\mathcal{I}$ in a distributed fashion [118].
4: for $v \in \mathcal{I}$ do
5:     Color $v$ with $C_1$, and arbitrarily pick $(k-1)$ vertices from the set $N_{G_\delta}(v)$ say $\{v'_1, v'_2, \ldots v'_{k-1}\}$ and colors them with $C_2 \ldots C_k$ respectively.
6: end for
7: Assign arbitrary colors to vertices which have not received any color so far.
```

We illustrate Algorithm 1 by a simple example.

Example 2.2.2 *We refer to Figure 2.1 for various steps of the example: (a) given a BRR instance with $k = 3$ and the edge weights representing pairwise distances; (b) computation of δ_{k-1} values for various vertices and the threshold δ_L is set to the maximum of these values; (c) threshold graph, G_{δ_L}: delete all edges with weights greater than δ_L and keep the rest (d) compute $G^2_{\delta_L}$: if u, v are within two hops of each other in G_{δ_L}, then we add an edge between them in $G^2_{\delta_L}$ and find a maximal independent set $\mathcal{I}$, for example, $\mathcal{I} = \{v_3, v_6\}$ is a candidate; (e) each member v of $\mathcal{I}$, assigns itself the color C_1 and colors its neighbors (in G_{δ_L}) arbitrarily using the remaining $k-1$ colors, using every color at least once.*

Theorem 2.2.3 *Algorithm 1 gives a 3-approximation for the BRR problem.*

Proof: We prove that for every vertex v and every color r, $d_r(v) \leq 3 \times \delta_L$. Since $\delta_L \leq \delta_{OPT}$, the result follows. If $v \in \mathcal{I}$, by construction $d_r(v) \leq \delta_L$. For vertices v, which are adjacent to some vertex (i.e., one hop distance) of $\mathcal{I}$ in G_δ, $d_r(v) \leq 2 \times \delta_L$ and for vertices at two hop distance from $\mathcal{I}$, $d_r(v) \leq 3 \times \delta_L$. There are no vertices

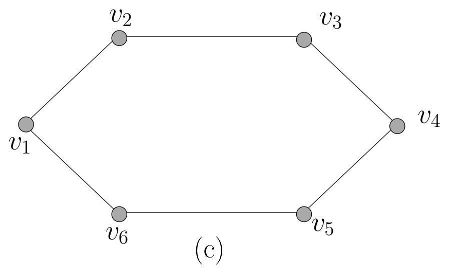


(a)

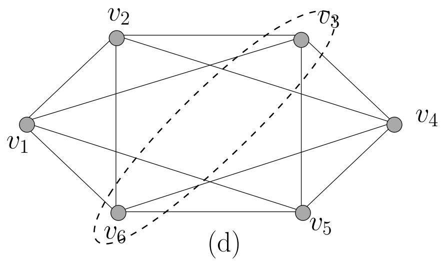
Vertex	$\delta_{\mathbf{k-1}}$
v_1	2
v_2	2
v_3	2
v_4	2
v_5	2
v_6	2

$\delta_L = 2$

(b)



(c)



(d)

Vertex	**Color**
v_1	C_3
v_2	C_2
v_3	C_1
v_4	C_3
v_5	C_2
v_6	C_1

(e)

Figure 2.1: An illustrative example for the BRR algorithm.

at ≥ 3 hop distance from $\mathcal{I}$, since the latter is a maximal independent set in $G^2_{\delta_L}$. ■

Limiting the amount of resource replicated. We first consider the following generalization of BRR: each color C_i has a bound $K \in \mathbb{N}$, which is the number of copies of C_i that can be used. This problem is also a natural generalization of the K-center problem (where there is a single resource with bound K). We note that a simple modification to Algorithm 1 solves this generalized version of BRR with capacity bound on colors. In fact the only difference between the algorithms for BRR and this version is how we choose δ (and we do not follow step 7). For this case, we must try out all the possible values of optimal δ (there are at most $O(n^2)$ such values) and choose the smallest δ which satisfies the following two properties : (1) each vertex of G_δ has degree $\geq k-1$; (2) the computed maximal independent set in G^2_δ has size at most K. Clearly, this gives a feasible solution for the problem, as we follow steps 4-6 of Algorithm 1 to assign color.

Lemma 2.2.4 *The optimal distance δ_{OPT}, for an instance of the generalized BRR, where at most K copies of each color may be used, must satisfy the following properties: (1) each vertex of $G_{\delta_{OPT}}$ has degree $\geq k-1$; (2) any maximal independent set in $G^2_{\delta_{OPT}}$ has a size at most K.*

Proof: The first condition is obvious. To see the second condition, suppose there exists a maximal independent set $\{u_1, u_2, \ldots, u_L\}$ in $G^2_{\delta_{OPT}}$ whose size $L > K$. Then, in $G_{\delta_{OPT}}$, it is not possible to satisfy all $u_1, u_2, \ldots, u_L$ by at most K copies of a single color, because then there exists at least a pair of vertices u_i and u_j, $i, j \in [1, L]$, within at most two hops. ■

The above lemma guarantees that the computed, $\delta \leq \delta_{OPT}$. Now, a 3-approximation guarantee follows immediately by an analogous argument to Theorem 2.2.3. Formally, we have the following theorem.

Theorem 2.2.5 *There is a polynomial time 3-approximation algorithm for the generalized BRR problem.*

Consider a further generalization of the BRR problem. Apart from the input given to the BRR instance, we are provided with placement cost of each resource j on a vertex i, c_{ij}. There can be two possible definitions of the weighted version of the problem (abbreviated as WBRR1 and WBRR2 respectively) -

1. WBRR1: Given a budget B, solve the BRR problem such that the total (sum of) cost of placement of various resources on the vertices must not exceed B.

2. WBRR2. Given budgets for each resource B_r, solve the BRR problem such that total cost of placement associated with each resource C_r does not exceed B_r.

For the first version of the problem, Algorithm 1 can be easily extended. This result generalizes the 3-approximation algorithm for weighted k-center problem [119].

Algorithm 2 3-approximation algorithm for weighted BRR

```
1: let $\mathcal{D}$ be the list of possible $\delta$ values, i.e., the list of pairwise distances between
   the vertices of $G$, arranged in the non-decreasing order.
2: for all $\delta \in \mathcal{D}$ such that the minimum degree in $G_\delta$ is $k-1$ do
3:   construct the threshold graph $G_\delta$ and the two-hop graph $G_\delta^2$.
4:   compute a maximal independent set $\mathcal{I}$ in $G_\delta^2$.
5:   for $v \in \mathcal{I}$ do
6:     construct a weighted complete bipartite graph as follows: neighbors of $v$,
       including $v$ itself, in $G_\delta$, are on one side (call left) and $k$ vertices, one for
       each color, on the other side (call right); the weight of an edge incident on
       the vertex $i$ and the color $j$ is assigned to $c_{ij}$, i.e., the cost of placing color
       $j$ on $i$.
7:     on the above bipartite graph, compute a minimum weight matching that
       saturates all the $k$ nodes on the right.
8:     we assign the color $j$ to a neighbor $i$ of $v$, if the edge $\bar{ij}$ is in the matching.
9:   end for
10:  if total weight of all matchings (corresponding to all $v \in \mathcal{I}$) is at most $B$
     then
11:    return current assignment of colors to vertices and exit.
12:  end if
13: end for
14: if no solution has been returned so far then
15:   return $\phi$ indicating no feasible solution exists.
16: end if
```

Theorem 2.2.6 *Algorithm 2 gives a 3-approximation for the* WBRR1 *problem.*

Proof: For the optimal δ, each $v \in \mathcal{I}$ has at least k neighbors (including itself) in G_δ. Hence, a minimum weighted matching saturating all the k nodes on the right side of the bipartite graph, defined on the Line 6 of Algorithm 2, is well defined. This ensures that v will have all k colors in its neighborhood and we are opening these in the cheapest possible way. Thus, for an optimal δ, the total cost should not exceed the budget B. We are returning a solution with minimum value of δ for which the total cost is at most B; hence this value serves as a lower bound to the optimum. Also, clearly, each vertex obtains all the required colors with in 3δ distance. ■

We now observe that there is no constant approximation for the second version of weighted basic resource replication (WBRR2) problem. We reduce the classic NP-hard problem of *subset sum* [120] to an instance of this problem. In an instance of the subset sum problem, we are given a set of elements S, with each element $e \in S$ having a weight w_e and an real number bound B. The goal is to compute a subset S' of S such that the total sum of weights of elements in S' is exactly B.

Theorem 2.2.7 *Assuming $P \neq NP$, there is no polynomial time constant approximation algorithm for the* WBRR2 *problem.*

Proof: For some constant $c > 0$, suppose there is a c-approximation algorithm, denoted by A, for the WBRR2 problem. Given an instance of the subset sum problem, $I = (S, B)$, we construct the following instance of WBRR2, $I' = (G, \mathcal{C} = \{C_1, C_2\}, \mathcal{B} = \{B_1, B_2\})$. The graph G is of a collection of independent edges, one for each element of S. The distance between any vertices on two distinct edges is $\geq c+1$ and that between end points of the same edge is 1. There are two colors in the instance C_1, C_2 and every vertex requires both of them. We mark one vertex on each edge as positive and other vertex as zero: the placement cost of either colors on the positive vertex is the weight of the corresponding element in the set S and the placement costs on the zero vertices is 0. We now place the weight bounds, on colors C_1 and C_2 respectively as, $B_1 = B$ and $B_2 = (\sum_{e \in S} w_e) - B$, where w_e is the weight of element e. It is easy to observe that I is a *yes* instance if and only if the c-approximation algorithm A returns the value 1 as the solution. ■

2.2.2 Subset resource replication problem

In the BRR model each client requires all the data items. But in general each client might be interested in a subset of resources instead of all the resources. The servers might also have capacity to hold several data items. This substantially more generalized version of resource replication problem, which we call the *subset resource replication* problem (SRR) was considered by Ko and Rubenstein in a subsequent paper [2]. Formally,

Definition 2.2.8 (The subset resource replication (SRR) problem.) *In an instance of the problem, we are given, a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$, a metric $d : V \times V \to \mathbb{R}^+ \cup \{0\}$, and a set of colors $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$. Every vertex $v \in V$ has a subset $\mathcal{C}_v \subseteq \mathcal{C}$ of "required" colors and a non-negative integer s_v as the storage capacity, i.e., we can assign s_v colors to vertex v. The goal is to assign a list of colors $\phi(v) \subseteq \mathcal{C}$ to each vertex v, such that $|\phi(v)| \leq s_v$, with the following objective: $\delta = \underset{\phi}{\mathsf{Min}} \; \underset{\substack{v \in V \\ r \in \mathcal{C}_v}}{\mathsf{Max}} \; d_r(v)$, where $d_r(v)$ is the shortest distance from v to some vertex u, such that $C_r \in \mathcal{C}_u$.*

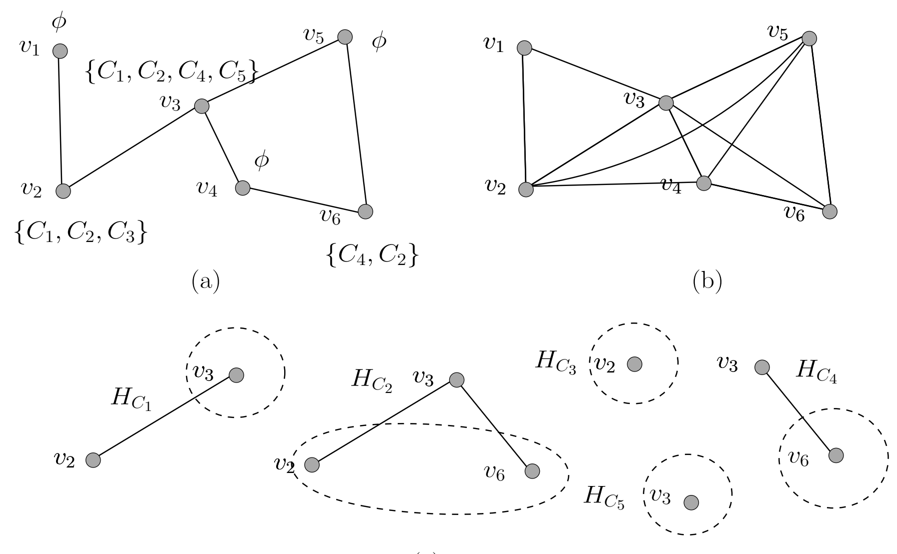
Ko and Rubenstein [2] extended their basic approach to this problem but had no guarantee on either the approximation ratio or the running time. We give the first centralized polynomial time 3-approximation algorithm (Algorithm 3) for the problem. Later, in Theorem 2.2.12, we will prove that this is the best possible approximation one can expect, assuming $P \neq NP$.

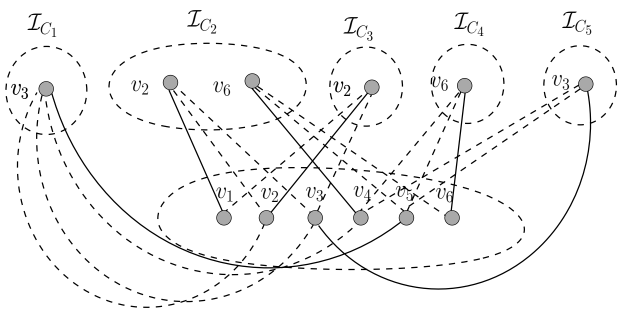
We again use the threshold graph technique. The optimal distance δ has to be the distance between one of the $O(n^2)$ pairs of vertices. Hence, it has only polynomial number of possible values and we can assume that the value of δ is known (trying out all possible values of δ will only add a polynomial factor). Assuming δ is known, we construct the threshold graph G_δ. We now square the graph G_δ to obtain G_δ^2, i.e., add an edge between two vertices $u, v \in V$ if they are at a distance at most two in G_δ. Consider a color r and let $H_r \subseteq G_\delta^2$ be the induced subgraph on vertices that need color r (among possibly other colors). Let $\mathcal{I}_r$ be a maximal independent set in the subgraph H_r. The following is a key observation about an optimal solution.

Observation 2.2.9 *For every vertex $v \in \mathcal{I}_r$, the optimal solution must assign a unique copy of r in the neighborhood of v in G_δ.* (†)

Indeed, in G_δ the neighborhoods corresponding to vertices in $\mathcal{I}_r$ must be mutually disjoint. If the neighborhoods corresponding to vertices $u, v \in \mathcal{I}_r$ intersect, there must exist an edge between u, v in G_δ^2, which is impossible as $\mathcal{I}_r$ forms an independent set in this graph. Since every vertex in $\mathcal{I}_r$ must be satisfied by some copy of r in its neighborhood in G_δ our observation holds. If for every vertex $v \in \mathcal{I}_r$ $d_r(v) \leq \delta$, then every vertex $u \in H_r$ has $d_r(u) \leq 3 \times \delta$. Thus, to find a 3-approximation we focus on satisfying vertices of such independent sets $\mathcal{I}_r$, for each color $r \in \mathcal{C}$. We cast this as a b-matching problem [121] on the graph $B = (X, Y)$ - where X is the union of independent sets $\mathcal{I}_r$, $\forall r \in \mathcal{C}$ (i.e., if a vertex belongs to s independent sets of the form I_r, we add s copies of the vertex to X) and Y is a copy of V with $b-$matching bounds s_v on each vertex $v \in V$. We add an edge across the partitions if its end points are at distance at most δ from each other. From observation (†), there must exist a b-matching that saturates all the vertices of X. The following example explains this construction.

Example 2.2.10 *We refer to the Figure 2.2 for various steps in the construction: (a) guess δ and construct the threshold graph, G_δ; the required color set for each vertex is shown next to it; (b) construction of G_δ^2; (c) for each color C_i, H_{C_i} is the subgraph of G_δ^2 over vertices that require C_i; the "dashed" circles represent maximal independent set $\mathcal{I}_{C_i}$ in each H_{C_i}; (d) bipartite graph construction: on one side we have copies of vertices in $\mathcal{I}_{C_i}$ and on the other side we have all the vertices in G_δ; we add edges ("dashed lines") between copies of two vertices, if they had an edge in the threshold graph G_δ; we then compute a b-matching on this graph; the solid lines represent a valid b-matching (where each vertex has a capacity of 1); (e) using the b-matching, we compute the final color assignments for each vertex.*





Vertex	**Color**
v_1	C_2
v_2	C_3
v_3	C_5
v_4	C_2
v_5	C_1
v_6	C_4

(e)

Figure 2.2: Illustrative example for the SRR algorithm.

Algorithm 3 A 3-approximation algorithm for SRR

```
1: let $\mathcal{D}$ be the list of possible $\delta$ values, i.e., the list of pairwise distances between
   the vertices of $G$, arranged in the non-decreasing order.
2: for all $\delta \in \mathcal{D}$ do
3:     for all colors $c$ do
4:         let $H_c$ be the subgraph of $G_\delta^2$ induced by the set of vertices that require
           color $c$.
5:         compute $\mathcal{I}_c$, which is a maximal independent set of $H_c$.
6:     end for
7:     let $X$ denote the set of vertices formed in the following way: if a vertex $v$ is
       contained in the independent set $\mathcal{I}_c$, corresponding to $\ell$ distinct colors $c$, then
       $\ell$ copies of $v$ are added to $X$. Let $Y$ be a copy of set of vertices in $V$ with
       non-zero storage capacities.
8:     construct the bipartite graph $B = (X, Y)$ : add an edge between $x \in X$ and
       $y \in Y$, if the nodes they represent are at distance at most $\delta$.
9:     compute a maximum $b$-matching in $B$ with bounds : 1 on vertices of $X$ and
       respective storage capacities on the nodes of $Y$.
10:    for every node $v \in Y$ having $S_v \subseteq X$ as the matched subset of nodes, assign
       the list of colors $L_v$ of nodes of $S_v$ to $v$. Call this coloring $\phi$.
11:    if the requirement of every vertex of $G$ is satisfied within a distance of $3\delta$
       then
12:        return $\phi$ (and exit the function)
13:    end if
14: end for
```

Theorem 2.2.11 *Algorithm 3 is a 3-approximation for the subset resource replication problem.*

Proof: Assuming δ is the optimum distance, we start by proving that the maximum b-matching, found in Step 9 of Algorithm 3, completely saturates X. It is sufficient to show that there exists of b-matching which saturates X because this in turn implies the maximum b-matching also does so. In an optimal list coloring, i.e., an assignment of colors to nodes that satisfies every vertex within distance δ, let L_v^{opt} denote the list of colors placed on $v \in V$. For feasibility, $|L_v^{opt}| \leq s_v$, where s_v is the storage capacity of v. For a color i and a vertex $v \in \mathcal{I}_i$, we denote the corresponding copy of v in X by v_i. We note that, for every v that requires a color i, there exists a vertex $u \in Y$ that is within a distance δ of v and has i in its list of colors L_u^{opt}. We now claim that the following edge set forms a b-matching that saturates X. The edge set, denoted by bM, consists one edge for each $v_i \in X$, namely $\overline{v_i u}$, where u is some vertex within distance δ of v_i such that $i \in L_u^{opt}$. We only have to show that bM is a feasible b-matching because it saturates X by its definition.

In order to prove that bM is a feasible b-matching, we show that the number of edges incident on each vertex is within the allocated bounds, s_v, for $v \in Y$ and 1 for $v_i \in X$. The latter bounds are, by definition, satisfied. To prove that the bounds s_v are not violated, we observe that no two vertices of X with same color index i, say v_i and w_i, are matched to the same vertex $u \in Y$ with respect to bM. Indeed, this would imply that v and w are adjacent in G_δ^2, which is a contradiction to the fact that they belong to a maximal independent set (in some induced subgraph of G_δ^2). Thus, the number of edges of bM incident on u is at most $|L_u^{opt}| \leq s_u$. Hence, bM is a valid b-matching which saturates all the vertices of X.

To finish the proof, we now show that every node v requiring a color i finds a node hosting i at distance at most 3δ. Indeed, there exists some $u_i \in X$, such that u is a neighbor of v in H_i (note that the distance between such u and v is at most 2δ). Now, if $\overline{u_i w} \in bM$, w is the vertex hosting i at distance at most 3δ. Hence, Algorithm 3 is a 3-approximation algorithm for the subset resource replication problem. ∎

2.2.3 Hardness results for resource replication problems

We now prove some lower bounds on the above problems. The following theorem shows that Algorithm 3 provides the best possible guarantee for the SRR problem, while there is a small gap between the algorithm and the lower bound proven for the BRR problem.

Theorem 2.2.12 *Assuming $P \neq NP$, there is no polynomial time algorithm that guarantees an approximation ratio better than,* (1) $2 - \epsilon$ *for the basic resource replication problem, and* (2) $3 - \epsilon$ *for the subset resource replication problem.*

Proof: (1) The following problem is called the *domatic partition problem* and was shown to be NP-complete in [120]: given a simple undirected graph $G = (V, E)$ and an integer k, does there exist a partition of V into k disjoint subsets $V_i : i \in [1, k]$ such that each V_i is a dominating set of G. We reduce any instance $(G = (V, E), k)$ of the domatic partition problem into an instance of BRR $(V, \mathcal{C} = \{C_1, C_2 \dots C_k\}, d : V \times V \to \{0, 1, 2\})$ in the following way. The function d is defined as follows, for any $u \neq v$,

$$d(\overline{uv}) = \begin{cases} 1 & \text{if } \overline{uv} \in E \\ \\ 2 & \text{otherwise} \end{cases}$$

It is easy to check that d is a metric. We note that, for the above instance of BRR, there are only two possible values for optimal distance, namely 1 or 2 (assuming a non-trivial instance with non-zero optimal distance). We claim that (G, k) is a *yes* instance of the domatic partition problem if and only if the optimum distance for the BRR instance is 1. Indeed, if V can be partitioned into k dominating sets, we assign colors to nodes in V such that the members of each set get a unique color. Every vertex, by the definition of dominating sets, must be adjacent to some vertex in each of the other dominating sets. Hence, each vertex will find all the k colors within distance 1. On the other hand, if every vertex finds a color within distance 1, each color group forms a dominating set, hence (G, k) will be a *yes* instance for the domatic partition problem. Thus, if (G, k) is a *yes* instance the optimal distance value is 1 and if it is a *no* instance the optimal distance is 2. Thus, if there exists an algorithm $\mathcal{A}$ with polynomial running time such that it approximates BRR within a factor $2-\epsilon$, it will be able to differentiate *yes* and *no* instances of domatic partition problem, in polynomial time. But this would imply $P = NP$.

(2) The following problem is NP-complete [122]: given a bipartite graph $B = (X, Y)$ partition Y into k subsets such that each subset dominates X. This problem is called, the *one-sided domatic partition* (ODP) problem. Now, given an instance of ODP, (B, k), we reduce it to an instance of the SRR, $\mathcal{I}' = (V, \mathcal{C} = \{C_1, C_2 \ldots C_k\}, \{\mathcal{C}_v \subseteq \mathcal{C} : v \in V\}, \{s_v : v \in V\}, d : V \times V \rightarrow \{0, 1, 2, 3\})$ in the following way. Set $V = X \cup Y$. All vertices of X have 0 capacity $(s_v = 0 : v \in X)$ and all vertices of Y have 1 capacity $(s_v = 1 : v \in Y)$. All vertices of X require every color in $[1, k]$ $(\mathcal{C}_v = \mathcal{C} : v \in X)$ and all vertices of Y require no color $(\mathcal{C}_v = \phi : v \in X)$.

The distance metric is

$$d(\overline{uv}) = \begin{cases} 1 & \text{if } \overline{uv} \in E(B) \\ 2 & \text{if } u, v \in X \text{ or } u, v \in Y \\ 3 & \text{otherwise} \end{cases}$$

Following the same argument as in part (1), it is easy to show (B, k) is a *yes* instance of ODP if and only if $\mathcal{I}'$ has an optimum distance 1. We observe that the above instance only takes values 1 or 3. Hence, if we could solve the SRR problem within an approximation guarantee of $3 - \epsilon$, ODP could be solved in polynomial time, thereby implying $P = NP$. ■

2.3 Dealing with outliers: Robust resource replication

The objective of minimizing the maximum distance over all vertices may result in a much larger distance if there are few distant "outliers". Even a good approximation algorithm, in this case, will raise δ to a very high value and many nodes could get a bad solution. It is therefore natural to study outlier versions of such problems. In such a model, the objective remains the same but we are allowed to ignore a few far away vertices (the outliers).

Example 2.3.1 *Figure 2.3 shows a simple example where "ignoring" a few far away vertices improves the quality of solution significantly. For $k = 4$, the non-*

outlier version has a cost of 100*, while if we ignore the vertices in the "dotted" circle, the solution has a cost of* 1*.*

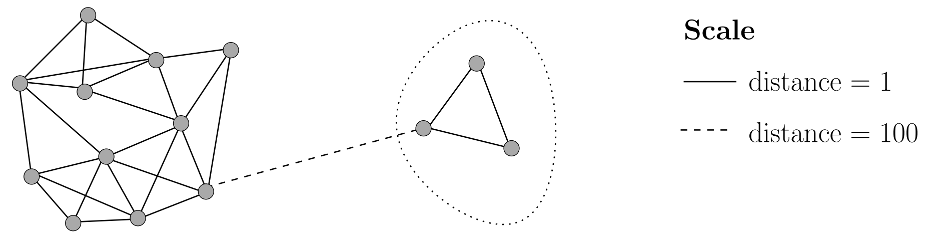


Figure 2.3: A case for the outlier version.

Several well known problems have been studied under the "outlier" model like outlier versions of k-center problem [49] (called *robust k-centers*), scheduling with outliers [62, 123, 124], outlier versions of facility location type problems [49, 125].

2.3.1 Robust basic resource replication

In this section, we initiate the problem of *robust basic resource replication* (RBRR) or the resource replication problem with outliers. In the RBRR problem the input is the same as the BRR problem except that we now have a lower bound M, which is the number of vertices that must be satisfied. Formally,

Definition 2.3.2 (The robust basic replication (RBRR) problem) *In an instance, $\mathcal{I} = (V, \mathcal{C}, M, d)$, we are given a set of vertices $V = \{v_1, v_2, \ldots v_n\}$, a metric $d : V \times V \to \mathbb{R}^+ \cup \{0\}$, a set of colors $\mathcal{C} = \{C_1, C_2 \ldots C_k\}$, an integer $M \in \mathbb{N}$. The goal is to find an optimal mapping $\phi : V \to \mathcal{C}$ with objective: $\underset{\substack{\phi \\ S \subseteq V \\ |S| \geq M}}{\mathsf{Min}} \quad \underset{\substack{v \in S \\ C_r \in \mathcal{C}}}{\mathsf{Max}} \quad d_r(v)$, where d_r is defined as the distance from v to a closest node u such that $\phi(u) = C_r$.*

We show that a simple extension to Algorithm 1 gives a 3-approximation for the RBRR. Algorithm 4 describes the procedure. Again, we begin by guessing the optimal value δ and construct the threshold graph G_δ. We note that there should be at least M vertices with degree at least $k-1$ for δ to be a feasible solution distance, because at least M vertices should get $k-1$ colors from their neighborhood to be satisfied. We then construct an independent set in G_δ^2 by adding only these "high" degree vertices as long as possible. Finally, for each vertex v in the independent set, we pick $k-1$ of its neighbors in G_δ and assign k colors one to each of the k vertices (v and its $k-1$ neighbors).

Algorithm 4 A 3-approximation algorithm for RBRR

```
1: let $\mathcal{D}$ be the list of possible $\delta$ values, i.e., the list of pairwise distances between
   the vertices of $G$, arranged in the non-decreasing order.
2: for all $\delta \in \mathcal{D}$ do
3:   construct $G_\delta$ and $G_\delta^2$, and mark vertices that have degree $\geq k-1$ in $G_\delta$.
4:   construct an independent set $\mathcal{I}$ of $G_\delta^2$ by adding marked vertices as long as
     possible, i.e., maximal with respect to marked vertices.
5:   for all $v \in \mathcal{I}$ do
6:     choose $k-1$ vertices from neighborhood of $v$ in $G_\delta$ and color these $k$ vertices
       with $k$ colors arbitrarily.
7:   end for
8:   color all uncolored vertices arbitrarily.
9:   if the number of vertices that are satisfied with in a distance of $3\delta$ is at least
     $M$ then
10:    return the current assignment and exit.
11:  end if
12: end for
```

Theorem 2.3.3 *Algorithm 4 gives a 3-approximation for the RBRR problem.*

Proof: Let δ be the optimum distance, such that at least M vertices have all the k colors within distance δ. We prove that Algorithm 4 satisfies at least M vertices within distance 3δ. We claim that every vertex of degree at least $k-1$ (in G_δ) has all the k colors within a distance 3δ. Indeed, if a vertex u has degree $k-1$ then either it belongs to $\mathcal{I}$ or has a node in $\mathcal{I}$ at distance at most 2δ. Since each $v \in \mathcal{I}$ has all colors within δ, every such vertex u is completely satisfied within 3δ. Clearly every vertex satisfied by the optimal algorithm must have degree $k-1$ and therefore there are at least M nodes of degree $k-1$. Hence, Algorithm 4 will satisfy at least M nodes within 3δ. ■

2.3.2 Budgeted robust basic resource replication

We now consider a more interesting generalization of the RBRR problem called the *K-robust basic resource replication* (K-RBRR) problem. In this problem, we only allow K copies of each resource, while the rest of input and output structure remains the same as RBRR. This problem is a natural generalization of the robust K-center problem, latter having $k = 1$ resource. The robust K-center problem is the outlier version of K-center problem and was studied, along with several other outlier variants of facility location type problems by Charikar et al. [49]. One variant of particular interest to our work is the robust K-supplier problem, for which Charikar et al. [49] give a 3 -approximation algorithm. The robust K-supplier is the outlier variant of K-supplier problem. In the K-supplier problem, we have a set of suppliers and a set of clients, embedded in a metric. The goal is to choose K suppliers which can hold a resource (there is only one resource here) such that the maximum "client to nearest resource distance" is minimized over all clients. In the robust K-supplier problem, we have the same objective but we must satisfy at least M clients, instead of all. We use the 3-approximation algorithm of [49] as a sub-routine and obtain a 5-approximation algorithm for K-RBRR problem. For the sake of completeness, we briefly describe the algorithm from [49] here.

For a given value δ, the algorithm of [49] proceeds in the following way.

- For each supplier v, construct G_v as the set of clients within distance δ and E_v as the set of clients within distance 3δ of v.
- Repeat the following steps K times:
  - Greedily pick a supplier v as a center whose set G_v covers most number of yet uncovered clients. (†)
  - Mark all the clients in E_v as covered.
- If at least M vertices are satisfied return the centers, or else return *no*.

For a proof on why this algorithm guarantees a 3-approximation, we refer the reader to [49]. We make a small modification to the above algorithm before using it as a sub-routine. In the step(†), if there are no more clients to be covered we can stop. Doing this will clearly not affect the performance or feasibility of the algorithm, but will make sure that there is at least one, so far uncovered, client that is covered on picking v as a center. We pick one such newly covered client arbitrarily and label it

$U(v)$. Note that this process assigns a distinct client to each supplier. Algorithm 5 gives a 5-approximation for the K-RBRR problem. We make the following claims about Algorithm 5.

Claim 2.3.4 *If δ is the optimum distance for an instance of K-RBRR, it is a feasible distance for the K-supplier instance in step 4 of Algorithm 5.*

Proof: Consider an optimal coloring of the graph which gives distance δ for the K-RBRR instance. Now, there are at least M vertices of degree at least $k-1$, which have a particular color C_1 within distance δ (in fact, these vertices have all the colors within δ). Now, just pick those nodes colored with color C_1 as centers. This implies M vertices of V_c (the set of clients in Algorithm 5) are satisfied, as all vertices with at least $k-1$ degree in G_δ are represented in V_c. Therefore δ is a feasible distance for the K-supplier instance constructed in step 4 of Algorithm 5. ■

Claim 2.3.5 *The set $\mathcal{I}$, formed in the Step 11 of Algorithm 5, is an independent set in G_δ^2.*

Proof: We prove that any two elements $U(v), U(w)$ are at distance strictly greater than 2δ from each other. Let us assume this is not the case. Let v be chosen as a center before w. Since the distance between v and $U(v)$ is $\leq \delta$ and distance between $U(v)$ and $U(w)$ is $\leq 2\delta$, the distance between v and $U(w)$ must be $\leq 3\delta$. But this implies $U(w)$ would be covered when v was picked (as $U(w) \in E_v$, as described in the robust K-supplier algorithm), a contradiction to the fact that $U(w)$ is an uncovered vertex when w was picked. ■

Algorithm 5 A 5-approximation algorithm for K-RBRR

```
1: let $\mathcal{D}$ be the list of possible $\delta$ values, i.e., the list of pairwise distances between
   the vertices of $G$, arranged in the non-decreasing order.
2: for all $\delta \in \mathcal{D}$ do
3:     construct $G_\delta$ and mark the subset, $V_c$, of "high" degree ($\geq k-1$) nodes.
4:     with $V_c$ as the set of clients, $V_s = V$ as the set of suppliers, distance between
       copies remaining the same as the original vertices, we solve the robust $K$-
       supplier problem [49] with $\delta$ as the input distance.
5:     if $\delta$ is infeasible for the above robust $K$-supplier instance then
6:         Claim 2.3.4 implies that $\delta$ is not the correct guess for optimal solution.
7:         continue to next $\delta$ value.
8:     else
9:         let $S \subseteq V_s$ be the set of centers returned.
10:    end if
11:    let $\mathcal{I} = \{U(v) : v \in S\}$. By Claim 2.3.5, $\mathcal{I}$ is an independent set in $G_\delta^2$.
       Further, each member of $\mathcal{I}$ has degree $\geq k-1$ in $G_\delta$ (because $\mathcal{I} \subseteq V_c$ and
       each $v \in V_c$ is of degree $\geq k-1$).
12:    for $v \in \mathcal{I}$ do
13:        pick $k-1$ neighbors of $v$ in $G_\delta$. Assign each of these vertices along with $v$,
           one color each of the $k$ colors.
14:    end for
15:    if the number of vertices satisfied within a distance $5\delta$ is at least $M$ then
16:        return the current assignment and exit.
17:    end if
18: end for
```

Theorem 2.3.6 *Algorithm 5 is a 5-approximation for the K-RBRR.*

Proof: Claim 2.3.4 guarantees S is valid and Claim 2.3.5 guarantees there is no clash during the coloring phase (Step 12) of Algorithm 5. Hence, Algorithm 5 generates a valid coloring. We now prove that at least M vertices get all k colors within 5δ distance. We note that S has the property that, at least M vertices are at distance at most 3δ from S (i.e., each of the M vertices has a vertex of S at distance at most 3δ). Since $\mathcal{I}$ was obtained by shifting the centers of S by at most δ, at least M vertices are at distance 4δ from $\mathcal{I}$. But each element of $\mathcal{I}$ has all k colors within δ, hence at least M vertices have all k colors within distance 5δ. ■

2.3.3 Robust subset resource replication

Let us now consider the *robust subset resource replication* (RSRR) problem. In this problem, we are provided with the input for the SRR problem along with a lower bound M on the number of vertices that must be satisfied with their requirement. The objective function is: $\underset{\substack{\phi \\ S\subset V \\ |S|\geq M}}{\mathsf{Min}} \quad \underset{\substack{v\in S \\ r\in\mathcal{C}_v}}{\mathsf{Max}} \quad d_r(v)$

Given that the outlier version of BRRand its extension with bound on each color has simple constant factor approximation algorithms, it is a natural question to ask whether similar bounds can be obtained for RSRR. But, somewhat surprisingly, we show not only there does not exist any constant factor approximation algorithm for RSRR, but in fact, assuming $P \neq NP$, there is no polynomial time algorithm that provides any nontrivial approximation guarantee. In Theorem 2.3.7, we prove that deciding if a given instance of RSRR is feasible, is NP-hard. We give a polynomial time reduction of the maximum k-clique problem to the problem of deciding the feasibility of RSRR.

Theorem 2.3.7 *Assuming $P \neq NP$, there is no polynomial time algorithm which*

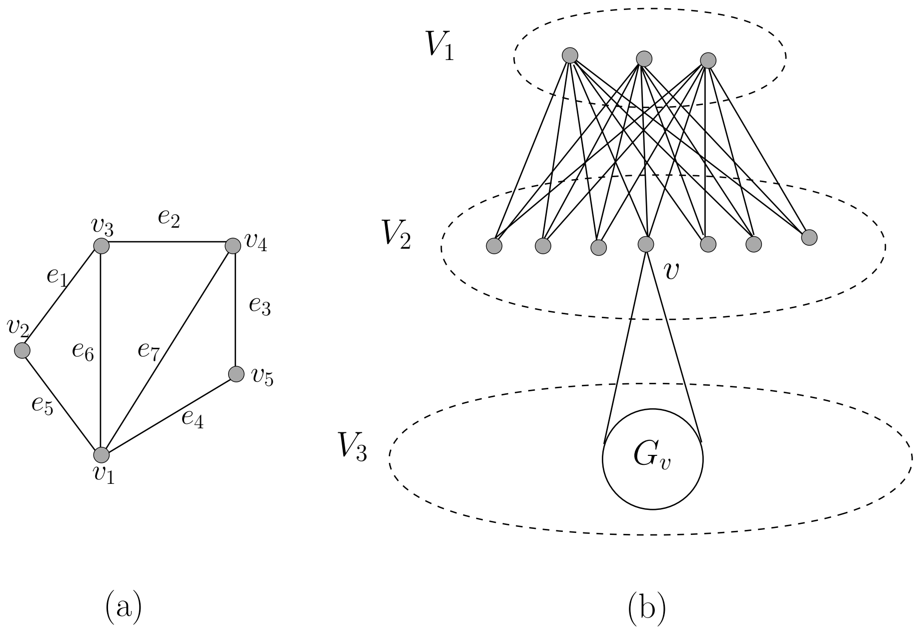


Figure 2.4: Reduction of maximum k-clique instance to an RSRR instance: (a) given instance of the maximum k-clique problem (for this example, $k = 3$); (b) a RSRR instance: the vertex set comprises of 3 parts: V_1 with $k = 3$ vertices, V_2 with vertices corresponding to the edges of G; each $v \in V_2$ has a m^2 clique G_v associated with it as shown; these gadgets form the set V_3; each edge represents distance 1 and the remaining distances are computed using shortest path metric.

gives a positive approximation ratio for robust subset resource replication problem, even in the case where all the storage capacities are unit.

Proof: In the decision version of the maximum k-clique problem, we have an instance of the form $\mathcal{I} = (G, k)$ and the goal is to decide if there is a complete subgraph (clique) of G with k vertices. We assume that $|V| = n$ and $|E| = m$

Reduction. Given an instance of maximum k-clique problem, $\mathcal{I} = (G = (V, E), k)$, we construct an instance of RSRR, $\mathcal{I}' = (G', M, \mathcal{C}, \{\mathcal{C}_v : \forall v \in G'\})$ as follows. First, color the vertices in V with distinct colors $c_1, c_2, \ldots, c_n$ arbitrarily. The vertex set of G' has 3 parts, namely, V_1, V_2, and V_3. V_1 has k vertices and V_2 has m vertices corresponding to the edges of G. The distance between any two vertices $u \in V_1, v \in V_2$ is 1. Each vertex $v \in V_2$ has a set of m^2 vertices, G_v, associated with itself. The distance between any vertex pair of $v \cup G_v$ is 1. Rest of the distances are computed using the shortest path metric. The set $\{\mathcal{C}_v : \forall v \in G'\}$ is specified in the following way - each vertex $u \in V_1$ requires 0 colors and hence is trivially satisfied. Each vertex $v \in V_2$ requires colors $\{a_v, c_i, c_j\}$ where a_v is a color associated uniquely with vertex v and c_i, c_j are the colors of the end points of the edge in G associated with v. Each vertex $w \in G_v$ requires colors $\{a_v, b_v^i : i \in [1 : m^2]\}$. Each one of $a_v, b_v^i : v \in V_2, i \in [1, m^2]$ is a distinct color. Let $L = \binom{k}{2}$. Set $M = m^3 + L + k$, the lower bound on the number of vertices that must be satisfied. Figure 2.4 shows the construction for a simple instance.

Claim: I is a *yes* instance of maximum k-clique problem if and only if I' is a feasible solution of RSRR problem. In other words, we prove that the feasibility question of RSRR problem is NP-hard. This would imply that there is no approximation algorithm for this problem.

Proof of the Claim. Let I be an *yes* instance of the maximum k-clique problem and let $H = \{v_1, v_2 \ldots v_k\}$ be the k vertices that induce a clique, that is $L = \binom{k}{2}$ edges in G. We present a feasible coloring for I' as follows -

- The k vertices of V_1 are colored with the k colors of H
- Each vertex $v \in V_2$ is colored with its associated color a_v.
- For each vertex $v \in V_2$, its m^2 associated vertices G_v are colored with m^2 colors of type b_v^i.

It is straightforward to check that the above coloring satisfies $M = m^3 + k + L$ vertices - all the vertices of V_3 are satisfied, all the vertices of V_1 are satisfied and at least L vertices of V_2 are satisfied. Now, we consider the other direction. Let there be a coloring of vertices of G' which certifies that I' is a feasible instance. We first observe that, all the m^3 colors of type b_v^i and the m colors of type a_v must be used - otherwise, there will be at least m^2 vertices out of $m^3 + m + k$ vertices which go unsatisfied and hence the bound M is not met. Since, we are only interested in the feasibility question, we can assume that m^2 vertices of G_v are colored with m^2 colors of type b_v^i and the m vertices $v \in V_2$ are colored with color a_v. Now at least $L = \binom{k}{2}$ vertices of V_2 must be satisfied and the k vertices of V_1 must be colored with k colors from $\{c_1, c_2 \ldots c_n\}$ - say $\{c_1, c_2 \ldots c_k\}$. We observe that the union of colors required by the $L = \binom{k}{2}$ vertices, apart from their associated colors, must be $\{c_1, c_2 \ldots c_k\}$. Hence, the L edges in G corresponding to these L vertices in V_2 must be completely incident on the vertices in V corresponding to these k colors. This implies the existence of k vertices in G that induce $\binom{k}{2}$ edges, that is an existence of a k clique. Hence the theorem. ■

Remark. If we insist only on a lower bound on the number of satisfied node-resource pairs as opposed to the number of completely satisfied nodes, the problem becomes significantly easier. We just need to create a copy of a vertex for each color that it desires and then run the robust version of BRR. The main hardness stems from the fact that in order for a vertex to be satisfied it requires all the desired colors.

2.4 Serving fairly: Capacitated basic resource replication

Another desired quality of an assignment scheme in client-server type problems is load balancing [115, 126, 127]. In this setting, we are not allowed to "overload" a server by assigning more than a bounded number of clients. Bar-Ilan, Kortsarz and Peleg [127], Khuller and Sussman [115] study the load balancing version of the k-center problem which is called the *capacitated k-center problem.* Khuller and Sussman [115] provide the current best approximation ratio of 5 for this problem. We initiate the study of basic resource replication problem in the load balancing setting. We call it the *capacitated basic resource replication problem* (CBRR). In this problem, the input instance is defined as $\mathcal{I} = (V, \mathcal{C} = \{C_1, C_2 \ldots C_k\}, d, L)$ and the goal is the same as the basic resource replication problem with an additional restriction that a vertex with a certain color is not allowed to serve more than L

other vertices (including itself). We give a 4-approximation algorithm (Algorithm 6) for this problem, provided $L \geq 2k - 1$. First we prove that the problem is infeasible if $L < k$.

Preposition 2.4.1 *Given an instance of CBRR, $\mathcal{I} = (V, \mathcal{C}, d, L)$, with $|\mathcal{C}| = k$, the following statements are true:*

1. *If $L \leq k - 1$, then $\mathcal{I}$ is infeasible.*

2. *If $L = k$, then $\mathcal{I}$ is feasible with value $\delta \iff$ the number of vertices in each component of G_δ (threshold graph on V) is a multiple of k.*

Proof: *1.* Let if possible, there exist a feasible solution when $L < k$. We construct the following directed graph D, on the vertex set V. For every pair of vertices $u, v \in V \times V$ (note that u and v need not be distinct), we add a directed edge $\overline{uv}$ them if v serves u in the feasible solution. Since each vertex requires all k colors, the out-degree of each vertex is at least k. Also we note that, since each vertex can serve at most L vertices the in-degree of each vertex is at most L . Given that the problem is feasible, we have:

$$\begin{aligned} k \times |V| &\leq \sum_{v \in D} \text{out-degree of v} \\ &= \sum_{v \in D} \text{in-degree of v} \\ &\leq L \times |V| \end{aligned}$$

This implies $L \geq k$, a contradiction.

2. Fixing a component C of G_δ, we will first prove that for the instance to be feasible, $|C|$ must be a multiple of k. Firstly, we note that the vertices of C can only be satisfied by other vertices of C. Consider the feasible assignment of colors C_i $i \in [1, k]$ to C. Group all the vertices that are given a color C_i into a class B_i. Let B_s be the smallest cardinality color class. Construct the directed graph on C as described in the part(a). Now, every vertex in C (including those in B_s) must have an edge directed into B_s (because every vertex requires the color C_s). Each vertex of B_s has an in degree $\leq L = k$. Hence, we have $|C| \leq k \times |B_s|$. This implies all the color classes have the same cardinality, which in turn implies $|C|$ is a multiple of k. ■

We now give a 4-approximation algorithm for the CBRR problem where $L \geq 2k - 1$. We refer to Algorithm 6 for pseudocode. The algorithm starts by guessing the optimal δ and constructs the threshold graph G_δ. Let $\mathcal{I}$ be some maximal independent set of G_δ^2. We divide all the vertices into three levels - *level* 0, *level* 1 and *level* 2. All the elements in $\mathcal{I}$ are at *level* 0. All vertices not in $\mathcal{I}$ but adjacent (with respect to G_δ) to some element in $\mathcal{I}$ are at *level* 1. Finally all the vertices not in *level* 0 or *level* 1 are in *level* 2. For each element v at *level* 0, its empire $Empire(v)$ consists of itself along with all the adjacent(with respect to G_δ) *level* 1 vertices. Since $\mathcal{I}$ is independent in G_δ^2, all the empires defined so far are mutually disjoint. Finally, all the *level* 2 vertices are adjacent to at least one *level* 1 vertex. For each *level* 2 vertex, we pick one such *level* 1 vertex arbitrarily and assign the former to the same empire as the latter. Thus we have assigned every vertex to exactly one empire.

Algorithm 6 A 4-approximation for CBRR

```
1: let $\mathcal{D}$ be the list of possible $\delta$ values, i.e., the list of pairwise distances between
   the vertices of $G$, arranged in the non-decreasing order.
2: **for all** $\delta \in \mathcal{D}$ **do**
3:   construct the graph $G_\delta$ and $G_\delta^2$.
4:   **if** minimum degree of $G_\delta < k - 1$ **then**
5:     **continue** onto the next $\delta$ value.
6:   **end if**
7:   let $\mathcal{I}$ be a maximal independent set in $G_\delta^2$.
8:   **for all** $v \in V$ **do**
9:     **if** $v \in \mathcal{I}$ **then**
10:      $Empire(v) = \{v\}$
11:    **end if**
12:    **if** $v \notin \mathcal{I}$ **then**
13:      **if** $v$ has a vertex $u \in \mathcal{I}$ at distance $\delta$. **then**
14:        Such a vertex is unique owing to the property that $\mathcal{I}$ is an independent
           set. Add $v$ to the empire of $u$, $Empire(u) = Empire(u) \cup \{v\}$.
15:      **else if** $v$ has a vertex in $\mathcal{I}$ at distance $2\delta$. **then**
16:        Pick one such vertex $u$ arbitrarily and add $v$ to the empire of $u$.
17:      **end if**
18:    **end if**
19:  **end for**
20:  **for all** $v \in \mathcal{I}$ **do**
21:    Each vertex $v$ has degree at least $k - 1$ in $G_\delta$. Hence, $|Empire(v)| \geq k$.
       Divide $Empire(v)$ into blocks, all of which have size exactly $k$ - except
       possibly the last one which has size at most $k$.
22:    Color each block of size exactly $k$ using $k$ colors, arbitrarily. The final block,
       whose size is at most $k$, has its color requirement satisfied from one such
       block. Since there is at least one block of size exactly $k$, such an assignment
       is valid.
23:  **end for**
24: **end for**
```

In the next step, we consider one empire at a time and split it into "blocks" of vertices. Every block consists of exactly k vertices, except the last block which might have less than k vertices. A key property of vertices in a block is the following - any two vertices are at a distance of at most 4δ from each other. We now color each block of size exactly k using all k colors (since the degree of each vertex is at least $k-1$ in G_δ, every empire has at least one block of size exactly k). A vertex in a block only serves other vertices in the same block, hence the load is not more than k currently on any vertex. The vertices of the final block (which might have $\leq k$ vertices) are now served by some block of size exactly size k. Thus the load on each vertex is at most $2k-1$.

Theorem 2.4.2 *Algorithm 6 is a 4-approximation algorithm for the problem of capacitated basic resource replication problem where the allowed load $L \geq 2k-1$.*

Proof: As mentioned in the discussion above, the key observation needed is that any two vertices in the same empire (of say a vertex v) are at distance at most 4δ from each other. Indeed, all the vertices in the empire of vertex v are at distance at most 2δ from v and hence at distance 4δ from each other. The only detail that needs to be verified is that the maximum load on any vertex is at most $2k-1$. A block of size exactly k satisfies the requirement of its own members along with at most one other block (of size $< k$). Hence the maximum load is $\leq 2k-1 \leq L$. ■

By using Preposition 2.4.1, we observe that Algorithm 6 is in fact a bicriteria approximation algorithm (for arbitrary load capacity) - it gives an approximation guarantee of 4 while exceeding the load by a factor of 2 at most.

We now show a simple 8-approximation algorithm for the CBRR problem, when the capacity $L = k$. We use the construction of [115] to obtain a maximal independent set, $\mathcal{I}$ on G_δ^2, which has the following useful property -

Property 2.4.3 *$\mathcal{I}$ can be represented as a rooted tree, $\mathcal{T}$, where any given vertex (apart from the root) has its parent (immediate ancestor) at distance $\leq 3\delta$.*

We also adopt the terminology of [115] and call each vertex in $\mathcal{I}$ a *monarch*, the rooted tree $\mathcal{T}$ a *monarch tree* and all the vertices assigned to it in a feasible solution its *empire*. Every monarch has all its neighbors in G_δ added to its empire. Every non-assigned vertex is at distance at most 2δ from some monarch (otherwise such a vertex can be added to $\mathcal{I}$) and we add the former to the latter's empire (breaking ties arbitrarily, if more than one such monarch exists).

Algorithm 7 8-approximation for load = k

```
1: Let $\mathcal{D}$ be the list of possible $\delta$ values, i.e., the list of pairwise distances between
   the vertices of $G$, arranged in the non-decreasing order.
2: for all $\delta \in \mathcal{D}$ do
3:    Construct $G_\delta$.
4:    if minimum degree of $G_\delta < k - 1$ then
5:       continue onto next $\delta$ value.
6:    end if
7:    Construct $G_\delta^2$.
8:    Construct the monarch tree $\mathcal{T}$. Let $r$ be its root.
9:    Call the recursive procedure, ProcessMonarch($r, \mathcal{T}$)
10: end for
```

Algorithm 8 Recursive procedure: ProcessMonarch$(r, \mathcal{T})$

```
1: Let $C$ be the set of *children* of $r$ in $\mathcal{T}$.
2: LeftOver $= \phi$
3: **for all** Child $c \in C$ **do**
4:     LeftOver $=$ LeftOver $\cup$ ProcessMonarch$(c, \mathcal{T})$.
5: **end for**
6: Divide the set empire$(r) \cup$ LeftOver, into blocks of size exactly $k$, with the
   possible exception of the final block, which is of size $< k$. In constructing such
   blocks, we give least preference to the vertices in the neighborhood of $r$ in $G_\delta$.
   Since the degree of $r$ in $G_\delta$ is at least $k$ (otherwise $\delta$ is infeasible), the final block
   $F$ is completely in the neighborhood of $r$ in $G_\delta$.
7: Color all the blocks of size exactly $k$ with $k$ colors.
8: **if** $F$ is uncolored **then**
9:     **return** $F$ as the set of "left over" vertices.
10: **else**
11:     **return** $\phi$.
12: **end if**
```

Theorem 2.4.4 *Algorithm 7 is an* 8*-approximation algorithm for* CBRR *with* $L = k$.

Proof: We prove the following two properties of Algorithm 7 which will imply the statement of the theorem:

1. Every vertex belongs to some "colored block", i.e., a block of size k which is colored using k colors. This will imply that every vertex has its requirement satisfied within the block.

2. For a given block, the maximum distance between any two vertices is at most 8δ. This will imply that Algorithm 7 is an 8-approximation.

From Preposition 2.4.1, we know that the number of vertices in a component is a multiple of k. This along with the fact that every block is of size k, implies that the LeftOver set must be empty when the Procedure 8 is called on the root r. For the second claim, lets consider a arbitrary block B which is colored when processing some monarch m. If B is completely contained in the empire of m, the maximum distance between any two vertices of B is 4δ. On the other hand, if B contains left over elements, we observe that these left over elements are from the empires of monarchs which are children of m in $\mathcal{T}$. Indeed, when we are creating blocks for a monarch, the left over vertices of its children are preferred and made into blocks first . Hence, each monarch has to deal with the left overs of its children alone. We also note that the only vertices passed on from a monarch to its parent monarch are the former's neighbors in G_δ. Hence, if u is an element in the set LeftOver of a monarch m, it must be at a distance $3\delta + \delta = 4\delta$ (since the monarch m', whose empire contains u, is a child of m and hence is at distance 3δ from it) from m. Thus, any two elements of the block are at a distance at most $4\delta + 4\delta = 8\delta$. ■

Chapter 3

The Container Selection Problem

3.1 Road map to the chapter

As mentioned in Chapter 1, the *container selection problem* is a special case of non-metric k-median, which is inapproximable unless we violate k significantly [21]. However, our problem still has sufficient geometric structure. This structure allows us to obtain near optimal algorithms that, in the case of continuous container selection, do not violate k, and in the discrete case violate k mildly.

In Section 3.2, we show that the *continuous container selection problem* admits a *PTAS*, for any fixed dimension d. Our analysis crucially relies on showing the existence of a near-optimal solution where every container point lies on one among a constant number of rays. Ensuring this structure costs us a $1 + \epsilon$ factor in the approximation ratio. The algorithm is itself then a dynamic program which optimally solves such a "restricted" container selection problem.

On the negative side, we show that the continuous container selection problem is NP-hard for $d \geq 3$. Interestingly, the flexibility of using container points in the continuous space is essential not just for our algorithm but for any approach: we show the discrete version is NP-hard to approximate to any factor when $d \geq 3$. The reduction is from a restricted version of planar vertex cover [128]. We also reduce the discrete container selection problem to the continuous version (not approximation preserving) which proves its NP-hardness when $d \geq 3$. All these hardness results form the content of Section 3.4.

On a positive note, in Section 3.3, we will discuss two different algorithms for the *discrete container selection problem*, both of which provide bi-approximation guarantees. The first algorithm (Section 3.3.1) is specialized to dimension two and is a $(1+\epsilon, 3)$-approximation. The main idea here is a partitioning of $\mathbb{R}_+^2$ into $O(\log n)$ "cells" where all points in a cell have roughly the same ℓ_1-norm and which allows a decoupling of "local assignments" within a single cell and "distant assignments" from one cell to another. This partitioning uses the definition of rays from the algorithm for the continuous problem. (Using a more standard partitioning yields $O(\log^2 n)$ cells which is too large for a polynomial-time algorithm.) The algorithm then uses enumeration to handle distant assignments and a dynamic-program for the local assignments. This decoupling is what necessitates the violation in the bound k.

The second algorithm for the discrete version (Section 3.3.2) works for any dimension d and yields a $(1+\epsilon, O(\frac{d}{\epsilon}\log dk))$-approximation. This is based on the natural linear programming relaxation used even for the non-metric k-median problem [21]. However, we obtain a sharper guarantee in the violation of k using the geometry specific to our setting. In particular, we show an LP-based reduction to hitting-set instances having VC-dimension $O(d)$. Then our algorithm just uses the well-known result of [129, 130] for such hitting-set instances. We note that a constant bi-approximation algorithm for $d = 2$ also follows from this approach, using a known $O(\frac{1}{\epsilon})$-size ϵ-net construction for "pseudo-disks" [131]. However the constant obtained here is much larger than our alternative direct approach.

Notation. For integers $a < b$, we use $[b] := \{1, 2, \cdots b\}$ and $[a, b] := \{a, a+1, \ldots, b\}$. A point $c(c_1, c_2, \ldots, c_d) \in \mathbb{R}^d$ *dominates* or *contains* another $p(x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ if, for all $i \in [d]$, $x_i \leq c_i$. By $p \prec c$, we mean c dominates p. Two points p_1 and p_2 are called *incomparable* if $p_2 \not\prec p_1$ and $p_1 \not\prec p_2$. The ℓ_1-norm of a point $c(c_1, c_2, \ldots, c_d)$ is denoted by $\|c\|$, i.e., $\|c\| = c_1 + c_2 \ldots + c_d$. For a subset of container points, S, we denote the total cost of assignment by $\mathsf{cost}(S)$. The cartesian product of two sets A and B is denoted by $A \times B$. Finally, we note that all the co-ordinates of points considered here are non-negative real numbers.

3.2 The continuous container selection problem

In this section, we describe a polynomial time approximation scheme for the continuous container selection problem. We start with a formal definition.

Definition 3.2.1 (continuous container selection) *In an instance of the problem, we are given a set of input points $\mathcal{C}$ in $\mathbb{R}^d$ and a budget k. The goal is to find a subset S of k container points in $\mathbb{R}^d$, such that the following cost is minimized:*

$$\mathop{\mathit{Min}}_{\substack{S \subseteq \mathbb{R}^d \\ |S| \leq k}} \sum_{p \in \mathcal{C}} \mathop{\mathit{Min}}_{\substack{c \in S \\ p \prec c}} \|c\|$$

We first describe the algorithm for $d = 2$ in Section 3.2.1 and subsequently, in Section 3.2.2, we extend this to general fixed dimension $d > 2$.

3.2.1 Two dimensions

We denote the set of input points by $\mathcal{C} = \{p_i(x_i, y_i) : i \in [n]\}$. Let S_{opt} denote some optimal set of k container points. We start with the following simple observation.

Observation 3.2.2 (potential container points) *For a given set of input points $\mathcal{C} = \{p_i(x_i, y_i) : i \in [n]\}$, let $X = \{x_i : i \in [n]\}$ and $Y = \{y_i : i \in [n]\}$. Then, $S_{opt} \subseteq X \times Y$. We call the set $X \times Y$ the set of potential container points and denote it by $\mathscr{F} = \{c_j(u_j, v_j) : j \in [m]\}$, where $m \leq n^2$.*

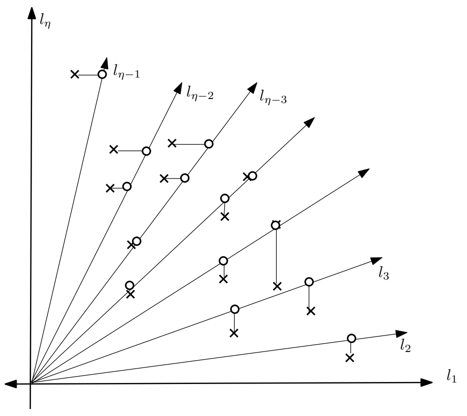
Algorithm outline. Given an instance of the problem, we transform it into an easier instance where all the chosen container points must lie on a certain family of rays. The number of rays in this family will be bounded by a constant that depends on ϵ, where $1 + \epsilon$ is the desired approximation ratio. Subsequently, we show that the restricted problem can be solved in polynomial time using a dynamic program.

Transformation. Fix a constant $\theta \approx \frac{\epsilon}{2} \in (0, \frac{\pi}{4}]$, such that $\eta = \frac{\pi}{2\theta}$ is an integer. Define the following lines $l_r \equiv y \cos{(r-1)\theta} - x \sin{(r-1)\theta} = 0$, for $r \in [\eta + 1]$. We define the following transformation of any point $c_j(u_j, v_j) \in \mathscr{F}$ to construct the set of potential container points $\mathscr{F}^T$. If c_j lies on some line l_r, for some $r \in [\eta]$, then $c_j^T = c_j$. Otherwise, c_j is contained in the region bounded by the lines l_r and l_{r+1}, for some $r \leq \eta$. Now define two points $c_j^u(u_j + \Delta u, v_j)$ and $c_j^v(u_j, v_j + \Delta v)$, such that

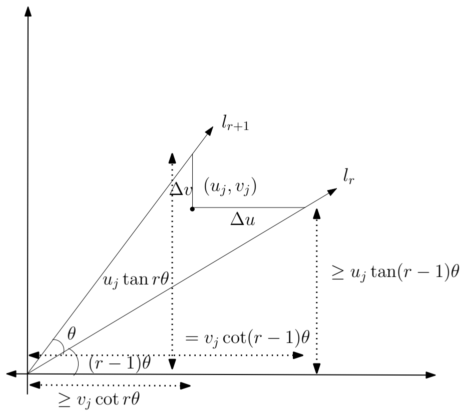
c_j^u is on l_r and c_j^v is on l_{r+1}. Now, the transformed point can be defined as follows:

$$c_j^T = \begin{cases} c_j^u, & \text{if } \Delta u \le \Delta v \\ c_j^v, & \text{otherwise} \end{cases}$$

Figure 3.1a illustrates this transformation.



(a) Transformation of containers



(b) Measurement of error

Figure 3.1: The continuous container selection problem.

We now show that under this transformation the optimal solution is preserved within an approximation factor of $(1 + \epsilon)$.

Lemma 3.2.3 *For instance $\mathcal{I} = (\mathcal{C}, k)$, let $S_{opt} = \{o_1, o_2, \ldots, o_k\}$ be an optimal solution. Further, let $S_{opt}^T = \{o_1^T, o_2^T, \ldots, o_k^T\} \subseteq \mathscr{F}^T$ be the set of transformed points corresponding to S_{opt}. Then, S_{opt}^T is a feasible solution to $\mathcal{I}$ and $\mathsf{cost}(S_{opt}^T) \le (1 + \epsilon)\mathsf{cost}(S_{opt})$.*

Proof: Recall that $\eta = \frac{\pi}{2\theta}$ and $\theta \approx \frac{\epsilon}{2}$. The feasibility of S_{opt}^T follows from the observation that if a point $p_i \in \mathcal{C}$ is dominated by a container $o_i \in S_{opt}$, it is also dominated by the point o_i^T. We now argue that $\mathsf{cost}(S_{opt}^T) \le (1 + \epsilon)\mathsf{cost}(S_{opt})$. It suffices to show that for every point $o_j = (u_j, v_j)$, $u_j^T + v_j^T \leq (1+\epsilon)(u_j + v_j)$, where $o_j^T = (u_j^T, v_j^T)$. The claim holds trivially in the case where o_j lies on a line l_r, for $r \in [1, 2, \ldots, \eta + 1]$. Hence, assume that o_j lies in the region bounded by the two lines l_r and l_{r+1}, where $r \in [1, 2, \ldots, \eta]$. Further, let $o_j^u = (u_j + \Delta u, v_j)$ and $o_j^v = (u_j, v_j + \Delta v)$, be the points on lines l_r and l_{r+1} respectively. By geometry (refer to the Figure 3.1b), we have the following equations:

$$\Delta u \leq v_j \left(\frac{\cos(r-1)\theta}{\sin(r-1)\theta} - \frac{\cos r\theta}{\sin r\theta} \right) = v_j \frac{\sin\theta}{\sin r\theta \sin(r-1)\theta} \tag{3.1}$$

$$\Delta y \leq u_j \left(\frac{\sin(r-1)\theta}{\cos r\theta} - \frac{\sin(r-1)\theta}{\cos(r-1)\theta} \right) = u_j \frac{\sin\theta}{\cos r\theta \cos(r-1)\theta} \tag{3.2}$$

Let $\Delta = \min(\Delta u, \Delta v)$. From Equations 3.1 and 3.2, we have,

$$(u_j + v_j)\sin\theta \geq \Delta(\sin r\theta \sin(r-1)\theta + \cos r\theta \cos(r-1)\theta) \;=\; \Delta\cos\theta.$$

$$\text{So } \Delta \leq (u_j + v_j)\tan\theta \leq (u_j + v_j)(2\theta) = (u_j + v_j)\epsilon. \tag{3.3}$$

Now, the claim follows from Equation 3.3 and the fact that $u_j^T + v_j^T = (u_j + v_j) + \Delta$.

■

In Section 3.2.3, we show that the following restricted problem can be solved in polynomial time, for any fixed dimensions, $d \geq 2$: given a set of input points $\mathcal{C}$, and a constant number of rays through the origin, choose k container points that lie on these rays, such that the total assignment cost of the input points is minimized. By Lemma 3.2.3, this implies a $(1+\epsilon)$ approximation for the original problem. We have the following theorem.

Theorem 3.2.4 *There is a PTAS for the 2D continuous container selection problem.*

3.2.2 Higher dimensions

We now consider the container selection problem in higher, but fixed, dimensions. Formally, an instance, $\mathcal{I} = (\mathcal{C}, k)$, of the d-dimensional container selection problem consists of a set of input points, $\mathcal{C} = \{p_i(x_1^i, x_2^i, \ldots, x_d^i) : i \in [n]\}$ and a budget k.

Potential container points. For each dimension $j \in [d]$, we define $X_j = \{x_j^i : i \in [n]\}$, as the set of j^{th} coordinates of all input points. An easy observation is that any container point chosen by any optimal solution must belong to $\mathscr{F} = X_1 \times X_2 \times \ldots \times X_d = \{c_i(u_1^i, u_2^i, \ldots, u_d^i) : i \in [m]\}$ where, $m \leq n^d$.

Algorithm outline. As in the two dimensional case, the main idea is a reduction to the following restricted problem. An instance is $\mathcal{I} = (\mathcal{C}, k, L_d)$ where $\mathcal{C}$ is a set of input points in $\mathbb{R}^d$, k is an integer and L_d is a family of rays in $\mathbb{R}_+^d$ with $|L_d| = O_d(1)$[1]. The goal is to choose k container points that lie on the rays in L_d, such that the total assignment cost of $\mathcal{C}$ is minimized.

Transformation. Fix a constant $\theta \approx \frac{\epsilon}{2} \in (0, \frac{\pi}{4}]$, such that $\eta = \frac{\pi}{2\theta}$ is an integer. In order to construct L_d, we use the recursive procedure described in Algorithm 9. Let $\bar{u}_i$ denote the i^{th} unit vector ($i \leq d$), i.e., $\bar{u}_i$ is a 0-1 vector with value 1 at the i^{th} coordinate and 0 elsewhere. Starting from the family L_2 of rays in two dimensions (using the transformation in Section 3.2.1), we add one dimension at a time and construct the corresponding families for higher dimensions. In the recursive step,

[1] $O_d(1)$ is a constant assuming that d is a constant

we start with the family L_{r-1} and observe that each of these rays will induce a 2-D plane in r-dimensions. Then, we use the two dimensional construction to handle the extra dimension. Observe that $|L_d| \leq (\pi/\theta)^d = O(1)$ for any fixed θ and d.

Algorithm 9 Construction of the family of lines in r-dimensions: L_r

```
1: let $\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_r$ be the unit vectors along the axis lines
2: if $r = 2$ then return equiangular rays in $\mathbb{R}^2_+$ from the Section 3.2.1 (see also
   Figure 3.1a)
3: construct the family $L_{r-1}$, recursively, with $r-1$ dimensions, $\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_{r-1}$
4: initiate: $L_r \leftarrow \emptyset$
5: for all $\ell \in L_{r-1}$ do
6:     let $\bar{\ell}$ be the unit vector along the line $\ell$
7:     consider the (two dimensional) plane $\Pi_\ell$ formed by the vectors $\bar{u}_r$ and $\bar{\ell}$
8:     let $Q_\ell$ be the family of rays obtained by applying the transformation of Sec-
       tion 3.2.1 to the plane $\Pi_\ell$
9:     $L_r \leftarrow L_r \cup Q_\ell$
10: end for
11: return $L_r$
```

Algorithm 10 describes a recursive procedure to transform a point $c(u_1, u_2, \ldots, u_d) \in \mathscr{F}$ to a point c^T that lies on some line in L_d. The idea is as follows: for any $r \geq 3$, first recursively transform the point $c_{r-1}(u_1, u_2, \ldots, u_{r-1}) \in \mathbb{R}^{r-1}$ into a point $c^T_{r-1}(u'_1, u'_2, \ldots, u'_{r-1})$ that lies on some line $\ell \in L_{r-1}$. Now, consider the point $c'_r(u'_1, u'_2, \ldots, u'_{r-1}, u_r)$, where u_r is the r^{th} coordinate of the original point c. The point c'_r lies on the 2D plane spanned by $\bar{\ell}$, the unit vector along the line ℓ, and $\bar{u}_r$. Using the 2D transformation we move c'_r to a point c^T_r that lies on some line in L_r.

Lemma 3.2.5 *For any $\theta = \frac{\epsilon}{2} \in (0, \frac{1}{2d-2}]$ and point $c(u_1, u_2, \ldots, u_d) \in \mathscr{F}$, applying Algorithm 10, we obtain $c^T = (u^T_1, u^T_2, \ldots, u^T_d)$ where $c \prec c^T$ and:*

$$\|c^T\| \leq (1 + 2(d-1)\epsilon)\|c\|.$$

Algorithm 10 The transformation of $c_r = (u_1, u_2, \dots u_r)$ onto L_r, $r \le d$

```
1: if $r = 2$ then use the 2D transformation from the Section 3.2.1 (see also Figure 3.1b)
2: $c_{r-1} \leftarrow (u_1, u_2, \dots, u_{r-1})$
3: recursively transform $c_{r-1}$ into a point on some line $\ell$ in $L_{r-1}$ and compute the transformed point $c^T_{r-1} = (u'_1, u'_2, \dots, u'_{r-1})$
4: $c'_r \leftarrow (u'_1, u'_2, \dots, u'_{r-1}, u_r)$, which lies on the plane $\Pi_\ell$ spanned by $\bar{u}_r$ and $\bar{\ell}$
5: let $Q_\ell$ denote the lines on plane $\Pi_\ell$ from Algorithm 9 step 8.
6: use the 2D transformation (Section 3.2.1) on plane $\Pi_\ell$ to move $c'_r$ onto a line in $Q_\ell$ and obtain $c^T_r = (u^T_1, u^T_2, \dots, u^T_{r-1}, u^T_r)$
7: **return** $c^T_r$
```

Proof: It is straightforward to see $c \prec c^T$. Using induction we will show that

$$\|c^T_r\| \le (1+\epsilon)^{r-1}\|c_r\|$$

The base case $r = 2$ follows from Lemma 3.2.3. Now consider $r \ge 3$ and assume the statement for $r-1$. In Algorithm 10, c^T_r is obtained by transforming the point c'_r in the 2D plane Π_ℓ. Note that c'_r has coordinates $\sqrt{(u'_1)^2 + (u'_2)^2 + \dots + (u'_{r-1})^2}$ and u_r in plane Π_ℓ. Hence, as shown in Lemma 3.2.3, we can obtain the following:

$$\begin{aligned} u^T_1 + u^T_2 + \dots + u^T_{r-1} + u^T_r &\le (1+\epsilon)(\sqrt{(u'_1)^2 + (u'_2)^2 + \dots + (u'_{r-1})^2} + u_r) \\ &\le (1+\epsilon)(u'_1 + u'_2 + \dots + u'_{r-1} + u_r) \qquad (3.4) \end{aligned}$$

By the inductive hypothesis, $u'_1 + u'_2 + \dots + u'_{r-1} = \|c^T_{r-1}\| \le (1+\epsilon)^{r-2}\|c_{r-1}\|$, i.e.

$$u'_1 + u'_2 + \dots + u'_{r-1} \le (1+\epsilon)^{r-2}(u_1 + u_2 + \dots + u_{r-1}) \qquad (3.5)$$

Using Equations 3.4, 3.5, we have

$$\begin{aligned}
u_1^T + u_2^T + \ldots + u_{r-1}^T + u_r^T &\leq (1+\epsilon)((u'_1 + u'_2 + \ldots + u'_{r-1}) + u_r) \\
&\leq (1+\epsilon)((1+\epsilon)^{r-2}(u_1 + u_2 + \ldots + u_{r-1}) + u_r) \\
&\leq (1+\epsilon)^{r-1}(u_1 + u_2 + \ldots + u_r)
\end{aligned}$$

Now since $(d-1)\epsilon \leq 1$, using $r = d$ above, $\|c^T\| \leq (1+\epsilon)^{d-1}\|c\| \leq (1+(2d-2)\epsilon)\cdot\|c\|$.

■

Thus, we have reduced the original instance to a restricted instance, where the potential container points lie on a family with a constant number of lines. Section 3.2.3 describes a polynomial time algorithm for this problem in d-dimensions. For any $\epsilon' > 0$, setting $\epsilon = \frac{\epsilon'}{2(d-1)}$, we can restrict the loss to a $(1+\epsilon')$ factor in this process.

Theorem 3.2.6 *There is a PTAS for continuous container selection in fixed dimension d.*

3.2.3 Dynamic program for the restricted problem

In this section, we discuss a dynamic programming based algorithm to solve the following restricted problem in d-dimensions.

Definition 3.2.7 (restricted container selection) *For a constant $\eta \geq 0$, let $L_d = \{l_1, l_2, \ldots, l_\eta\}$ be a given family of η rays in $\mathbb{R}_+^d$. The input is a set of points*

$\mathcal{C} \subseteq \mathbb{R}^d$, a set of potential container points $\mathscr{F}$ that lie on the lines in L_d and a budget k. The goal is to find a subset $S \subseteq \mathscr{F}$ with $|S| \leq k$ such that $\mathsf{cost}(S)$ is minimized.

Theorem 3.2.8 *There is a poly-time algorithm for the restricted container selection problem.*

We need the following notion of a profile of a given subset of container points.

Profile of a subset. For a given line l_i and $S \subseteq \mathscr{F}$, let $c_i \in S$ be the container point on l_i with maximum ℓ_1-norm; if there is no such point then c_i is set to the origin. We define the *profile* of S, denoted by $\Pi(S)$, as the ordered tuple $(c_1, c_2, \dots, c_\eta)$. The *feasible* region of a profile $\Pi(S) = (c_1, c_2, \dots, c_\eta)$, denoted by $\mathsf{feas}(\Pi(S))$, is the set of those input points that are dominated by at least one of the points c_i, $i \in [\eta]$. We slightly abuse this notation and refer to the tuple itself as a profile, without any mention of S. The following is a simple combinatorial argument.

Observation 3.2.9 *The number of distinct profiles is at most $\left(\frac{|\mathscr{F}|}{\eta}\right)^\eta$.*

Proof: Let n_i be the number of potential container points on the line l_i. The total number of distinct profiles is simply the number of ways of choosing the tuple $(c_1, c_2, \dots, c_\eta)$, which is equal to $n_1 n_2 \dots n_\eta \leq \left(\frac{\sum_{i=1}^{\eta} n_i}{\eta}\right)^\eta = \left(\frac{|\mathscr{F}|}{\eta}\right)^\eta$. ■

For a given profile $\Pi = (c_1, c_2, \dots, c_\eta)$, let c_m denote the profile point with maximum ℓ_1-norm, i.e., $c_m = \underset{c_i}{\mathsf{arg\,max}} \|c_i\|$. Further, let $c'_m < c_m$ be some potential container point such that both the points are on the line l_m; if c'_m does not exist we set it to the origin. We define the *child profile* of Π corresponding to c'_m, denoted by $\mathsf{chld}(\Pi, c'_m)$, as the profile $(c_1, c_2, \dots, c_{m-1}, c'_m, \dots, c_\eta)$. We note that for a specific subset S, the child profile of S is unique, because c_m and c'_m are uniquely defined. However, a given profile tuple could have multiple child profiles. The following observation is immediate from the definition of a child profile.

Observation 3.2.10 *Any profile tuple* Π *has at most* $|\mathscr{F}|$ *child profile tuples.*

The DP variable. For every possible profile tuple $\Pi = (c_1, c_2, \ldots, c_\eta)$ and all budgets $k' \leq k$, define the dynamic program variable, $\mathscr{M}(\Pi, k')$ as the cost of an optimal solution $S \subseteq \mathsf{feas}(\Pi) \cap \mathscr{F}$, to assign all the input points in $\mathsf{feas}(\Pi)$, such that $|S| \leq k'$, and $c_i \in S$, for $i \in [\eta]$. The following lemma allows us to set up the dynamic program recurrence.

Lemma 3.2.11 *Let* $\Pi = (c_1, c_2, \ldots, c_\eta)$ *be a profile with* c_m *as the point with maximum* ℓ_1*-norm. For a given child profile* $\mathsf{chld}(\Pi, c'_m)$ *of* Π*, let* $n(c'_m) = |\mathsf{feas}(\Pi) \setminus \mathsf{feas}(\mathsf{chld}(\Pi, c'_m))|$*. Then, for any* $k' \geq 1$*, the following holds.*

$$\mathscr{M}(\Pi, k') = \underset{c'_m}{\mathsf{Min}} \ \ (\mathscr{M}(\mathsf{chld}(\Pi, c'_m), k'-1) + n(c'_m)\|c_m\|)$$

Proof: We denote the optimal solution corresponding to the variable $\mathscr{M}(\Pi, k')$ by $S(\Pi, k')$. Firstly, note that, for any c'_m, the solution $S(\mathsf{chld}(\Pi, c'_m), k'-1) \cup \{c_m\}$ is a feasible candidate for the computation of $\mathscr{M}(\Pi, k')$. Hence, we have

$$\mathscr{M}(\Pi, k') \leq \underset{c'_m}{\mathsf{Min}} \ \ (\mathscr{M}(\mathsf{chld}(\Pi, c'_m), k'-1) + n(c'_m)\|c_m\|) \tag{3.6}$$

Let l_m be the ray containing the point c_m. Further, let $q_0 = (0^d), q_1, \ldots, q_{j-1}, q_j = p_i$ be the container points, on l_m and in $S(\Pi, k)$, in the increasing order of ℓ_1-norm. Now, we set $q' = q_{j-1}$ and prove that the child profile corresponding to q' satisfies the following equation:

$$\mathscr{M}(\Pi, k') = \mathscr{M}(\mathsf{chld}(\Pi, q'), k'-1) + n(q')\|c_m\|$$

To this end, we first observe that, without loss of generality, no point in $\mathsf{feas}(\mathsf{chld}(\Pi, q'))$ is assigned to c_m. Indeed, this follows from the fact that c_m is the container point with maximum cost and therefore, any point in the above feasible region can be assigned to some container point on the profile $\mathsf{chld}(\Pi, q')$ without increasing the solution cost. Further, any point in $\mathsf{feas}(\Pi) \setminus \mathsf{feas}(\mathsf{chld}(\Pi, q'))$ must be assigned to c_m, since it is the only potential container point that dominates these points. Now,

$$\begin{aligned} \mathscr{M}(\Pi, k') &= \mathscr{M}(\mathsf{chld}(\Pi, q'), k'-1) + n(q')\|c_m\| \\ &\geq \underset{c'_m}{\mathsf{Min}} \ \left(\mathscr{M}(\mathsf{chld}(\Pi, c'_m), k'-1) + n(c'_m)\|c_m\|\right) \end{aligned} \tag{3.7}$$

From Equations 3.6 and 3.7, we have our lemma. ■

Algorithm 11 describes the dynamic program.

3.3 The discrete container selection problem

In this section, we consider the discrete version of the container selection problem. We start with the problem definition.

Definition 3.3.1 (discrete container selection) *In an instance of the problem, $\mathcal{I} = (\mathcal{C}, \mathscr{F}, k)$, we are given a set of input points $\mathcal{C} \subset \mathbb{R}^d$, a set of potential container points $\mathscr{F} \subset \mathbb{R}^d$ and a budget k. The goal is to find a subset of container points*

Algorithm 11 Dynamic program for the restricted container selection problem

```
Input: Family of lines $L_d = \{l_1, l_2 \ldots, l_\eta\}$, input points $\mathcal{C}$, potential container points
    set $\mathscr{F}$ on $L_d$ and a budget $k$
 1: for all profile tuples $\Pi$ (w.r.t $L_d$) and integers $k' \leq k$ do
 2:   if $k' = 0$ then
 3:     if $\Pi = ((0^d), (0^d), \ldots, (0^d))$ then
 4:       $\mathscr{M}(\Pi, k') = 0$
 5:     else
 6:       $\mathscr{M}(\Pi, k') = \infty$
 7:     end if
 8:   else
 9:     let $c_m$ be the container point with maximum $\ell_1$-norm in $\Pi$
10:     for all $c'_m \prec c_m$ such that both $c_m$ and $c'_m$ lie on the same line $l_m$ do
11:       $n(c'_m) \leftarrow |\mathsf{feas}(\Pi) \setminus \mathsf{feas}(\mathsf{chld}(\Pi, c'_m))|$
12:       $f(c'_m) \leftarrow (\mathscr{M}(\mathsf{chld}(\Pi, c'_m), k' - 1) + n(c'_m)\|c_m\|)$
13:     end for
14:     $\mathscr{M}(\Pi, k') \leftarrow \underset{c'_m}{\mathsf{Min}}\, f(c'_m)$
15:   end if
16: end for
17: return  profile $\Pi$ with least cost $\mathscr{M}(\Pi, k)$ such that $\mathcal{C} = \mathsf{feas}(\Pi)$.
```

$S \subseteq \mathscr{F}$, such that $|S| \leq k$ and the total assignment cost of all the input points, $\mathsf{cost}(S)$ is minimized.

This problem is considerably harder than the continuous version, as we show that there is no true approximation algorithm for this problem, unless $P = NP$, for $d \geq 3$. Hence, we look for bi-approximation algorithms defined as follows. An (α, β) bi-approximation algorithm obtains a solution S such that $|S| \leq \beta k$ and $\mathsf{cost}(S) \leq \alpha \mathsf{cost}(S_{opt})$.

Theorem 3.3.2 (two-dimenions) *For $d = 2$, and any constant $\epsilon > 0$, there is a $(1 + \epsilon, 3)$-bi-approximation algorithm for the discrete container selection problem.*

Theorem 3.3.3 (higher-dimensions) *For any dimension $d > 2$ and $\epsilon > 0$, there is a $\left(1 + \epsilon, O(\frac{d}{\epsilon} \log dk)\right)$-bi-approximation algorithm for the discrete container selec-*

tion problem.

3.3.1 Two dimensions

Algorithm Outline. The first step is to partition the plane into a logarithmic number of "cells" such that the ℓ_1-norms of points in a particular cell are approximately uniform. One standard way of doing this, where we create a two-dimensional grid with logarithmic number of lines in each dimension, fails because such a process would yield $\Omega(\log^2 n)$ cells. Our approach uses the *rays partitioning* idea. Given such a partitioning, we "guess" the "good" cells that have any container points belonging to a fixed optimal solution. For each one of these good cells, we then pick two representative container points. These points are chosen such that if in the optimal solution an input point i outside a cell e is assigned to a container point inside e, at least one of the representative points in e dominates i. This enables us to make "local decisions" for each cell independently. We then solve this localized instance, using k more container points. Hence, in total we use $3k$ container points.

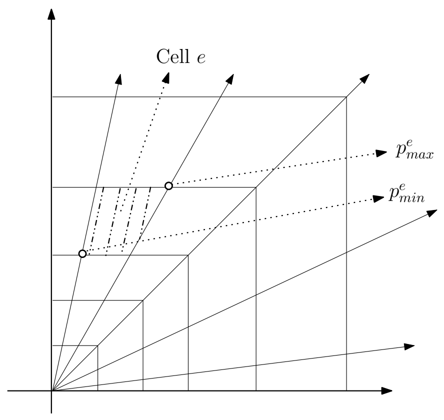


Figure 3.2: Description of the cells.

The algorithm. Fix $\delta = \frac{\epsilon}{11}$, be chosen such that $\frac{\pi}{4\delta} = \eta$ is an integer. We first use a simple scaling argument to bound the maximum to minimum ratio of ℓ_1-norms by $O(n)$. We guess the maximum norm container point p_{max} that is used in some fixed optimal solution (there are only $|\mathscr{F}|$ guesses) and delete all larger points from $\mathscr{F}$. Let p_{min} be the point in $\mathcal{C} \cup \mathscr{F}$ with minimum norm. We increase the x-coordinates of all the input points and the container points by $\frac{\delta}{n}\|p_{max}\|$ and then divide all the co-ordinates of all points by $\|p_{min}\|$.

Observation 3.3.4 *Let S_{opt} and S'_{opt} be the optimal solutions of a given instance before and after scaling respectively.* $\|p_{min}\|\mathsf{cost}(S'_{opt}) \leq \mathsf{cost}(S_{opt})(1+\delta)$

Proof: Since all the points are increased and scaled uniformly, the feasibility is maintained. Further, we note that $\mathsf{cost}(S_{opt}) \geq \|p_{max}\|$ since our guess $p_{max} \in S_{opt}$. If the cost of assignment of any input point is C in the original instance, the new cost is $= (C + \frac{\delta}{n}\|p_{max}\|)/\|p_{min}\|$ and the lemma follows. ■

From now on, we assume that all the points are scaled as above and therefore $\|p_{min}\| = 1$ and $\|p_{max}\| \leq \frac{n}{\delta}$. Let $t = \log_{1+\delta}\|p_{max}\|$ and define the following families of rays.

$$\mathcal{L}_1 = \{x\sin(r\delta) - y\cos(r\delta) = 0 : r \in [0,\eta)\} \quad \mathcal{L}_3 = \{y = (1+\delta)^i : i \in [0,t]\}$$

$$\mathcal{L}_2 = \{x\sin(r\delta) - y\cos(r\delta) = 0 : r \in [\eta,2\eta]\} \quad \mathcal{L}_4 = \{x = (1+\delta)^i : i \in [0,t]\}$$

Cells. We define the notion of cell as exemplified in the Figure 3.2. A *cell* is a quadrilateral formed with the following bounding lines: either, two consecutive lines in $\mathcal{L}_1$ and two consecutive lines in $\mathcal{L}_4$, or, two consecutive lines in $\mathcal{L}_2$ and two consecutive lines in $\mathcal{L}_3$. We observe that the number of cells formed is at most $(2\eta+1)t = O(\log n)$

Lemma 3.3.5 *For a given cell e, let p^e_{min} and p^e_{max} be the points of minimum and maximum cost, respectively. Then,*

$$\|p^e_{max}\| \leq (1+\epsilon)(\|p^e_{min}\|)$$

Proof: Without loss of generality, let e be formed by lines $y = (1+\delta)^i$, $y = (1+\delta)^{i+1}$, $x\sin\theta - y\cos\theta = 0$ and $x\sin(\theta+\delta) - y\cos(\theta+\delta) = 0$, where $\theta \geq \frac{\pi}{4}$. Clearly, as shown in the Figure 3.2, we have

$$p^e_{min} = ((1+\delta)^i\cot(\theta+\delta), (1+\delta)^i)$$

$$p^e_{max} = ((1+\delta)^{i+1}\cot\theta, (1+\delta)^{i+1})$$

$$\begin{aligned}
\frac{\|p^e_{max}\|}{\|p^e_{min}\|} &= \frac{(1+\delta)^{i+1}(1+\cot\theta)}{(1+\delta)^i(1+\cot(\theta+\delta))} = (1+\delta)\frac{(\sin\theta+\cos\theta)\sin(\theta+\delta)}{(\sin(\theta+\delta)+\cos(\theta+\delta))\sin\theta} \\
&= (1+\delta)\frac{\sin\theta\sin(\theta+\delta)+\cos\theta\sin(\theta+\delta)}{\sin(\theta+\delta)\sin\theta+\cos(\theta+\delta)\sin\theta} \\
&= (1+\delta)\left(1+\frac{\cos\theta\sin(\theta+\delta)-\cos(\theta+\delta)\sin\theta}{\sin(\theta+\delta)\sin\theta+\cos(\theta+\delta)\sin\theta}\right) \\
&= (1+\delta)\left(1+\frac{\sin\delta}{\sin(\theta+\delta)\sin\theta+\cos(\theta+\delta)\sin\theta}\right) \leq (1+\delta)\left(1+\frac{\sin\delta}{\sin^2\theta}\right) \\
&\leq (1+\delta)(1+2\delta) = (1+3\delta+2(\delta)^2) \leq (1+\epsilon)
\end{aligned}$$

We note that the last inequality follows from the fact that $\sin^2\theta \geq \sin^2\frac{\pi}{4} \geq \frac{1}{2}$. ∎

Representative points. For a given optimal solution, a cell is good if at least one container point is chosen from it (we break the ties between two cells sharing an edge arbitrarily). Since, there are $O(\log n)$ cells, there are a polynomial number of good-bad classifications. Therefore, we can try out all possible configurations and assume that we know which cells are good. For each good cell e, let p_x^e be the container point with maximum x-coordinate and p_y^e the one with maximum y-coordinate. We define the set of representative points, $\mathcal{R} = \{\ p_x^e,\ p_y^e : \forall\ e \text{ good cell }\}$. Clearly $|\mathcal{R}| \leq 2k$. We will show (in Lemma 3.3.7) that any input point that is not assigned to a "local container" (one in the same cell) in the optimal solution, can be re-assigned to some point of $\mathcal{R}$ at approximately the same cost.

Localized container selection problem. In an instance of the *localized container selection problem*, $\{\mathcal{C}, \mathscr{F}_1, \mathscr{F}_2, k\}$, we are given a set of input points $\mathcal{C}$, a set of potential container points $\mathscr{F}_1$, a set of pre-chosen container points $\mathscr{F}_2$ and a budget k. Moreover, for each cell e, the points in $\mathscr{F}_1 \cap e$ are all incomparable to each other. For a cell e, let $\Delta_{max}^e = \underset{p \in \mathscr{F}_1 \cap e}{\mathsf{Max}} \|p\|$, be the maximum ℓ_1-norm of any container point in e. The cost of assignment of any input point to any point, in $\mathscr{F}_1 \cap e$, is uniform and equal to Δ_{max}^e. The cost of assignment of an input point to a container point $c \in \mathscr{F}_2$ is $\|c\|$. Further, any input point p in the cell e, can only be assigned to:

- a container point $c \in \mathscr{F}_2$ such that $p \prec c$, or
- a container point $c \in \mathscr{F}_1$ such that c belongs to e and $p \prec c$.

Given an instance of the discrete container selection problem, $\mathcal{I} = (\mathcal{C}, \mathscr{F}, k)$, we construct the following instance of the *localized container selection problem*, $\mathcal{I}' = (\mathcal{C}, \mathscr{F}_1, \mathscr{F}_2, k)$.

Construction 3.3.6 *The input point set $\mathcal{C}$, remains the same and $\mathscr{F}_2$ is the set of representative points, i.e., $\mathscr{F}_2 = \mathcal{R}$. $\mathscr{F}_1$ is constructed as follows: starting with $\mathscr{F}_1 = \mathscr{F} \setminus \mathcal{R}$, while there are two points p and p' in $\mathscr{F}_1$ that belong to same cell e and $p \prec p'$, delete p from $\mathscr{F}_1$.*

Lemma 3.3.7 *For a given instance of the discrete container selection problem, $\mathcal{I} = (\mathcal{C}, \mathscr{F}, k)$, with the optimal solution cost OPT, the corresponding localized container selection instance $\mathcal{I}' = (\mathcal{C}, \mathscr{F}_1, \mathscr{F}_2, k)$ has an optimal cost of at least $(1+\epsilon)OPT$.*

Proof: Suppose S is an optimal solution for the instance $\mathcal{I}$. We iteratively construct a solution, S', for the instance $\mathcal{I}'$. Initiating $S' = \phi$, we add exactly one container point for every container point $c \in S$ in the following way: let c belong to a cell e. If $c \in \mathscr{F}_1$, then we add c to S'; otherwise, we add some $c' \in \mathscr{F}_1 \cap e$, such that $c \prec c'$, which must exist by Construction 3.3.6. Clearly $|S'| \le |S| \le k$. We show that S' is a feasible solution, with a cost at least $(1+\epsilon)OPT$, for the instance $\mathcal{I}'$.

Consider an input point p that is assigned to some container point $c \in S$, in the optimal solution for $\mathcal{I}$. Suppose, firstly, that c and p are contained in the same cell e. By the construction of S', there must be some $c' \in S' \cap e$ (possibly $c = c'$) such that $c \prec c'$ and we can assign p to c'. Further, note that since p and c' belong to the same cell this is a valid "local" assignment and by Lemma 3.3.5, the cost of assignment equals $\Delta^e_{max} \le \|c\|(1+\epsilon)$.

Subsequently, assume that p belongs to a cell e_1 and c belongs to a cell e_2, such that $e_1 \ne e_2$. We show that p can be assigned to one of the two representative points of e_2, namely $p_x^{e_2}$ or $p_y^{e_2}$. Recall that $p_x^{e_2}$ (resp. $p_y^{e_2}$) is a container point in e_2 with maximum x-coordinate (resp. y-coordinate). We first claim that there must exist a separating line $y = mx + C$ with slope $m \geq 0$, such that e_1 and e_2 lie on the opposite sides of this line (they could share a boundary along this line). We overload notation and allow $m = \infty$ in which the line is $x + C = 0$. So when $m = 0$ the line ($y = C$) is parallel to the x-axis and when $m = \infty$ the line ($x = -C$) is parallel to the y-axis.

Observe that by our construction, all the boundary lines have non-negative slopes. Therefore, if e_1 and e_2 share a boundary line segment, this will be our separating line. Suppose, on the other hand, that they do not share a boundary line segment and therefore are disjoint. If e_1 and e_2 are on the opposite sides of the line $y = x$, this will be our separating line. So, we assume that both the cells are on the same side of $y = x$, without loss of generality say above $y = x$. Then both these cells must be bounded by lines from the families $\mathcal{L}_2$ and $\mathcal{L}_3$. Let the lines bounding e_1 and e_2, respectively be, $B_1 = \{y = (1+\delta)^i,\ y = (1+\delta)^{i+1},\ x\sin\theta - y\cos\theta = 0,\ x\sin(\theta+\delta) - y\cos(\theta+\delta) = 0\}$ and $B_2 = \{y = (1+\delta)^j,\ y = (1+\delta)^{j+1},\ x\sin\theta' - y\cos\theta' = 0,\ x\sin(\theta'+\delta) - y\cos(\theta'+\delta) = 0\}$. Now, if $i = j$, then for the cells not to intersect, we must have $\theta \geq \theta' + \delta$ or $\theta' \geq \theta + \delta$. Without loss of generality, let $\theta \geq \theta' + \delta$. In this case, clearly the separating line is $x\sin\theta - y\cos\theta = 0$. In the case, where $i > j$ (resp. $i < j$), $y = (1+\delta)^j$ (resp. $y = (1+\delta)^i$) is a separating line.

We consider two different cases based on the value of m and prove that p can be assigned to some representative point in e_2.

Case 1: $m \in \{0, \infty\}$. The separating line between e_1 and e_2 is axis parallel, say

$x = a$, without loss of generality. Since $p \prec c$, we have that the x-co-ordinates of all points in e_1 are less a and x-coordinates of all points in e_2 are more than a. Hence, clearly the point with maximum y-coordinate in e_2, namely $p_y^{e_2}$ must dominate p.

Case 2: $m > 0$ and finite. Let the separating line be $y = mx + C$. There are two further cases here. First assume that p lies below the $y = mx + C$ and c lies above it. Letting $p = (x_1, y_1)$, $c = (x_2, y_2)$ and $p_x^{e_2} = (x_3, y_3)$, we have $y_1 \leq mx_1 + C$ and $y_2 \geq mx_2 + C$ and $y_3 \geq mx_3 + C$. By definition, $x_1 \leq x_2 \leq x_3$ and we focus on showing that $y_1 \leq y_3$. Indeed we have $y_1 \leq mx_1 + C \leq mx_2 + C \leq mx_3 + C \leq y_3$. Thus, $p \prec p_x^{e_2}$. Next, we assume that p lies above $y = mx + C$ and c lies below it. Letting $p = (x_1, y_1)$, $c = (x_2, y_2)$ and $p_y^{e_2} = (x_3, y_3)$, we have $y_1 \geq mx_1 + C$, $y_2 \leq mx_2 + C$ and $y_3 \leq mx_3 + C$. By definition, $y_1 \leq y_2 \leq y_3$. Further, $x_1 \leq y_1/m - C/m \leq y_2/m - C/m \leq y_3/m - C/m \leq x_3$. Hence, $p \prec p_y^{e_2}$. Therefore, we have shown that if p is assigned to c, we can assign it to a representative point, c_r, that lies in the same cell as c. From Lemma 3.3.5, this implies that our cost of assignment is $\|c_r\| \leq (1 + \epsilon)\|c\|$. Hence, the lemma. ■

We describe a polynomial time algorithm to solve the *localized container selection problem*. The approach is dynamic program based.

Dynamic program for the localized container selection problem. We define the dynamic program variable, $\mathcal{M}(e, k_e)$, for a given cell e, as the optimal cost of assigning all input points in e, to $k_e \leq k$ newly chosen container points in e, along with the set $\mathcal{R}$ of representative container points. We note that this variable can be computed in polynomial time using ideas in [132]. For completeness, we describe a simple algorithm to compute this variable for every e and $k_e \leq k$.

Dynamic program to compute $\mathcal{M}(e, k_e)$. We recall that by the problem definition, all the container points in e are incomparable and have the same cost, C. Let $c_1(x_1, y_1), c_2(x_2, y_2), \ldots, c_l(x_l, y_l)$ be the ordering of the container points in e, in the descending order of the y_i. That is $y_1 \geq y_2 \geq \ldots \geq y_l$ and $x_1 \leq x_2 \leq \ldots \leq x_l$. For a given index $i \in [l]$ and integer $k_i \leq k_e$, we define the variable $\mathcal{N}(i, k_i, j)$ is the optimal cost of assigning every input point, (x, y), in e, such that $y > y_{i+1}$, by choosing k_i container points with index $\leq i$, with $j \leq i$ being the highest index container point chosen (that is c_j is chosen and none of $c_{j+1}, \ldots, c_i$ are chosen). The following recurrence computes the variable $\mathcal{N}(i, k_i, j)$. Let n_i be the number of input points contained by c_i, whose y-co-ordinates are $> y_{i+1}$. If c_i is chosen,

$$\mathcal{N}(i, k_i, i) = \underset{j<i}{\mathsf{Min}}\ \mathcal{N}[i-1, k_i - 1, j] + n_i \times C$$

Now, if c_i is not chosen and c_j is the highest index container point chosen, with $j \leq i$, we assign the input points contained in c_i with x-coordinate $> x_j$ and y-coordinate $> y_{i+1}$ to the nearest representative container point (if no such point exists, then the cost of assignment is ∞). Further, we assign those, so far, unassigned input points with y-co-ordinate $> y_{i+1}$ and x-co-ordinate $\leq x_j$ to c_j. Let C_i denote the total cost of assignment of all these input points. We have

$$\mathcal{N}(i, k_i, j) = M[i-1, k_i, j] + C_i$$

We can compute $\mathcal{M}$, using the following equation: $\mathcal{M}(e, k_e) = \underset{j \leq l}{\mathsf{Min}}\ \mathcal{N}[l, k_e, j]$

Let there be μ cells in total. We order them arbitrarily as $e_1, e_2 \dots e_\mu$. We define the variable $\mathcal{D}(i, k_i)$ as the total cost of assigning all the input points in the cells e_j, for $j \in [i]$, while choosing k_i new container points from these cells and using the representative set $\mathcal{R}$. The following simple recurrence defines the dynamic program: $\mathcal{D}[i, k_i] = \operatorname*{Min}_{\ell \le k_i} \ \mathcal{D}[i-1, k_i - \ell] + \mathcal{M}[e_i, \ell]$

The optimal solution has a cost $\mathcal{D}[\mu, k]$. This completes the proof of Theorem 3.3.2.

Remark. This approach does not extend directly to dimension $d = 3$. There are issues in both main steps of the algorithm (1) we do not know a similar construction with $O(\log n)$ cells, and (2) the localized container selection problem also appears hard.

3.3.2 Higher dimensions

We now consider the discrete container selection problem in any dimension $d > 2$. Recall that $\mathcal{C}$ denotes the input points and $\mathscr{F}$ the potential container points. We prove Theorem 3.3.3. Our algorithm is based on the linear programming relaxation in Figure 3.3.

When the x and y variables are restricted to lie in $\{0, 1\}$ note that we obtain an exact formulation. This LP relaxation is similar to the

$$\begin{aligned}
\mathsf{Min} \quad & \sum_{i \in \mathscr{F}} \|i\| \sum_{j \in \mathcal{C}} y_{ij} \\
s.t. \quad & y_{ij} \le x_i, \qquad \forall i \in \mathscr{F},\ j \in \mathcal{C}, \\
& y_{ij} = 0, \qquad \forall j \not\preceq i, \\
& \sum_{i \in \mathscr{F}} y_{ij} \ge 1, \qquad \forall j \in \mathcal{C}, \\
& \sum_{i \in \mathscr{F}} x_i \le k, \\
& x, y \ge 0.
\end{aligned}$$

Figure 3.3: LP relaxation.

one for (non-metric) facility location [21]. Indeed, our problem is a special case of non-metric k-median, for which the result of [21] implies a $\big(1+\epsilon, O(\frac{1}{\epsilon}\log n)\big)$-bicriteria approximation algorithm. Our result (Theorem 3.3.3) is an improvement for fixed dimensions since $k \le n$.

The first step in our algorithm is to solve the LP. Let (x, y) denote an optimal LP solution. The second step performs a filtering of the y variables, as in [21]. Let $C_j^* = \sum_{i\in\mathscr{F}} \|i\| \cdot y_{ij}$ denote the contribution of input point $j \in \mathcal{C}$ to the optimal LP objective. Define:

$$\overline{y}_{ij} = \begin{cases} (1+\frac{1}{\epsilon})y_{ij} & \text{if } \|i\| \le (1+\epsilon)C_j^* \\ 0 & \text{otherwise.} \end{cases}$$

Also define $\overline{x}_i = (1+\frac{1}{\epsilon})x_i$ for all $i \in \mathscr{F}$, and $\overline{C}_j = (1+\epsilon)C_j^*$ for $j \in \mathcal{C}$.

Claim 3.3.8 *For each $j \in \mathcal{C}$, $\sum_{i\in\mathscr{F}} \overline{y}_{ij} \ge 1$. For each $j \in \mathcal{C}$ and $i \in \mathscr{F}$, $\overline{y}_{ij} \le \overline{x}_i$.*

Proof: Fix any $j \in \mathcal{C}$ and let $F_j = \{i \in \mathscr{F} : \|i\| > (1+\epsilon)C_j^*\}$. By Markov's inequality we have $\sum_{i\in F_j} y_{ij} < \frac{1}{1+\epsilon}$. So $\sum_{i\in\mathscr{F}} \overline{y}_{ij} = (1+\frac{1}{\epsilon})\sum_{i\in\mathscr{F}\setminus F_j} y_{ij} \ge 1$. ■

The third step of our algorithm formulates a geometric hitting-set problem with VC-dimension d. For each input point $j \in \mathcal{C}$, define a polytope $P_j \subseteq \mathbb{R}^d$ given by:

$$P_j = \{v \in \mathbb{R}^d : j \prec v \text{ and } \|v\| \le \overline{C}_j\} = \left\{v \in \mathbb{R}^d : v_r \ge j_r \,\forall r \in [d],\ \sum_{r=1}^{d} v_r \le \overline{C}_j\right\}$$

Note that each P_j is described by $d+1$ *parallel inequalities* of the form:

$$\{-e_r^t v \le -j_r\}_{r=1}^d \cup \{e^t v \le \overline{C}_j\}.$$

Above e_r denotes the r^{th} coordinate unit vector and $e = (1, 1, \cdots 1)$.

Claim 3.3.9 *For each $j \in \mathcal{C}$, $\sum_{i \in \mathscr{F} \cap P_j} \overline{x}_i \ge 1$.*

Proof: This follows directly from Claim 3.3.8 since $\overline{y}_{ij} = 0$ for all $j \in \mathcal{C}$ and $i \notin P_j$.

■

VC dimension bound. We use the following fact about the VC-dimension of a range space $(\mathscr{F}, \mathcal{P})$ where P is a finite set of points in $\mathbb{R}^d$ and $\mathcal{P}$ consists of all positive scaling and translations of a fixed polytope $Q \subseteq \mathbb{R}^d$ with $q \ge d$ facets.

Lemma 3.3.10 *The VC-dimension of $(\mathscr{F}, \mathcal{P})$ is at most q.*

Proof: This may be a known result; in any case we give a short proof here. Let polytope $Q = \{x \in \mathbb{R}^d : \alpha_r^t x \le \beta_r, \forall r \in [q]\}$ where each $\alpha_r \in \mathbb{R}^d$ and $\beta_r \in \mathbb{R}$.

The VC-dimension is the size of the largest subset $A \subseteq \mathscr{F}$ such that $\{A \cap P : P \in \mathcal{P}\} = 2^A$. Consider any such set A. Suppose (for contradiction) that $|A| > q$, then we will show a subset $A' \subseteq A$ such that there is no $P \in \mathcal{P}$ with $A \cap P = A'$. This would prove the claim.

For each constraint $r \in [q]$ let $a_r \in A$ denote a point that maximizes $\{\alpha_r^t x : x \in A\}$. Set $A' = \{a_r\}_{r=1}^q$. Note that there is some $a' \in A \setminus A'$ since $|A| > q$ and $|A'| \le q$; moreover, by the choice of a_rs, we have $\alpha_r^t a' \le \alpha_r^t a_r$ for all $r \in [q]$.

Suppose $P \in \mathcal{P}$ is any polytope that contains all points in A'. Note that $P = \{x \in \mathbb{R}^d : \alpha_r^t x \leq \gamma_r, \forall r \in [q]\}$ for some $\{\gamma_r \in \mathbb{R}\}_{r=1}^q$ since it is a scaled translation of the fixed polytope Q. Since $a_r \in P$ for each $r \in [q]$, we have $\gamma_r \geq \alpha_r^t a_r \geq \alpha_r^t a'$. This means that $a' \in P$ as well. Hence there is no set $P \in \mathcal{P}$ with $P \cap A = A'$. ■

Applying Lemma 3.3.10 we obtain $(\mathscr{F}, \{P_j : j \in \mathcal{C}\})$ has VC-dimension at most $d+1$. Moreover, by Claim 3.3.9 the hitting set instance $(\mathscr{F}, \{P_j : j \in \mathcal{C}\})$ has a fractional hitting set $\{\overline{x}_i : i \in \mathscr{F}\}$ of size $(1 + \frac{1}{\epsilon})k$. Thus we can use the following well-known result:

Theorem 3.3.11 ([129, 130]) *Given any hitting set instance on a set-system with VC-dimension d and a fractional hitting set of size k, there is a polynomial time algorithm to compute an integral hitting set of size $O(d \log(dk)) \cdot k$.*

This completes the proof of Theorem 3.3.3.

Remark: We can also use this LP-based approach to obtain a constant-factor bicriteria approximation for the discrete container selection problem in $\mathbb{R}^2$. This is based on the ϵ-net result for "pseudo-disks" in $\mathbb{R}^2$ [131] and the observation that in dimension two the above set-system $(\mathscr{F}, \{P_j : j \in \mathcal{C}\})$ is a collection of pseudo-disks. However, the constant factor obtained via this approach is much worse than the direct approach in Section 3.3.1.

3.4 Hardness results

In this section, we provide hardness results for the continuous and discrete container selection problems in dimension $d = 3$. The reductions are based on the planar degree 3 vertex cover problem. The following restriction of this problem is also known to be NP-hard [128].

Definition 3.4.1 (Plane degree 3 vertex cover (PVC)) *The input is a bound k and a plane drawing of a degree 3 planar graph $G = (V, E)$ with girth at least 4, where the distance between any pair $u, v \in V$ of vertices is exactly one if $(u, v) \in E$ and at least $\sqrt{3}$ if $(u, v) \notin E$. The decision problem is to determine whether G has a vertex cover of size at most k.*

We first show that the following auxiliary problem is NP-hard.

Definition 3.4.2 (Δ-hitting problem) *The input is a bound k, a set V of points in the plane where each pairwise distance is at least one and a set $\{\Delta_e\}_{e \in E}$ of equilateral triangles with side $s := \frac{2}{\sqrt{3}}$ that are all translates of each other. The goal is to find a subset $T \subseteq V$ with $|T| \leq k$ such that $T \cap \Delta_e \neq \emptyset$ for all $e \in E$.*

Theorem 3.4.3 *The Δ-hitting problem is NP-hard.*

Proof: We reduce from the NP-hard PVC problem. An instance of PVC consists of a plane drawing of graph $G = (V, E)$ and bound k. We construct an instance of the Δ-hitting problem as follows. The set of points is V and the bound is k. Note that the the distance between each pair of points is at least one, by Definition 3.4.1. For each edge $e = (u, v) \in E$ we can find (in polynomial time) an equilateral triangle

Δ_e with side $s = \frac{2}{\sqrt{3}}$ such that $V \cap \Delta_e = \{u, v\}$. To see this, first note that we can easily find $\Delta_e \ni u, v$ as $d(u, v) = 1$. Since the diameter of Δ_e is $\frac{2}{\sqrt{3}} < \sqrt{3}$ the vertices $V \cap \Delta_e$ form a clique in G, and as G has girth 4 we must have $|V \cap \Delta_e| = 2$. The set of triangles in the Δ-hitting problem is $\{\Delta_e\}_{e \in E}$. Moreover, we can ensure that the triangles $\{\Delta_e\}_{e \in E}$ are all translates of some canonical triangle. It is now clear that the Δ-hitting problem is a *yes*-instance if and only if the PVC instance has a vertex cover of size at most k. ■

Theorem 3.4.4 *The 3-dimensional discrete container selection problem is NP-hard.*

Proof: We reduce from the Δ-hitting problem. Consider an instance as described in Definition 3.4.2. We construct an instance of the discrete problem in $\mathbb{R}^3$ as follows. Set $A = 2|V|$ and let Π denote the plane $x + y + z = A$. We place the points V and triangles $\{\Delta_e\}_{e \in E}$ of the Δ-hitting instance on plane Π oriented so that every triangle Δ_e is parallel to the triangle $\{(A, 0, 0), (0, A, 0), (0, 0, A)\}$. We can ensure that all points in V are in the positive orthant since A is large. The potential container points are V. Observe that for each triangle Δ_e there is a unique point $p_e \in \mathbb{R}^3$ such that $\Delta_e = \Pi \cap \{x \in \mathbb{R}^3 : p_e \prec x\}$. The set of input points is $\{p_e\}_{e \in E}$. The bound k is same as for the Δ-hitting problem.

It is easy to see that the discrete container selection instance has a feasible solution with k containers if and only if the Δ-hitting instance is a *yes*-instance. ■

We immediately have the following corollary of the Theorem 3.4.4.

Corollary 3.4.5 *It is NP-hard to approximate the 3-dimensional discrete container selection problem within any approximation guarantee.*

Theorem 3.4.6 *The 3-dimensional continuous container selection problem is NP-hard.*

Proof: We reduce from the discrete container selection problem. We also rely on the structure of hard instances from Theorem 3.4.4. Let $\mathcal{I}_1 = (\mathcal{C}, \mathscr{F}, k)$ denote an instance of the discrete container selection problem from Theorem 3.4.4 where $\mathcal{C}$ are the input points and $\mathscr{F}$ denotes the potential container points. Note that all points of $\mathscr{F}$ lie on the plane $x + y + z = A$, and the distance between every pair of points in $\mathscr{F}$ is at least one. Observe that the latter property implies that the points in $\mathscr{F}$ are incomparable.

We construct an instance $\mathcal{I}_2 = (\mathcal{C}', k')$, of the continuous problem in the following way. Fix parameter $\delta < \frac{1}{2}$. For every point $c \in \mathscr{F}$ we define another point $\hat{c} := c + \delta(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$; note that $\|\hat{c}\| = \|c\| + \delta$ and $\hat{c}$ dominates c but no other point in $\mathscr{F} \setminus \{c\}$. Let $\hat{\mathscr{F}} = \{\hat{c} : c \in \mathscr{F}\}$. Observe that this is well-defined: since the distance between every pair of points in $\mathscr{F}$ is at least one, any point dominating more than one point of $\mathscr{F}$ costs at least $A + 1$.

Now, the set $\mathcal{C}'$ of input points is constructed as follows. Let $M_1 \gg |\mathcal{C}|A$ and $M_2 \gg 2(|\mathcal{C}|A + |\mathscr{F}|M_1)$ be two sufficiently large integers. For each $c \in \mathscr{F}$, we create M_1 input points at c and M_2 input points at $\hat{c}$, which are added to $\mathcal{C}'$. Finally we also add the points $\mathcal{C}$ to $\mathcal{C}'$. The bound $k' := k + |\mathscr{F}|$. We claim that $\mathcal{I}_1$ is feasible if and only if $\mathcal{I}_2$ has a solution of cost at most $T := |\mathcal{C}|A + |\mathscr{F}|(M_1 + M_2)(A + \delta) - kM_1\delta$.

Forward direction. Let $S = \{c_1, c_2, \ldots, c_k\}$ be the set of container points chosen by a feasible solution of $\mathcal{I}_1$. Consider the set $S' = S \cup \hat{\mathscr{F}}$. Observe that S' is a feasible solution for the instance $\mathcal{I}_2$. We now compute the assignment cost of this solution.

- The assignment cost for each point in $\mathscr{C}$ is A (it is covered by S).
- The input points at locations of S have assignment cost A (there are kM_1 such points).
- The remaining $(|\mathscr{F}| - k)M_1 + |\mathscr{F}|M_2$ input points have assignment cost $A + \delta$ each.

Therefore the total cost of this solution is exactly T.

Backward direction. Let S' with $|S'| = k + |\mathscr{F}|$ be a feasible solution to $\mathcal{I}_2$ of cost at most T. We first argue that $\hat{\mathscr{F}} \subseteq S'$. Indeed, assume that it is not true. Observe that, in this case, the input points at $\hat{\mathscr{F}}$ should be dominated by $< |\mathscr{F}|$ container points. So some container point $s \in S'$ should dominate input points at two distinct locations $\hat{c_i}$ and $\hat{c_j}$. Note that $|s| \geq A + 1$ since $c_i, c_j \prec s$ (using the distance one separation between points of $\mathscr{F}$). Hence any such solution has assignment cost at least $AM_1|\mathscr{F}| + (A+\delta)M_2|\mathscr{F}| + (1-\delta)M_2 > T$ using the definition of M_2. We now assume $\hat{\mathscr{F}} \subseteq S'$. Next we show that each of the remaining k container points in S' dominates at most one point of $\mathscr{F}$. If $s \in S'$ dominates two distinct locations c_i and c_j, its cost $|s| \geq A + 1$ as noted above. However, any input point can be assigned to one of the container points in $\hat{\mathscr{F}}$ at cost $A + \delta < A + 1$, which makes point s redundant.

Now we show that each of the k container points $S' \setminus \hat{\mathscr{F}}$ dominates some point of $\mathscr{F}$. If not, consider a container point $s' \in S'$ that does not dominate any $\mathscr{F}$ point. Let $f \in \mathscr{F}$ be some point which is not dominated by any $S' \setminus \hat{\mathscr{F}}$; note that this must exist since each $S' \setminus \hat{\mathscr{F}}$ dominates at most one $\mathscr{F}$-point and $|S' \setminus \hat{\mathscr{F}}| = k \leq |\mathscr{F}|$. Suppose we modify the solution by removing s' and adding f: the increase in cost is at most $|\mathcal{C}|(A+\delta) + M_1 A - M_1(A+\delta) < 0$ by the definition of M_1. Thus, $\hat{\mathscr{F}} \subseteq S'$ and $S'' = S' \setminus \hat{\mathscr{F}} \subseteq \mathscr{F}$. We now claim that S'' dominates every point of $\mathcal{C}$. For a contradiction, suppose there is some point of $\mathcal{C}$ that is not dominated by S'' : then this point has assignment cost $A+\delta$. Every other points of $\mathcal{C}$ has assignment cost at least A. The assignment cost of points at $\hat{\mathscr{F}} \cup \mathscr{F}$ is $|\mathscr{F}|(M_1 + M_2)(A + \delta) - kM_1\delta$. So the total assignment cost is at least $T + \delta$, a contradiction. Hence S'' is a feasible solution for $\mathcal{I}_1$. ■

Chapter 4

Partial and Budgeted Connected Dominating Set.

4.1 Roadmap to the chapter

This chapter discusses nearly tight (upto a constant factor) approximation algorithms for the partial and budgeted variants of the connected dominating set problem. As we will discuss later in this section, none of the previous approaches that work for CDS problem give any good guarantees for these problem. The key idea is to *define a natural, greedy based, profit function and use a classic result of the well studied problem of quota Steiner tree.* The *quota Steiner tree* problem is an integral component of all our results in this chapter. Formally, the problem is the following.

Definition 4.1.1 (Quota Steiner tree problem(QST)) *Given an undirected graph $G = (V, E)$, a profit function $p : V \to Z^+ \cup \{0\}$ on the vertices, a cost function $c : E \to Z^+ \cup \{0\}$ on the edges, and an integer (quota) q, find a subtree T that minimizes $\sum_{e \in E(T)} c(e)$, subject to $\sum_{v \in V(T)} p(v) \geq q$.*

An important special case, where the profit function is uniform, is called the *k-minimum spanning tree* (k-MST) problem. Johnson et al. [58] studied the QST problem and showed that an α-approximation algorithm for the k-MST problem can be adapted to obtain an α-approximation algorithm for the QST problem. Using this result along with the 2-approximation for k-MST by Garg [48], gives us the following theorem.

Theorem 4.1.2 ([48, 58]) *There is a 2-approximation algorithm for the* QST *problem.*

In Section 4.2, we discuss a $O(\ln \Delta)$ approximation algorithm for the PCDS problem. To be precise, our approximation guarantee is $4 \ln \Delta + 2 + o(1)$, where Δ is the maximum degree. Section 4.3 details a $\frac{1}{13}(1 - \frac{1}{e})$-approximation algorithm for the BCDS problem. This is the first constant approximation known for BCDS. In Section 4.4, we generalize the above problems to a special kind of submodular optimization problem (to be defined later), which has the *weighted profit connected dominating set* problem as a special case. Again, we obtain $O(\ln q)$ and $\frac{1}{13}(1 - \frac{1}{e})$ approximation algorithms for the partial and budgeted version of this problem respectively where q denotes the quota for the partial version.

Previous approaches. We now describe the three approaches taken by Guha and Khuller [8] to solve the CDS problem and show why none of these approaches extend directly for the budgeted and partial coverage variants.

Algorithm 1. The first algorithm is a "one step look-ahead" greedy algorithm where they iteratively grow a tree by selecting a *pair* of vertices that together cover the most number of previously uncovered vertices. Figure 4.1 shows a bad instance on which a c-step look-ahead greedy algorithm fails for the BCDS and PCDS. The instance contains k "spiders" whose heads (vertices with degree > 2) are connected by paths of length $c + 1$. The spider heads are the only vertices that offer profit greater than 3. We show that on this graph, there are BCDS and PCDS instances

that can perform very poorly. Consider a BCDS instance on the graph, with a budget $k+(c+1)(k-1)$. Clearly the optimal solution picks the path connecting all the spider heads, so that the total coverage is $(M+1)k+(c+1)(k-1)$. On the other hand, the c-step look-ahead greedy algorithm, might get stuck inside one of the spiders and may end up selecting as many as $M+1$ vertices from it. This is because, despite the look-ahead capability of the algorithm, the spider legs will become indistinguishable from the optimal path. For a sufficiently large value of M, the c-step look-ahead algorithm might use up all its budget on a single spider, there by obtaining a coverage of $O(M+k)$. Thus in the worst case the look-ahead greedy algorithm could have a $\Omega(k)$ approximation guarantee. Using a similar argument, we can show that, for the PCDS instance on the graph with quota Mk, the approximation guarantee could be $\Omega(M)$.

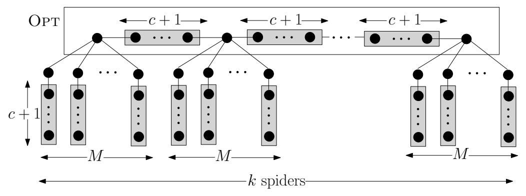


Figure 4.1: A bad example for the c-step look-ahead greedy algorithm

Algorithm 2. The second algorithm is to find a dominating set D and run a Steiner tree algorithm with the vertices in D as terminals. Since the optimal connected dominating set, by definition, is a tree that dominates D, we can show that there exists a Steiner tree of low cost with the set D as terminals. Using a constant factor approximation algorithm for the Steiner tree problem, we obtain a $O(\ln n)$ approximation for the connected dominating set. However, for the partial and budgeted versions, the optimal solution does not dominate all vertices and hence it's

not possible to bound the cost of the Steiner tree in terms of the optimal solution.

Algorithm 3. The final algorithm builds unconnected components greedily and owing to the fact that every vertex has to be dominated, makes sure that the constructed components be connected cheaply. Again this approach fails in the partial and budgeted case because the components created when we have dominated a specified number of vertices could be far apart.

4.2 Partial connected dominating set

The partial connected dominating set can be defined formally as follows.

Definition 4.2.1 (The partial connected dominating set (PCDS)) *Given an undirected graph $G = (V, E)$, and an integer (quota) n', find a minimum size subset $S \subseteq V$ of vertices such that the subgraph induced by S is connected, and S dominates at least n' vertices.*

We will discuss a $4 \ln \Delta + 2 + o(1)$-approximation algorithm for the PCDS problem in this section.

4.2.1 Algorithm

We now give a high level overview of the algorithm. The algorithm itself is very simple but to show that it is indeed a $O(\log \Delta)$ approximation requires non-trivial analysis. The algorithm proceeds in the following manner. We first run a simple greedy algorithm to find a (not necessarily connected) dominating set D. In each iteration, the greedy algorithm chooses a vertex that dominates the maximum number of previously undominated vertices.

We call this number the "profit" associated with the chosen vertex. Given this profit function on the nodes, we now apply a 2-approximation algorithm for the QST problem, with quota of n' to obtain a connected solution. This is a little surprising, since the profit function depends on the choices made by the greedy algorithm in the first phase. However, we can show that there is a subset of vertices $D' \subseteq D$, of cardinality at most $|\mathsf{OPT}| \ln \Delta + 1$ whose profits sum up to at least n' where $|\mathsf{OPT}|$ is the size of the optimum solution of the PCDS instance. Furthermore the vertices in D' can be connected with additional $(\ln \Delta + 1)|\mathsf{OPT}| + 1$ vertices. Thus, if we could find the smallest tree with total profit at least n', such a tree would cost (number of edges in the tree) no more than $(2 \ln \Delta + 1)|\mathsf{OPT}| + 2 - 1 = (2 \ln \Delta + 1)|\mathsf{OPT}| + 1$. This is a special case of the QST problem (with unit edge costs) and hence we can apply Theorem 4.2.3 to obtain a tree of size (cost) no more than $2((2 \ln \Delta + 1)|\mathsf{OPT}| + 1) = (4 \ln \Delta + 2)|\mathsf{OPT}| + 2$. Thus, we obtain a $(4 \ln \Delta + 2 + o(1))$-approximate solution for the PCDS problem.

Algorithm 12 Greedy profit labeling algorithm for PCDS.

```
Input: graph $G = (V, E)$ and $n' \in \mathbb{Z}^+ \cup \{0\}$.
Output: tree $T$ with at least $n'$ coverage.
1: compute the greedy dominating set $D$ and the corresponding profit function
   $p : V \to \mathbb{N}$ using the Algorithm 13.
2: use the 2-approximation algorithm for QST problem [58] on the instance $(G, p)$
   to obtain a tree $T$ with profit at least $n'$.
```

Algorithm 13 Greedy dominating set.

Input: graph $G = (V, E)$.

Output: dominating set D and its profit function p.

```
1: initiate: $D \leftarrow \phi$ and $U \leftarrow V$
2: for all $v \in V$ do
3:     $p(v) \leftarrow 0$;
4: end for
5: while $U \neq \phi$ do
6:     compute: $v \leftarrow \operatorname{arg\,max}_{v \in V \setminus D} |N_U(v)|$
7:     update: $p(v) \leftarrow |C_v|$, $U \leftarrow U \setminus N_U(v)$ and $D \leftarrow D \cup \{v\}$
8: end while
```

4.2.2 Analysis

We first introduce some required notation.

Notation: For every vertex $v \in D$ that is chosen by the greedy algorithm, let C_v denote the set of *new* vertices that v dominates i.e., we have $p(v) = |C_v|$. We say that v "covers" a vertex w if and only if $w \in C_v$. For the sake of analysis, we partition the vertices of the graph G into layers. Let $L_1 = \mathsf{OPT}$ be the vertices in an optimal solution for the PCDS instance, L_2 be the set of vertices that are not in L_1 and have at least one neighbor in L_1, and $R = V \setminus \{L_1 \cup L_2\}$ be the remaining vertices. Let L_3 be the subset of vertices of R that have a neighbor in L_2. Furthermore let $L'_i = D \cap L_i, 1 \leq i \leq 3$ where D is the dominating set chosen by the greedy algorithm. Figure 4.2 clarifies this notation regarding the layers L_i.

We first show the following.

Lemma 4.2.2 *There is a subset $D' \subseteq L'_1 \cup L'_2 \cup L'_3$ such that $|D'| \leq |\mathsf{OPT}| \ln \Delta + 1$*

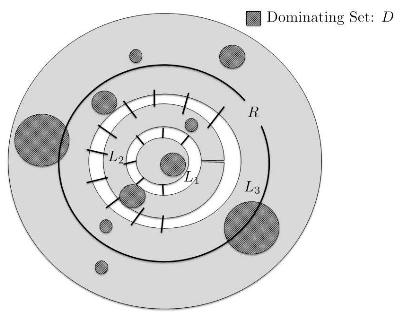


Figure 4.2: Pictorial representation of different layers. (a) L_1 is an optimal solution (b) L_2 is set of the vertices adjacent to L_1 (c) L_3 is the subsequent layer (d) R is the set of all vertices other than $L_1 \cup L_2$ and (e) $L'_i = L_i \cap D$.

and the total profit of vertices in D' is at least n', i.e. $\sum_{v \in D'} p(v) \geq n'$.

Proof: Let $L'_1 \cup L'_2 \cup L'_3 = \{v_1, v_2, \ldots, v_l\}$ where the vertices are arranged according to the order in which they were selected by the greedy algorithm. Since all vertices in $L_1 \cup L_2$ are dominated by $L'_1 \cup L'_2 \cup L'_3$, we have $\sum_{i=1}^{l} p(v_i) \geq |L_1 \cup L_2| \geq n'$ where the second inequality follows from the fact that L_1 is a feasible solution (in fact optimal feasible solution). Choose t such that $\sum_{i=1}^{t} p(v_i) < n'$ and $\sum_{i=1}^{t+1} p(v_i) \geq n'$. Let $\mathcal{S} = \{v_1, v_2, \ldots, v_t\}$ denote the set of the first t vertices chosen from the set $L'_1 \cup L'_2 \cup L'_3$. We now show that $|\mathcal{S}| = t \leq |\mathsf{OPT}| \ln \Delta$ and hence $D' = \mathcal{S} \cup \{v_{t+1}\}$ satisfies the requirements of the claim.

Let C_{12} be the set of vertices in $L_1 \cup L_2$ that are covered by $\mathcal{S}$ in the original greedy step i.e., $C_{12} = \cup_{v \in \mathcal{S}} \{C_v \cap (L_1 \cup L_2)\}$. Let $UC_{12} = (L_1 \cup L_2) \setminus C_{12}$ be the vertices in $L_1 \cup L_2$ that are not covered by $\mathcal{S}$. Similarly define $C_R = \cup_{v \in \mathcal{S}} \{C_v \cap R\}$ as the set of vertices in R covered by $\mathcal{S}$ (as per the greedy step). Then, we have that $|C_R| + |C_{12}| < n' \leq |L_1 \cup L_2| = |C_{12}| + |UC_{12}|$, where the first inequality follows from the definition of $\mathcal{S}$. Therefore we have $|C_R| < |UC_{12}|$.

We can thus assign every vertex in C_R to a unique vertex in UC_{12}, i.e. let $I : C_R \to UC_{12}$ denote a one to one function from C_R to UC_{12}. In the subsequent charging argument, any cost that we charge to a vertex $x \in C_R$ is transferred to its assigned vertex $I(x) \in UC_{12}$. Hence, after this charge transfer, only vertices in $L_1 \cup L_2$ will be charged. We will now use a charging argument to show that $|\mathcal{S}| \leq |\mathsf{OPT}| \ln \Delta$.

Consider a vertex $u \in \mathcal{S}$. We recall that C_u is the set of vertices covered for the first time by u in the greedy step. We assign every $w \in C_u$ a charge $\rho(w) = \frac{1}{|C_u|}$. It is clear that the total charge on all vertices is equal to the size of $\mathcal{S}$. As described above, the charge of a vertex in $w \in R$ is transfered to its mapped vertex in $I(w) \in UC_{12}$. Let v be a vertex in the optimal solution set L_1. We denote the set of neighbors of v, including itself, by $\mathcal{N}(v)$. We claim that the total charge on the vertices of $\mathcal{N}(v)$ is at most $\ln \Delta$. Initially, none of the vertices in $\mathcal{N}(v)$ are charged. Let $u_1, u_2 \dots, u_l$ be the vertices in $\mathcal{S}$ which charge some vertices of $\mathcal{N}(v)$ in that order. This charge could either be the direct charge or a transfer of charge from some vertex in R. For $i \in [l]$, let $O_i \subseteq \mathcal{N}(v)$ denote the set of vertices that remain uncharged (either directly or through a transfer), after the vertex u_i is picked into $\mathcal{S}$.

Let $O_0 = \mathcal{N}(v)$. We will now show that, for every u_i, $|C_{u_i}| \geq |O_{i-1}|$. Let us consider the iteration of the greedy algorithm in which u_i is picked. We claim that none of the vertices in O_{i-1} can be dominated by any vertex chosen before u_i in the greedy algorithm. Let $w \in O_{i-1}$ be some vertex which is dominated by some vertex u' chosen by greedy before u_i, such that $w \in C_{u'}$. Clearly $u' \in L_1' \cup L_2' \cup L_3'$ should hold, because no vertex in $R \setminus L_3$ can dominate w. But since u' was chosen before

u_i and $u' \in L'_1 \cup L'_2 \cup L'_3$, u' must be chosen into $\mathcal{S}$ before u_i. Hence, w cannot be an uncharged vertex in the current iteration leading to a contradiction.

Thus, in the iteration where the greedy algorithm was about to choose u_i, none of the vertices O_{i-1} have been dominated. Hence if the greedy were to choose v, then $p(v) \geq |O_{i-1}|$. Since the greedy algorithm chooses vertex u_i instead of v, we have $|C_{u_i}| \geq |O_{i-1}|$.

The total charge in this iteration $(C_{u_i} \cap \mathcal{N}(v))$ is thus at most $\frac{|O_{i-1}|-|O_i|}{|O_{i-1}|}$. Adding these charges over all l iterations, we get, using an analysis very similar to the set cover analysis [133], $\sum_{w \in \mathcal{N}(v)} \rho(w) \leq H(\Delta)$, where H is the harmonic function and Δ is the maximum degree. Adding up the charges over all vertices in L_1, we get $\sum_{u \in C_{12} \cup UC_{12}} \rho(u) \leq \sum_{v \in L_1} \sum_{w \in \mathcal{N}(v)} \rho(w) \leq |\mathsf{OPT}| \ln \Delta$. Hence we have $|\mathcal{S}| \leq |\mathsf{OPT}| \ln \Delta$. Since $\mathcal{S}$ was a maximal set having profit at most n', we obtain a set D' with $|D'| = |\mathcal{S}| + 1$ with profit at least n' by adding a single vertex to $\mathcal{S}$, which gives us the desired result. ■

Theorem 4.2.3 *Let OPT be the optimal solution set for an instance of PCDS. There exists a tree $\hat{T}$ with at most $2|\mathsf{OPT}| \ln \Delta + |\mathsf{OPT}| + 1$ edges such that $\sum_{v \in \hat{T}} p(v) \geq n'$.*

Proof: In Lemma 4.2.2, we have shown that there exists a subset $D' \subseteq L'_1 \cup L'_2 \cup L'_3$ of size $|\mathsf{OPT}| \ln \Delta + 1$ that has profit at least n'. However this set D' need not be connected. We now show that this set D' can be connected without paying too much. Firstly we note that for every vertex $v \in L_3 \cap D'$, there exists a vertex

$w \in L_2$ such that w dominates v. Thus we can pick a subset $D'' \subseteq L_2$ of size at most $|L_3 \cap D'| \leq |\mathsf{OPT}| \ln \Delta + 1$ which dominates all vertices of $L_3 \cap D'$. Now, it is sufficient to ensure that all the vertices of $(D' \cap L_2) \cup D''$ are connected. This can be achieved by simply adding all the vertices of L_1 to our solution. Thus we have shown that $\hat{D} = D' \cup D'' \cup L_1$ induces a connected subgraph with profit at least n' and the number of vertices in $\hat{D} \leq |D'| + |D''| + |L_1| \leq 2|\mathsf{OPT}| \ln \Delta + |\mathsf{OPT}| + 2$. Hence there exists subtree $\hat{T}$ on these vertices with at most $(2 \ln \Delta + 1)|\mathsf{OPT}| + 1$ edges with the requisite total profit. ■

Corollary 4.2.4 *Algorithm 12 is a* $4 \ln \Delta + 2 + o(1)$*-approximation algorithm for* *PCDS.*

Proof: Let OPT be the optimal solution of the PCDS instance. As per Theorem 4.2.3, we know that there exists a Steiner tree $\hat{T}$ with at most $2|\mathsf{OPT}| \ln \Delta + |\mathsf{OPT}| + 1$ edges whose total profit exceeds the quota n'. Hence, the tree T returned by the 2-approximation algorithm for the QST problem has at most $4|\mathsf{OPT}| \ln \Delta + 2|\mathsf{OPT}| + 2$ edges. Thus, we obtain a $4 \ln \Delta + 2 + o(1)$ approximation algorithm. ■

4.3 Budgeted connected dominating set

We now turn our attention to the *budgeted connected dominating set* (BCDS) problem. Formally,

Definition 4.3.1 (The budgeted connected dominating set (BCDS)) *Given* *an undirected graph $G = (V, E)$, and an integer (budget) k, find a subset $S \subseteq V$ of at most k vertices such that the graph induced by S is connected, and the number of vertices dominated by S is maximized.*

4.3.1 Algorithm

Algorithm 14 is very similar to the one we used to obtain a partial connected dominating set. We start by running the standard greedy algorithm to find a dominating set D in the graph. We set the profits of vertices in D as the number of newly covered vertices at each step of the greedy algorithm, while we assign zero profit for the remaining vertices in $V \setminus D$. In the analysis section, we show that there is a tree on at most $3k$ vertices that has a total profit of at least $(1 - \frac{1}{e})\mathsf{OPT}$ where OPT is the number of vertices dominated by an optimal solution. Note that we may assume that we have guessed OPT by trying out values between k and n using, say, binary search. We run the 2 approximation algorithm for the QST problem on this instance with the quota being set to $(1 - \frac{1}{e})\mathsf{OPT}$. This will result in a tree with at most $6k$ nodes with total profit at least $(1 - \frac{1}{e})\mathsf{OPT}$. Thus we obtain a $(6, 1 - \frac{1}{e})$ bicriteria approximation algorithm. To convert this bicriteria approximation into a true approximation, we use a dynamic program (Section 4.3.2.2) to find the "best" subtree on at most k vertices from this tree of $6k$ vertices. We use a simple tree decomposition scheme to show that the best tree dominates at least $\frac{1}{13}(1 - \frac{1}{e})\mathsf{OPT}$ nodes.

Algorithm 14 Greedy profit labeling algorithm for BCDS.

```
Input: graph $G = (V, E)$ and $k \in \mathbb{N}$.
Output: tree $\tilde{T}$ with cost at most $k$.
1: compute the greedy dominating set $D$ and the corresponding profit function
   $p : V \to \mathbb{N}$ using the Algorithm 13.
2: OPT ← number of vertices dominated by an optimal solution (guess using binary
   search between $k$ and $n$).
3: use the 2-approximation algorithm for QST problem [58] to obtain a tree $T$ with
   profit at least $(1 - \frac{1}{e})$OPT (we show that $|T| \leq 6k$).
4: use the dynamic program of Section 4.3.2.2 to find $\tilde{T}$, the best subtree of $T$
   having at most $k$ vertices.
```

4.3.2 Analysis

Let L_1 denote the vertices in an optimal solution. Let layers L_2, L_3, R, and L'_i be defined as in Section 4.2. $\mathsf{OPT} = |L_1 \cup L_2|$ is the number of vertices dominated by the optimal solution.

Let $L'_1 \cup L'_2 \cup L'_3 = \{v_1, v_2, \ldots, v_l\}$ where the vertices are according to the order in which they were selected by the greedy algorithm. Let $D' = \{v_1, v_2, \ldots, v_k\}$ denote the first k vertices from $L'_1 \cup L'_2 \cup L'_3$. In Lemma 4.3.3, we prove that the total profit of $D' = \sum_{v \in D'} p(v)$ is at least $(1 - \frac{1}{e})\mathsf{OPT}$. Next, we can show that these k vertices can be connected by using at most $2k$ more vertices, thus proving the existence of a tree with at most $3k$ vertices having the desired total profit.

Let g_i denote the total profit after picking the first i vertices from D', i.e., $g_i = \sum_{j=1}^{i} p(v_j)$. We start by proving that the following recurrence holds for every $i = 0$ to $k - 1$.

Claim 4.3.2 $g_{i+1} - g_i \geq \frac{1}{k}(\mathit{OPT} - g_i)$

Proof: Consider the iteration of the greedy algorithm, where vertex v_{i+1} is being picked. We first show that at most g_i vertices of $L_1 \cup L_2$ have been already been dominated. Note that any vertex $w \in L_1 \cup L_2$ that has been already dominated must have been dominated by a vertex in $\{v_1, v_2, \dots v_i\}$. This is because no vertex from $R \setminus L_3$ can neighbor w. Since $g_i = \sum_{j=1}^{i} p(v_j)$ is the total profit gained so far, it follows that at most g_i vertices from $L_1 \cup L_2$ have been dominated. Hence we have that there are at least $\mathsf{OPT} - g_i$ undominated vertices in $L_1 \cup L_2$. Since the k vertices of L_1 together dominate all of these, it follows that there exists at least one vertex $v \in L_1$ which neighbors at least $\frac{1}{k}(\mathsf{OPT} - g_i)$ undominated vertices.

We conclude this proof by noting that since the greedy algorithm chose to pick v_{i+1} at this stage, instead of the v above, it follows that $p(v_{i+1}) = g_{i+1} - g_i \geq \frac{1}{k}(\mathsf{OPT} - g_i)$.

■

Lemma 4.3.3 *Let OPT be the number of vertices dominated by an optimal solution for BCDS. Then there exists a subset $D' \subseteq D$ of size k with total profit at least $(1 - \frac{1}{e})\mathit{OPT}$. Further, D' can be connected using at most $2k$ Steiner vertices.*

Proof: From the Claim 4.3.2, the profit after $i + 1$ iterations is given by

$$g_{i+1} \geq \frac{\mathsf{OPT}}{k} + g_i(1 - \frac{1}{k}).$$

By solving this recurrence, we get $g_i \geq (1 - (1 - \frac{1}{k})^i)\mathsf{OPT}$. Hence, we obtain the following.

$$\sum_{v \in D'} p(v) = g_k \geq (1 - (1 - \frac{1}{k})^k)\mathsf{OPT} \geq (1 - \frac{1}{e})\mathsf{OPT}$$

We show that D' can be connected by at most $2k$ Steiner nodes to form a connected tree. Note that for every vertex $v \in L_3 \cap D'$, there exists a vertex $w \in L_2$ such that w neighbors v. Thus we can pick a subset $D'' \subseteq L_2$ of size at most $|L_3 \cap D'| \leq k$ which dominates all vertices of $L_3 \cap D'$. Now, it is sufficient to ensure that all the vertices of $(D' \cap L_2) \cup D''$ are connected. This can be achieved by simply adding all the k vertices of L_1. Thus we have shown that $\hat{D} = D' \cup D'' \cup L_1$ induces a connected subgraph with profit at least $(1 - \frac{1}{e})\mathsf{OPT}$ and $|\hat{D}| \leq |D'| + |D''| + |L_1| \leq 3k$.

■

Lemma 4.3.4 *There is a* $(6, (1 - \frac{1}{e}))$ *bicriteria approximation algorithm for the* BCDS *problem.*

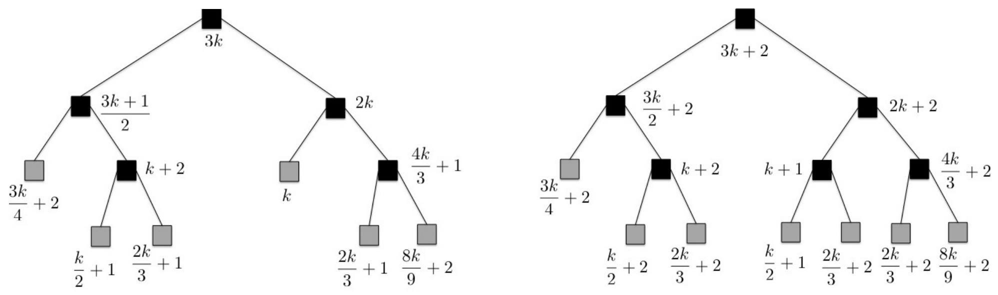
Proof: Lemma 4.3.3 shows that there exists a Steiner tree with at most $3k$ vertices having total profit greater than a quota of $(1 - \frac{1}{e})\mathsf{OPT}$. Hence, using the 2-approximation algorithm for the QST problem, we obtain a tree T of at most $6k$ nodes and total profit at least $(1 - \frac{1}{e})\mathsf{OPT}$. Thus we obtain a $(6, (1 - \frac{1}{e}))$ bicriteria approximation algorithm for the BCDS problem. ■

4.3.2.1 **Obtaining a true approximation**

In order to obtain a true approximate solution (solution of size k), we need a technique to find a small subtree $\tilde{T} \subseteq T$ of k vertices which has high total profit. In Section 4.3.2.2, we show that this problem can be easily solved in polynomial time using dynamic programming. However, simply finding the subtree which maximizes the profit is not enough to give a good approximation ratio. We need a way to compare the total profit of the subtree $\tilde{T}$ with the entire profit $P = \sum_{v \in T} p(v)$. We now show that if $n = 6k$, we can obtain a subtree having profit at least $\frac{1}{13}P$.

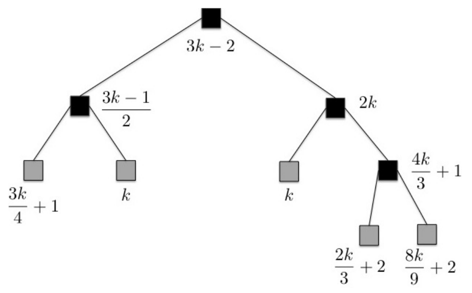
The following lemma is well known in folklore and can be easily proven by induction. It can also be seen as an easy consequence of a theorem by Jordan [134].

Lemma 4.3.5 (Jordan [134]) *Given any tree on n vertices, we can decompose it into two trees (by replicating a single vertex) such that the smaller tree has at most $\lceil \frac{n}{2} \rceil$ nodes and the larger tree has at most $\lceil \frac{2n}{3} \rceil$ nodes.*

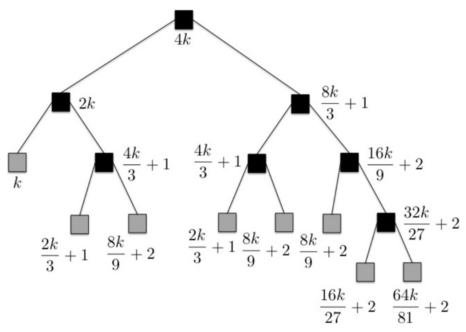


(a) Decomposes into 6 subtrees (b) Decomposes into 7 subtrees

Figure 4.3: Decomposition of trees into 13 subtrees



(a) Decomposes into 5 subtrees



(b) Decomposes into 8 subtrees

Figure 4.4: Decomposition of trees into 13 subtrees

We now show the following:

Lemma 4.3.6 *Let k be greater than a sufficiently large constant. Given a tree T with $6k$ nodes, we can decompose it into 13 trees of size at most k nodes each.*

Proof: We use Lemma 4.3.5 to decompose the tree into two trees T_1 and T_2 such that $|T_1| \leq |T_2|$. In this decomposition, at most one vertex is duplicated, therefore $|T_1| + |T_2| \leq 6k + 1$. Also, we have $|T_1| \leq 3k$. We now have two cases:

Case 1: $|T_1| \geq 3k - 1$. In this case, $|T_2| \leq 6k + 1 - |T_1| \leq 3k + 2$. Now repeatedly using Lemma 4.3.5 we can see that T_1 can be decomposed into at most 6 trees and T_2 can be decomposed into at most 7 trees of size at most k. This is shown in the Figure 4.3. Hence, in this case, we can decompose the tree T into 13 trees.

Case 2: $|T_1| \leq 3k - 2$. In this case, $|T_2| \leq 4k$. In this case, we can decompose T_1 into 5 trees and T_2 can be decomposed into 8 trees. This is shown in Figure 4.4. Thus in this case, we can decompose T into 13 trees. ■

Using Lemma 4.3.6, we can convert the bicriteria approximation algorithm for BCDS to a true approximation algorithm. In particular, we show the following -

Theorem 4.3.7 *There is a $\frac{1}{13}(1-\frac{1}{e})$ approximation algorithm for the* BCDS *problem.*

Proof: By Lemma 4.3.4, we obtain a tree T with at most $6k$ nodes with profit $(1-\frac{1}{e})\mathsf{OPT}$. Now using Lemma 4.3.6, we obtain 13 trees in the worst case, say $T_1, T_2, \ldots T_{13}$. Finally, out of these 13 trees (each of size at most k), we pick the tree $\tilde{T}$ with the highest total profit. Let, $p(T) = \sum_{v \in T} p(v)$ denote the total profit of tree T. Then we have,

$$p(\tilde{T}) \geq \frac{1}{13}\sum_{i=1}^{13} p(T_i) \geq \frac{1}{13} p(T) \geq \frac{1}{13}(1-\frac{1}{e})\mathsf{OPT}$$

Thus we have a $\frac{1}{13}(1-\frac{1}{e})$ approximation guarantee. ■

4.3.2.2 Finding the best subtree

Although the decomposition Lemma 4.3.6 is useful to prove a theoretical bound, from a practical perspective it is better to use a dynamic programming approach to find the best k sub-tree. Formally, we have the following problem. Given a tree $T = (V, E)$ of n vertices, profits on vertices $p : V \to \mathbb{Z}^+ \cup \{0\}$, and an integer k, find a subtree $\tilde{T}$ of k vertices which maximizes the total profit $\tilde{P} = \sum_{v \in \tilde{T}} p(v)$. We show that this problem can be solved in polynomial time using dynamic programming. Let the tree T be rooted at an arbitrary vertex and T_v denote the subtree rooted at a vertex v. We define the following:

$F(v, i) \leftarrow$ best solution of at most i vertices completely contained inside T_v.

$G(v,i) \leftarrow$ best solution of at most i vertices completely contained inside T_v such that v is a part of the solution.

The desired solution is thus at $F(root,k)$. The base cases (when v is a leaf) are trivial. Let $v_1, v_2, \ldots, v_l$ denote the children of vertex v. We now have the following recurrence :

$$F(v,i) = \max\left\{\max_{1\le j\le l}\{F(v_j,i)\}, G(v,i)\right\}$$

$$G(v,i) = p(v) + M(l, i-1)$$

Here $M(j,i')$ denotes the best way to distribute a budget of i' among the first j children of v. In other words,

$$M(l,i-1) = \max_{i_1+i_2+\ldots+i_l=i-1}\left\{\sum_j G(v_j,i_j)\right\}$$

$M(j,i')$ is computed using another dynamic program as follows. Again the base cases when $j=0$ or $i'=0$ are trivial. For $1 \le j \le l$ and $1 \le i' \le i-1$, we have the following recurrence:

$$M(j,i') = \max_{0\le i^*\le i'}\{M(j-1,i^*) + G(v_j, i'-i^*)\}$$

4.4 Budgeted generalized CDS

In this section, we show that our approach extends to more general budgeted connected domination problems. We first define a special kind of submodular function.

Let $G=(V,E)$ be an arbitrary graph. A function $f : 2^V \to \mathbb{Z}^+ \cup \{0\}$, is said to have the *special submodular* property if it satisfies the following-

- f is submodular. That is $f(A \cup \{v\}) - f(A) \geq f(B \cup \{v\}) - f(B) \quad \forall A, B, v$ such that $A \subseteq B \subseteq V$.
- $f_A(X) = f_{A \cup B}(X)$, if $N(X) \cap N(B) = \phi \quad \forall X, A, B \subseteq V$.

where $f_A(X) = f(A \cup X) - f(A)$ is the marginal profit of X given A and $N(X)$ denotes the neighborhood of X, including X itself.

Definition 4.4.1 (Budgeted generalized connected dominating set (BGCDS)) *Given a graph $G = (V, E)$, a budget k, and a monotone* special submodular *profit function $f : 2^V \to \mathbb{Z}^+ \cup \{0\}$, find a subset $S \subseteq V$ which maximizes $f(S)$ such that $|S| \leq k$ and S induces a connected subgraph of G.*

This problem captures the budgeted variant of the weighted profit connected dominating set problem. *Weighted profit connected dominating set.* In this variant, each vertex has an arbitrary profit which is obtained if it is dominated by some chosen vertex.

4.4.1 Algorithm.

Algorithm 15 begins by running the standard greedy algorithm to find a basis of the polymatroid associated with f. In other words, we greedily pick a vertex v with the maximum marginal profit $f(D \cup \{v\}) - f(D)$ until all vertices have zero marginal profit. With every selected vertex, we associate the marginal profit gained, and associate zero profit with the other vertices. Finally, we run a quota Steiner tree algorithm using these profits to find the smallest tree that yields a profit of at least $(1-\frac{1}{e})\mathsf{OPT}$ where OPT is the optimal profit (which we guess). In the analysis section, we show that there exists a tree $\hat{T}$ of size at most $3k$ with $f(\hat{T}) \geq (1-\frac{1}{e})\mathsf{OPT}$. Hence, the 2-approximation approximation for the quota Steiner tree yields a tree T of size at most $6k$ yielding the desired profit. Finally using the tree decomposition described earlier, we show that we can obtain a tree $\tilde{T}$ of size at most k with $f(\tilde{T}) \geq \frac{1}{13}(1-\frac{1}{e})\mathsf{OPT}$.

4.4.2 Analysis.

Let the L_1 denote the vertices in the optimal solution and $f(L_1) = \mathsf{OPT}$. Let L_2 denote the set of vertices which have at least one neighbor in L_1, and similarly let L_3 denote the set of vertices having a neighbor in L_2 (and NOT in L_1). Let $R = V \setminus \{L_1 \cup L_2 \cup L_3\}$ denote the rest of the vertices. Let $L'_i = D \cap L_i$ where D is the set of vertices chosen by the greedy algorithm.

Further, let D' denote the first k vertices picked by the greedy algorithm from $L'_1 \cup L'_2 \cup L'_3$. To simplify notation, let $D' = \{v_1, v_2, \ldots, v_k\}$ and let D_i denote the the set of vertices already picked by the greedy algorithm when the vertex v_{i+1} is being chosen. Hence we have $v_{i+1} = \mathsf{arg\,max}_{v \in V \setminus D_i} f(D_i \cup \{v\}) - f(D_i)$ and $p(v_{i+1}) = f(D_i \cup \{v_{i+1}\}) - f(D_i)$. Note that in particular $D_i \subseteq D$ but may not be a subset of D'. Also let $D'_i = \cup_{j=1}^{i} v_j$ denote the first i vertices in D'. Let $P(D'_i) = \sum_{v \in D'_i} p(v)$ denote the total profit associated with the set D'_i. Finally let $D''_i = D_i \setminus D'_i$ be the vertices in $D_i \cap R$.

Claim 4.4.2 $p(v_{i+1}) = P(D'_{i+1}) - P(D'_i) \geq \frac{1}{k}(\mathit{OPT} - P(D'_i))$

Proof: Consider the marginal profit of the set $L_1 \setminus D'_i$. Since $N(D''_i)$ does not intersect with $N(L_1)$, we have,

$$\begin{aligned}
f_{D'_i}(L_1 \setminus D'_i) &= f_{D'_i \cup D''_i}(L_1 \setminus D'_i) \\
&= f_{D''_i}(D'_i \cup (L_1 \setminus D'_i)) - f_{D''_i}(D'_i) \\
&\geq f_{D''_i}(L_1) - f_{D''_i}(D'_i) \\
&= f(L_1) - f_{D''_i}(D'_i) \\
&= \mathsf{OPT} - f_{D''_i}(D'_i) \qquad (4.1)
\end{aligned}$$

Let us now consider the term $f_{D''_i}(D'_i)$. Adding up over successive marginal profits,

$$\begin{aligned}
f_{D''_i}(D'_i) &= \sum_{j=1}^{i} f_{D''_i \cup D'_{j-1}}(v_j) \leq \sum_{j=1}^{i} f_{D_{j-1}}(v_j) \qquad (4.2) \\
&= \sum_{j=1}^{i} p(v_j) = P(D'_i)
\end{aligned}$$

From Eq (4.1) and Eq (4.2),

$$f_{D'_i}(L_1 \setminus D'_i) \geq \mathsf{OPT} - P(D'_i)$$

As f is submodular, we have

$$f_{D'_i}(L_1 \setminus D'_i) \leq \sum_{w \in L_1 \setminus D'_i} f_{D'_i}(\{w\})$$

Since $|L_1 \setminus D'_i| \leq k$, there exists at least one vertex $w \in L_1 \setminus D'_i$ satisfying

$$f_{D'_i}(\{w\}) \geq \frac{1}{k} f_{D'_i}(L_1 \setminus D'_i) \geq \frac{1}{k}(\mathsf{OPT} - P(D'_i))$$

Using $f_{D_i}(\{w\}) = f_{D'_i}(\{w\})$ and the fact that greedy picked v_{i+1} at this stage

$$\begin{aligned} p(v_{i+1}) = f_{D_i}(\{v_{i+1}\}) &\geq f_{D_i}(\{w\}) \\ &\geq \frac{1}{k}(\mathsf{OPT} - P(D'_i)) \end{aligned}$$

■

Solving the recurrence of Claim 4.4.2, we have $P(D') \geq (1 - \frac{1}{e})\mathsf{OPT}$. We thus have a set D' of size k which yields a total profit of at least $(1 - \frac{1}{e})\mathsf{OPT}$. We now proceed to show that the above set D' can be connected at a relatively low cost. Since every vertex in D' can be connected to L_1 using at most one vertex (from L_2), we can obtain a connected subset $\hat{T}$ of size at most $3k$ by choosing D', L_1 and vertices in L_2 as described. Hence, the 2-approximation for the QST problem will yield a tree T of size at most $6k$ which would give a profit of at least $(1 - \frac{1}{e})\mathsf{OPT}$. Finally applying the tree decomposition described earlier we obtain a tree $\tilde{T}$ of size

$\leq k$ with $f(\tilde{T}) \geq P(\tilde{T}) \geq \frac{1}{13}(1-\frac{1}{e})\mathsf{OPT}$.

Algorithm 15 Greedy profit labeling algorithm for BGCDS.

Input: graph $G = (V, E)$, a monotone special submodular function $f : 2^V \to \mathbb{Z}^+ \cup \{0\}$ and $k \in \mathbb{Z}^+ \cup \{0\}$.

Output: tree $\tilde{T}$ with at most k vertices.

```
1: run the generalized greedy dominating set routine (Algorithm 16) on $(G, f)$ to
   obtain a subset $D$ and a profit function $p : V \to \mathbb{N}$.
2: $\mathsf{OPT} \leftarrow$ profit of an optimal solution. (Guess using binary search 0 and $f(V)$).
3: $T \leftarrow$ 2-approximation for QST with quota $(1-\frac{1}{e})\mathsf{OPT}$.
4: use the dynamic program of Section 4.3.2.2 to find $\tilde{T}$, the best subtree of $T$
   having at most $k$ vertices.
```

Algorithm 16 Generalized greedy dominating set.

Input: graph $G = (V, E)$ and a monotone special submodular function $f : 2^V \to \mathbb{Z}^+ \cup \{0\}$.

Output: $D \subseteq V$ such that $f(D) = f(V)$ and profit function $p : V \to \mathbb{Z}^+ \cup \{0\}$.

```
1: initiate: $D \leftarrow \phi$
2: while $f(D) \neq f(V)$ do
3:     compute: $v \leftarrow \arg\max_{v \in V \setminus D} f(D \cup \{v\}) - f(D)$
4:     update: $p(v) \leftarrow f(D \cup \{v\}) - f(D)$ and $D \leftarrow D \cup \{v\}$
5: end while
6: for all $v \in V \setminus D$ do
7:     $p(v) \leftarrow 0$
8: end for
```

4.5 Partial generalized connected domination

We now consider a partial coverage version of the generalized connected domination presented in Section 4.4. In this problem, the goal is to find the smallest subset of vertices which induce a connected subgraph and have total profit at least q (quota).

Just as for the budgeted case, the algorithm proceeds by finding a spanning subset greedily. Using profits as defined by the greedy algorithm, we then find a QST having total profit at least q. In the analysis section, we show that there exists a tree $\hat{T}$ of size at most $2k \ln q + k$ with total profit at least q. Hence, the 2-approximation for QST yields a tree T of size at most $4k \ln q + 2k$ leading to a $O(4 \ln q)$ approximation.

4.5.1 Analysis

We reuse notation from Section 4.4 regarding the layers L_i and L'_i. Let D' denote the first $k \ln q + 1$ vertices picked by the greedy algorithm from $L'_1 \cup L'_2 \cup L'_3$. We now show that the total profit of vertices in D' is at least q.

Claim 4.5.1 $P(D') \geq q$

Proof: As per Claim 4.4.2, we obtain the following recurrence

$$P(D'_{i+1}) \geq (1 - (1 - \frac{1}{k})^{i+1})q \tag{4.3}$$

Substituting $i + 1 = k \ln q$, we get,

$$P(D'_{k \ln q}) \geq (1 - (1 - \frac{1}{k})^{k \ln q})q \tag{4.4}$$

$$\geq (1 - \frac{1}{q})q \geq q - 1 \tag{4.5}$$

Since profit function f is integral, we have

$$P(D'_{k \ln q+1}) \geq q \tag{4.6}$$

■

Theorem 4.5.2 *Given that the optimal solution is of size k, there exists a tree $\hat{T}$ of size at most $k \ln q + k + 2$ such that $\sum_{v \in \hat{T}} p(v) \geq q$*

Proof: In Claim 4.5.1 above, we have demonstrated the existence of a set of size at most $k \ln q + 1$ with the requisite total profit. We now show that this set can be connected at low cost. As in Theorem 4.2.3, we can see that by selecting at most $k \ln q + 1$ more vertices from layer L_2 and at most k vertices from layer L_1, the set D' can be connected to form a tree $\hat{T}$. ■

Finally using the 2-approximation for QST, we obtain a $O(4 \ln q)$ approximation.

Chapter 5

Covering Problems in Sensor and Radio Networks

5.1 Roadmap to the chapter

In Section 5.2, we study the *max-min k-cover* problem and describe a $\frac{1}{3}(1-\frac{1}{e})$ approximation algorithm for it. In fact, we present the same result for a more general *submodular max-min k-cover* problem.

The *minimum conflict-free coloring* problem is hard to approximate and an algorithm, with tight approximation guarantee, was obtained by Pach and Tardos [76]. We describe an alternate $2\sqrt{m}$ approximation algorithm for the *minimum conflict-free coloring* problem in general networks. These are detailed in Section 5.3.

We relax the *minimum conflict-free coloring* problem in the following two ways, in Section 5.4

1. We allow a small fraction of vertices in T to be left uncovered. We show that we can obtain a partition of expected size $O(\frac{\log m \log n}{\epsilon})$ such that at least $(1-\epsilon)m$ vertices in T are uniquely covered in expectation.

2. We relax the disjointness requirement by allowing a vertex $s \in S$ to appear in at most $O(\log n)$ sets. We now obtain a family of size $O(\log n \log m)$ that uniquely covers every $t \in T$.

5.2 The max-min k-cover problem

As mentioned in Chapter 1, one of the drawbacks of the *set k-cover* problem is that some set S_i in even the optimal solution can have very low coverage. In this section, we consider the *max-min k-cover* problem, defined formally as follows:

Definition 5.2.1 (*max-min k-cover*) *Given a sensor-target network $N = (S, T, E)$, the max-min k-cover problem is to partition S into exactly k disjoint sets $\mathcal{P} = (S_1, S_2 \ldots, S_k)$, such that we maximize the minimum coverage of any S_i, i.e., $\underset{\mathcal{P}}{\mathsf{Max}} \ \underset{i}{\mathsf{Min}} \ \mathcal{C}(S_i)$.*

Interestingly, we now show that an α-approximation algorithm for the *set k-cover* problem can be used to obtain comparable guarantees for the *max-min k-cover* problem.

Theorem 5.2.2 *Given a polynomial time α-approximation algorithm for the* set k-cover *problem, we can obtain a polynomial time $\alpha/3$-approximation algorithm for the max-min k-cover problem.*

Theorem 5.2.2 immediately yields the following result, as a corollary.

Corollary 5.2.3 *There is a $\frac{1}{3}(1 - \frac{1}{e})$ approximation algorithm for the max-min k-cover problem.*

Proof: Theorem 5.2.2 along with the $(1 - \frac{1}{e})$-approximation algorithm for the *set k-cover* problem by Abrams et al. [67] gives the result. ■

Instead of proving Theorem 5.2.2, we prove the theorem for the following more general submodular optimization problem.

Definition 5.2.4 (*submodular max-min k-cover*) *Given a set S (of sensors), an integer k, and a monotone, submodular profit function $f : 2^S \rightarrow \mathbb{Z}^+ \cup \{0\}$, the goal is to find a partition $\mathcal{P} = \{S_1, S_2 \ldots S_k\}$ of S so as to maximize $\underset{i}{\mathsf{Min}}\ f(S_i)$.*

This problem has the *max-min* version of the *capacitated set k-cover* problem studied by Deshpande et al. [70] as a special case. We now show the following.

Theorem 5.2.5 *Given a polynomial time α-approximation algorithm for the submodular set k-cover problem, we can obtain a polynomial time $\alpha/3$-approximation algorithm for the submodular max-min k-cover problem.*

Proof: Given the sensors set S and k, we first guess the optimal value, say M[1], such that there exists some partitioning of S into k sets such that each set S_i has $f(S_i) \geq M$.

We will first deal with the large profit vertices in S. For a constant μ_1 (to be specified later), if there is a sensor s with $f(\{s\}) \geq \mu_1 M$, we construct a singleton set with that sensor alone and delete it from S. Let $\mathcal{H}$ (resp. H) be the family of singleton sets (resp. set of high profit vertices) corresponding to high profit sensors and let $|\mathcal{H}| = k_1$. If $k_1 \geq k$, we stop. Otherwise, our goal is to now construct $k_2 = k - k_1$ more sets. Consider the reduced set $S' = S \setminus H$ after deleting the high profit sensors. We first observe that the optimal *submodular set k-cover* partitioning, into k_2 sets, has a total profit at least Mk_2 on this reduced instance. To see this, consider the optimal *submodular max-min k-cover* partitioning of the original instance. The removal of the k_1 high profit sensors affects at most k_1 sets

[1]Clearly, since $M \leq f(U)$, we have at most $f(U)$ different choices for M. Although $f(U)$ might be exponential, using a standard binary search approach, we can restrict ourselves to at most $\log f(U)$ different choices, i.e., a polynomial number of choices in the input size

in the optimal solution. Thus, there is a partitioning of the reduced instance into $k_2 = k - k_1$ sets, such that each set has a profit of at least M.

We now use the α-approximation algorithm for *submodular set k-cover* problem, to obtain a partition $\mathcal{P} = \{S_1, S_2 \dots S_{k_2}\}$ of $S \backslash H$ such that such that $\sum_{i \in [k_2]} f(S_i) \geq \alpha M k_2$. We will now describe a procedure that will construct a partition $\mathcal{P}'$ such that the minimum profit of each set is at least $\alpha M/3$.

Let μ_2 be a constant to be determined later. Let the set S_i have profit $f(S_i) \in [r_i M(\mu_1 + \mu_2) - M\mu_2, r_i M(\mu_1 + \mu_2) + M\mu_1)$, where $r_i \geq 0$. We will now show that we can decompose S_i into at least r_i disjoint subsets such that each subset has a profit of at least $M\mu_2$. It is easy to see that the claim is true when $r_i = 0$. We construct a new subset of S_i namely S_{new}, by removing arbitrary sensors from S_i and adding them to S_{new} until its profit just crosses $M\mu_2$. We note that since we have deleted all the sensors of profit $\geq M\mu_1$, we have, by submodularity of f, that $f(S_{new}) \leq M(\mu_1 + \mu_2)$. Clearly now, $f(S_i \setminus S_{new}) \geq f(S_i) - f(S_{new}) \geq (r_i - 1)M(\mu_1 + \mu_2) - M\mu_2$ and the claim follows from mathematical induction. In this way, we decompose each set S_i into r_i subsets.

At the end of the above decomposition, we call a subset "big" if its profit $\geq M\mu_2$, otherwise it is "small". Let $\mathcal{B}$ and $\mathcal{S}$ be the families of big and small subsets respectively. We now need to choose μ_1 and μ_2 to be such that the size of $\mathcal{B}$ is at least k_2. For this, we need the following to hold:

$$\sum_{i \in [k_2]} r_i \geq k_2 \tag{5.1}$$

But, by definition of r_i, we have the following:

$$\sum_{i\in[k_2]}(r_iM(\mu_1+\mu_2)+M\mu_1) \geq \sum_{i\in[k_2]} f(S_i) \geq \alpha M k_2$$

$$\therefore (\sum_{i\in[k_2]} r_i)(\mu_1+\mu_2)+k_2\mu_1 \geq \alpha k_2$$

Thus for Equation 5.1 to hold, it is sufficient to have

$$2\mu_1+\mu_2 \leq \alpha$$

Once this condition is satisfied, notice that $\mathcal{B}\cup\mathcal{H}$ together have a cardinality of at least $k_1+k_2=k$ such that the minimum profit of any set is $\mathsf{Min}\,(\mu_1,\mu_2)M$. The best approximation guarantee is thus obtained when $\mu_1=\mu_2=\frac{\alpha}{3}$. Thus we obtain a $\frac{1}{3}(1-\frac{1}{e})$ approximation. ■

Corollary 5.2.6 *There is a $\frac{1}{3}(1-\frac{1}{e})$ approximation algorithm for the submodular max-min k-cover problem.*

Proof: We first obtain a $(1-\frac{1}{e})$-approximation algorithm for the *submodular set k-cover* problem. To achieve this, we reduce it to the well known monotone submodular maximization problem subject to a matroid constraint [135], where the goal is : given a ground set U, a matroid $\mathcal{M}=(U,\mathcal{I})$ and a monotone submodular function $f':2^U\rightarrow\mathbb{Z}^+\cup\{0\}$, compute a subset $U'\in\mathcal{I}$ such that $f'(U')$ is maximized. This problem has a well known deterministic $1-\frac{1}{e}$ approximation algorithm.

Given an instance of the *submodular set k-cover* problem, define $U = \{(v, j) : v \in S,\ j \in [k]\}$ and for any $U' \subseteq U,\ f'(U') = \sum_j f(\{v : (v, j) \in U'\})$. To complete the reduction, we need to impose the restriction of the form if $(u, j) \in U'$ and $(u, j') \in U'$, then $j = j'$. This is achieved by imposing the (partition) matroid constraint where we set that at most one element can be chosen from the group $G_v = \{(v, j) : j \in [k]\}$. Now, the corollary follows from Theorem 5.2.5. ■

5.3 The minimum conflict-free coloring problem

In this section, we study the *minimum conflict-free coloring* (MCFC) problem, that can be formally defined as follows:

Definition 5.3.1 (conflict-free coloring (MCFC)) *We are given a radio network $N = (S, T, E)$ where $n = |S|$ and $m = |T|$. The goal is to find a partition $\mathcal{P}$ of S of minimum size, such that every vertex $t \in T$ is uniquely covered by $\mathcal{P}$.*

Equivalently, the goal is to color vertices of S using minimum number of colors such that for every $t \in T$ there is at least one color that is assigned to exactly one vertex in t's neighborhood. This problem can be viewed as a "disjoint" variant of the well studied *selective families* problem [77, 136, 137]. Clementi et al. [77] show that it is possible to choose $O(\log n \log m)$ subsets of S that cover all the vertices in T uniquely. However, the additional restriction of requiring disjoint subsets makes the problem significantly harder. The following theorem is implicit in the work of Even et al. [69] and we include the proof for completeness.

Theorem 5.3.2 *It is NP-hard to approximate the* MCFC *problem within a factor of* Max $(n^{1-\epsilon}, m^{1/2-\epsilon})$.

Proof: We show that if we can solve this problem within an approximation factor of Max $(n^{1-\epsilon}, m^{1/2-\epsilon})$ then we can approximate the chromatic number of a graph with an approximation factor of $n^{1-\epsilon}$ which is known to be NP-hard [138]. Given an instance $G = (V, E), k$ of the graph coloring problem, we construct a bipartite graph $N = (S, T, E')$ as follows. Set $S = V$ and $T = E$. For every edge $e = (u, v) \in E$, we add edges (e, u) and (e, v) in E'. We claim that G is k-colorable if and only if S can be partitioned into k sets such that every vertex in T is covered uniquely by at least one part.

Forward direction. If G has a feasible k-coloring, consider the partition of S where every set is a color class. Now, consider any vertex $e \in T$ and let $e = (u, v)$ be the corresponding edge in G. By definition of a feasible coloring u and v are assigned to different color classes, and hence e is uniquely covered (by both the color classes).

Backward direction. On the other hand, suppose S can be feasibly partitioned into k sets. As each $e \in T$ is uniquely covered, it must be the case that it's endpoints belong to different sets. Hence, assigning a unique color to each set creates a valid k-coloring of G.

Clearly in the above instance $m = |T| \leq n^2$. Therefore a $m^{1/2-\epsilon}$ approximation directly gives a $n^{1-\epsilon}$ approximation for the k-coloring problem. As it is NP-hard to approximate the chromatic number within $n^{1-\epsilon}$, the *minimum conflict-free coloring* problem is hard to approximate better than $\mathsf{Max}\ (n^{1-\epsilon}, m^{1/2-\epsilon})$. ■

To complement the $\Omega(m^{\frac{1}{2}-\epsilon})$ hardness result for MCFC, Pach and Tardos [76] show that a simple greedy algorithm can be used to find a partition of S of size $O(\sqrt{m})$ that uniquely covers all vertices of T. We present an alternate simple algorithm that provides a similar guarantee. The details are shown in Algorithm 17. The algorithm proceeds in iterations and constructs one new subset S_{new} in each iteration. We construct S_{new} to be a set such that every $s \in (S \setminus S_{new})$ is adjacent to a vertex t having degree 1 and every $t \in T$ has a neighbor in $S \setminus S_{new}$. We then remove all the degree 1 vertices in T and repeat this process until there are at most $\sqrt{m}$ vertices left in S. The key insight here is that once a vertex $t \in T$ attains degree 1 in some iteration, it can essentially be eliminated as it is guaranteed to be uniquely covered in some future iteration.

Algorithm 17 An $O(\sqrt{m})$-approximation for the *minimum conflict-free coloring* problem

```
1: Input: given a network $N = (S, T, E)$ with $|S| = n$ and $|T| = m$
2: Output: partition of size $k = 2\sqrt{m}$
3: $\mathcal{F} = \phi$
4: while $|S| \geq \sqrt{m}$ do
5:    $S_{new} = \phi$
6:    $U$ = set of degree 1 vertices in $T$
7:    while $\exists v \in S$ such that $N(v) \cap U = \phi$ do
8:       $S = S \setminus \{v\}$
9:       $S_{new} = S_{new} \cup \{v\}$
10:      add all newly formed one degree vertices to $U$
11:   end while
12:   $\mathcal{F} = \mathcal{F} \cup \{S_{new}\}$
13:   $T = T \setminus U$
14: end while
15: for $v \in S$ do
16:   $S = S \setminus \{v\}$
17:   $\mathcal{F} = \mathcal{F} \cup \{\{v\}\}$
18: end for
```

Theorem 5.3.3 *Given a radio network $N = (S, T, E)$, such that $|S| = n$ and $|T| = m$, there is a polynomial time algorithm that constructs a partition of size $2\sqrt{m}$, such that every vertex in T is covered exactly once by some subset of the partition. Thus, there is an algorithm with $2\sqrt{m}/k$ approximation guarantee for the MCFC problem, where k is the optimal partition size.*

Proof: For the Algorithm 17, we show that the following hold true -

1. Every vertex in T is uniquely covered: A vertex $t \in T$ is deleted in some iteration only if it has a single neighbor $s \in S$. Such a vertex t is guaranteed

to be uniquely covered by s irrespective of which subset s ends up in. On the other hand, if t is not deleted in any iteration, then it is uniquely covered trivially as all its neighbors form singleton sets.

2. The size of the partition obtained is at most $2\sqrt{m}$: We create one set in each iteration of the loop at step 4. By construction of S_{new}, every vertex in S at the end of an iteration is adjacent to a vertex in $t \in U$. Thus, $|U| \geq |S|$ at the end of each iteration. As $|S| \geq \sqrt{m}$ (except after the last iteration), we delete at least $\sqrt{m}$ vertices from T and hence can have at most $\sqrt{m}$ iterations. In addition, we create at most $\sqrt{m}$ singleton sets in Step 15.

■

5.4 Two relaxations of the MCFC problem

From Theorem 5.3.2, we infer that in the MCFC problem, the condition of "satisfying" all the vertices while maintaining disjointness property of various subsets of S, is too strict to achieve anything useful. In this subsection, we relax at least one of these conditions to obtain two interesting problems, that will hopefully be more tractable.

1. R1-MCFC. In this relaxation, we still require disjoint subsets of S but we only need to uniquely cover almost all vertices in T.

2. R2-MCFC. In this case, we are required to uniquely cover all the vertices,

however we relax on the disjointness property. We are now allowed to re-use a vertex a bounded number of times.

We now consider both of these relaxations.

5.4.1 The R1-MCFC problem

Interestingly, we show that if an $\epsilon > 0$ fraction of vertices in T are "sacrificed", it is possible to obtain much better solutions. Specifically, our algorithm partitions S into $O(\frac{1}{\epsilon} \log n \log m)$ disjoint sets, so that at least $(1-\epsilon)m$ of the nodes in T are uniquely covered.

Our algorithm is very simple: iteratively, use a subroutine for the *unique coverage* problem as long as possible. The key challenge in the analysis is to bound the number of vertices in T that are not uniquely covered and instead are left with no neighbors. The subroutine we use is a slightly modified version of Demaine et al. [68]'s algorithm. Algorithm 18 gives a formal description of this subroutine.

Algorithm 18 Modified subroutine for *unique coverage*

```
1: Input: graph $N = (S, T, E)$ and $\epsilon > 0$
2: partition vertices of $T$ into $\log n$ groups by degree as follows : $G_i \leftarrow \{t \in T \mid 2^i \leq deg(t) \leq 2^{i+1} - 1\}$
3: $i^* \leftarrow \arg\max_i |G_i|$
4: $S' \leftarrow$ sample each vertex from $S$ with probability $\frac{\epsilon}{2^{i^*}}$
5: return $S'$
```

The following proposition was essentially proved in [68], with slightly different probability calculations. We give the proof for completeness

Proposition 5.4.1 (Demaine et al. [68]) *In expectation, the number of vertices in T uniquely covered by S', returned by Algorithm 18, is at least $\frac{\epsilon m}{e^2 \log n}$.*

Proof: Since G_{i^*} is the group with the highest cardinality, $|G_{i^*}| \geq \frac{m}{\log n}$. Consider any vertex $v \in G_{i^*}$ having degree d. The probability that v is uniquely covered by S' is $d(\frac{\epsilon}{2^{i^*}})(1 - \frac{\epsilon}{2^{i^*}})^{d-1} \geq \frac{\epsilon}{e^2}$. Thus in expectation, S' covers at least $\frac{\epsilon m}{e^2 \log n}$ vertices uniquely. ■

It is now easy to describe our algorithm (refer to Algorithm 19). In each iteration, we pick a subset of vertices in S using the modified *unique coverage* subroutine (Algorithm 18). To ensure that different subsets chosen are disjoint, we delete the vertices in S that have already been selected. We also delete any vertex $t \in T$ that has been uniquely covered or is left without a neighbor in S.

Algorithm 19 Algorithm for R1-MCFC

```
1: Input: graph $N = (S, T, E)$, $\epsilon > 0$
2: initialize : $\mathcal{F} = \phi$ and $i = 1$
3: while $T \neq \phi$ do
4:     $S_i \leftarrow$ modified unique coverage algorithm (Algorithm 18) on $(N, \epsilon)$
5:     $S = S \setminus S_i$ $U_i \leftarrow$ vertices uniquely covered by $S_i$
6:     $D_i \leftarrow \{v : v \in T$ and $deg(v) = 0\}$
7:     update : $T = T \setminus (U_i \cup D_i)$, $\mathcal{F} = \mathcal{F} \cup \{S_i\}$ and $i = i + 1$
8: end while
9: return $\mathcal{F}$
```

Theorem 5.4.2 *For $\epsilon > 0$, the following hold true in Algorithm 19*

(a) The expected size of the partition is $O(\frac{1}{\epsilon} \log n \log m)$.

(b) The expected number of vertices in T that are uniquely covered is at least $(1 - \epsilon)m$.

Proof: *(a)* In Algorithm 19, we create a new subset in each iteration and thus the size of the partition is equal to the number of iterations. In iteration i, let t_i be the random variable corresponding to the cardinality of $U_i \cup D_i$, i.e., $t_i = |U_i \cup D_i|$. Also let T_i be a random variable denoting the number of vertices in T remaining after i iterations. We have,

$$T_i = T_{i-1} - t_i$$
$$\mathbb{E}(T_i) = \mathbb{E}(T_{i-1}) - \mathbb{E}(t_i)$$

From Proposition 5.4.1, we know that, for $c \geq \frac{e^2}{\epsilon}$

$$\mathbb{E}(t_i) \geq \frac{\mathbb{E}(T_{i-1})}{c \log n}$$
$$\mathbb{E}(T_i) \leq \mathbb{E}(T_{i-1}))(1 - \frac{1}{c \log n}) \tag{5.2}$$

Solving the above recurrence (Equation 5.2), we obtain that $\mathbb{E}(T_i) \leq me^{\frac{-i}{c \log n}}$. Thus, if we stop after $c \log n \log m$ iterations, the expected number of vertices in T_i is at most 1. The last remaining vertex $t \in T_i$ has at least one remaining neighbor $s \in S$ (otherwise $t \in D_j$ for some $j < i$) and can thus be satisfied by adding s as singleton. Hence we obtain a partition of expected size $O(\frac{1}{\epsilon} \log n \log m)$.

(b) We first note that a vertex of T is not uniquely covered if and only if it was included in the set D_i of some iteration of Algorithm 19. We now upper bound the probability of this "bad" event occurring by ϵ. To this end, we fix a vertex $t \in T$,

an iteration j of Step 3 and an integer $d \geq 2$. We now define the following events:

$E_t^{good} \lhd$ Event that t is uniquely covered in some iteration

$E_t^{bad} \lhd$ Event that t is not uniquely covered in any iteration

$E_{t,j,d} \lhd$ Event that t has d neighbors at the start of iteration j

$E_{t,j,d}^{last} \lhd$ Event that $E_{t,j,d}$ holds and t is removed from T at the end of iteration j

We claim that, $Pr(E_t^{bad}|E_{t,j,d}^{last}) \leq \epsilon$. Let $p_j(= \frac{\epsilon}{2^{i^*}} < \frac{1}{2})$ be the sampling probability for iteration j. We now have the following equations,

$$\frac{Pr(E_t^{bad}|E_{t,j,d}^{last})}{Pr(E_t^{good}|E_{t,j,d}^{last})} = \frac{Pr(E_t^{bad} \wedge E_{t,j,d}^{last})}{Pr(E_t^{good} \wedge E_{t,j,d}^{last})} = \frac{Pr(E_t^{bad} \wedge E_{t,j,d}^{last}|E_{t,j,d})}{Pr(E_t^{good} \wedge E_{t,j,d}^{last}|E_{t,j,d})} = \frac{(p_j)^d}{dp_j(1-p_j)^{d-1}}$$

The second equality follows from $E_{t,j,d}^{last} \wedge E_{t,j,d} = E_{t,j,d}^{last}$. Also, we have that $Pr(E_t^{bad}|E_{t,j,d}^{last}) + Pr(E_t^{good}|E_{t,j,d}^{last}) = 1$ and hence,

$$Pr(E_t^{bad}|E_{t,j,d}^{last}) = \frac{(p_j)^d}{(p_j)^d + dp_j(1-p_j)^{d-1}} \leq p_j \leq \epsilon$$

The second inequality follows from basic algebra using $p_j < \frac{1}{2}$ and $d \geq 2$. Finally, we have the following

$$Pr(E_t^{bad}) = \sum_{j,d} Pr(E_t^{bad}|E_{t,j,d}^{last}) Pr(E_{t,j,d}^{last}) \leq \epsilon \sum_{j,d} Pr(E_{t,j,d}^{last}) \leq \epsilon$$

Thus $Pr(E_t^{good}) \geq (1-\epsilon)$. Hence, we have $\mathbb{E}[\#$ vertices covered uniquely$] \geq (1 -$

$\epsilon)m$. ■

5.4.2 The R2-**MCFC** problem

We now consider the second relaxation, where we are allowed to re-use a vertex from S in a bounded number of subsets and our goal is to uniquely cover all the vertices in T. It is interesting to note that the randomized algorithm of Clementi et al. [77], that yields a partition of size $O(\log n \log m)$, also implicitly guarantees that a vertex is used $O(\log m)$ times in *expectation*. Our algorithm improves this result, by obtaining a partition of size $O(\log n \log m)$ such that every vertex is (deterministically) used $O(\log m)$ times. As a byproduct, we show that we can obtain a partition of size $O(\log n)$ that uniquely covers at least a constant number of vertices in T. The key ingredient of our algorithm is to sample vertices in a dependent fashion, in contrast with the previous approaches.

Theorem 5.4.3 *Given any bipartite graph $N = (S, T, E)$, we can partition S into $\log n$ sets such that at least $\frac{m}{e^2}$ vertices in T are uniquely covered.*

Proof: Consider the following randomized scheme:

- create $\log n$ empty sets - $S_1, S_2, \ldots, S_{\log n}$
- for every vertex $s \in S$
  - assign s to one of $S_1, S_2, \ldots, S_{\log n}$ such that the probability that s is assigned to $S_i = \frac{1}{2^i}$

The random process described above is well defined as :

$$\sum_i Pr(s \text{ is assigned to } S_i) = \sum_{i=1}^{\log n} \frac{1}{2^i} \leq \sum_{i=1}^{\infty} \frac{1}{2^i} = 1$$

For the sake of analysis, partition the nodes in T by their degree. Let G_i denote the nodes in T having degree d such that $2^i \leq d < 2^{i+1}$. We now show that set S_i uniquely covers at least $\frac{1}{e^2}$ fraction of nodes in G_i in expectation. Let v be any node in G_i.

$$Pr(v \text{ is uniquely covered by } S_i) = (\frac{d}{2^i})(1 - \frac{1}{2^i})^{d-1} \geq \frac{1}{e^2}$$
$$\therefore \mathbb{E}[\# \text{ of vertices uniquely covered by } S_i] \geq \frac{1}{e^2}|G_i|$$

Summing over all i, we get

$$\mathbb{E}[\# \text{ of vertices uniquely covered}] \geq \frac{1}{e^2}\sum_i |G_i| = \frac{m}{e^2}$$

It is easy to derandomize the above algorithm, using the standard technique of conditional expectation [139]. ■

Corollary 5.4.4 *There is a polynomial time algorithm, that given a network $N = (S, T, E)$ with $|S| = n$ and $|T| = m$, yields a family of subsets $\mathcal{F}$, such that $|\mathcal{F}| = O(\log n \log m)$ and every vertex of T is covered exactly once by at least one subset and every vertex $s \in S$, belongs to $O(\log m)$ different subsets.*

Proof: Repeat the algorithm in Theorem 5.4.3, until all the vertices in T are uniquely covered, while removing the vertices in T that have already been uniquely covered. Since, in each iteration, the number of vertices in T decreases by a constant factor, in $O(\log m)$ iterations, all the vertices in T are uniquely covered. Clearly, every vertex is "reused" at most once in every iteration. Hence, we have our claim. ■

Chapter 6

Steiner Tree and Cheapest Tour in Oracle Model

6.1 Road map to the chapter

As discussed in Chapter 1, Thorup and Zwick's work [11] directly yields oracles for *Steiner tree* and *cheapest tour* problems that can answer these queries with an approximation guarantee of $4\ell - 2$ and $3\ell - 1.5$ respectively, for any given parameter $\ell \geq 1$. The preprocessing time for these data structures is $O(\ell m n^{1/\ell})$ time and the data structure size itself is $O(\ell n^{1+1/\ell})$, same as the distance oracle. A natural open problem that we consider in this chapter is to improve the approximation guarantees for the *Steiner tree* and *cheapest tour* queries while maintaining the same space-time complexity for the preprocessing and query algorithms.

Given a weighted graph $G = (V, E)$ with $n = |V|$, $m = |E|$, and a query set, S, with $|S| = k$. In Section 6.3, we prove the following main result.

Theorem 6.1.1 *For a given $\ell \geq 1 \in \mathbb{N}$, we can preprocess G in $O(\ell m n^{1/\ell})$ time and construct a data structure of size $O(\ell n^{1+1/\ell})$ such that a Steiner tree query can be answered in $O(\ell k^2)$ time with an approximation guarantee $3\ell + 2$.*

Subsequently, we will show a similar result for the *cheapest tour* problem.

Theorem 6.1.2 *For a given $\ell \geq 1 \in \mathbb{N}$, we can preprocess G in $O(\ell m n^{1/\ell})$ time and construct a data structure of size $O(\ell n^{1+1/\ell})$ such that a cheapest tour query* *can be answered in $O(\ell k^2)$ time with an approximation guarantee $2.5\ell + 1.5$.*

Our preprocessing algorithm is a modified version of TZ's data structure. The key technical difficulty comes in the query processing algorithm and its analysis. Since we rely on the distance oracle ideas heavily, we start with a brief discussion in Section 6.2.

6.2 A brief introduction to distance oracles

We start with some notation. The graph that we work on is generally denoted by $G = (V, E)$, where $|V| = n$ and $|E| = m$. The query set for the Steiner tree and cheapest tour problems is a subset of vertices denoted by $S \subseteq V$, where $|S| = k$. The distance between a pair of vertices u, v in G is denoted by $d_G(u, v)$ or when clear from the context just $d(u, v)$. A shortest path metric graph on a subset of vertices K is a complete graph on K where edge weights are the lengths of the shortest paths between the end points. For a given set of terminals K and a subgraph H we denote

- the shortest path metric graph on K with respect to H by $H[K]$
- an optimal Steiner tree and cheapest tour on K with respect to G by $\mathsf{OST}(K)$ and $\mathsf{OCT}(K)$ respectively
- a minimum spanning tree in a subgraph H by $\mathsf{MST}(H)$
- the sum of edge weights of H by $\mathsf{cost}(H)$.

We start with a brief overview of the work of Thorup and Zwick [11]. There are two aspects involved namely a preprocessing stage that generates a data structure from the given graph, a query processing algorithm that uses this data structure to answer queries.

Sampling preprocessing algorithm [11]. The algorithm works by randomly and recursively sampling vertices of the graph. Given a graph $G = (V, E)$, it constructs a series of randomized subsets $A_{\ell-1} \subseteq A_{\ell-2} \subseteq A_{\ell-3} \ldots \subseteq A_1 \subseteq A_0 = V$, as shown in Algorithm 20. We denote the distance between two vertices u, v by $d(u, v)$. Algorithm 20 constructs for every vertex a set of *landmark* nodes B_v and computes and stores distances from v to each vertex in B_v. TZ show that the size of each B_v is $O(\ell n^{1/\ell})$ and therefore the total size of the data structure is $O(\ell n^{1+1/\ell})$. Slightly modifiying Dijkstra's algorithm, TZ further show that all such distances may be computed in time $O(\ell m n^{1/\ell})$.

Algorithm 20 TZ's sampling preprocessing algorithm

```
 1: initialize $A_0 \leftarrow V$
 2: for all $i = 1$ to $\ell - 1$ do
 3:     sample vertices of $A_{i-1}$ with uniform probability $n^{-1/\ell}$ to obtain $A_i$
 4: end for
 5: for all $v \in V$ do
 6:     for all $i \in [0, \ell - 1]$ do
 7:         $s_i(v) \leftarrow \mathsf{arg\,min}_{w \in A_i} d(v, w)$
 8:         $B_v^i \leftarrow \{w : w \in A_{i-1} \text{ and } d(v, w) \leq d(v, s_i(v))\}$
 9:     end for
10:     $B_v = \bigcup_{i \in [0,\ell-1]} B_v^i$
11:     compute and store distances from $v$ to every vertex in $B_v$
12: end for
```

Oscillating query algorithm [11]. The second phase is the query processing algorithm. Given two vertices u, v, the query processing algorithm returns the (approximate) distance between them. Algorithm 21 is a formal description of TZ's query algorithm. We call it an *oscillating* algorithm because in every iteration we swap x (initialized to u) and y (initialized to v). TZ show that if the algorithm does not terminate when the iterator $i = l$, then $d(x, s_{l+1}(x)) \leq (l+1) \times d(u, v)$. Further, it can be shown [11] that $d(x,z)+d(y,z) \leq d(x,z)+d(x,z)+d(x,y) \leq (2\ell-1)d(u,v)$.

Algorithm 21 TZ's oscillating query algorithm

```
1: initialize $x \leftarrow u$, $y \leftarrow v$
2: for all $i \in [0, \ell - 1]$ do
3:     $z \leftarrow s_i(x)$
4:     if $z \in B_y$ then
5:         return $d(x,z) + d(z,y)$
6:     end if
7:     swap $x \leftrightarrow y$
8: end for
```

Facts [**11**]. We now state some of the facts established by TZ [11].

1. **Fact 1**. If the oscillating algorithm does not terminate when $i = l$, then $d(x, s_{l+1}(x)) \leq (l+1) \times d(u,v)$. This is shown in Lemma 3.3 of [11].

2. **Fact 2**. If the oscillating algorithm does terminate when $i = l$, $d(x, z) + d(z, y) \leq (2l + 1)d(u, v)$. This is the approximate distance between (u, v) found by the Algorithm 21, denoted by $d_{alg}(u,v)$. Since $i \leq \ell - 1$, we have $d_{alg} \leq (2\ell - 1)d(u,v)$. This is proven in the Lemma 3.3 of [11].

3. **Fact 3**. The expected size of B_v is $O(\ell n^{1/\ell})$ and we can check membership in B_v in $O(1)$ amortized time using a 2-hashing data structure. This is shown in Lemma 3.2 of [11].

4. **Fact 4**. The expected size of A_i is atmost $n^{1-i/\ell}$. This follows from the fact the A_i is formed by sampling vertices of A_{i-1} with probability $n^{-1/\ell}$.

6.3 Steiner tree and cheapest tour oracles

As mentioned earlier, Kou, Markowsky and Berman [92] showed that computing the shortest path metric on a given set of vertices S, followed by the computation of a minimum spanning tree yields a 2-approximation algorithm for the Steiner tree problem. Formally, the problem of Steiner tree can be defined as follows.

Definition 6.3.1 (Steiner tree) *In an instance $(G(V,E), S, w)$, we are given an undirected simple graph $G = (V,E)$, a weight function $w : E \to \mathbb{R}^+ \cup \{0\}$ and a subset of vertices $S \subseteq V$, the goal is to find a minimum cost tree that contains all the vertices in S.*

Theorem 6.3.2 ([92]) *Let $(G(V,E), w, S)$ be an instance of the Steiner tree problem. If ξ is an edge of maximum weight in $\mathsf{MST}(G[S])$*

$$\mathsf{cost}(\mathsf{MST}(G[S])) \leq 2\mathsf{cost}(\mathsf{OST}(S)) - w(\xi)$$

In a slightly weaker sense, a minimum spanning tree on $G[S]$ is a $2-1/|S|$-approximation of the optimal Steiner tree on S in G. A direct corollary of Theorem 6.3.2 is that if we use an α approximate shortest path metric (where weights of an edge is the α approximate distance between its end points), we obtain a 2α approximation for the Steiner tree problem. Therefore, using the $2\ell - 1$ approximate distance oracle, we can compute a $2\ell - 1$ approximate shortest path metric in time $O(\ell k^2)$. From the corollary of Theorem 6.3.2, this directly yields a $2(2\ell - 1) = 4\ell - 2$ approximation guarantee for the Steiner tree query problem.

Formally the cheapest tour problem can be defined as follows.

Definition 6.3.3 (cheapest tour) *In an instance $(G(V, E), S, w)$, we are given an undirected simple graph $G = (V, E)$, a weight function $w : E \to \mathbb{R}^+ \cup \{0\}$ and a subset of vertices $S \subseteq V$, the goal is to find a minimum cost tour that contains all the vertices in S.*

In the case of this problem, an algorithm due to Christofides [93] gives a 3/2 approximation guarantee. Algorithm 22 is a brief description of the Christofides algorithm. It can be shown that the cost of T_S is at most the cost of the optimal cheapest tour on S, in G and $\mathsf{cost}(M_O)$ is at most half the cost of the optimal cheapest tour. Thus, this algorithm gives a 3/2 approximation guarantee. Clearly, if we have access to an α approximate distance oracle, this approximation guarantee blows up to $3\alpha/2$. Hence, using TZ's distance oracle, we immediately obtain a $3(2\ell - 1)/2 = 3\ell - 1.5$ approximation guarantee for the cheapest tour problem.

Algorithm 22 Christofides algorithm for the cheapest tour problem

```
1: compute the shortest path metric $G[S]$ on $S$
2: compute a minimum spanning tree $T_S$ on $G[S]$
3: let $O$ be the set of odd degree vertices in $T_S$ and let $M_O$ be the minimum weight
   perfect matching, in $G[S]$, on the vertices of $O$
4: return $T_S \cup M_O$
```

We now our describe our new preprocessing and query algorithms for the problems of Steiner tree and cheapest tour.

6.3.1 **Preprocessing algorithm**

We use a slightly modified version of TZ's preprocessing algorithm. In this modified data structure, along with the distances stored by TZ's distance oracle, we store the distances corresponding to every pair of vertices in A_r, where $r = \lceil \frac{\ell-1}{2} \rceil$. Algorithm 23 gives a formal description of our modified preprocessing algorithm. We prove the following easy observation.

Observation 6.3.4 *The total additional space required by the data structure is* $O(n^{1+1/\ell})$

Proof: From the Fact 4, in Section 6.2, we have $|A_r| \leq n^{1-r/\ell} \leq n^{1-(\ell-1)/2\ell} = n^{1/2+1/2\ell}$. Therefore, the total space required is $O(|A_r|^2) = O(n^{(1/2+1/2\ell)2}) = O(n^{1+1/\ell})$.

■

We note that the total size of our data structure is of the same order as that of the TZ's data structure.

Algorithm 23 Modified preprocessing algorithm

```
1: Input: edge weighted graph $G = (V, E)$ and $\ell \geq 1$
2: Output: distance oracle data structure: $\mathcal{D}$
3: construct TZ [11]'s distance oracle, $\mathcal{D}'$, on $G$ with parameter $\ell$   $r \leftarrow \lceil \frac{\ell-1}{2} \rceil$
4: compute and store the distance between every pair of vertices in $A_r$.
5: let $\mathcal{D}''$ denote these additional shortest paths
6: return $\mathcal{D} = \mathcal{D}' \cup \mathcal{D}''$
```

6.3.2 Query algorithms

In this subsection, we describe improved query algorithms for the Steiner tree and cheapest tour problems. We begin with some notation.

Notation. We recall that the preprocessing step constructs a family of recursively sampled subsets $A_{\ell-1} \subseteq A_{\ell-2} \subseteq \ldots \subseteq A_0$. For any vertex v, let $s_i(v)$ denote a vertex in A_i (for any $i \in [0, \ell-1]$), that is closest to v. We call the vertex $s_r(v)$, where $r = \lceil \frac{\ell-1}{2} \rceil$, the hook vertex of v denoted by $\mathfrak{h}(v)$, and the shortest path connecting v to $\mathfrak{h}(v)$ as the hook path of v, denoted by $\mathfrak{hp}(v)$. We note that all the hook path weights are stored in our modified distance oracle. Further for any $S' \subseteq S$, we define the hook vertex set, as the set of hook vertices $\mathcal{H}(S') = \{\mathfrak{h}(v) : v \in S'\}$ and the hook path set, as the set of hook paths $\mathcal{HP}(S') = \{\mathfrak{hp}(v) : v \in S'\}$.

Intuition. The intuition behind our query algorithm is the following. If the total cost of the hook paths of S (i.e., $\mathcal{HP}(S)$) is "low", then the cost of optimal Steiner tree (similarly cheapest tour) on S and that on $\mathcal{H}(S)$ must be nearly equal. Indeed, adding $\mathcal{HP}(S)$ to an Steiner tree on $\mathcal{H}(S)$, directly yields an Steiner tree on S (and vice-versa). Now, since we have stored the exact distances between all pairs of vertices in $\mathcal{H}(S) \subseteq A_r$, we can use Theorem 6.3.2 to obtain a 2 approximation guarantee for Steiner tree on $\mathcal{H}(S)$. This in turn yields a good approximation for Steiner tree on S. On the other hand if the cost of hook paths set is "high" in some sense, it can be shown that TZ's algorithm terminates within r iterations for most of the pairs of vertices in S and therefore, on these pairs we only loose a factor of $\ell - 1$ on the distance computation (instead of $2\ell - 1$). This prompts us to run the oscillating algorithm only upto r iterations. For a given pair of vertices, if the oscillating algorithm terminates within r iterations, we will use that approximate distance. Otherwise, we will "hook" these vertices to A_r and compute a Steiner tree on the resulting hook vertex set. To capture this intuition, we introduce the notion of a *gray-black* graph.

Gray-black graph construction. Given a query set S, the gray-black graph is a complete weighted graph, with weight function $w : S \times S \to \mathbb{R}^+ \cup \{0\}$, constructed in the following way. For every pair of vertices u, v, we run the oscillating algorithm for $r = \lceil \frac{\ell - 1}{2} \rceil$ iterations. If the algorithm terminates within r iterations, we have a $2r - 1$ approximate distance between the pair of vertices u, v, denoted by $d_{alg}(u, v)$. We color such an edge gray and set the weight of the edge $w(u, v)$ to $d_{alg}(u, v)$. Otherwise, we color the edge black and set the weight of the edge to 2 times the maximum of the hook edges of u and v. The gray edges are "real" as they represent true paths of weight within an $\ell - 1$ factor of the actual distance. On the other hand, black edges are merely placeholders and need to be further handled. The formal details of the construction are given in Algorithm 24.

Algorithm 24 Construction of gray-black graph.

```
1: Input: a distance oracle $\mathcal{D}$ on graph $G = (V, E)$ and a query set $S \subseteq V$
2: Output: a gray-black graph $\mathcal{GB}$ on $S$
3: $r \leftarrow \lceil \frac{\ell-1}{2} \rceil$
4: for $v \in S$ do
5:     $m_v \leftarrow d(v, s_r(v))$
6: end for
7: initialize the gray-black graph $\mathcal{GB} = (S, E_{GB} = \phi)$
8: for $u, v \in S$ do
9:     add $e = uv$ to $E_{GB}$, that is, $E_{GB} \leftarrow E_{GB} + e$
10:    run the oscillating algorithm on $u, v$ for at most $r$ iterations
11:    if oscillating algorithm terminates before $j < r$ iterations then
12:        set $w(u, v) = d(u, s_j(u)) + d(v, s_j(v))$ and color $e$ gray
13:    else
14:        set $w(u, v) = 2\,\mathsf{Max}(m_u, m_v)$ and color $e$ black
15:    end if
16: end for
17: return $\mathcal{GB}$
```

Query algorithm for the Steiner tree problem. Algorithm 25 is a formal description of the query algorithm for computing a Steiner tree on S. We begin by constructing a gray-black graph $\mathcal{GB}$ on S and a minimum spanning tree $\mathsf{MST}(\mathcal{GB})$ over $\mathcal{GB}$. As noted earlier, while gray edges represent "real" paths, black edges do not and therefore cannot be used. Hence, the black edges are deleted from $\mathsf{MST}(\mathcal{GB})$ to obtain a forest F_{gr} with components $C_1, C_2, \ldots, C_p$. Now, to obtain a valid Steiner tree, we need to connect these components in some way. This is done by choosing representative vertices in A_r, for each component C_i, and then connecting these representative vertices. More precisely, we do the following. From each component C_i, we choose a vertex, w_i, with least cost hook path. We call the hook path of w_i as the hook path of the component C_i and denote, the set of all such vertices w_i, by

$\mathcal{R}$. Now, since $\mathcal{H}(\mathcal{R}) \subseteq A_r$ and because the distance between every pair of vertices is A_r is stored in our data structure, we have access to the shortest path metric on $\mathcal{H}(\mathcal{R})$. Hence, using Theorem 6.3.2 we can compute a good Steiner tree, denoted by $\hat{T}$, on $\mathcal{H}(\mathcal{R})$. We return $F_{gr} \cup \hat{T} \cup \mathcal{HP}(\mathcal{R})$ as the final Steiner tree, denoted by T_{alg}.

Algorithm 25 Steiner tree query algorithm.

```
1: Input:   a distance oracle $\mathcal{D}$ on graph $G = (V, E)$ and a query set $S \subseteq V$
2: Output:   a Steiner tree $T_{alg}$ on $S$
3: construct the gray-black graph on $S$, $\mathcal{GB}$, using Algorithm 24
4: compute the minimum spanning tree $\mathsf{MST}(\mathcal{GB})$ on $\mathcal{GB}$
5: delete all the black edges from $\mathsf{MST}(\mathcal{GB})$ to obtain a forest $F_{gr}$ that has
   $C_1, C_2, \ldots, C_p$ as components
6: let $\mathcal{R} = \{w_i : w_i \in C_i$, where $w_i$ is a vertex in $C_i$ with least cost hook path$\}$
7: use Theorem 6.3.2 to compute the Steiner tree $\hat{T}$ on $\mathcal{H}(\mathcal{R})$
8: return  $T_{alg} = \hat{T} \cup F_{gr} \cup \mathcal{HP}(\mathcal{R})$
```

Query algorithm for the cheapest tour problem. Algorithm 26 gives the formal details of the query algorithm for the cheapest tour problem. Again, we start by constructing the gray-black graph $\mathcal{GB}$ on S and compute the minimum spanning tree $\mathsf{MST}(\mathcal{GB})$ on $\mathcal{GB}$. We then delete all the black edges of $\mathsf{MST}(\mathcal{GB})$ to obtain a forest F_{gr} with components $C_1, C_2 \ldots C_p$. As in the query algorithm for Steiner tree we then define the set $\mathcal{R}$ and compute an approximate cheapest tour $\hat{C}$ on the vertices $\mathcal{H}(\mathcal{R})$. For any given subgraph H, we denote by H^{dbl} the subgraph obtained by duplicating the edges of H. We return the tour $C_{alg} = \hat{C} \cup \mathcal{HP}(\mathcal{R})^{dbl} \cup F_{gr}^{dbl}$.

Algorithm 26 Cheapest tour query algorithm.

```
1: Input: a distance oracle $\mathcal{D}$ on graph $G = (V, E)$ and a query set $S \subseteq V$
2: Output: a tour $C_{alg}$ on $S$
3: construct the gray-black graph on $S$, $\mathcal{GB}$, using Algorithm 24
4: compute the minimum spanning tree $\mathsf{MST}(\mathcal{GB})$ on $\mathcal{GB}$
5: delete all the black edges from $\mathsf{MST}(\mathcal{GB})$ to obtain a forest $F_{gr}$ that has
   $C_1, C_2, \ldots, C_p$ as components
6: let $\mathcal{R} = \{w_i : w_i \in C_i\}$, where $w_i$ is a vertex with least cost hook path in $C_i$
7: using Christofides [93] algorithm compute the cheapest tour $\hat{C}$ on $\mathcal{H}(\mathcal{R})$
8: return $C_{alg} = \hat{C} \cup F_{gr}^{dbl} \cup \mathcal{HP}(\mathcal{R})^{dbl}$
```

Analysis. We denote the shortest path metric on the set of vertices S, with respect to the original graph G by $G[S]$. We start by bounding the cost of the minimum spanning tree $\mathrm{T} = \mathsf{MST}(\mathcal{GB})$ on the gray-black graph in terms of the minimum spanning tree $\mathsf{MST}(G[S])$ on $G[S]$. For a weighted graph H, let $\mathsf{cost}(H)$ denote the aggregate weight of its edges. Further, let ξ denote an edge of maximum weight in T. We recall that $\mathsf{OST}(S)$ denotes the optimal Steiner tree on S.

Lemma 6.3.5 $\mathit{cost}(T) \leq \ell \times \mathit{cost}(\mathit{MST}(G[S])) \leq 2\ell \times \mathit{cost}(\mathit{OST}(S)) - w(\xi)$

Proof: We show that for any pair of vertices, u, v, in the gray-blak graph $\mathcal{GB}$, we have $w(u, v) \leq \ell \cdot d(u, v)$. Indeed, if $e = uv$ is colored gray, then by definition the oscillating algorithm terminates before $i < r = \lceil \frac{\ell-1}{2} \rceil$ iterations. From Fact 2, in Section 6.2, it follows that $w(u, v) = d_{alg} \leq (2(r-1)+1)d(u, v) \leq (\ell-1)d(u, v)$. On the other hand, let e be colored black. From Fact 1, in Section 6.2, we have $\mathsf{Max}(m_u, m_v) = \mathsf{Max}\ (d(u, s_r(u)), d(v, s_r(v))) \leq r \cdot d(u, v)$. Therefore, $w(u, v) = 2\,\mathsf{Max}(m_u, m_v) \leq 2r \cdot d(u, v) \leq \ell \cdot d(u, v)$. For the sake of analysis, we consider the metric $G'[S]$ on S with distance function $d'(u, v) = \ell \cdot d(u, v)$. We have established that $w(u,v) \leq d'(u,v)$. Let $\mathfrak{e}'$ be the edge with maximum weight in $G'[S]$. We have the following equations

$$\mathsf{cost}(T) \leq \mathsf{cost}(\mathsf{MST}(G'[S])) = \ell \cdot \mathsf{cost}(\mathsf{MST}(G[S])) \tag{6.1}$$

By Theorem 6.3.2 and Equation 6.1

$$\mathsf{cost}(T) \leq 2\ell \cdot \mathsf{cost}(\mathsf{OST}(S)) - w(\xi') \tag{6.2}$$

Also, since $w(\xi) \leq \ell \cdot d(\xi) \leq \ell \cdot d(\xi')$, by Equation 6.2

$$\mathsf{cost}(T) \leq 2\ell \cdot \mathsf{cost}(\mathsf{OST}(S))) - w(\xi)$$

Hence, the lemma. ∎

We now proceed to prove a crucial lemma that bounds the cost of hook path set $\mathcal{HP}(\mathcal{R})$ and the gray forest F_{gr}.

Lemma 6.3.6 *Let F_{gr} and $\mathcal{HP}(\mathcal{R})$ be as defined by Algorithm 25 and Algorithm 26. The following bounds hold.*

1. *$\mathsf{cost}(F_{gr}) \leq 2\ell \times \mathsf{cost}(\mathsf{OST}(S)) - 2\mathsf{cost}(\mathcal{HP}(\mathcal{R}))$; Therefore, $\mathsf{cost}(\mathcal{HP}(\mathcal{R})) \leq \ell \times \mathsf{cost}(\mathsf{OST}(S))$*

2. *$\mathsf{cost}(F_{gr}) \leq \ell \times \mathsf{cost}(\mathsf{OCT}(S)) - 2\mathsf{cost}(\mathcal{HP}(\mathcal{R}))$; Therefore, $\mathsf{cost}(\mathcal{HP}(\mathcal{R})) \leq \frac{\ell}{2} \times \mathsf{cost}(\mathsf{OCT}(S))$*

Proof: We recall that F_{gr} is formed by deleting all the black edges from the minimum spanning tree T of the gray-black. Let the components of F_{gr} be C_1, C_2, …, C_p. We construct a tree T_{blk} from T by shrinking each component C_i to a single vertex v_i. Among all the hook paths of vertices in C_i we pick the least hook path m_i and associate it with v_i.

We first obtain a lower bound on the weight of T_{blk} in terms of the cost of $\mathcal{HP}(\mathcal{R})$. To this end, we use a charging argument to show that we can pay for every hook path, except one, twice using the weight on the edges of T_{blk}. We note that each vertex of T_{blk} has at most one hook path in $\mathcal{HP}(\mathcal{R})$ associated with it.

Charging Scheme. We root T_{blk} at a vertex r that has the least weight hook path m_r in $\mathcal{HP}(\mathcal{R})$. Now, we recursively use the following charging scheme until all the edges from T_{blk} deleted. Pick a leaf of T_{blk}, say i. Let C_i be the component in F_{gr} that on contraction resulted in the leaf i. Also, let m_i be the hook path in $\mathcal{HP}(\mathcal{R})$ associated with C_i and $e = (u, v)$ be the *only* black edge incident on C_i (since i is a leaf in T_{blk}), with $v \in C_i$. By definition, since e is a black edge - $2m_v \leq 2\,\mathsf{Max}(m_u, m_v) = w(u, v)$. By the choice of m_i, we have $m_i = (\min_{w \in C_i} m_w) \leq m_v$. Therefore, $2m_i \leq w(u, v)$. Hence, we can charge off m_i twice onto the edge e. We now delete the hook path m_i from $\mathcal{HP}(\mathcal{R})$ and e from T_{blk}. Clearly, at the end of the above charging scheme, each edge of T_{blk} is charged twice by exactly one hook path. Additionally, the only hook path that we did not charge is m_r. Thus, we have $\mathsf{cost}(T_{blk}) \geq 2\mathsf{cost}(\mathcal{HP}(\mathcal{R})) - 2m_r$.

Now, for any black edge $e'_b = (u, v)$ incident on r, where $v \in C_r$, we have

$2m_r \leq 2m_v \leq w(e'_b)$. The following holds.

$$\mathsf{cost}(T_{blk}) \geq 2\mathsf{cost}(\mathcal{HP}(\mathcal{R})) - 2m_r \geq 2\mathsf{cost}(\mathcal{HP}(\mathcal{R})) - w(e'_b) \tag{6.3}$$

Choosing e'_b to be a black edge in T_{blk} and incident on r, and recalling that ξ is the edge of maximum weight in T, we have

$$\mathsf{cost}(T_{blk}) \geq 2\mathsf{cost}(\mathcal{HP}(\mathcal{R})) - w(\xi) \quad (\because w(e'_b) \leq w(\xi))$$

Applying Lemma 6.3.5

$$\mathsf{cost}(F_{gr}) = \mathsf{cost}(T) - \mathsf{cost}(T_{blk}) \leq (2\ell \times \mathsf{cost}(\mathsf{OST}(S)) - w(\xi)) - (2\mathsf{cost}(\mathcal{HP}(\mathcal{R})) - w(\xi))$$
$$\leq 2\ell \times \mathsf{cost}(\mathsf{OST}(S)) - 2\mathsf{cost}(\mathcal{HP}(\mathcal{R}))$$

Using a similar computation for the cheapest tour, we obtain that $\mathsf{cost}(F_{gr}) \leq \ell \times \mathsf{cost}(\mathsf{OCT}(S)) - 2\mathsf{cost}(\mathcal{HP}(\mathcal{R}))$. Finally, since $\mathsf{cost}(F_{gr}) \geq 0$, we have $\mathsf{cost}(\mathcal{HP}(\mathcal{R})) \leq \ell \times \mathsf{cost}(\mathsf{OST}(S))$ and $\mathsf{cost}(\mathcal{HP}(\mathcal{R})) \leq \frac{\ell}{2} \times \mathsf{cost}(\mathsf{OCT}(S))$ ■

Theorem 6.3.7 *Algorithm 25 yields a $3\ell + 2$ approximation for the Steiner tree problem.*

Proof: Let $T_{alg} = \hat{T} \cup F_{gr} \cup X$. It is easy to verify that T_{alg} is indeed a Steiner tree on the set of terminals S. We will first bound the cost of $\hat{T}$. Since, we have access to the shortest path metric on A_r and hence on $\mathcal{H}(\mathcal{R})$, using

Theorem 6.3.2 we can construct a Steiner tree on $\mathcal{H}(\mathcal{R})$ with a 2 approximation guarantee. As $\mathsf{OST}(S)\ \cup \mathcal{HP}(\mathcal{R})$ is a feasible Steiner tree on $\mathcal{H}(\mathcal{R})$, we have $\mathsf{cost}(\mathsf{OST}(\mathcal{H}(\mathcal{R}))) \leq \mathsf{cost}(\mathsf{OST}(S)) + \mathsf{cost}(\mathcal{HP}(\mathcal{R}))$. Also, from Theorem 6.3.2 we have $\mathsf{cost}(\hat{T}) \leq 2\mathsf{cost}(\mathsf{OST}(\mathcal{H}(R)))$. Thus $\mathsf{cost}(\hat{T}) \leq 2\mathsf{cost}(\mathsf{OST}(S)) + 2\mathsf{cost}(\mathcal{HP}(\mathcal{R}))$. From Lemma 6.3.6, we have $\mathsf{cost}(\mathcal{HP}(\mathcal{R})) \leq \ell \times \mathsf{cost}(\mathsf{OST}(S))$. Therefore, we have

$$\begin{aligned}
\mathsf{cost}(T_{alg}) &\leq \mathsf{cost}(\hat{T}) + \mathsf{cost}(F_{gr}) + \mathsf{cost}(\mathcal{HP}(\mathcal{R})) \\
&\leq 2\mathsf{cost}(\mathsf{OST}(S)) + 2\mathsf{cost}(\mathcal{HP}(\mathcal{R})) + \mathsf{cost}(F_{gr}) + \mathsf{cost}(\mathcal{HP}(\mathcal{R})) \\
&\leq 2\mathsf{cost}(\mathsf{OST}(S)) + 3\mathsf{cost}(\mathcal{HP}(\mathcal{R})) + 2\ell \times \mathsf{cost}(\mathsf{OST}(S)) - 2\mathsf{cost}(\mathcal{HP}(\mathcal{R})) \\
&\leq (2\ell + 2)\mathsf{cost}(\mathsf{OST}(S)) + \mathsf{cost}(\mathcal{HP}(\mathcal{R})) \leq (3\ell + 2)\mathsf{cost}(\mathsf{OST}(S))
\end{aligned}$$

Hence, the theorem. ■

Theorem 6.3.8 *Algorithm 26 yields a $2.5\ell + 1.5$ approximation for the cheapest tour problem.*

Proof: We have the following

$$\begin{aligned}
\mathsf{cost}(F_{gr}^{dbl}) = 2\mathsf{cost}(F_{gr}) &\leq 2[\ell \times \mathsf{cost}(\mathsf{OCT}(S)) - 2\mathsf{cost}(\mathcal{HP}(\mathcal{R}))] \\
&\leq 2\ell \times \mathsf{cost}(\mathsf{OCT}(S)) - 4\mathsf{cost}(\mathcal{HP}(\mathcal{R})) \qquad (6.4)
\end{aligned}$$

Since, $\mathsf{OCT}(S) \cup \mathcal{HP}(\mathcal{R})^{dbl}$ is a feasible tour on $\mathcal{H}(\mathcal{R})$

$$\mathsf{cost}(\hat{C}) \leq 1.5 \times \mathsf{cost}(\mathsf{OCT}(\mathcal{H}(\mathcal{R}))) \leq 1.5 \times \mathsf{cost}(\mathsf{OCT}(S)) + 3\mathsf{cost}(\mathcal{H}(\mathcal{R})) \tag{6.5}$$

Finally, we have

$\mathsf{cost}(C_{alg}) \leq \mathsf{cost}(\hat{C}) + \mathsf{cost}(F_{gr}^{dbl}) + \mathsf{cost}(\mathcal{H}(\mathcal{R})^{dbl})$

$\leq 3\mathsf{cost}(\mathcal{H}(\mathcal{R})) + 1.5\mathsf{cost}(\mathsf{OCT}(S)) + 2\ell \times \mathsf{cost}(\mathsf{OCT}(S)) - 4\mathsf{cost}(\mathcal{H}(\mathcal{R})) + 2\mathsf{cost}(\mathcal{H}(\mathcal{R}))$

$\leq 1.5\mathsf{cost}(\mathsf{OCT}(S)) + 2\ell \times \mathsf{cost}(\mathsf{OCT}(S)) + \mathsf{cost}(\mathcal{H}(\mathcal{R})) \leq (2.5\ell + 1.5)\mathsf{cost}(\mathsf{OCT}(S))$

Hence, the theorem. ■

Chapter 7

Future Work

In this thesis, we consider several optimization problems that arise from applications in computer networks. In all these applications, nodes of the networks have a certain *limiting* resource like storage capacity, computational resource or energy resources. Several problems remain widely open. We now list these problems, arranged according to the topic.

Resource replication problems. Although most of our results for the variants of resource replication problems are reasonably tight, some questions still remain open.

1. For the basic resource replication problem, Theorem 2.2.3 gives a 3 approximation guarantee. On the negative side, Theorem 2.2.12 shows that it is hard to obtain an algorithm with better than 2 approximation guarantee. An interesting problem is to close this gap, i.e., is there a better than 3 approximation algorithm for the basic resource replication problem? Similarly, is there a better than 5 approximation algorithm for the K-robust resource replication problem studied in Section 2.3?

2. Another interesting question arises in the context of capacitated resource replication problem. We obtain a $(4, 2)$ bi-approximation algorithm (Theorem 2.4.2) for the basic version of this problem. Is there a true approximation

algorithm for this problem? It would also be interesting to study the capacitated version of the subset resource replication problem.

Container selection problem. In Chapter 3, we study discrete and continuous variants of the container selection problem in d dimensions. We make considerable progress on this problem and yet several basic questions still remain unanswered.

1. In Section 3.4, we show prove that for $d \geq 3$ both the problems are NP-hard. Unfortunately, we do not have a reduction to prove the hardness in the case of $d = 2$. Further, for the discrete version of the problem, we know that there is no true approximation algorithm possible in the case of $d \geq 3$ dimensions. But is there a true approximation algorithm for the discrete problem in two dimensions? As noted in Chapter 3, the discrete version (even in higher dimensions) is a special case of the non-metric k-median problem. Hence, there is a known $(1, O(\log n))$ bi-approximation algorithm. We improve this guarantee to $(1, O(\log k))$ but we leave the question of whether one can find a $(1, O(1))$ bi-approximation guarantee open.

2. All our results have been focused on the fixed dimensions case. An interesting question is to study the problem in general arbitrary (non-fixed) number of dimensions. Interestingly here, although the discrete version is still a special case of the non-metric k-median, it is unclear how to approximate the continuous version with a guarantee of $(1, O(\log n))$ or better.

Connected dominating set problem. In Chapter 4, we considered partial and budgeted versions of the well studied connected dominating set problem. We obtain the first $O(\log n)$ approximation for the partial connected dominating set problem and a $\frac{1}{13}(1-\frac{1}{e})$ approximation for the budgeted version. We also extend our results to a *special submodular* problem, which includes capacitated and weighted profit versions of the PCDS and BCDS problems as special cases. Our results are tight up to a constant factor in all the cases. Apart from the natural open question of improving the constants in these approximation guarantees, the following are the key questions left open in this area.

1. Polynomial time approximation schemes (PTAS) are known for the basic connected dominating set problem in the special graphs like geometric graphs [10] and planar graphs [32]. Are there similar schemes for the partial and budgeted versions of the problem ?

2. The connected dominating set problem is well studied in the distributed framework [9]. It would be very interesting to also study the partial variants in this setting.

Miscellaneous. Finally, improving the approximation guarantees of the max-min k-cover problem, and query results for the Steiner tree and cheapest tour oracles are some of the important open problems for future work.

Bibliography

[1] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

[2] Bong-Jun Ko and Dan Rubenstein. Distributed server replication in large scale networks. In *International Workshop on Network and Operating Systems support for Digital Audio and Video*, pages 127–132, 2004.

[3] Bong-Jun Ko and Dan Rubenstein. Distributed, self-stabilizing placement of replicated resources in emerging networks. In *IEEE International Conference on Network Protocols*, pages 6–15, 2003.

[4] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, volume 11, pages 24–24, 2011.

[5] Joel Wolf, Zubair Nabi, Viswanath Nagarajan, Robert Saccone, Rohit Wagle, Kirsten Hildrum, Edward Pring, and Kanthi Sarpatwar. The x-flex cross-platform scheduler: Who's the fairest of them all? In *Proceedings of the Middleware Industry Track*, Industry papers, pages 1:1–1:7, New York, NY, USA, 2014. ACM.

[6] Anthony Ephremides, Jeffrey E Wieselthier, and Dennis J Baker. A design concept for reliable mobile radio networks with frequency hopping signaling. *Proceedings of the IEEE*, 75(1):56–73, 1987.

[7] Atish Das Sarma, Sreenivas Gollapudi, Marc Najork, and Rina Panigrahy. A sketch-based distance oracle for web-scale graphs. In *WSDM*, pages 401–410, 2010.

[8] Sudipto Guha and Samir Khuller. Approximation algorithms for connected dominating sets. In *ESA*, pages 179–193, 1996.

[9] Rajiv Gandhi and Srinivasan Parthasarathy. Distributed algorithms for connected domination in wireless networks. *Journal of Parallel and Distributed Computing*, 67(7):848–862, 2007.

[10] Xiuzhen Cheng, Xiao Huang, Deying Li, Weili Wu, and Ding-Zhu Du. A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. *Networks*, 42(4):202–208, 2003.

[11] Mikkel Thorup and Uri Zwick. Approximate distance oracles. In *STOC*, pages 183–192, 2001.

[12] Ivan D. Baev, Rajmohan Rajaraman, and Chaitanya Swamy. Approximation algorithms for data placement problems. *SIAM J. Comput.*, 38(4):1411–1429, 2008.

[13] Leana Golubchik, Sanjeev Khanna, Samir Khuller, Ramakrishna Thurimella, and An Zhu. Approximation algorithms for data placement on parallel disks. *ACM Transactions on Algorithms*, 5(4), 2009.

[14] Private cloud. In `http://wikipedia.org/wiki/Cloud_computing#Private_cloud`.

[15] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony Joseph, Scott Shenker, and Ion Stoica. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *NSDI (Boston, MA)*, 2011.

[16] V. Vavilapalli, A. Murthy, C. Douglis, A. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwiele. Apache Hadoop YARN: Yet Another Resource Negotiator. In *SoCC (Santa Clara, CA)*, 2013.

[17] Amazon EC2. In `http://aws.amazon.com/ec2/`.

[18] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3), September 1999.

[19] Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In *STOC*, pages 901–910, 2013.

[20] Hyung-Chan An, Aditya Bhaskara, Chandra Chekuri, Shalmoli Gupta, Vivek Madan, and Ola Svensson. Centrality of trees for capacitated k-center. In *IPCO*, pages 52–63, 2014.

[21] Jyh-Han Lin and Jeffrey Scott Vitter. epsilon-approximations with minimum packing constraint violation (extended abstract). In *STOC*, pages 771–782, 1992.

[22] Jaroslaw Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median, and positive correlation in budgeted optimization. In *SODA*, 2015.

[23] Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for euclidean k-medians and related problems. In *STOC*, pages 106–113, 1998.

[24] Peng-Jun Wan, Khaled M Alzoubi, and Ophir Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. In *INFOCOM*, volume 3, pages 1597–1604, 2002.

[25] Devdatt P. Dubhashi, Alessandro Mei, Alessandro Panconesi, Jaikumar Radhakrishnan, and Aravind Srinivasan. Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons. In *SODA*, pages 717–724, 2003.

[26] Jie Wu and Hailan Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *DIALM*, pages 7–14, 1999.

[27] Bevan Das and Vaduvur Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *Communications, 1997. ICC 97 Montreal,'Towards the Knowledge Millennium'. 1997 IEEE International Conference on*, volume 1, pages 376–380. IEEE, 1997.

[28] D.Z. Du and P.J. Wan. *Connected Dominating Set: Theory and Applications.* Springer Optimization and Its Applications. Springer New York, 2013.

[29] Yuzhen Liu and Weifa Liang. Approximate coverage in wireless sensor networks. In *ICN*, pages 68–75, 2005.

[30] Maggie Xiaoyan Cheng, Lu Ruan, and Weili Wu. Achieving minimum coverage breach under bandwidth constraints in wireless sensor networks. In *INFOCOM*, volume 4, pages 2638–2645, 2005.

[31] Maggie X Cheng, Lu Ruan, and Weili Wu. Coverage breach problems in bandwidth-constrained sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 3(2):12, 2007.

[32] Erik D Demaine and MohammadTaghi Hajiaghayi. Bidimensionality: New connections between fpt algorithms and ptass. In *SODA*, pages 590–601, 2005.

[33] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, July 1998.

[34] Konstantin Avrachenkov, Prithwish Basu, Giovanni Neglia, Bruno F. Ribeiro, and Don Towsley. Online myopic network covering. *CoRR*, abs/1212.5035, 2012.

[35] Christian Borgs, Michael Brautbar, Jennifer Chayes, Sanjeev Khanna, and Brendan Lucier. The power of local information in social networks. In *Internet and Network Economics*, pages 406–419. Springer, 2012.

[36] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions I. *Mathematical Programming*, 14(1):265–294, 1978.

[37] Samir Khuller, Anna Moss, and Joseph (Seffi) Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39 – 45, 1999.

[38] Michael J Kearns. *The computational complexity of machine learning.* The MIT Press, 1990.

[39] Petr Slavík. Improved performance of the greedy algorithm for partial cover. *Information Processing Letters*, 64(5):251–254, 1997.

[40] Laurence A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.

[41] Rajiv Gandhi, Samir Khuller, and Aravind Srinivasan. Approximation algorithms for partial covering problems. *J. Algorithms*, 53(1):55–84, 2004.

[42] Reuven Bar-Yehuda. Using homogenous weights for approximating the partial cover problem. In *SODA*, pages 71–75, 1999.

[43] R. Ravi, R. Sundaram, M. V. Marathe, D. J. Rosenkrantz, and S. S. Ravi. Spanning trees—short or small. *SIAM J. Discret. Math.*, 9(2):178–200, May 1996.

[44] Sanjeev Arora and George Karakostas. A $2+\epsilon$ approximation algorithm for the k-mst problem. In *SODA*, pages 754–759, 2000.

[45] Baruch Awerbuch, Yossi Azar, Avrim Blum, and Santosh Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. In *STOC*, pages 277–283, 1995.

[46] Avrim Blum, R. Ravi, and Santosh Vempala. A constant-factor approximation algorithm for the k mst problem (extended abstract). In *STOC*, 1996.

[47] Naveen Garg. A 3-approximation for the minimum tree spanning k vertices. In *FOCS*, 1996.

[48] Naveen Garg. Saving an epsilon: a 2-approximation for the k-mst problem in graphs. In *STOC*, 2005.

[49] Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *SODA*, pages 642–651, 2001.

[50] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

[51] Dorit S Hochbaum and David B Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM (JACM)*, 33(3):533–550, 1986.

[52] Ke Chen. A constant factor approximation algorithm for k-median clustering with outliers. In *SODA*, pages 826–835, 2008.

[53] Aravind Srinivasan. Distributions on level-sets with applications to approximation algorithms. In *FOCS*, pages 588–597, 2001.

[54] Reuven Bar-Yehuda, Guy Flysher, Julián Mestre, and Dror Rawitz. Approximation of partial capacitated vertex cover. In *ESA*, pages 335–346, 2007.

[55] Julián Mestre. A primal-dual approximation algorithm for partial vertex cover: Making educated guesses. *Algorithmica*, 55(1):227–239, 2009.

[56] Nader H Bshouty and Lynn Burroughs. Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem. In *STACS*, pages 298–308, 1998.

[57] Eran Halperin and Aravind Srinivasan. Improved approximation algorithms for the partial vertex cover problem. In *Approximation Algorithms for Combinatorial Optimization*, pages 161–174. 2002.

[58] David S Johnson, Maria Minkoff, and Steven Phillips. The prize collecting steiner tree problem: theory and practice. In *SODA*, pages 760–769, 2000.

[59] Anna Moss and Yuval Rabani. Approximation algorithms for constrained for constrained node weighted steiner tree problems. In *STOC*, 2001.

[60] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Vahid Liaghat. Improved approximation algorithms for (budgeted) node-weighted steiner problems. In *ICALP*, 2013.

[61] Moses Charikar and Samir Khuller. A robust maximum completion time measure for scheduling. In *SODA*, pages 324–333, 2006.

[62] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Danny Segev. Scheduling with outliers. In *APPROX-RANDOM*, 2009.

[63] Sasha Slijepcevic and Miodrag Potkonjak. Power efficient organization of wireless sensor networks. In *Communications, 2001. ICC 2001. IEEE International Conference on*, volume 2, pages 472–476. IEEE, 2001.

[64] Gruia Călinescu, Chandra Chekuri, and Jan Vondrák. Disjoint bases in a polymatroid. *Random Structures & Algorithms*, 35(4):418–430, 2009.

[65] Uriel Feige, Magnús M Halldórsson, Guy Kortsarz, and Aravind Srinivasan. Approximating the domatic number. *SIAM Journal on computing*, 32(1):172–195, 2002.

[66] Limin Wang and Sandeep S Kulkarni. Sacrificing a little coverage can substantially increase network lifetime. In *Sensor and Ad Hoc Communications and Networks, 2006. SECON'06. 2006 3rd Annual IEEE Communications Society on*, volume 1, pages 326–335. IEEE, 2006.

[67] Zoë Abrams, Ashish Goel, and Serge Plotkin. Set k-cover algorithms for energy efficient monitoring in wireless sensor networks. In *IPSN*, pages 424–432. ACM, 2004.

[68] Erik D Demaine, Uriel Feige, MohammadTaghi Hajiaghayi, and Mohammad R Salavatipour. Combination can be hard: Approximability of the unique coverage problem. *SIAM Journal on Computing*, 38(4):1464–1483, 2008.

[69] Guy Even, Zvi Lotker, Dana Ron, and Shakhar Smorodinsky. Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. *SIAM Journal on Computing*, 33(1):94–136, 2003.

[70] Amol Deshpande, Samir Khuller, Azarakhsh Malekian, and Mohammed Toossi. Energy efficient monitoring in sensor networks. In *LATIN 2008: Theoretical Informatics*, pages 436–448. Springer, 2008.

[71] Maggie X Cheng, Lu Ruan, and Weili Wu. Coverage breach problems in bandwidth-constrained sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 3(2):12, 2007.

[72] Maggie Xiaoyan Cheng, Lu Ruan, and Weili Wu. Achieving minimum coverage breach under bandwidth constraints in wireless sensor networks. In *INFOCOM*, volume 4, pages 2638–2645. IEEE, 2005.

[73] Chen Wang, My T Thai, Yingshu Li, Feng Wang, and Weili Wu. Minimum coverage breach and maximum network lifetime in wireless sensor networks. In *GLOBECOM*, pages 1118–1123. IEEE, 2007.

[74] Mihaela Cardei and Jie Wu. Energy-efficient coverage problems in wireless ad-hoc sensor networks. *Computer communications*, 29(4):413–420, 2006.

[75] Bang Wang. *Coverage control in sensor networks*. Springer, 2010.

[76] János Pach and Gábor Tardos. Conflict-free colourings of graphs and hypergraphs. *Combinatorics, Probability & Computing*, 18(5):819–834, 2009.

[77] Andrea EF Clementi, Pilu Crescenzi, Angelo Monti, Paolo Penna, and Riccardo Silvestri. On computing ad-hoc selective families. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*, pages 211–222. Springer, 2001.

[78] Christian Sommer, Elad Verbin, and Wei Yu. Distance oracles for sparse graphs. In *FOCS 2009*, pages 703–712, 2009.

[79] Surender Baswana and Telikepalli Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *FOCS 2006*, pages 591–602. IEEE, 2006.

[80] Mihai Patrascu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 815–823. IEEE, 2010.

[81] Laurent Flindt Muller and Martin Zachariasen. Fast and compact oracles for approximate distances in planar graphs. In *ESA 2007*, pages 657–668. Springer, 2007.

[82] Shiri Chechik. Improved distance oracles and spanners for vertex-labeled graphs. In *ESA 2012*, pages 325–336. Springer, 2012.

[83] Danny Hermelin, Avivit Levy, Oren Weimann, and Raphael Yuster. Distance oracles for vertex-labeled graphs. In *ICALP 2011*, pages 490–501. Springer, 2011.

[84] Wei Chen, Christian Sommer, Shang-Hua Teng, and Yajun Wang. Compact routing in power-law graphs. In *Distributed Computing*, pages 379–391. Springer, 2009.

[85] Andrey Gubichev and Thomas Neumann. Fast approximation of steiner trees in large graphs. In *CIKM*, pages 1497–1501, 2012.

[86] Olga Russakovsky and Andrew Y Ng. A steiner tree approach to efficient object detection. In *CVPR 2010*, pages 1070–1077. IEEE, 2010.

[87] Sabih H Gerez. *Algorithms for VLSI design automation*, volume 8. Wiley Chichester, England, 1999.

[88] Frank K Hwang, Dana S Richards, and Pawel Winter. *The Steiner tree problem.* Elsevier, 1992.

[89] Barry L Brumitt and Anthony Stentz. GRAMMPS: A generalized mission planner for multiple mobile robots in unstructured environments. In *ICRA 1998*, volume 2, pages 1564–1571. IEEE, 1998.

[90] Eugene L Lawler, Jan Karel Lenstra, AHG Rinnooy Kan, and David B Shmoys. *The traveling salesman problem: a guided tour of combinatorial optimization*, volume 3. Wiley Chichester, 1985.

[91] Marek Cygan, Lukasz Kowalik, Marcin Mucha, Marcin Pilipczuk, and Piotr Sankowski. Fast approximation in subspaces by doubling metric decomposition. *ESA*, pages 72–83, 2010.

[92] L Kou, George Markowsky, and Leonard Berman. A fast algorithm for steiner trees. *Acta informatica*, 15(2):141–145, 1981.

[93] Nicos Christofides. Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem. Technical report, DTIC Document, 1976.

[94] Hiromitsu Takahashi and Akira Matsuyama. An approximate solution for the steiner problem in graphs. *Math. Japonica*, 24(6):573–577, 1980.

[95] Alexander Z Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9(5):463–470, 1993.

[96] Piotr Berman and Viswanathan Ramaiyer. Improved approximations for the steiner tree problem. In *SODA 1992*, pages 325–334, 1992.

[97] Alexander Zelikovsky. Better approximation bounds for the network and euclidean steiner tree problems. 1996.

[98] Marek Karpinski and Alexander Zelikovsky. New approximation algorithms for the steiner tree problems. *Journal of Combinatorial Optimization*, 1(1):47–65, 1997.

[99] Gabriel Robins and Alexander Zelikovsky. Improved steiner tree approximation in graphs. In *SODA 2000*, pages 770–779. Society for Industrial and Applied Mathematics, 2000.

[100] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved lp-based approximation for steiner tree. In *Proceedings of the 42nd ACM symposium on Theory of computing*, pages 583–592. ACM, 2010.

[101] Marshall Bern and Paul Plassmann. The steiner problem with edge lengths 1 and 2. *IPL*, 32(4):171–176, 1989.

[102] Glencora Borradaile, Claire Kenyon-Mathieu, and Philip Klein. A polynomial-time approximation scheme for steiner tree in planar graphs. In *SODA 2007*, pages 1285–1294. Society for Industrial and Applied Mathematics, 2007.

[103] Sanjeev Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *FOCS 1996*, pages 2–11. IEEE, 1996.

[104] Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In *FOCS 2011*, pages 550–559. IEEE, 2011.

[105] Joseph SB Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric tsp, k-mst, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999.

[106] Sanjeev Arora, Michelangelo Grigni, David Karger, Philip Klein, and Andrzej Woloszyn. A polynomial-time approximation scheme for weighted planar graph tsp. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 33–41, 1998.

[107] Christos H Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[108] Samir Khuller, Barna Saha, and Kanthi K Sarpatwar. New approximation results for resource replication problems. *Approximation, Randomization, and Combinatorial Optimization*, page 218, 2012.

[109] Samir Khuller, Barna Saha, and Kanthi K. Sarpatwar. New approximation results for resource replication problems. *Algorithmica*, pages 1–23, 2015.

[110] Viswanath Nagarajan, Kanthi K. Sarpatwar, Baruch Schieber, Hadas Shachnai, and Joel L. Wolf. Container selection with applications to cloud computing. *Submitted*, 2015.

[111] Samir Khuller, Manish Purohit, and Kanthi K. Sarpatwar. Analyzing the optimal neighborhood: Algorithms for budgeted and partial connected dominating set problems. In *SODA*, pages 1702–1713, 2014.

[112] Samir Khuller, Manish Purohit, and Kanthi Sarpatwar. Approximation algorithms for covering problems in energy constrained wireless networks. *Submitted*, 2015.

[113] Randeep Bhatia, Bhawna Gupta, and Kanthi K. Sarpatwar. Improved approximation algorithms for steiner tree and cheapest tour oracles. *Submitted*, 2015.

[114] Jack Edmonds and Delbert R. Fulkerson. Bottleneck extrema. *Journal of Combinatorial Theory*, 8(3):299 – 306, 1970.

[115] Samir Khuller and Yoram J. Sussmann. The capacitated k-center problem. *SIAM J. Discrete Math.*, 13(3):403–418, 2000.

[116] Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33(3):533–550, 1986.

[117] Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[118] Michael Luby. A simple parallel algorithm for the maximal independent set problem. In *ACM Symposium on the Theory of Computing*, pages 1–10, 1985.

[119] V.V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[120] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[121] Jack Edmonds. Paths, trees, and flowers. In Ira Gessel and Gian-Carlo Rota, editors, *Classic Papers in Combinatorics*, Modern Birkhuser Classics, pages 361–379. Birkhuser Boston, 1987.

[122] Uriel Feige, Magnús M. Halldórsson, and Guy Kortsarz. Approximating the domatic number. In *ACM Symposium on the Theory of Computing*, pages 134–143, 2000.

[123] Moses Charikar and Samir Khuller. A robust maximum completion time measure for scheduling. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 324–333, 2006.

[124] Barna Saha and Aravind Srinivasan. A new approximation technique for resource-allocation problems. In *Innovations in Computer Science*, pages 342–357, 2010.

[125] Ravishankar Krishnaswamy, Amit Kumar, Viswanath Nagarajan, Yogish Sabharwal, and Barna Saha. The matroid median problem. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1117–1130, 2011.

[126] Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1–10, 1998.

[127] Judit Bar-Ilan, Guy Kortsarz, and David Peleg. How to allocate network centers. *J. Algorithms*, 15(3):385–415, 1993.

[128] Tomás Feder and Daniel H. Greene. Optimal algorithms for approximate clustering. In *STOC*, pages 434–444, 1988.

[129] David Haussler and Emo Welzl. epsilon-nets and simplex range queries. *Discrete & Computational Geometry*, 2:127–151, 1987.

[130] Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

[131] Evangelia Pyrga and Saurabh Ray. New existence proofs epsilon-nets. In *SOCG*, pages 199–207, 2008.

[132] Baruch Schieber. Computing a minimum weight k-link path in graphs with the concave monge property. *Journal of Algorithms*, 29(2):204–222, 1998.

[133] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 2009.

[134] Camille Jordan. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik*, 70:185–190, 1869.

[135] Y. Filmus and J. Ward. A tight combinatorial algorithm for submodular maximization subject to a matroid constraint. In *FOCS, 2012*, pages 659–668, 2012.

[136] Andrea EF Clementi, Angelo Monti, and Riccardo Silvestri. Selective families, superimposed codes, and broadcasting on unknown radio networks. In *SODA*, pages 709–718. Society for Industrial and Applied Mathematics, 2001.

[137] R Rado. Selective families of sets. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 372(1750):307–315, 1980.

[138] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *STOC*, pages 681–690. ACM, 2006.

[139] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 2010.