Mathematics is not about numbers, equations, computations, or algorithms: it is about understanding.

– William P. Thurston.

University of Alberta

APPROXIMATION ALGORITHMS FOR CLUSTERING PROBLEMS

by

Babak Behsaz

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

To my parents, Fatemeh and Ebrahim,
and my wife, Farzaneh.

Abstract

In this thesis, we present some approximation algorithms for the following clustering problems: Minimum Sum of Radii (MSR), Minimum Sum of Diameters (MSD), and Unsplittable Capacitated Facility Location (UCFL).

Given a metric (V, d) and an integer k, we consider the problem of partitioning the points of V into k clusters so as to minimize the sum of radii (MSR) or the sum of diameters (MSD) of these clusters. We call a cluster containing a single point, a *singleton* cluster. For the MSR problem when singleton clusters are not allowed, we give an exact algorithm for metrics induced by unweighted graphs. For the MSD problem on the plane with Euclidean distances, we present a polynomial time approximation scheme. In addition, we settle the open problem of complexity of the MSD problem with constant k by giving a polynomial time exact algorithm in this case.

In the (uniform) UCFL problem, we are given a set of clients and a set of facilities where client j has demand d_j, each facility i has capacity u and opening cost f_i, and a metric cost c_{ij} which denotes the cost of serving one unit of demand of client j at facility i. The goal is to open a subset of facilities and assign each client to *exactly one* open facility so that the total amount of demand assigned to each open facility is no more than u, while minimizing the total cost of opening facilities and serving clients. As it is **NP**-hard to give a solution without violating the capacities, we consider bicriteria (α, β)-approximation algorithms, where these algorithms return a solution whose cost is within factor α of the optimum and violates the capacity constraints within factor β. We present the first constant approximations with violation factor less than 2. In addition, we present a quasi-polynomial time $(1 + \epsilon, 1 + \epsilon)$-approximation for the (uniform) UCFLP in Euclidean metrics, for any constant $\epsilon > 0$.

Acknowledgements

This thesis would not have been possible without the advice, help, and support of the kind people around me. Above all, I would like to thank my supervisor, Dr. Mohammad R. Salavatipour. Not only, I benefited from his knowledge and the opportunities that he provided me, but also, he taught me many important aspects of being a researcher. In addition, he was always more than happy to help me with non-academic issues and I am very grateful for his understanding and support at a personal level.

I also would like to thank my fellow students and post doctoral researchers: Amin Jorati, Zachary Friggstad, Reza Khani, Imran A. Pirwani, Zoya Svitkina, and Mohammad A. Safari. I had many inspiring and fruitful discussions with them that made my experience and knowledge more complete. I am grateful to Imran A. Pirwani for introducing me to the Minimum Sum of Radii and Diameters problems, which form a large part of this thesis, and our early discussions on this topic.

For personal funding and research scholarships, I must thank University of Alberta and Alberta Innovates Technology Futures (former iCore). In addition, I thank my references Mohammad R. Salavatipour, Mike MacGregor, and Lorna Stewart for supporting me when I was applying for scholarships and jobs.

I also would like to thank three persons who had a great influence on where I am now. I am grateful to my fourth and fifth grade teacher, Mr. Mousavi, who made me interested in mathematics and encouraged me to learn this beautiful subject. I am grateful to Mr. Yasser A. Fouladi who made me interested in discrete mathematics and computer science when I was at high school. Finally, I would like to thank Prof. Ebadollah S. Mahmoodian who boosted my passion for discrete mathematics and theoretical computer science and had a great influence on the start of my doctoral studies.

Last, but not least, I am very grateful to my parents, Fatemeh and Ebrahim, for their unconditional love and support that helped me throughout my life. They always put my well-being, education, and happiness ahead of themselves. I am also very grateful to my wife, Farzaneh, for her support, understanding, and patience. She was my best friend and of invaluable assistance when I was facing difficult situations during the period of completing this thesis.

Table of Contents

List of Figures

Chapter 1

Introduction

Clustering, one of the fundamental techniques in information technology, has been used in a wide variety of application areas [ELLS11]. The main goal of this technique is to partition a set of objects into homogeneous subsets, called *clusters*. In other words, the objects in the same cluster are similar in some sense. For example, a web search engine partitions the web documents into clusters of relevant documents. Then, instead of matching a search query with all the documents on the web, it finds the relevant cluster and matches the query only with the documents in that cluster and as a result, it saves a huge amount of time [MRS08].

In any clustering method, we define a distance measure between each pair of objects to indicate how similar those objects are. Then, we use this distance measure to guide our clustering algorithm to form meaningful clusters. In most clustering algorithms, we try to find a clustering that optimizes an objective function based on the distance measure. This objective function is chosen in a way that its value for a clustering shows how desirable this clustering is and optimizing it yields a meaningful clustering for the application of our interest. Unfortunately, most of these optimization problems (clustering problems) are **NP**-hard, which means it is unlikely to find an algorithm that finds an optimal solution in polynomial time. Therefore, our goal in these optimization problems is to find a solution with objective value as close as possible to the optimum value. This goal can be captured by the notion of *approximation algorithms* (see the next section for a formal definition). In this thesis, we present some approximation algorithms for some clustering problems.

1.1 Preliminaries

In this section, we discuss some preliminaries required for the presentation and understanding of the rest of this thesis.

1.1.1 Our Modelling

To model and discuss our problems, we extensively use graphs and graph-theoretic concepts. The reader may refer to [Wes00] for an introduction to this subject. We only want to emphasize two graph-theoretic concepts: the radius and the diameter of a graph. These concepts are defined on a weighted graph. Assume $w : E \to \mathbb{R}_{\geq 0}$ is a weight function defined on the edges of a graph G, where $\mathbb{R}_{\geq 0}$ is the set of non-negative real numbers. For two vertices u and v, the shortest path distance from u to v with respect to w is denoted by $d(u, v)$. The *eccentricity* of a vertex u is the maximum distance of u from any other vertex in G, *i.e.*, $\max_{v \in V(G)} d(u, v)$. The vertices with the minimum eccentricity are called the *centers* of G and their eccentricity value is the *radius* of the graph, denoted by $\operatorname{rad}(G)$, *i.e.*, $\operatorname{rad}(G) = \min_{u \in V(G)} \max_{v \in V(G)} d(u, v)$. The *diameter* of a graph G, denoted by $\operatorname{diam}(G)$, is the largest distance between any pair of vertices in G, *i.e.,* $\operatorname{diam}(G) = \max_{u,v \in V(G)} d(u, v)$.

In the clustering problems that we consider, we model the space of objects and their relations with a weighted graph. We represent the objects with the vertices and show the direct connection between two objects with an edge. Here, the weight of an edge shows the distance between the objects corresponding to its endpoints. This distance is a measure of similarity between these objects and as one expects, a smaller distance shows more similarity. Most of the time, the edge weights (distances) satisfy the triangle inequality and indeed, form a *metric* on the vertices (see next section for the definition). It should be noted that we can view the distance between two objects as the cost of putting them in the same cluster. Thus, we may use the terms distance and cost interchangeably in the following discussions.

1.1.2 Metrics

A metric (V, d) is an ordered pair where $d : V \times V \to \mathbb{R}_{\geq 0}$. For all u, v, w in V, the distance function d must satisfy the following properties:

1. $d(u, v) = 0$ if and only if $u = v$,
2. $d(u, v) = d(v, u)$,
3. $d(u, v) \leq d(u, w) + d(w, v)$

In our discussions, we relax the first property and may have two different u and v with $d(u, v) = 0$, but we always have the property that $d(u, u) = 0$. The second property is called the *symmetry* property. In addition, the third property is called the *triangle inequality*, which has very important algorithmic implications.

A well-known example of metrics is the *Euclidean metric* where the elements of V are from a Euclidean space and the distance function is simply the Euclidean distance function. A less famous example is the *doubling-dimension* metric, which in a sense generalizes the Euclidean metric. Let u be a point in V. The set of points from V having distance at most r from u is called the *ball of radius r centered at* u and is denoted by $B(u, r)$. In other words, $B(u, r) = \{v : v \in V, d(u, v) \leq r\}$. The doubling dimension of a finite metric space V is the smallest $k > 0$ such that every ball in the metric can be covered by 2^k balls of half the radius. This definition is based on volume growth, and simple volume estimates imply that a d-dimensional Euclidean metric has doubling dimension $\Omega(d)$. Also, it can be shown that its doubling dimension is in $O(d)$ [Ass83].

Notice that to represent a metric, we need to have $|V|(|V|-1)/2$ entries. Sometimes, we can represent a metric in a more compact way by a non-complete edge-weighted graph G. Here, the members of V are the vertices of the graph. In this metric, the distance between any pair of vertices is the shortest path distance of these vertices in G. This is called the *metric induced by graph G*. This way we can have fewer entries to represent the metric. In particular, the *tree metrics* are the metrics induced by a tree, which can be represented by $|V|-1$ entries.

Quasi-metrics

Sometimes, we even relax the symmetry property. The distance functions that satisfy the properties of a metric except the symmetry property are called *quasi-metrics*. In theoretical computer science community, these metrics are also called *asymmetric metrics* and when we discuss a problem defined on these metrics, we call it the asymmetric version.

1.1.3 Approximation Algorithms

An *optimization problem* Π is either a minimization or maximization problem. Each valid instance I of Π comes with a set of feasible solutions. A *feasible solution* is a solution that satisfies all the problem constraints. There is an *objective function*, which assigns a real valued number called the objective value to each feasible solution. A feasible solution having the optimal objective value (*i.e.*, the minimum or maximum value) is called an *optimal solution*. The goal of an optimization problem is to find an optimal solution.

Generally, we like to design polynomial time algorithms for optimization problems, but many optimization problems are **NP**-hard. In less technical words, it is very unlikely to find an exact algorithm for these problems that runs in polynomial time. Thus, we try to approximately solve these problems. An *α-approximation algorithm* is a polynomial time algorithm, which for all valid instances of the optimization problem produces a solution whose value is within factor α of the optimum value. In the literature, α is called the *approximation ratio*, *approximation factor*, or *performance guarantee*.

For example, consider the minimum vertex cover problem. Given a graph $G=(V,E)$, we want to choose a minimum size subset of vertices S such that each edge has an endpoint in S. This problem is a well-known **NP**-hard problem [GJ79]. Here is a simple 2-approximation algorithm: find a maximal matching[1] M in the graph and return the endpoints of the edges in the matching. First, notice that the endpoints of M cover all the edges in the graph, because if there is an uncovered edge, we can add this edge to M, which is a contradiction to the maximality of M. Therefore, the algorithm indeed returns a vertex cover. Second, we prove that the size of this vertex cover, $2|M|$,

[1]A matching is a subset of edges that do not share any vertex. A maximal matching is a matching that we cannot add any more edges to.

is at most twice the size of a minimum vertex cover. To do this, we simply point out that the size of a minimum vertex cover is at least $|M|$, because we need at least $|M|$ vertices to cover the edges of the matching (*i.e.,* each vertex can cover at most one edge of M).

As one can expect, we want to find an approximation algorithm with ratio as close to 1 as possible. The best ratio that we can hope for is $1 + \epsilon$ where $\epsilon > 0$ is an arbitrary parameter that controls the trade-off between the running time and accuracy. We can increase accuracy by choosing smaller ϵ, but the running time increases with this choice. In particular, a polynomial time approximation scheme (PTAS) is a $(1 + \epsilon)$-approximation algorithm that runs in time polynomial in the size of input (but it can be exponential in $1/\epsilon$). For example, the running time can be in $O(n^{1/\epsilon})$, where n is the size of input. Notice that for a fixed ϵ, this running time is polynomial time.

One can define different complexity classes with respect to the approximability of the **NP**-hard optimization problems. In particular, the class of problems that have a polynomial time constant factor approximation algorithm is called **APX**. A PTAS reduction from an optimization problem Π to a problem Π' has the property that given a PTAS for problem Π', we can obtain a PTAS for problem Π. A problem is said to be **APX**-hard if there is a PTAS reduction from every problem in **APX** to that problem. Therefore, given a PTAS for an **APX**-hard problem, we can obtain a PTAS for every problem in **APX**. It is shown that unless **P**= **NP**, some problems in **APX** cannot have a PTAS. As a consequence, unless **P**= **NP**, an **APX**-hard problem cannot have a PTAS.

Generalizations

Sometimes, we may allow an approximation algorithm to violate some problem constraints (*e.g.*, the number of clusters). A *bicriteria* (α, β)*-approximation algorithm* is a polynomial time algorithm that guarantees its solution value is within factor α of the optimum value and it violates a group of constraints within factor β. Here, we are comparing our solution value with the optimum value of a solution that does *not* violate the constraints.

An algorithm has a *quasi-polynomial* running time if there is a constant c such that the algorithm runs in time $O(n^{\log^c n})$. It is widely believed that it is not possible to solve an **NP**-hard problem in quasi-polynomial time. The definition of approximation algorithms may be extended to include algorithms whose running time is quasi-polynomial. In these cases, we explicitly state that the running time is quasi-polynomial time. Similar to the definition of a PTAS, we can define a quasi PTAS: a quasi-polynomial time approximation scheme (QPTAS) is a $(1 + \epsilon)$-approximation algorithm that runs in time quasi-polynomial in the size of input.

Techniques based on Linear Programming

One of the main steps in the analysis of an approximation algorithm is to find a suitable lower bound or upper bound for the optimum value. Consider a minimization problem and suppose that we want to prove an algorithm returns a solution within factor α of the optimum. One natural way to do this is to find a suitable lower bound for the optimum and prove that the algorithm always returns a solution within factor α of this lower bound.

Linear programming (LP)[2] provides us with a powerful tool to find a lower bound of the optimum. The reason is that we can model most combinatorial optimization problems with an integer program. We relax the integer requirement on the variables of an integer program to get its linear programming relaxation. Since any feasible solution for the integer program is a feasible solution for the LP relaxation, the optimum value of this relaxation is a lower bound for the optimum value of the integer program. Therefore, to prove an approximation ratio of α, it is sufficient to find a solution with objective value within factor α of the optimum value of the LP relaxation.

One way to utilize the above idea is to solve the LP and find an optimal fractional solution. Then, we round this fractional solution to an integer solution in a way that we do not lose much in the process and we can prove the value of this solution is within factor α of the optimal fractional value. This technique is called the *LP rounding* technique. Let us see this in an example for the minimum vertex cover problem. Recall that we have already shown a combinatorial 2-approximation algorithm for this problem.

A natural integer program for the minimum vertex cover problem is as follows:

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{u \in V} x_u & \\ \text{subject to} & x_u + x_v \geq 1 & \forall \{u, v\} \in E \\ & x_u \in \{0, 1\} & \forall u \in V, \end{array}$$

where $x_u = 1$ indicates that u is in the vertex cover. To get the LP relaxation, we relax the variables to take fractional values between zero and one, *i.e.*, $0 \leq x_u \leq 1$ for all $u \in V$. We can find an optimal solution x^* to this LP by a polynomial time LP solver. One can show that this optimal solution is half integral, *i.e.*, x_u^* is 0, $\frac{1}{2}$, or 1. To round this solution to an integral one, we round up all variables with value $\frac{1}{2}$ to 1. First, notice that for any edge $\{u, v\}$, one of x_u^* or x_v^* is non-zero and hence, in the integral solution at least one of these variables is 1, which means we choose one of the endpoints of this edge. Therefore, this is a valid vertex cover. Second, it is clear that we only double the value of the variables with value $\frac{1}{2}$. Thus, the objective value of the integral solution (*i.e.,* the number of vertices in the vertex cover) is at most twice the optimal fractional value, which is a lower bound for the optimum value. As a result, this solution is within factor 2 of the optimum value and our algorithm is a 2-approximation algorithm for the minimum vertex cover problem.

Finally, we introduce the notion of the *integrality gap* for an LP relaxation. Consider an LP relaxation of a minimization problem. Let $\mathcal{I}$ be the set of instances for the problem, $\text{OPT}(I)$ be the optimal value for an instance $I \in \mathcal{I}$, and $\text{OPT}_f(I)$ be the optimal fractional value (LP value) for

[2]We also refer to Linear Programs by the acronym LP.

this instance. Then, the integrality gap of the LP is defined as

$$\sup_{I \in \mathcal{I}} \frac{\mathrm{OPT}(I)}{\mathrm{OPT}_f(I)}.$$

The integrality gap of an LP shows the best achievable approximation ratio using the optimum value of this LP as our lower bound. For example, the integrality gap of the above LP for the minimum vertex cover problem is at least $2 - 2/n$ for graphs with n vertices. Consider a complete graph with n vertices as our instance I'. For this graph, we have $\mathrm{OPT}(I') = n - 1$, because if we leave two vertices out of a cover, it does not cover the edge between those two vertices. Now, we have $\mathrm{OPT}_f(I') \leq n/2$, because the solution that sets $x_u = 1/2$ for all $u \in V$ is a feasible solution with value $n/2$. Therefore, we have $\sup_{I \in \mathcal{I}} \frac{\mathrm{OPT}(I)}{\mathrm{OPT}_f(I)} \geq \frac{\mathrm{OPT}(I')}{\mathrm{OPT}_f(I')} \geq 2 - 2/n$. This means that the simple factor 2 rounding algorithm is very close to the best ratio possible using this LP as our lower bound.

Local Search Technique

The *local search technique* is one of the fundamental techniques in optimization. In this technique, we move from one solution to another solution in the space of feasible solutions by means of some local operations. These operations just modify a local part of a solution to change it to a new solution. As a result, the new solution is similar to the previous one. If one defines a suitable distance function on the space of feasible solutions, these two solutions will be close and in the neighbourhood of each other.

More formally, assume that our current solution is S. We have k local operations $f_1, \ldots, f_k$ where the result of applying the ith operation on S is $f_i(S)$. Assume our problem is a minimization problem. In a local search algorithm, we apply each of these k operations on S. If the value of $f_i(S)$ for some i^* is less than value of S, we update the current solution to be $f_{i^*}(S)$ and do the same in the next iteration. Here, a *local optimal solution* is the solution that we cannot improve with any of the operations and a *global optimal solution* is an optimal solution for the problem. To prove an approximation ratio α for a local search algorithm, we need to prove that the ratio of the value of any local optimal solution is within factor α of the global optimum value. The maximum ratio between the value of a local optimal solution and the value of a global optimum is called *locality gap* value and clearly, we cannot have a better approximation ratio than the locality gap value using these local search operations. While the description of a local search algorithm is typically simple, the analysis is more involved. We conclude this section with a simple example of a local search algorithm and its analysis.

Consider the unweighted maximum cut problem. In this problem, given an unweighted graph $G = (V, E)$, we want to partition V into two sets V_1 and V_2 such that we maximize the number of edges between the two partitions. Our local search algorithm starts with an arbitrary partition V_1 and V_2. Then, it performs one of the following two local operations as long as it improves the size of cut (V_1, V_2):

- Move some vertex $v \in V_1$ to V_2, or
- Move some vertex $v \in V_2$ to V_1.

The local search algorithm having these operations finishes in polynomial time. The reason is that in each iteration, we increase the number of edges between two partition by at least one. Since there are $O(n^2)$ edges where $n = |V|$, we have $O(n^2)$ iterations. Each iteration takes $O(n)$ time for each vertex and $O(n^2)$ in total. Therefore, the total running time is in $O(n^4)$. Now, we analyze the approximation ratio of this simple algorithm. Let $d(v)$ be the number of edges incident to a vertex v in the graph, *i.e.*, be the degree of v. We claim that in a local optimal solution, each vertex has at least $d(v)/2$ incident edges between V_1 and V_2, because otherwise we can move it from one partition to the other and increase the objective value. This is a contradiction with the local optimality of the solution. Therefore, the number of edges between the partitions is at least $|E(G)|/2$. Since the optimum value is at most $|E(G)|$, the value of our solution is within factor 2 of the optimum.

1.2 Clustering Problems

To have meaningful clustering, the objective function must restrict the number of clusters; otherwise, the clustering that puts every single object in a cluster (*i.e.*, the number of clusters is equal to the number of objects) gives the optimum solution for any reasonable objective function. There are two main groups of clustering problems based on how the objective function restricts the number of clusters: *k-clustering problems* and *facility location problems*.

1.2.1 k-clustering problems

In k-clustering problems, given an integer k in input, we want to cluster some objects into at most k clusters and optimize an objective function, which shows the desirability of a clustering. Here we directly restrict the number of clusters to a given number. We present some well-known k-clustering problems in the following.

k-median: In this problem, we are given a complete bipartite graph $G = (V, E)$ with bipartition (F, C) and a metric cost function $c : E \to \mathbb{Q}_{\geq 0}$. Here, F and C can be considered as the sets of facilities and clients, respectively. We want to partition C into k sets (clusters) $C_1, C_2, \ldots, C_k$ and assign a facility to each set (cluster) such that we minimize $\sum_{i=1}^{k} \sum_{j \in C_i} c_{y_i j}$ where y_i is the facility assigned to cluster C_i. In a series of works, the best approximation ratio for this problem improved to its current best factor of $3 + \frac{2}{p}$ for any fixed integer $p \geq 1$ with running time $n^{O(p)}$ due to Arya *et al.* [AGK$^+$01]. Jain *et al.* [JMS02] proved that the k-median problem cannot be approximated within a factor strictly less than $1 + 2/e \approx 1.735$, unless **NP** $\subseteq$ DTIME$[n^{O(\log \log n)}]$, where DTIME$(t(n))$ is the collection of languages that are decidable by an $O(t(n))$ time deterministic Turing machine.

k-center and k-cluster: In these problems, we are given a complete graph $G = (V, E)$ with metric cost function $c : E \to \mathbb{Q}_{\geq 0}$ and we want to partition V into k sets $C_1, C_2, \ldots, C_k$. In the k-

center problem, we want to minimize $\max_{i=1}^{k} \mathrm{rad}(C_i)$. The k-center problem has a 2-approximation algorithm and this is the best possible unless **P**= **NP** [HS85]. In the k-cluster problem, we want to minimize $\max_{i=1}^{k} \mathrm{diam}(C_i)$. For this problem, there is a 2-approximation algorithm as well and this is the best possible unless **P**= **NP** [Gon85].

1.2.2 Facility Location Problems

Sometimes, fixing the maximum number of clusters is not appropriate, because we do not know how to set this parameter. Setting k too low or too high may result in an ineffective clustering. Thus, we are interested in restricting the number of clusters without directly specifying an upper bound k. In the second group of clustering problems, there is a cost associated with each possible cluster and for each clustering, the objective value depends on the cost of clusters in it. To restrict the number of clusters, we define these costs in a way that the optimal solution cannot have many clusters unless it is unavoidable (*i.e.*, most of the objects are different and it is not optimal to put them in the same cluster). The clustering problems of this type are called *facility location problems*. In these problems, there are some clients that must be served by some facilities. In other words, we want to cluster clients and assign a facility to each cluster. To use each facility, we must pay an opening cost.

Now, we present some well-known facility location problems. In the following problems, we are given a complete bipartite graph $G = (V, E)$ with metric cost function $c : E \to \mathbb{Q}_{\geq 0}$ where $V = F \cup C$, and F and C are the bipartition of vertices. The sets F and C are sets of facilities and clients, respectively. Each client j has a demand d_j and each facility i has an opening cost f_i. We denote the cost of serving one unit of demand of client j at facility i by c_{ij}. Let x_{ij} be the amount of demand of client j that will be served by facility i in our solution. Note that we must have $\sum_{i \in F} x_{ij} = d_j$ and $x_{ij} > 0$ only if facility i is open in our solution.

Uncapacitated Facility Location: In this problem, there is no limit on the capacity of a facility. We want to choose a subset I of facilities to open and assign the demands of clients to these facilities in a way that minimizes $\sum_{i \in C} x_{ij} c_{ij} + \sum_{i \in I} f_i$. Shmoys, Tardos and Aardal [STA97] gave the first constant approximation for this problem with ratio 3.16. In a long series of works, the approximation ratio for this problem was improved to 1.488 [Li11]. Furthermore, a result of Guha and Khuller [GK98], combined with an observation of Sviridenko (personal communication cited in [CW99]), implies that no polynomial time algorithm for this problem can have an approximation factor better than 1.463 unless **P**= **NP**.

Capacitated Facility Location: In this problem, facility i can serve at most u_i units of demands. Again, we want to choose a subset I of facilities to open and assign the demand of clients to these facilities in a way that minimizes $\sum_{i \in C} x_{ij} c_{ij} + \sum_{i \in I} f_i$. The current best approximation algorithm for this problem has factor 5 [BGG12] and the best hardness result is the same as the uncapacitated version.

Lower-bounded Facility Location: In this problem, if we open facility i, it must serve at least l_i units of demands and the objective function is the same as the previous two problems. Svitkina gave the first constant approximation algorithm with ratio 448 for this problem [Svi10]. Later, an improved approximation algorithm with ratio 82.6 was given by [AS11]. The best hardness result is the same as the uncapacitated version.

1.3 Problems Considered

Now, we introduce the problems that we consider in this thesis. The first group of problems are from the k-clustering problems and the second group of problems are from the facility location problems.

1.3.1 The Minimum Sum of Radii and Diameters Problems

The k-center clustering is an important and traditional clustering method. Unfortunately, in some applications, using k-center objective function produces a *dissection effect* [HJ87]. This effect causes objects that should be placed in the same cluster to be assigned to different clusters. To avoid this effect, it is proposed to minimize the sum of cluster radii or diameters. This leads to the Minimum Sum of Radii (MSR) and the Minimum Sum of Diameters (MSD) problems, respectively. In each of these problems, one is given a set of points V in a metric space d and the goal is to partition V into k clusters so as to minimize the sum of radii of clusters (in MSR) or the sum of diameters of the clusters (in MSD). We can consider these points as the vertices of a graph with a metric cost function on the edges. More formally, we are given a graph $G = (V, E)$ with edge costs (or distances) $d : E \rightarrow \mathbb{Q}_{\geq 0}$. The goal is to partition V into k sets $V_1, V_2, \ldots, V_k$. In the MSR problem, we want to minimize $\sum_{i=1}^{k} \mathrm{rad}(V_i)$. In the MSD problem, we want to minimize $\sum_{i=1}^{k} \mathrm{diam}(V_i)$.

1.3.2 The Unsplittable Capacitated Facility Location Problem

We also consider the Unsplittable Capacitated Facility Location Problem (UCFLP) with *uniform* capacities. In this problem, we are given a set of clients C and facilities F where each client j has demand d_j and each facility i has capacity u and opening cost f_i. We have a metric cost function c_{ij} which denotes the cost of serving one unit of demand of client j at facility i. The goal is to open a subset of facilities $I \subseteq F$ and assign each client j to *exactly one* open facility $\phi(j)$ to serve its entire demand d_j so that the total amount of demand assigned to each open facility is no more than u, while minimizing the total cost of opening facilities and serving clients, i.e. minimizing $\sum_{i \in I} f_i + \sum_{j \in C} d_j c_{\phi(j)j}$. This problem generalizes some important combinatorial optimization problems.

Facility location problems have been studied extensively in operations research and management sciences and even a few books are devoted to these problems (*e.g.*, see [DH04, LMW88]). They are also well studied in theoretical computer science and various approximation algorithms are designed for them. While these problems arise in a wide range of practical applications, the most common context of their employment has been supply chain. In a supply chain that consists of suppliers, distribution centers, or warehouses and customers, these problems emerge in making location decisions [Ver11]. When we deal with indivisible goods, the unsplittable demand assumption is a necessity. In particular, the UCFLP has been studied in Operations Research literature from the eighties, where it is called the capacitated facility location problem with single sourcing or the capacitated concentrator location problem [DH04]. The latter name comes from the problem of assigning a set of terminals or workstations to some concentrator devices in telecommunications networks. Here, each terminal has a demand that must be served by exactly one concentrator and each concentrator has a capacity that shows the amount of traffic that it can manage.

1.4 Previous Works

In this section, we present the previous works on the problems that we consider. We discuss these works in more details in the following chapters.

1.4.1 The MSR and MSD Problems

Both the MSR and MSD problems are well studied in general and Euclidean metrics. When the cost function is a metric distance function, a simple observation is that for any graph (or cluster) G: $\mathrm{rad}(G) \leq \mathrm{diam}(G) \leq 2\,\mathrm{rad}(G)$. Thus, an α-approximation algorithm for MSD yields a 2α-approximation algorithm for MSR and vice versa. Doddi *et al.* [DMR$^+$00a] considered the MSD problem and showed that unless **P**= **NP**, for any $\epsilon > 0$, there is no $(2 - \epsilon)$-approximation algorithm for the MSD problem even when the graph is unweighed (i.e. the metric is the shortest-path metric of an unweighted graph). Note that this result does not imply **NP**-hardness of MSR. They also presented a bicriteria algorithm that returns a solution with $O(k)$ clusters whose cost is within $O(\log(n/k))$ factor of the optimum. Charikar and Panigrahy [CP04] significantly improved this result by giving a $(3.504 + \epsilon)$-approximation algorithm for MSR, and consequently a $(7.008 + \epsilon)$-approximation algorithm for MSD, that runs in time $n^{O(\frac{1}{\epsilon})}$. These are the current best ratios for the MSR and MSD problems on general metrics. Recently, in an interesting result, Gibson *et al.* [GKK$^+$10] designed an exact algorithm for the MSR problem which runs in time $n^{O(\log n \log \Delta)}$ where Δ is the ratio of the largest distance over the smallest non-zero distance. They translate this result to a quasi-polynomial time approximation scheme for general metrics.

There are also several results for the special cases of these problems. When $k = 2$, the MSD problem is solvable in polynomial time by a reduction to the 2-SAT problem [HJ87]. When k is fixed and the metric is Euclidean, Capoyleas *et al.* [CRW91] present an exact algorithm for MSD. When k is fixed, for general metrics, there is a 2-approximation for MSD [DMR$^+$00a]. This result can be obtained from a simple exact algorithm for MSR [DMR$^+$00a], which uses the observation that the number of distinct clusters are polynomially bounded. For Euclidean MSR, there is an exact polynomial time algorithm [GKK$^+$12]. This result also extends to L_1 and L_∞ norms. This also implies a 2-approximation for MSD on Euclidean plane, which is the current best ratio. In contrast, the MSR problem is **NP**-hard even in metrics induced by weighted planar graphs and in metrics of constant doubling dimension [GKK$^+$10].

1.4.2 The UCFLP

As Bateni and Hajiaghayi [BH09] pointed out, solving the UCFLP without relaxation of capacities is NP-hard even in very special cases. Thus, research has focused on the design of bicriteria approximation algorithms. An (α, β)*-approximation* for the UCFLP returns a solution whose cost is within factor α of the optimum and violates the capacity constraints within factor β. Here, we compare our solution with an optimal solution that does *not* violate any constraints. It should be noted that if we violate capacity of a facility within factor β, we must pay β times its opening cost.

In the context of approximation algorithms, Shmoys, Tardos, and Aardal [STA97] were the first to consider this problem and presented a $(9, 4)$-approximation algorithm. They used a filtering and rounding technique to get an approximation algorithm for the splittable version and used a rounding for a generalized version of the matching problem, called generalized assignment problem (GAP) [ST93] to obtain their algorithm for the unsplittable version (see Section 3.1.4 for a formal definition of GAP). This technique of reducing the unsplittable version using the rounding for the GAP to the splittable version was a cornerstone of the subsequent approximation algorithms. In addition, in the same place, by a randomized version of the filtering technique, they got a new algorithm with the ratio $(7.62, 4.29)$.

Korupolu, Plaxton, and Rajaraman [KPR98] gave the first constant factor approximation algorithm for the splittable hard capacitated version, and applied the GAP rounding technique of [STA97] to get a $(O(1), 2)$-approximation algorithm for the UCFLP. All subsequent constant factor approximation algorithms for the splittable capacitated version combined with the GAP rounding technique give a $(O(1), 2)$-approximation algorithm for the UCFLP, even though this may not be explicitly stated by their authors. Applying the current best approximation algorithms for the splittable capacitated version with non-uniform capacities [BGG12] and uniform capacities [AAB$^+$10], one can get factor $(9, 2)$ and $(5, 2)$ approximation algorithms for the UCFLP with non-uniform and uniform capacities, respectively.

Recently, Bateni and Hajiaghayi [BH09] modeled an assignment problem in content distribution networks by the UCFLP. This assignment problem has been first considered by Alzoubi *et al.* [ALR$^+$08] and is basically the assignment of downloadable objects, such as media files or softwares, to some servers. We cannot split a downloadable object and we need to store it in a single server. As Alzoubi *et al.* mention, the server capacities is very crucial in practice and a high overload amount on a server can disrupt a large number of connections. Motivated by this strict requirement on capacities, the authors of [BH09] designed a $(1+\epsilon, 1+\epsilon)$-approximation algorithm for *tree metrics* (for any constant $\epsilon > 0$) using a dynamic programming approach. They also presented a quasi-polynomial time $(1+\epsilon, 1+\epsilon)$-approximation algorithm (again for trees) for the non-uniform capacity case, i.e. when each facility i has a given capacity u_i. Using Fakcharoenphol *et al.*'s [FRT03] improvement of Bartal's machinery [Bar98], this implies a polynomial time $(O(\log n), 1+\epsilon)$-approximation algorithm for almost uniform capacities and a quasi-polynomial time $(O(\log n), 1+\epsilon)$-approximation algorithm for non-uniform case for an arbitrary constant $\epsilon > 0$.

All the known constant-factor algorithms for the UCFLP violate the capacity constraints by a factor of at least 2 which is mainly due to using the rounding algorithm for GAP [ST93]; and the algorithm of [BH09] (although has $1+\epsilon$ violation) is not a constant factor approximation.

1.5 Our Contributions and Thesis Structure

Here, we present the new results of this thesis. The details of results for the MSR and MSD problems come in Chapter 2. Then, the details of results for the UCFLP come in Chapter 3. Finally, we conclude this thesis in Chapter 4.

1.5.1 The MSR and MSD Problems

For graphs with polynomially bounded Δ (for instance for unweighted graphs), the exact algorithm of Gibson *et al.* [GKK$^+$08b] for the MSR problem runs in time $n^{O(\log^2 n)}$. This gives us strong evidence that the MSR problem for these metrics is not **NP**-hard and perhaps, solvable in polynomial time. We make some progress in this direction and give a polynomial time exact algorithm for metrics induced by unweighted graphs in the case that no singleton clusters (*i.e.*, clusters having a single point) are allowed. This result reduces the unweighted MSR problem to the problem of finding the singleton clusters. In other words, it shows the difficult core of the problem is to determine which points should be a cluster by themselves. We also show that finding the best single cluster of each connected component is a $\frac{3}{2}$-approximation algorithm for the unweighted MSR problem without singletons. This algorithm has a better running time than the above exact algorithm. We contrast this simple algorithm with an integrality gap of at least essentially $3/2$ for a natural LP relaxation of the problem.

For Euclidean MSD (*i.e.*, points in $\mathbb{Q}^2$ and Euclidean metric), the exact algorithm of Capoyleas *et al.* [CRW91] for fixed k raises a question about the complexity of this problem for variable k. This is first asked by Doddi *et al.* as an open problem (see Section 6 in [DMR$^+$00a]). By giving a PTAS for the Euclidean MSD, we show that this problem is not **APX**-hard unless **P**= **NP**.

Recall that Hansen and Jaumard [HJ87] gave an exact algorithm for the MSD problem when $k = 2$. The best known approximation algorithm for the MSD problem with constant $k > 2$ is the 2-approximation algorithm that comes from the exact algorithm of the MSR problem in this case. Doddi et al. raised an open question (see Section 1.3 in [DMR$^+$00a]) about the complexity of this problem in this case. We answer this question by giving a polynomial time exact algorithm for the MSD problem with constant k.

1.5.2 The UCFLP

We present the first constant factor approximation algorithms with capacity violation factor less than 2. Particularly, we present two approximation algorithms with factors $(9, 3/2)$ and $(29.315, 4/3)$ for the UCFLP. In this process, we reduce the UCFLP to a more restricted version that we believe can be an important step in finding a constant approximation algorithm with $(1 + \epsilon)$ violation of capacities for any $\epsilon > 0$. Note that an $(O(1), 1 + \epsilon)$-approximation algorithm for the UCFLP is the best possible (up to the constant factor hidden in $O(.)$) as the UCFLP does not admit any $(1.463 - \epsilon, \beta)$-approximation for any constant $\epsilon > 0$ and any $\beta \geq 1$ unless **P**= **NP**. This is because the standard facility location problem is 1.463-hard [GK98]. We also consider the UCFLP restricted to Euclidean metrics and give a $(1 + \epsilon, 1 + \epsilon)$-approximation that runs in quasi-polynomial time.

Chapter 2

The Minimum Sum of Radii and Diameters Problem

In this chapter, we consider the Minimum Sum of Radii (MSR) and the Minimum Sum of Diameters (MSD) problems. These two problems are closely related to each other. The input and output for both problems are the same. The only difference is in the objective function. The formal definitions of these problems are as follows. We are given a set of points V and a metric d defined on these points and an integer k. The goal is to partition V into k clusters (subsets of V) so as to minimize the sum of radii of clusters in the MSR problem or the sum of diameters of the clusters in the MSD problem. We can consider these points as the vertices of a complete graph with a metric cost function on the edges. More formally, we are given a complete graph $G = (V, E)$ with metric edge cost (or distance) function $d : E \to \mathbb{Q}^+$ and an integer k. The goal is to partition V into k sets $V_1, V_2, \ldots, V_k$. In the MSR problem, we want to minimize $\sum_{i=1}^{k} \mathrm{rad}(V_i)$, where $\mathrm{rad}(V_i)$ is the radius of V_i. In the MSD problem, we want to minimize $\sum_{i=1}^{k} \mathrm{diam}(V_i)$, where $\mathrm{diam}(V_i)$ is the diameter of V_i. We call a cluster containing a single vertex a *singleton* or *zero ball*.

Notice that when we have k sets covering V, *i.e.*, each vertex of V is in at least one of these sets, we can easily obtain a partition of V from these sets without increasing their total radii or diameters. As a consequence, we also consider any cover of V as a feasible solution.

Recall that, sometimes, we can represent a metric in a more compact way by a non-complete edge-weighted graph G, where the distance between any pair of vertices is the shortest path distance of these vertices in G.

2.1 Overview of Related Works

In this section, we briefly review the works related to the MSR and MSD problems and discuss their connection to our results.

2.1.1 Connection between the MSR and MSD Problems

There is a strong connection between the MSR and MSD problems. This connection comes from the similarity of their objective functions and the relationship between radius and diameter of a graph. This well known relationship is as follows:

Observation 2.1.1 *In an edge-weighted graph G with metric weights, we have* $\mathrm{rad}(G) \leq \mathrm{diam}(G) \leq 2\,\mathrm{rad}(G)$.

Doddi *et al.* [DMR$^+$00b] noticed that the above observation has an important implication for the MSR and MSD problems. As it is shown in the following proposition, any α-approximation algorithm for the MSR (MSD) problem is a 2α-approximation algorithm for the MSD (MSR, respectively) problem. Thus, by designing a good approximation algorithm for one problem, we get a reasonably good approximation algorithm for the other problem.

Proposition 2.1.1 *[DMR$^+$00b] An α-approximate solution for the MSR (MSD) problem is a 2α-approximate solution for the MSD (MSR, respectively) problem.*

Proof. Let G be the input graph, and OPT_R and OPT_D be the optimal values for the MSR and MSD problems, respectively. Assume that clusters $V_1^D, \ldots, V_k^D$ give the optimum value for the MSD problem. Also, let $V_1^R, \ldots, V_k^R$ be a clustering that gives the optimum value for the MSR problem. We have

$$\mathrm{OPT}_R \leq \sum_{i=1}^{k} \mathrm{rad}(V_i^D) \leq \sum_{i=1}^{k} \mathrm{diam}(V_i^D) = \mathrm{OPT}_D, \tag{2.1}$$

where the first inequality holds, because OPT_R is the optimum value for the MSR problem and the second holds because of Observation 2.1.1. Similarly, we have

$$\mathrm{OPT}_D \leq \sum_{i=1}^{k} \mathrm{diam}(V_i^R) \leq \sum_{i=1}^{k} 2\,\mathrm{rad}(V_i^R) = 2\mathrm{OPT}_R. \tag{2.2}$$

Now, let $V_1, \ldots, V_k$ be an α-approximate solution for the MSR problem. The cost of this solution for the MSD problem is

$$\sum_{i=1}^{k} \mathrm{diam}(V_i) \leq \sum_{i=1}^{k} 2\,\mathrm{rad}(V_i) \leq 2\alpha\mathrm{OPT}_R \leq 2\alpha\mathrm{OPT}_D,$$

where we used Observation 2.1.1 and Equation (2.1) in the first and last inequality, respectively. In an analogous manner, by using Observation 2.1.1 and Equation (2.2), we can prove an α-approximate solution for the MSD problem is a 2α-approximate solution for the MSR problem. ■

While we have the above interesting connection, these problems have a very important difference. We call a cluster *maximal* for the MSR (MSD) problem if one cannot add any vertex to it without increasing its radius (diameter, respectively). Any solution can be turned into one having only maximal clusters without increasing the cost. Therefore, we can ignore non-maximal clusters and only consider maximal clusters in the solutions for the MSR and MSD problems.

Consider a maximal cluster of radius r for the MSR problem and let v be an arbitrary center of this cluster. By definition, this cluster must contain all the vertices in the graph having distance at most r from v. Recall that we call the set of such vertices the *ball of radius r around v* and denote it by $B(v, r)$. As is shown in the following proposition, the number of distinct maximal clusters for the MSR problem is at most n^2, while the number of distinct maximal clusters for the MSD problem can be exponential. This yields a very important advantage when we deal with the MSR problem. We can enumerate all subsets of the maximal clusters having size at most l in time $O(n^{2l})$. When l is a constant, this is polynomial time and when l is $O(\log^c(n))$ for some constant c, this is quasi-polynomial time. This enumeration is an important part of many related results. The following proposition was first observed by Doddi *et al.* [DMR$^+$00b].

Proposition 2.1.2 *[DMR$^+$00b] In the MSR problem, the number of distinct maximal clusters is at most n^2.*

Proof. We claim that each vertex can be the center of at most n distinct maximal clusters and hence, there are at most n^2 distinct maximal clusters overall. Consider an arbitrary vertex v and sort the vertices in the graph in the ascending order of their distances from v. Let $0 = r_0 \leq r_1 \leq \cdots \leq r_l$ be the distinct distances. It is not hard to see that for any $1 \leq i < l$, if $r_i \leq r < r_{i+1}$, we have $B(v, r_i) = B(v, r)$. Therefore, the only distinct maximal clusters centered at v are $B(v, r_i)$ for $1 \leq i \leq l$ and v is center of $l \leq n$ distinct balls. ■

From now on, when we discuss the clusters in the MSR problem, we mean the clusters from the above set of distinct maximal clusters, which has size at most n^2.

2.1.2 General Metrics

There are only three separate works for the general case of the MSR and MSD problems. Beating the current best approximation ratios for the MSR and MSD problems has been an open problem for the last eleven years. It seems that achieving this needs the introduction of new techniques or perhaps some important new structural observations.

The First Approximation Result

Doddi *et al.* [DMR$^+$00b] gave the first approximation result for the MSD problem (and consequently, for the MSR problem by Proposition 2.1.1). They designed an algorithm that yields a solution with no more than $10k$ clusters having cost within factor $O(\log(n/k))$ of cost of an optimal solution, which has k clusters. Clearly, this is a bicriteria approximation algorithm. Here, we briefly explain the high-level idea of their algorithm.

They start with the trivial clustering that contains n singletons and iteratively decrease the number of clusters to at most $10k$. First, they observe the simple fact that if two clusters with diameters

D_1 and D_2 intersect, the maximum distance between any pair of their combined vertices is at most $D_1 + D_2$ and we can cover all their vertices with a single cluster of diameter $D_1 + D_2$. In other words, we can merge them without increasing the total diameter[1]. Second, they show that given a graph with l vertices, one can find in polynomial time a clustering with at most $3k(1 + \ln(l/k))$ clusters having cost at most $O(\ln(l/k))\text{OPT}$. Now, assume at the start of ith iteration the clusters are $C_1, \ldots, C_l$. Choose a random representative from each cluster and consider the graph and metric induced on this representative vertices. They give this graph to the algorithm mentioned above and get at most $3k(1 + \ln(l/k))$ clusters with cost at most $O(\ln(l/k))\text{OPT}$. Now, add these clusters to $C_1, \ldots, C_l$. It results in some intersecting clusters that after merging them, we must have at most $3k(1 + \ln(l/k))$ clusters, while we increased the cost only by $O(\ln(l/k))\text{OPT}$.

Let n_i be the number of clusters after the ith iteration and let $n_0 = n$. It is easy to see that $n_{i+1} \leq 3k(1 + \ln(n_i/k))$ and the number of clusters decreases very fast. They show that after $5 + \log\log(n/k)$ iterations the number of clusters decreases from n to at most $10k$ and hence, the running time is polynomial time. Also, the cost of this solution is at most

$$\sum_{i=1}^{5+\log\log(n/k)} O(\ln(n_{i-1}/k))\text{OPT}.$$

They show that the first term of summation dominates the other terms and the cost is actually within factor $O(\ln(n/k))$ of the optimum value.

Later, in the journal version of their paper [DMR$^+$00a], Doddi *et al.* slightly improved their result. For a fixed ϵ, they gave a polynomial time algorithm that returns a solution with at most $((1 + \frac{2}{x})\frac{e}{e-1} + \delta)k$ clusters whose total diameter is within a factor of $(1 + \epsilon)(x + 2)\ln(\frac{n}{k\delta})$ of the optimum, where the parameters x, δ, and ϵ, act to control a tradeoff between the cost of the solution, the number of clusters returned, and the running time. The reader can verify that even with this improvement, we cannot have a clustering with number of clusters arbitrarily close to k.

The Current Best Ratio

Soon after the first result, Charikar and Panigrahy significantly improved it [CP01]. They gave a $(3.504 + \epsilon)$-approximation for the MSR problem that runs in time $n^{O(1/\epsilon)}$. Note that in contrast to the previous result, this is a true approximation algorithm, not a bicriteria one. In addition, the logarithmic approximation ratio is improved to a constant factor. By Proposition 2.1.1, this also yields a $(7.008 + \epsilon)$-approximation algorithm for the MSD problem. These are currently the best ratios known for these problems for general metrics. We give a high-level description of this algorithm here.

[1]The same fact is true for the sum of radii in the case of unweighted graphs (see Lemma 2.3.1), but not weighted graphs as in the case of sum of diameters

The natural LP relaxation of the MSR problem is as follows:

$$\begin{aligned}
&\text{minimize} && \sum_{u \in V} \sum_{r} r.y_u^{(r)} && \\
&\text{subject to} && \sum_{u \in V} \sum_{r: d(u,v) \leq r} y_u^{(r)} \geq 1 && \forall v \in V \\
& && \sum_{u \in V} \sum_{r} y_u^{(r)} = k && \qquad (2.3) \\
& && y_u^r \geq 0 && \forall u \in V, \forall r,
\end{aligned}$$

where $y_u^{(r)} = 1$ indicates a cluster of radius r is centered on vertex u. They first use the Lagrangian relaxation technique to simplify the problem by removing Constraint (2.3). The Lagrangian relaxation LP is as follows:

$$\begin{aligned}
&\text{minimize} && \sum_{u \in V} \sum_{r} (r + \lambda).y_u^{(r)} - k\lambda && \\
&\text{subject to} && \sum_{u \in V} \sum_{r: d(u,v) \leq r} y_u^{(r)} \geq 1 && \forall v \in V \\
& && y_u^r \geq 0 && \forall u \in V, \forall r.
\end{aligned}$$

This LP can be considered as a relaxation for the problem in which we do not have a hard limit on the number of clusters, but instead, we must pay a creation cost of λ in addition to its radius for each cluster (in other words, they transformed the problem to a facility location clustering). They designed a 3-approximation algorithm for this problem with an extra property that is necessary for the later steps.

Notice that when λ is small, the optimal solution has many clusters. For instance, when $\lambda = 0$, the optimal solution has n singleton clusters. Also, when λ is large, the optimal solution has just a few clusters. For example, when $\lambda = \mathrm{rad}(G)$, the single cluster containing the whole graph is an optimal solution. In other words, the number of clusters in an optimal solution does not increase when λ increases. Thus, one can use λ as a control parameter for the number of clusters.

One can use binary search on the value of λ to get clusterings that their number of clusters are closer and closer to k. If we indeed find a clustering with exactly k clusters, it can be shown that it is a 3-approximate solution for the MSR problem. Otherwise, we must be able to find two solutions S_1 and S_2 with k_1 and k_2 clusters ($k_1 < k < k_2$), respectively, such that their corresponding λ values, λ_1 and λ_2, are very close to each other. Let a and b be the coefficients such that $ak_1 + bk_2 = k$ and $a + b = 1$. Because λ_1 and λ_2 are very close to each other, it can be shown that $a. \mathrm{cost}(S_1) + b. \mathrm{cost}(S_2)$ is a close lower bound for the optimum. Finally, they give a randomized algorithm that chooses at most k clusters from $S_1 \cup S_2$ with expected cost within factor at most $\frac{3}{8\sqrt{3}-13} + \epsilon \approx 3.504 + \epsilon$ of the optimum in time $n^{O(1/\epsilon)}$.

The Exact Algorithm in Quasi-polynomial time

The last result for the general case belongs to Gibson *et al.* [GKK$^+$08b]. They presented a very interesting exact algorithm for the MSR problem that runs in time $n^{O(\log n \log \Delta)}$. Here, Δ is the aspect ratio of graph, which is the ratio of the cost of heaviest edge over the lightest edge. Later, they showed how this result can be transformed to a QPTAS for the MSR problem [GKK$^+$10]. A brief description of this algorithm is as follows.

First, they run a randomized algorithm to partition any set Q of vertices into two or more non-empty sets of *half diameter* such that the expected number of clusters cut in an optimal solution for Q is at most $32\log(|Q|)$. This randomized algorithm is exactly the same as the partitioning algorithm of [FRT03] for embedding general metrics into tree metrics. By Markov's inequality, the probability that this partition cuts more than $64\log(|Q|)$ clusters of an optimum is not more than $1/2$. We can guess these at most $64\log(|Q|)$ clusters by enumerating all subsets of at most $64\log(|Q|)$ clusters. By Proposition 2.1.2, this can be done in time $n^{O(\log |Q|)}$. For any of these subsets, we remove the at most $64\log(|Q|)$ chosen clusters from the partitions and recursively solve the problem on the remaining vertices of each partition. One can show that using a standard dynamic programming technique, we can combine the answers of the recursive calls to find the best cover with respect to the subset of chosen clusters that we removed. As a result, we can simply check each of these subsets of size at most $64\log(|Q|)$ and return the best solution among the solutions that we obtain corresponding to each subset. Then, the proof of correctness of this algorithm can be done with an inductive argument. Note that the branching factor of the recursion tree is $n^{O(\log |Q|)}$ and the depth of recursion is at most $\log \Delta$, because we cut the diameter in half when we partition any set of vertices Q. Therefore, the running time is in $n^{O(\log n \log \Delta)}$

The Gap between Hardness and Algorithmic Results

Doddi *et al.* [DMR$^+$00b] showed that unless **P**= **NP**, for any $\epsilon > 0$, there is no $(2-\epsilon)$-approximation algorithm for the MSD problem even when the metric is induced by an unweighed graph. Note that this result does not imply **NP**-hardness of the MSR problem, because an exact algorithm of the MSR problem yields a 2-approximation algorithm for the MSD problem and does not violate the above hardness. Later, Gibson *et al.* [GKK$^+$08b] showed that the MSR problem is **NP**-hard even in metrics induced by weighted planar graphs and in metrics of constant doubling dimension. As one can expect, the aspect ratios of the graphs used for the reductions are exponential in n.

The above **NP**-hardness result shows that in terms of approximation ratio a PTAS is the best that we can hope for. Interestingly enough, the QPTAS found by Gibson *et al.* shows that the MSR problem is very unlikely to be **APX**-hard. The reason is that if someone can show that it is **NP**-hard to beat some approximation ratio c where c is a small constant, we can use the QPTAS algorithm to get an approximation ratio less than c for the MSR problem. This means that we can solve all problems in **NP** in quasi-polynomial time, which is widely believed to be false. The fact that the

MSR problem is unlikely to be **APX**-hard, gives us a strong evidence that it should be possible to design a PTAS for it. Unfortunately, there is a large gap between what should be possible, *i.e.*, a PTAS, and the current best approximation ratio of $3.504 + \epsilon$. Note that the above argument also gives a strong evidence for existence of a $(2 + \epsilon)$-approximation algorithm for the MSD problem, while the current best ratio is $7.008 + \epsilon$ and we also have a large gap here.

2.1.3 Special Cases

There are also several results for special cases of the MSR and MSD problems. When $k = 2$, the MSD problem in general metrics is solvable in polynomial time by a reduction to the 2-SAT problem. This is shown by Hansen and Jaumard [HJ87]. Moreover, they showed that for two clusters, the task of minimizing any function of their diameters can be done in polynomial time. Since we give a polynomial time exact algorithm for the MSD problem with constant k (see Section 2.5), we briefly explain their reduction to 2-SAT.

They do the following steps for any pairs of possible diameters for the two clusters. Note that these diameters are from one of the at most $O(n^2)$ different distances that we have in the metric. Fix two diameters d_1 and d_2 where $d_1 \leq d_2$. For any vertex u, they associate a boolean variable x_u. This variable takes value 0 if u is in the first cluster and takes value 1 if u is in the second cluster. Now, for any pair of vertices u and v, we add between zero to two clauses to the SAT formula. If $d(u,v) \leq d_1$, we do not add any clause. If $d_1 < d(u,v) \leq d_2$, we add the clause $(x_u \vee x_v)$, which forbids both u and v being in the first cluster. Finally, if $d_2 < d(u,v)$, we add the clauses $(x_u \vee x_v) \wedge (\neg x_v \vee \neg x_u)$, which forbids u and v being in the same cluster. After determining the 2-SAT formula, we check to see whether it is satisfiable or not. If it is not, this pair of diameters is impossible in a feasible solution. If it is, it gives an assignment of vertices to the first and the second clusters with diameters at most d_1 and d_2, respectively. They returned a feasible clustering obtained from a pair of diameters with minimum $d_1 + d_2$.

When k is *fixed* and the metric is *Euclidean*, Capoyleas *et al.* [CRW91] presented an exact algorithm for the MSD and MSR problems. In fact, their algorithm is more general and works for the minimization of any monotone function of the diameters and radii. The main part of this result is to show that any two clusters in some optimal solution can be separated by a line. Then, the rest is straight forward. It can be shown that we have at most $3k - 6$ distinct separating lines. We can move any of these lines to hit two vertices of the input. Therefore, we can enumerate all subsets of $3k - 6$ lines that contain two vertices of the input in time $O(n^{6k-12})$, partition the points with these lines and find the best solution with at most k partitions among all the partitions.

When k is *fixed*, for general metrics, there is a 2-approximation for MSD [DMR$^+$00b]. Although it is explained in a different way, one can verify that the algorithm of [DMR$^+$00b] is the following simple exact algorithm for the MSR problem (which is a 2-approximation algorithm for the MSD problem by Proposition 2.1.2): By proposition 2.1.2, we can enumerate any subsets of size at most k

of maximal clusters in time $O(n^{2k})$. For a constant k, this is polynomial time and we simply return the best feasible solution out of these $O(n^{2k})$ solutions that we get. For *Euclidean* MSR, there is an exact polynomial time algorithm [GKK$^+$08a]. This result also extends to L_1 and L_∞ metrics. This also implies a 2-approximation for the MSD problem on Euclidean plane, which is the current best ratio. As we use Gibson *et al.*'s framework to give a PTAS for the MSD problem (see Section 2.4), we give a detailed explanation of this algorithm here with reference to the journal version of this result [GKK$^+$12]. We call this algorithm the *GKKPV algorithm*.

The GKKPV Algorithm for Euclidean MSR

Consider an instance of the Euclidean MSR problem which consists of a set of points V on the plane along with an integer k. Here, the distance between any pair of points is the Euclidean distance of these points in the plane. Let $\mathcal{D}$ be the set of distinct maximal clusters which, in our special case, is the set of discs with a center $p \in V$ and radius $\|pq\|$ for some $q \in V$. Note that by Proposition 2.1.2, we have $|\mathcal{D}| \leq n^2$.

The high level description of the algorithm is as follows. An axis parallel rectangle is called *balanced* if the ratio of its width to length is at least $1/3$. The GKKPV algorithm is a dynamic programming algorithm which uses balanced rectangles to define the subproblems (i.e. the set of points enclosed in a balanced rectangle to be covered with a given number of discs). A *separator* for a (balanced) rectangle is any line which is perpendicular to its longer side and cuts it in the middle third of its longer side. The algorithm starts with a rectangle containing all the points, denoted by R_0, and cuts it into two smaller rectangles by selecting a separator line and solves the subproblems corresponding to smaller rectangles recursively.

A vertical or horizontal line is called *critical* if it either goes through a point $p \in V$ or if it is tangent to some disk in $\mathcal{D}$. It is not hard to see that all vertical lines between two consecutive critical vertical lines intersect the same set of discs. Thus, it is enough to fix an arbitrary vertical line between any two consecutive critical vertical lines, one to the left of the leftmost critical vertical line and one to the right of the rightmost critical vertical line and only consider these lines as potential vertical separators. We can also do the same for the horizontal lines to reduce the number of important horizontal separators. Thus, there are only $\Theta(n^2)$ vertical or horizontal lines to consider as separators. Let $L(R)$ denote the separators of a rectangle R from these fixed set of lines and the leftmost and rightmost separators of it.

The algorithm has a recursive procedure $\mathrm{DC}(R, \kappa, T)$ shown below, which takes as input a rectangle R, an integer $\kappa \geq 0$ and a subset $T \subseteq \mathcal{D}$ and computes an optimum MSR solution using at most κ discs for the set of points in $Q = \{q : q \in V \cap R,\ q \text{ is not covered by } T\}$. The idea is that T is the set of discs in the optimal solution that intersect R and are chosen in the higher levels of recursion. The algorithm calls $\mathrm{DC}(R_0, k, \emptyset)$ to find the best cover for V. The value of the sub-problem for a recursive call is stored in a dynamic programming table $\textsc{Table}[V \cap R, \kappa, T]$.

They prove that we only need to consider $|T| \leq \beta = 424$ to get an optimal solution (we explain this below); this combined with the fact that the number of distinct $V \cap R$ is $O(n^4)$ and $\kappa \in O(n)$, implies that the size of the dynamic programming table (and hence sub-problems to compute) is $O(n^{2\beta+5})$, which is polynomially bounded. In the following algorithm, we assume that I is a dummy disc of radius ∞.

Algorithm 2.1.1 Recursive Clustering: DC(R, κ, T)

```
1: If TABLE$[V \cap R, \kappa, T]$ is created then return its value; otherwise create it.
2: Let $Q = \{q : q \in V \cap R,\ q$ is not covered by $T\}$. If $Q = \emptyset$ then TABLE$[V \cap R, \kappa, T] \leftarrow \emptyset$ and
   return $\emptyset$.
3: If $\kappa = 0$, let TABLE$[V \cap R, \kappa, T] \leftarrow \{I\}$ (infeasible) and return $\{I\}$.
4: If $|Q| = 1$, let TABLE$[V \cap R, \kappa, T]$ be the solution with a singleton cluster and return its value.
5: Let $R'$ be a minimal balanced rectangle containing $V \cap R$ (this rectangle always can be found
   in polynomial time).
6: Initialize $\mathcal{D}' \leftarrow \{I\}$.
7: for all choices $\ell \in L(R')$ do
8:    for all choices of a set $\mathcal{D}_0 \subseteq \mathcal{D}$ of size at most 12 that intersect $\ell$ do
9:       for all choices of $\kappa_1, \kappa_2 \geq 0$ with $\kappa_1 + \kappa_2 + |\mathcal{D}_0| \leq \kappa$ do
10:         Let $R_1$ and $R_2$ be the two rectangles obtained from cutting $R'$ by $\ell$. Let $T_1 = \{D \in
            T \cup \mathcal{D}_0 :\ D$ intersects $R_1\}$ and $T_2 = \{D \in T \cup \mathcal{D}_0 :\ D$ intersects $R_2\}$.
11:         if $|T_1| \leq \beta$ and $|T_2| \leq \beta$ then
12:            Recursively call DC$(R_1, \kappa_1, T_1)$ and DC$(R_2, \kappa_2, T_2)$.
13:            if cost$(\mathcal{D}_0 \cup$ TABLE$[V \cap R_1, \kappa_1, T_1] \cup$ TABLE$[V \cap R_2, \kappa_2, T_2]) <$ cost$(\mathcal{D}')$ then
14:               Update $\mathcal{D}' \leftarrow \mathcal{D}_0 \cup$ TABLE$[V \cap R_1, \kappa_1, T_1] \cup$ TABLE$[V \cap R_2, \kappa_2, T_2]$.
15: Assign TABLE$[V \cap R, \kappa, T] \leftarrow \mathcal{D}'$ and return its value.
```

For the proof of correctness of GKKPV, the authors of [GKK$^+$12] prove the following lemmas. The first one is used to show that in Step 8, it is sufficient to only consider subsets of size at most 12.

Lemma 2.1.1 (Lemma 2.1 in [GKK$^+$12]) *If R is a rectangle containing a set of points $P \subseteq V$ and $\mathcal{O}$ is an optimum solution for P, then there is a separator for R that intersects at most 12 discs in $\mathcal{O}$.*

The next lemma essentially bounds the number of large discs of optimum intersecting a rectangle and is used to show that it is sufficient to only consider the choices of T_1 and T_2 as in Step 11. The proof of this lemma is based on the fact that the centers of discs of an optimum solution cannot contain the centers of other discs; so you cannot pack too many of them close to each other.

Lemma 2.1.2 (Lemma 2.2 in [GKK$^+$12]) *If $\mathcal{O}$ is an optimum solution for a set of points $P \subseteq \mathbb{Q}^2$ and R is an arbitrary rectangle of length $a > 0$, then the number of discs in $\mathcal{O}$ of radius at least a that intersect R is at most 40.*

The following lemma puts a lower bound on the distance of separator lines of nested rectangles. Its proof follows from the definition of separator lines.

Lemma 2.1.3 (Lemma 3.1 in [GKK$^+$12]) *Let DC$(R_i, \cdot, \cdot)$ be an ancestor of DC$(R_j, \cdot, \cdot)$ in the recursion tree. Let a be the length of R_j. Let ℓ_i be the separator that resulted in the recursion* *subtree that contains* $\mathrm{DC}(R_j, \cdot, \cdot)$ *and let* ℓ_j *be an arbitrary separator for* R_j*. If both of these separators are vertical or horizontal, then the distance between them is at least* $a/3$*.*

The main part of the proof of correctness is Lemma 3.2 in [GKK$^+$12] which inductively proves the correctness of procedure $\mathrm{DC}(R, \kappa, T)$. Let OPT be an optimal solution. The heart of this proof is the induction step. For that, suppose that $T \subseteq \mathrm{OPT}$ contains every cluster of OPT that contains a point in $V \cap R$ as well as a point in $V \setminus R$, OPT' is the set of clusters of optimum that have a point in $Q = \{q : q \in V \cap R,\ q \text{ is not covered by } T\}$, and $\kappa \geq |\mathrm{OPT}'|$. Note that the sole purpose of OPT' is to cover Q and hence, OPT' is an optimum cover for Q. From Lemma 2.1.1, R' has a separator ℓ' that intersects a set $\tilde{\mathcal{D}_0} \subseteq \mathcal{D}$ of at most 12 clusters of OPT'. Without loss of generality, assume ℓ' is vertical. We can move ℓ' (to left or right) to become a line $\ell \in L(R')$ that intersects the same set $\tilde{\mathcal{D}_0}$. Consider the choice of $\mathcal{D}_0 = \tilde{\mathcal{D}_0}$. Let OPT_1 and OPT_2 be the clusters of OPT' to the left and right of ℓ and consider choices of $\kappa_1 = |\mathrm{OPT}_1|$ and $\kappa_2 = |\mathrm{OPT}_2|$. Let R_1, R_2, T_1, T_2 be exactly as in the algorithm.

Then, the proof of correctness follows inductively. One can show that, by the induction hypothesis, $\mathrm{DC}(R_1, \kappa_1, T_1)$ and $\mathrm{DC}(R_2, \kappa_2, T_2)$ return an optimal solution. To complete the proof of correctness, we only need to prove that $|T_1| \leq \beta$ and $|T_2| \leq \beta$. The proof for $|T_1| \leq \beta$ is as follows. The proof for T_2 is analogous.

Every disc in T_1 come from a guessed $\mathcal{D}_0$ for a separator in a higher level of the nested recursive calls. We partition the separators of higher level recursive calls, into two groups and bound the number of discs from each set. Assume R_1 is a rectangle of length a. Let $\bar{R}$ be a square of side $5a$ centered at the center of R_1. Consider the separators of higher level calls that do not intersect $\bar{R}$. A disc that intersects these separators and R_1 has radius at least a and by Lemma 2.1.1, there can be only 40 of them in T_1, because $T_1 \subseteq \mathrm{OPT}$. Now, consider the rest of separators. These separators intersect $\bar{R}$. By Lemma 2.1.3, it can be seen that there are at most $5a/(a/3) + 1 = 16$ vertical (horizontal) such separators. Therefore, there are at most $32 \times 12 = 384$ discs coming from these separators in T_1 that can intersect R_1. Thus, we must have $|T_1| \leq 384 + 40 = 424 = \beta$.

2.1.4 Generalizations

One can generalize the MSR and MSD clustering problems to the case that the distance between vertices is asymmetric metric or even not metric. Recall that in asymmetric metrics, we have the triangle inequality, but not the symmetry. In the case of the non-metric distances, the graph must be complete and we do not have even the triangle inequality. The diameter and radius have the same definition in presence of these kind of distances, but the the property that $\mathrm{rad}(G) \leq \mathrm{diam}(G) \leq 2\,\mathrm{rad}(G)$ does not exist anymore. As a result, Proposition 2.1.1 does not hold and the strong connection between the two problems does not exist. In contrast, Proposition 2.1.2 still holds even for these distances.

Doddi *et al.* [DMR$^+$00a] showed that unless **P**= **NP**, for any polynomial time computable function $\alpha(n)$, the non-metric MSD problem cannot be approximated within a factor $\alpha(n)$. This statement holds even when $k = 3$. In contrast, the polynomial time exact algorithm of Hansen and Jaumard [HJ87] for MSD with $k = 2$ works even if the cost function is not a metric.

For the non-metric MSR problem, when k is constant, the simple algorithm that enumerates all subsets of size at most k of clusters still works, because as mentioned above, Proposition 2.1.2 still holds. This shows a great difference with the non-metric MSD problem. When k is not constant, the hardness of the non-metric MSR problem is unknown. We just know that it is trivially as hard as the asymmetric MSR problem in this case.

For the asymmetric MSR problem, there is an (H_{n-k})-approximation algorithm by Charikar and Panigrahy [CP04], where H_i is the ith Harmonic number. This is the only known result for asymmetric metrics.

2.2 New Results

For graphs with polynomially bounded Δ (for instance for unweighted graphs), the exact algorithm of Gibson *et al.* for the MSR problem [GKK+08b] runs in time $n^{O(\log^2 n)}$. This gives us strong evidence that the MSR problem for these metrics is not **NP**-hard and one should be able to design an exact algorithm for the MSR problem when restricted to these metrics. In contrast, the best known polynomial time algorithm even for this case is the $(3.504+\epsilon)$-approximation algorithm of Charikar and Panigrahy [CP01]. We make some progress in this direction and give a polynomial time exact algorithm for metrics of unweighted graphs in the case that no singleton clusters are allowed.

Theorem 2.2.1 *There is a polynomial time exact algorithm for the unweighted MSR problem when no singletons are allowed.*

It should be noted that having no singleton clusters may be a reasonable assumption for some applications. In addition, our result reduces the unweighted MSR problem to the problem of finding the singleton clusters. In other words, it shows the difficult core of the problem is to determine which points should be a cluster by themselves.

Moreover, we show that there is a simple $\frac{3}{2}$-approximation algorithm for unweighted graphs in the case that no singleton clusters are allowed. This algorithm has a better running time than the above exact algorithm. We contrast this simple algorithm with an integrality gap of at least essentially $\frac{3}{2}$ for a natural LP relaxation of the problem.

Theorem 2.2.2 *Finding the best single cluster of each connected component is a $\frac{3}{2}$-approximation algorithm for the unweighted MSR problem without singletons.*

For Euclidean MSD (*i.e.*, points in $\mathbb{Q}^2$ and Euclidean metric), the exact algorithm of Capoyleas *et al.* for fixed k raises a question about the complexity of this problem for variable k. This is first asked by Doddi *et al.* as an open problem (see Section 6 in [DMR+00a]). Recall that the exact algorithm of [GKK+12] gives a 2-approximation for Euclidean MSD. In contrast, there is a ratio 2 hardness for the general case [DMR$^+$00a]. Thus, it is not trivial if we can beat this factor of 2. By giving a PTAS for the Euclidean MSD, we show that this problem is not **APX**-hard unless **P**= **NP**.

Theorem 2.2.3 *There is an algorithm such that when given a set of n points V in $\mathbb{Q}^2$, an integer k, and an error bound $\epsilon > 0$, finds in time $n^{O(1/\epsilon)}$ a $(1+\epsilon)$-approximate solution to the MSD problem on the given input.*

Recall that Hansen and Jaumard [HJ87] gave an exact algorithm for the MSD problem when $k = 2$. The best known approximation algorithm for the MSD problem with constant $k > 2$ is the 2-approximation algorithm that comes from the exact algorithm of the MSR problem in this case. Doddi et al. raised an open question (see Section 1.3 in [DMR$^+$00a]) about the complexity of this problem in this case. We answer this question by giving an exact algorithm for the MSD problem with constant k.

Theorem 2.2.4 *There is a polynomial time exact algorithm for the MSD problem when k is constant.*

Finally, we have two simple observations (see Section 2.6) for the generalization to asymmetric metrics.

2.3 MSR Restricted to Unweighted Graphs

In this section, we focus on the MSR problem when the metric is the shortest path metric of an unweighted graph. First, note that if one can optimally solve the MSR problem for some arbitrary metric in polynomial time for connected graphs, then using a standard dynamic programming approach, one can optimally solve the problem for that metric for all graphs:

Proposition 2.3.1 *The MSR problem reduces in polynomial time to the MSR problem for connected graphs.*

Proof. Assume we know how to solve the problem for connected graphs. First, for each $\kappa \leq k$, we find an optimal solution of κ clusters for each connected component. Let $H_1, H_2, \ldots, H_l$ be the connected components. We present how we can find the solution by adding one connected component at a time. For $1 \leq i < l$, we show how we can find an optimal solution of κ clusters for the graph $\cup_{j=1}^{i+1} H_j$ given the optimal solutions for the graph $\cup_{j=1}^{i} H_j$ for all $\kappa_1 \leq k$. For all $\kappa_1, \kappa_2 > 0$ such that $\kappa_1 + \kappa_2 = \kappa$, we combine the optimal solution of $\cup_{j=1}^{i} H_j$ with κ_1 clusters with the optimal solution of H_{i+1} with κ_2 clusters and store the best of these $\kappa - 1$ combined solutions as the optimal solution. ■

As a consequence, we are going to assume that the input graph is connected in the rest of this section. Also, recall that when we discuss the clusters in the MSR problem, we mean the clusters from the set of distinct maximal clusters and a maximal cluster for the MSR problem is a *ball* centered at some vertex of V. We start with some definitions that we use.

Definition 2.3.1 We call a ball of positive radius a *non-zero ball*. We say two balls *intersect* if they share at least one common vertex. We say two balls are *adjacent* if they do not intersect and there is an edge that connects two vertices of these balls. Among the optimal solutions for covering graph with at most k clusters, a *canonical optimal* solution is a solution with the minimum possible number of balls.

We have the following lemmas about the structure of a canonical optimal solution in an unweighted graph.

Lemma 2.3.1 *A canonical optimal solution does not have any intersecting balls.*

Proof. We show that two intersecting balls can be merged into one ball without increasing the total cost. Assume the intersecting balls are $B(v_1, r_1)$ and $B(v_2, r_2)$, where $r_1 \geq r_2$, and vertex u belongs to both of them (see Figure 2.1). Consider the shortest path between v_1 and u. Let v be the vertex on this path that has distance $\min(d(v_1, u), r_2)$ from v_1.

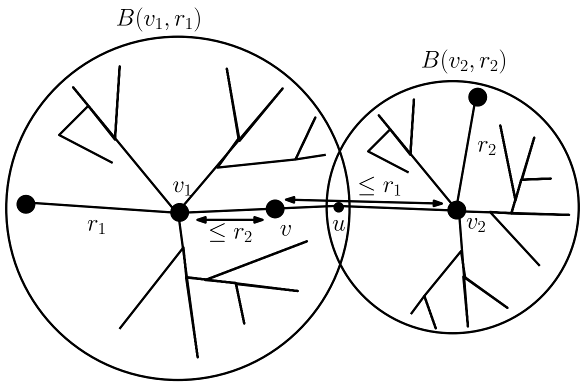


Figure 2.1: The intersecting balls $B(v_1, r_1)$ and $B(v_2, r_2)$. The ball $B(v, r_1 + r_2)$ covers all the vertices in these two balls.

The distance of v to all the vertices in $B(v_1, r_1)$ is at most $r_1 + r_2$, because the distance of v to v_1 is at most r_2 and the distance v_1 to the vertices in $B(v_1, r_1)$ is at most r_1. Also, the distance of v to all the vertices in $B(v_2, r_2)$ is at most $r_1 + r_2$, because the distance of v to u is at most $r_1 - r_2$, the distance of u to v_2 is at most r_2, and the distance of v_2 to the vertices in $B(v_2, r_2)$ is at most r_2. Therefore, $B(v, r_1 + r_2)$ contains all the vertices in $B(v_1, r_1)$ and $B(v_2, r_2)$ and in a solution, we can replace these two balls with $B(v, r_1 + r_2)$ without increasing the cost. ■

Remark 2.3.1 With a similar proof, one can show that we can cover two adjacent balls with radii r_1 and r_2 with a ball of radius $r_1 + r_2 + 1$. This shows that for the case of unweighted graphs, if we increase the number clusters by one, the optimum value decreases by at most one.

Lemma 2.3.2 *In a canonical optimal solution, each ball is adjacent to at most two non-zero balls.*

Proof. Let $B(v, r)$ be a ball in a canonical optimal solution and let $B(v_1, r_1), B(v_2, r_2), \dots B(v_l, r_l)$ be its adjacent balls such that $r_1 \geq r_2 \geq \cdots \geq r_l$ and assume that $l \geq 3$. We show that we can find a ball with radius $r + r_1 + r_2 + 1$ that covers $B(v, r)$ and all its adjacent balls. As a result, if $r_3 > 0$, we can decrease the number of balls in the canonical optimal solution without increasing the cost, which is a contradiction. Consider the shortest path between v and v_1. Let u be the vertex on this path that has distance $\min(d(v, v_1), r_1 - r_2)$ from v (see Figure 2.2). Since the shortest path has length at most $r + 1 + r_1$, the distance of u to v_1 is at most $r + r_1 + 1 - (r_1 - r_2) = r + r_2 + 1$. Clearly, the distance of u to v is at most $r_1 - r_2$.

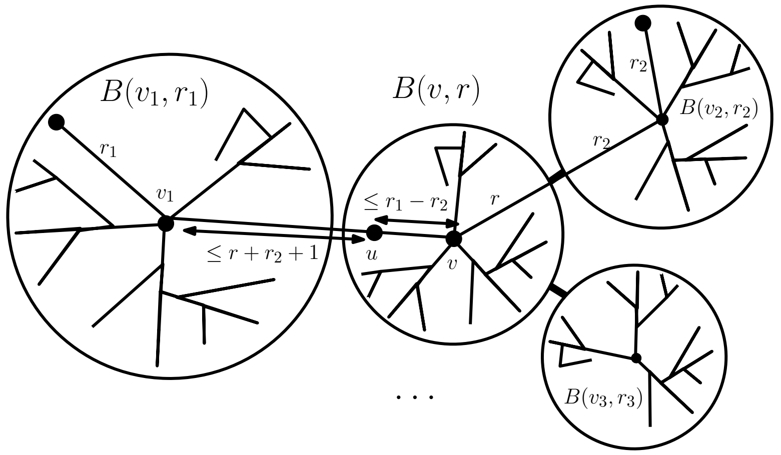


Figure 2.2: The ball $B(v, r)$ and its adjacent balls. The ball $B(u, r + r_1 + r_2 + 1)$ covers $B(v, r)$ and all its adjacent balls.

We claim that the ball $B(u, r+r_1+r_2+1)$ covers $B(v, r)$ and all its adjacent balls. The distance of u to all the vertices in $B(v_1, r_1)$ is at most $r + r_1 + r_2 + 1$, because the distance of u to v_1 is at most $r + r_2 + 1$ and the distance of v_1 to the vertices in the ball $B(v_1, r_1)$ is at most r_1. The distance of u to all the vertices in $B(v_i, r_i)$ for $i \geq 2$ is at most $r + r_1 + r_2 + 1$, because the distance of u to v is at most $r_1 - r_2$, the distance v to v_i is at most $r + 1 + r_i \leq r + r_2 + 1$, and the distance of v_i to the vertices in the ball $B(v_i, r_i)$ is at most $r_i \leq r_2$. Similarly, the distance of u to all the vertices in $B(v, r)$ is at most $r_1 - r_2 + r \leq r + r_1 + r_2 + 1$. ■

Remark 2.3.2 It is easy to see that the above two lemmas hold even when we allow singleton clusters.

Suppose (G, k) is an instance of the (unweighted) MSR problem where no zero balls are allowed. Consider a canonical optimal solution $\mathcal{S}^*$ and suppose that we have k^* balls in $\mathcal{S}^*$. Since we can run the algorithm for all values $q \leq k$ and take the best solution of all, for simplicity of exposition we assume $k^* = k$. By Lemmas 2.3.1 and 2.3.2 and because no zero balls is allowed, the balls in $\mathcal{S}^*$ are disjoint and each has at most two adjacent balls. Therefore, the balls in $\mathcal{S}^*$ form a path or cycle. Assume that these balls form a path, say $B_1^*, B_2^*, \dots, B_k^*$, where each B_i^* is adjacent to B_{i+1}^*,

$1 \leq i < k$ (the case of a cycle reduces to the case of a path as we show shortly). We give an exact algorithm for this case. Let $\mathcal{B}$ be the set of all possible distinct balls. By Proposition 2.1.2, we have $|\mathcal{B}| \leq n^2$.

The general idea of our algorithm is as follows. For every $V' \subseteq V$, let $G[V']$ denote the induced subgraph of G on V'. Let $G_i = G[\cup_{l=1}^{i} B_l^*]$ (i.e. the subgraph of G induced by the vertices in the first i balls of the path). It is easy to see that for all $1 \leq i \leq k$, the solution $B_1^*, B_2^*, ..., B_i^*$ is an optimal solution for covering G_i with i balls. We present a recursive algorithm, called BESTCOVER, that given a graph H and the number of clusters j, returns a feasible solution, and this feasible solution is optimal when $H = G_i$ and $j = i$, for any $1 \leq i \leq k$. For the moment suppose that the algorithm works as described for values of $j < l$ (for some $l \leq k$). When $j = l$, we guess the ball B_l^* (by enumerating over all possible balls in $\mathcal{B}$) and remove this ball to get a new graph H'. Then, run BESTCOVER with parameters H' and $l - 1$ and return the union of the guessed ball and the solution of the recursive call. Note that regardless of whether $H = G_l$ or not, assuming that the recursive call returns a feasible solution for H', we have a feasible solution for H. Furthermore, if $H = G_l$ then for the guessed ball (namely B_l^*), $H' = G_{l-1}$ and the solution returned by the recursive call is optimal and has the same cost as the solution $\{B_1^*, B_2^*, ..., B_{l-1}^*\}$. Adding the guessed ball B_l^* to the returned solution, we get an optimal solution for G_l.

The difficulty with the above general idea is that even if we use a dynamic programming approach and save the solution of each distinct input of BESTCOVER that we encounter, there may still be exponentially many different inputs corresponding to exponentially many subsets of $V(G)$. We fix this problem with the crucial observation that we are interested to solve only the subproblems corresponding to graphs G_i. Of course, we do not know $\mathcal{S}^*$ and the subsets $\cup_{l=1}^{i} B_l^*$ in advance, but we show that we can find a polynomial size family of subsets of $V(G)$, called candidate family $\mathcal{F}$, that contains subsets $\cup_{l=1}^{i} B_l^*$ for $1 \leq i \leq k$. Then we only solve the problem for graphs induced by subsets in $\mathcal{F}$, which gives the solution for graph G as well, because $V(G) = \cup_{l=1}^{k} B_l^*$ is in $\mathcal{F}$.

Definition 2.3.2 A *candidate family* $\mathcal{F}$, is a set of subsets of $V(G)$ which consists of the following sets: a subset $S \subseteq V(G)$ is in $\mathcal{F}$ if $S = V(G)$ or if there exists a ball B such that $G \setminus B$ has at most two connected components and the set of vertices in one of those components is S.

Lemma 2.3.3 *A candidate family can be computed in polynomial time, has at most* $2n^2+1$ *members and contains subsets* $\cup_{l=1}^{i} B_l^*$ *for all* $1 \leq i \leq k$.

Proof. Recall that $|\mathcal{B}| \leq n^2$. We remove each of these $|\mathcal{B}|$ balls from G. If the number of connected components is at most two, we add the set of vertices in each component to $\mathcal{F}$. We add $V(G)$ to $\mathcal{F}$ as well. This can be done in polynomial time and we must have considered all members of $\mathcal{F}$ after checking all the balls. The number of subsets obtained this way can be at most $2|\mathcal{B}| + 1 \leq 2n^2 + 1$. When in this process, we remove B_i^* for some $1 < i \leq k$, we get at most two connected components. Therefore, we add the set of vertices $\cup_{l=1}^{i-1} B_l^*$ to $\mathcal{F}$ for all $1 < i \leq k$. Also, we added

$V(G) = \cup_{l=1}^{k} B_l^*$ to $\mathcal{F}$ as well. Thus, $\mathcal{F}$ contains subsets $\cup_{l=1}^{i} B_l^*$ for all $1 \leq i \leq k$. ■

The procedure BESTCOVER is presented below.

Algorithm 2.3.1 BESTCOVER(H, l)

```
Require: A graph $H$ and an integer $l > 0$
Ensure: A subset of at most $l$ balls covering $H$, which is optimal when $H = G_i$ and $l = i$ for some
    $1 \leq i \leq k$
 1: If TABLE$[V(H), l] \neq \emptyset$, return TABLE$[V(H), l]$.
 2: If $V(H) = \emptyset$, return $\emptyset$.
 3: If $l = 0$ then return “infeasible”.
 4: Find $v$ the center of $H$ and $r = \mathrm{rad}(H)$.
 5: If $l = 1$, return $B(v, r)$.
 6: for all choices of a ball $B \in \mathcal{B}$ do
 7:     if $V(H) \setminus B \in \mathcal{F}$ then
 8:         Store the union of $B$ and the result of BESTCOVER$(H \setminus B, l - 1)$ in the set of solutions $\mathcal{C}$.
 9: If $\mathcal{C}$ is empty, store $B(v, r)$ in TABLE$[V(H), l]$ and return it.
10: Choose a solution in $\mathcal{C}$ having the minimum cost, store it in TABLE$[V(H), l]$ and return it.
```

Algorithm 2.3.2

```
Require: A graph $G$ and an integer $k > 0$
Ensure: A subset $\mathcal{S}^*$ of at most $k$ balls covering $G$ having optimal cost
 1: for all choices of a ball $B \in \mathcal{B}$ do
 2:     if $H = G \setminus B$ is connected then
 3:         Compute a candidate family $\mathcal{F}$ for $H$ as in Lemma 2.3.3.
 4:         Define an array TABLE. For all $S \in \mathcal{F}$ and for all $1 \leq l < k$, set TABLE$[S, l] = \emptyset$.
 5:         for all $1 \leq q \leq k - 1$ do
 6:             Store the union of $B$ and the result of BESTCOVER$(H, q)$ in the set of solutions $\mathcal{C}$.
 7: Let $u$ be a center of $G$. Add the ball $B(u, \mathrm{rad}(G))$ as a soltuion to $\mathcal{C}$. Return the minimum cost
    solution in $\mathcal{C}$.
```

We prove the following which is a re-statement of Theorem 2.2.1. In this theorem, a canonical optimal solution can form a path or cycle of balls.

Theorem 2.3.1 *Algorithm 2.3.2 is a polynomial time exact algorithm for the unweighted MSR problem when no singletons are allowed.*

Proof. First, we show that Algorithm 2.3.2 runs in polynomial time. By Lemma 2.3.3, we can compute $\mathcal{F}$ in polynomial time and the number of table entries is polynomial. Thus, we just need to prove that the call BESTCOVER(H, q) runs in polynomial time. We only call BESTCOVER$(.,.)$ (including in recursive calls) for graphs H with $V(H) \in \mathcal{F}$. Since by Lemma 2.3.3, $|\mathcal{F}| \leq 2n^2 + 1$ and $l \leq k \leq n$, the number of distinct instances that are given as parameter to BESTCOVER$(.,.)$ is polynomial. Also, for a fixed H and l, BESTCOVER(H, l) makes a total of at most $|\mathcal{B}| \leq n^2$ calls to other instances of BESTCOVER$(.,.)$. As a result, the overall number of calls of BESTCOVER$(.,.)$ is polynomial.

Now, we prove the correctness of Algorithm 2.3.2. By Lemmas 2.3.1 and 2.3.2 and because no zero balls are allowed, the balls in a canonical optimal solution form a path or a cycle. Suppose that the set of balls in a canonical optimal solution are $B_1^*, \ldots, B_{k^*}^*$ (for some $k^* \leq k$) where each B_i^* is adjacent to B_{i+1}^*, $1 \leq i < k^*$ and when the balls form a cycle, $B_{k^*}^*$ and B_1^* are also adjacent. If $k^* = 1$, since we consider the best single cluster in Step 7 of the algorithm, we find the optimum solution. Assume $k^* > 1$. Consider the iteration in the main loop that $B = B_{k^*}^*$. Then, there is a canonical optimal solution in $H = G \setminus B$, which forms the path $B_1^*, \ldots, B_{k^*-1}^*$. Consider the iteration in Step 5 where $q = k^* - 1$. We need to prove that BESTCOVER(H, q) returns an optimal solution in this case.

We prove that BESTCOVER(H, l) returns a subset of at most l balls covering H, and the cost of this solution is optimal when $H = G_i$ and $l = i$ for all $1 \leq i \leq q = k^* - 1$. It is easy to see that BESTCOVER$(.,.)$ always returns a feasible solution. We prove the rest by induction on l. When $l = 1$, the function returns the ball with minimum radius that covers all the vertices in $V(H)$, which is trivially optimal. In particular, when $H = G_1$ this is a ball with the same radius as B_1^*, and is the optimum solution for H. Assume that when $H = G_{j-1}$ and $l = j - 1$, the function returns an optimal solution and consider the case $H = G_j$ and $l = j$. In Step 6 of Algorithm 2.3.1, when B is equal to B_j^*, $V(H) \setminus B$ will be equal to $\cup_{l=1}^{j-1} B_l^* = V(G_{j-1})$. By Lemma 2.3.3, the set $\cup_{l=1}^{j-1} B_l^*$ is in $\mathcal{F}$. Thus, in Step 8 of Algorithm 2.3.1, BESTCOVER$(G_{j-1}, j-1)$ will be called and by the induction hypothesis, an optimal solution will be returned having cost the same as cost of $\cup_{l=1}^{j-1} B_l^*$. Therefore, the union of this solution and B_j^* has the same cost as cost of $\cup_{l=1}^{j} B_l^*$ and is optimal. Hence, an optimal cover will be returned in Step 10 of Algorithm 2.3.1. ■

Corollary 2.3.1 *Given the location of singleton balls, there exists an exact algorithm for the unweighted MSR problem.*

The above corollary shows that we essentially reduced the unweighted MSR problem to the problem of finding the singleton clusters in an optimal solution.

Here, we show that the simple algorithm that returns the best single cluster is a $\frac{3}{2}$-approximation for the unweighted MSR without singletons; this is Theorem 2.2.2.

Proof of Theorem 2.2.2. By Lemmas 2.3.1 and 2.3.2 and because no zero balls are allowed, the balls in a canonical optimal solution form a path or a cycle. Suppose that the set of balls in a canonical optimal solution are $B_1^*, \ldots, B_{k^*}^*$ (for some $k^* \leq k$) where each B_i^* is adjacent to B_{i+1}^*, for all $1 \leq i < k^*$ and when the balls form a cycle, $B_{k^*}^*$ and B_1^* are also adjacent. Color all balls with odd indices with red and all balls with even indices with blue. Let $\text{OPT} = \text{OPT}_r + \text{OPT}_b$ where OPT_r and OPT_b are the total radii of red and blue balls, respectively. Let n_r and n_b be the number of red and blue balls, respectively. Consider the following two modifications: first increase the radius of red balls by one to get a solution with value at most $\text{OPT}_r + n_r + \text{OPT}_b \leq 2\text{OPT}_r + \text{OPT}_b$. Now, every two adjacent balls intersect and we can merge all the balls into a single ball without increasing the cost (similar to what we did in Lemma 2.3.1). By doing the same thing for the blue balls, we can get a single ball with cost at most $\mathrm{OPT}_r + 2\mathrm{OPT}_b$. Therefore, the solution of our algorithm is less than or equal to $\min(2\mathrm{OPT}_r + \mathrm{OPT}_b, \mathrm{OPT}_r + 2\mathrm{OPT}_b) \leq \frac{3}{2}\mathrm{OPT}$. ■

Remark 2.3.3 *It is interesting that we cannot beat this simple algorithm by using the natural LP relaxation for this problem as we can show it has an integrality gap of at least essentially $\frac{3}{2}$. See Section 2.7.2.*

We conclude this section with a simple observation. If we are given the center of balls in a canonical optimal solution, but we do not know the radius corresponding to each center then there is a simple 2-approximation algorithm. Note that this may be useful for a local search heuristic that given a set of centers swaps each center with another vertex in hope of getting a better solution.

Proposition 2.3.2 *There exists a 2-approximation algorithm for the unweighted MSR problem when the center of balls in a canonical optimal solution is given.*

Proof. Let S be the set of centers which are adjacent to another center. Find the best single ball in each component of $G \setminus S$ and return these balls and zero balls centered around the vertices in S as our solution. This algorithm is a 2-approximation. Notice that if two centers are adjacent in a canonical solution, both of them must be centers of zero balls. Consider an optimal solution and increase the size of all its non-zero balls by one. This at most doubles the cost of solution and makes all non-zero balls intersect their adjacent balls. This also covers all zeros ball not in S, because they must be adjacent to some non-zero ball. After merging all the intersecting balls, we obtain a solution with cost at most twice of the optimum and a single ball in each component of $G \setminus S$. Clearly, the cost of our solution is not more than the cost of this solution, which completes the proof. ■

2.4 PTAS for Euclidean MSD

In this section, we present a PTAS for the MSD problem in $\mathbb{Q}^2$. Throughout this section, we assume that $\epsilon > 0$ is a given fixed constant. We build upon the framework of Gibson *et al.* [GKK$^+$12] and introduce some new ideas to make it work for the MSD problem. Recall that Gibson *et al.* present an exact algorithm (that we called GKKPV) for the MSR problem restricted to Euclidean plane. Since our algorithm follows the same steps as the GKKPV algorithm for MSR, recall the overview of that algorithm along with the necessary lemmas for its proof of correctness that we presented in Section 2.1.3.

Let $\mathrm{CH}(C)$ be the convex hull of the points in a cluster C. Consider two clusters C_1 and C_2 of an optimum solution. If $\mathrm{CH}(C_1)$ and $\mathrm{CH}(C_2)$ intersect, we can replace the two clusters with one containing all the points in $C_1 \cup C_2$ without increasing the total diameter. Thus, without loss of generality, we can focus on the solutions in which the convex-hulls of their clusters are disjoint. Therefore, each cluster can be unambiguously defined by its convex-hull. This convex-hull belongs to the set of convex polygons having all corners from V. Let $\mathcal{P}$ be the set of these convex polygons. For this reason, from now on, when we say a *cluster*, we consider a convex polygon in $\mathcal{P}$.

There are three main difficulties in extending the arguments of GKKPV for MSR to MSD. First, as we have seen before in the MSR problem, one can bound the number of distinct maximal clusters that can appear in a solution by n^2. This allows one to enumerate over all possible clusters (and more generally over all constant size subsets of clusters) that appear in an optimal solution. This fact is critically used in the GKKPV algorithm. For the MSD problem, we do not have such a nice (i.e. polynomially bounded) characterization of clusters that can appear in an optimum solution. To overcome this obstacle, we show that for every cluster C, there is a cluster C' enclosing it whose diameter is at most a factor $(1 + \epsilon)$ of the diameter of C, and C' belongs to a polynomial size set of polygons. The critical property of C' is that it is simpler to describe: it is determined by $O(1/\epsilon)$ points of input.

The second difficulty in adapting the GKKPV algorithm is that they show one cannot have too many large clusters close to a rectangle of comparable length in any optimum solution (Lemma 2.1.2). To prove this, they use the simple fact that no disc (cluster) in an optimum solution can contain the center of another disc (cluster) as otherwise one can merge the two clusters into one with smaller total radius. In the MSD problem, clusters are not necessarily defined by a disc and there is no notion of center here. We can potentially have many thin (e.g. non-fat triangles) and non-overlapping polygons of an optimum solution inside a region. We can still argue that in an optimum solution, we cannot have too many large clusters in a bounded region but the packing argument is different from that of [GKK$^+$12] (see Lemma 2.4.4). Third, when we fix the first two issues, some parts of the correctness proof fails and we need to prove some extra properties to handle this new difficulty.

We start by stating a special case of Jung's theorem about the minimum enclosing ball of a set of points in $\mathbb{R}^n$ (when $n = 2$) and a definition that will be used later.

Lemma 2.4.1 *[Jun01] A convex polygon in $\mathbb{R}^2$ with diameter D has an enclosing circle with radius at most $\sqrt{3}D/3$.*

Definition 2.4.1 A *σ-restricted polygon* is a convex polygon with at most σ sides such that each side contains (not necessarily at the corners) at least two points of V.

We emphasize that each corner of a σ-restricted polygon is not necessarily a point of V; it is obtained from the intersection of two lines each of which contains at least two points of V.

Lemma 2.4.2 *For any convex polygon P and fixed $\epsilon > 0$, there is a $\sigma(\epsilon)$-restricted polygon Q with $\sigma(\epsilon) = 13 + \frac{4\sqrt{3}\pi}{3}\frac{1}{\epsilon} \approx 13 + \frac{7.26}{\epsilon}$, satisfying the following properties:*

1. *Q contains P,*

2. $\operatorname{diam}(Q) \leq (1+\epsilon)\operatorname{diam}(P)$*,*

3. For any vertical or horizontal line ℓ, ℓ intersects P if and only if ℓ intersects Q.

Proof. The sides of Q are extensions of a subset of sides of P that we choose. In other words, we choose a subset of sides of P, consider the straight lines extending these sides and the convex polygon defined by these lines is the polygon Q. The number of sides of Q is exactly equal to the size of the chosen subset of sides of P. Clearly, if we choose Q this way, it encloses P and therefore contains P. We show there is a subset of size $\sigma(\epsilon)$ of sides of P, call it S, such that the resulting polygon Q satisfies Properties 2 and 3 as well.

Let us order the vertices of P in clock-wise order $u_1, u_2, \ldots$ (starting from the left-most corner of P). We traverse the sides of P in clockwise order and choose some of them along the way, starting with selecting u_1u_2, and select only a small number of sides in total. In order to satisfy Property 3, we also make sure that the left-most, top-most, right-most, and bottom-most corners of P are selected to be in Q (along with the two sides defining those corners of P). So first, we choose the (at most) 8 sides defining each of these (at most) 4 corners of P and let S consists of these sides. The remaining sides added to Q are obtained by traversing on the sides of P (in clock-wise order) starting from u_1u_2 and we add them (if not already among those 8 sides) according to the following rules.

Let D be $\operatorname{diam}(P)$. Suppose the last side chosen was u_iu_{i+1}. We call u_ju_{j+1} ($j \geq i+1$) *dismissible* with respect to u_iu_{i+1} if it satisfies both of the following conditions (see Figure 2.3):

- its angle, say θ, with u_iu_{i+1} is less than $\pi/2$,
- $\operatorname{dist}(u_{i+1}, u_j) \leq \epsilon D/2$.

Let j be the largest index such that u_ju_{j+1} is dismissible with respect to the last chosen side. Then, we add this side to S, and continue until we arrive at u_1u_2. We let Q be the convex polygon defined by the set of straight lines obtained from the sides in S. Note that Q satisfies Property 3.

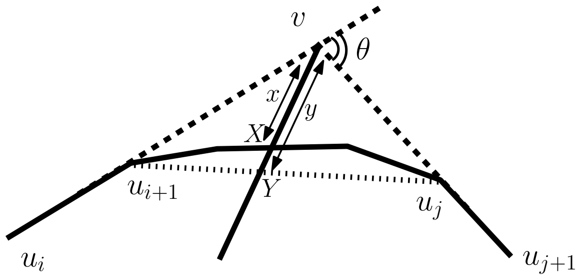


Figure 2.3: The solid sides u_iu_{i+1} and u_ju_{j+1} belong to P. The extension of sides u_iu_{i+1} and u_ju_{j+1} in polygon Q are shown with dashed lines. Line ℓ is shown with a solid line. The part of ℓ outside P has length x which is less than the length of side $u_{i+1}u_j$.

Let R be the perimeter of P. By Lemma 2.4.1, P can be enclosed by a circle of perimeter $2\sqrt{3}\pi D/3$ and hence, we have $R \leq 2\sqrt{3}\pi D/3$. It is not hard to see that we choose at most

$2\pi/(\pi/2)$ sides, because of the first condition in definition of a dismissible side and we choose at most $R/(\epsilon D/2)$ sides, because of the second condition. Therefore, we choose a total of at most $1 + 4 + \frac{4\sqrt{3}\pi}{3}\frac{1}{\epsilon}$ sides in our traversal. Given that we add at most 8 sides for the left-most, top-most, right-most, and bottom-most vertices of P, we have $|S| \leq 13 + \frac{4\sqrt{3}\pi}{3}\frac{1}{\epsilon}$.

It only remains to show that $\operatorname{diam}(Q) \leq (1+\epsilon)D$. It is enough to show that for any two corners of Q their distance is at most $(1+\epsilon)D$. Consider two arbitrary corners of Q, say v and w, and the line segment ℓ connecting these points. At most two segments of ℓ may be outside of P and this happens when both v and w are not corners of P. The section of ℓ that lies inside P has clearly length at most D. We prove that each of the (at most) two segments of ℓ which lies in between P and Q has length at most $\epsilon D/2$. Let us assume that v is the corner obtained from the two lines that contain sides $u_i u_{i+1}$ and $u_j u_{j+1}$ of P and consider the segment of ℓ that has v as an end-point (see Figure 2.3). Let X be the other end-point of it, which is a cross point of P and ℓ. We also name the cross-point of vw and $u_{i+1}u_j$ point Y. Let x be the length of vX and let y be the length of vY. Because of the convexity of P, we must have $y \geq x$. Consider the triangle $u_{i+1}vu_j$. Since the angle of lines u_iu_{i+1} and u_ju_{j+1}, θ, is less than $\pi/2$, the angle $u_{i+\hat{1}}vu_j \geq \pi/2$, which implies $u_{i+1}u_j$ is the longest side of triangle $u_{i+1}vu_j$. We know that the length of $u_{i+1}u_j$ is at most $\epsilon D/2$. We also know that a segment enclosed in a triangle has length at most equal to the longest side of the triangle. Thus, we must have $y \leq \epsilon D/2$ which implies $x \leq \epsilon D/2$, as wanted. ■

Let OPT denote an optimal solution. The above lemma implies that for every cluster P of OPT, there is a $\sigma(\epsilon)$-restricted polygon Q containing all the points of P such that $\operatorname{diam}(Q) \leq (1+\epsilon)\operatorname{diam}(P)$. Thus, if we replace each cluster P with the set of points in the corresponding $\sigma(\epsilon)$-restricted polygon (breaking the ties for the points that belong to different $\sigma(\epsilon)$-restricted polygons arbitrarily), we find a $(1+\epsilon)$-approximate solution. This enables us to use essentially the same Algorithm 2.1.1 (*i.e.*, GKKPV algorithm), where we enumerate over $\sigma(\epsilon)$-restricted polygons (instead of all clusters), which is a polynomially bounded set of polygons.

Let R_0 be a rectangle containing all the points of V. First, we define a $\Theta(n)$ size set of separators. The notion of a balanced rectangle and a separator line is the same. A vertical (horizontal) line is called *critical* if it passes through a point in the input. It is not hard to see that all vertical (horizontal) lines between two consecutive critical vertical (horizontal) lines are equivalent in the sense that they intersect the same set of clusters (recall that the clusters are convex polygons). Choose an arbitrary vertical (horizontal) line between each consecutive vertical (horizontal) critical lines. As before, let $L(R)$ show the separators of a rectangle R from these chosen lines and the leftmost and rightmost separators of it.

We apply the following modifications to Algorithm 2.1.1 to obtain our PTAS (Algorithm 2.4.1). We substitute $\mathcal{D}$ with $\mathcal{Q}$, which is the set of all $\sigma(\epsilon)$-restricted polygons, *i.e.*, for each $\sigma(\epsilon)$-restricted polygon Q, the set of points in Q forms a cluster in $\mathcal{Q}$. By definition of a $\sigma(\epsilon)$-restricted polygon,

$|\mathcal{Q}| \in O(n^{2\sigma(\epsilon)})$ which is a polynomial in the size of input for a fixed $\epsilon > 0$.

Algorithm 2.4.1 Recursive Clustering for Euclidean MSD: $\text{DC}(R, \kappa, T)$

```
1: If TABLE$[V \cap R, \kappa, T]$ is created then return its value; otherwise create it.
2: Let $X = \{x : x \in V \cap R,\ x$ is not covered by $T\}$. If $X = \emptyset$ then TABLE$[V \cap R, \kappa, T] \leftarrow \emptyset$
   and return $\emptyset$.
3: If $\kappa = 0$, let TABLE$[V \cap R, \kappa, T] \leftarrow \{I\}$ (infeasible) and return $\{I\}$.
4: If $|X| = 1$, let TABLE$[V \cap R, \kappa, T]$ be the solution with a singleton cluster and return its value.
5: Let $R'$ be a minimal balanced rectangle containing $V \cap R$.
6: Initialize $\mathcal{Q}' \leftarrow \{I\}$.
7: for all choices $\ell \in L(R')$ do
8:    for all choices of a set $\mathcal{Q}_0 \subseteq \mathcal{Q}$ of size at most 4 that intersect $\ell$ do
9:       for all choices of $\kappa_1, \kappa_2 \geq 0$ with $\kappa_1 + \kappa_2 + |\mathcal{Q}_0| \leq \kappa$ do
10:         Let $R_1$ and $R_2$ be the two rectangles obtained from cutting $R'$ by $\ell$. Let $T_1 = \{Q \in$
            $T \cup \mathcal{Q}_0 :\ Q$ intersects $R_1\}$ and $T_2 = \{Q \in T \cup \mathcal{Q}_0 :\ Q$ intersects $R_2\}$.
11:         if $|T_1| \leq \beta$ and $|T_2| \leq \beta$ then
12:            Recursively call DC$(R_1, \kappa_1, T_1)$ and DC$(R_2, \kappa_2, T_2)$.
13:            if cost$(\mathcal{Q}_0 \cup$ TABLE$[V \cap R_1, \kappa_1, T_1] \cup$ TABLE$[V \cap R_2, \kappa_2, T_2]) <$ cost$(\mathcal{Q}')$ then
14:               Update $\mathcal{Q}' \leftarrow \mathcal{Q}_0 \cup$ TABLE$[V \cap R_1, \kappa_1, T_1] \cup$ TABLE$[V \cap R_2, \kappa_2, T_2]$.
15: Assign TABLE$[V \cap R, \kappa, T] \leftarrow \mathcal{Q}'$ and return its value.
```

Note that for each cluster $P \in \mathcal{P}$, there is a cluster $Q \in \mathcal{Q}$ with $\text{diam}(Q) \leq (1 + \epsilon) \cdot \text{diam}(P)$, by Lemma 2.4.2. We fix an arbitrary such cluster Q and call it the representative of P and denote it by $\text{Rep}(P)$. Note that (again by Lemma 2.4.2):

Corollary 2.4.1 *A vertical (or horizontal) line intersects a cluster P if and only if it intersects* $\text{Rep}(P)$. *In particular for every axis-parallel rectangle R, P intersects R if and only if* $\text{Rep}(P)$ *intersects R.*

Note that although we have an exponential number of clusters in $\mathcal{P}$, we have a polynomially bounded size set of representatives. For a set of clusters $\tilde{\mathcal{P}}$, we use $\text{Rep}(\tilde{\mathcal{P}})$ to denote the set of clusters in $\mathcal{Q}$ that are representative of clusters in $\tilde{\mathcal{P}}$. In particular, $\text{Rep}(\text{OPT})$ are the representative clusters of OPT.

We keep the same dynamic programming table, TABLE$[V \cap R, \kappa, T]$, which stores a cover with at most κ clusters of $\mathcal{Q}$ having cost within $(1 + \epsilon)$ factor of the optimum cover for the set of points $\{q : q \in V \cap R, \text{q is not covered by } T\}$. We only consider sets $T \subseteq \mathcal{Q}$ with size at most $\beta \leq 83$. Similar to GKKPV algorithm, we later show that considering sets of at most this size is enough to get a solution within factor $1 + \epsilon$ of the optimum value. As a result, the size of the table is polynomially bounded. Also, we substitute $\mathcal{D}_0$ with a set $\mathcal{Q}_0 \subseteq \mathcal{Q}$ in line 8 and change the bound of 12 for $|\mathcal{D}_0|$ to a bound of 4 for $|\mathcal{Q}_0|$. This comes from the Lemma 2.4.3 which is the equivalent version of Lemma 2.1.1 for the Euclidean MSR problem. In addition, we use $\beta = 83$ for the bounds of $|T_1|$ and $|T_2|$ in line 11. Also, Lemma 2.4.4 is the equivalent version of Lemma 2.1.2. As we stated before, Gibson *et al.*' proof strategy does not work here and we use a new packing argument to prove this lemma. The new lemmas are as follows:

Lemma 2.4.3 *Suppose R is a rectangle containing a set of points $V' \subseteq V$ and $\mathcal{O}$ is an optimum solution of MSD on V'. Let $\mathcal{Q}'$ be the set of representatives of clusters of $\mathcal{O}$. Then there is a separator ℓ for R such that it intersects at most 4 clusters of $\mathcal{O}$ (and their representatives in $\mathcal{Q}'$).*

Proof. Let a be the length of the longer side of R. If one chooses a separator ℓ uniformly at random, then the probability that ℓ intersects a cluster $P \in \mathcal{O}$ is at most $\operatorname{diam}(P)/(a/3)$ (recall that the separators are in the middle one third of a rectangle). So the expected number of clusters of $\mathcal{O}$ that intersect ℓ is $\sum_{P\in\mathcal{O}} 3\operatorname{diam}(P)/a \le 3 \cdot \operatorname{cost}(\mathcal{O})/a \le 3\sqrt{2} < 4.25$, because $\operatorname{cost}(\mathcal{O}) \le \sqrt{2}a$ as they all are inside R. So there is a separator ℓ that intersects at most 4 clusters of $\mathcal{O}$, and by Corollary 2.4.1, it intersects at most 4 representative clusters of them in $\mathcal{Q}'$. ■

Lemma 2.4.4 *A rectangle R of length a intersects at most 3 clusters of* OPT *with diameter at least a.*

Proof. Let OPT' be the set of clusters of OPT with diameter at least a that are intersecting R. By way of contradiction, assume that C_1, C_2, C_3, and C_4 are four clusters of OPT' with diameters d_1, d_2, d_3, and d_4 each intersecting R. Without loss of generality, assume that $d_1 \ge d_2 \ge d_3 \ge d_4 \ge a$. This implies that the distance of any two points $u, v \in C_1 \cup C_2 \cup C_3 \cup C_4$ is at most $d_1 + d_2 + \sqrt{2}a$ since the total distance from u and v to R is at most $d_1 + d_2$ and the diameter of R is at most $\sqrt{2}a$. But $d_1 + d_2 + \sqrt{2}a < d_1 + d_2 + d_3 + d_4$, so if one replaces these 4 clusters of OPT with one single cluster (containing all those points), the total diameter decreases, which is a contradiction. ■

Corollary 2.4.2 *A rectangle R of length a intersects at most 3 clusters, which are representatives of clusters of diameter at least a in* OPT.

Note that Lemma 2.1.3 still holds here as it is based on the properties of balanced rectangles and separator lines, which we did not change. We prove the following Lemma which is the equivalent version of Lemma 3.2 in [GKK$^+$12].

Lemma 2.4.5 *Suppose that* $\mathrm{DC}(R, \kappa, T)$ *is called at some level of recursion by top-level invocation to $DC(R_0, k, \emptyset)$ in Algorithm 2.4.1. Let $T' \subseteq$* OPT *be those clusters of* OPT *that have a point in both $V \cap R$ as well as a point in $V \setminus R$ and suppose $T = \mathrm{Rep}(T')$. Also, let $X' = \{x : x \in V \cap R,\ x \text{ is not covered by } T'\}$ and $X = \{x : x \in V \cap R,\ x \text{ is not covered by } T\}$, and* OPT' *be the set of clusters of* OPT *that contain a point in X'. Suppose that $\kappa \ge |\mathrm{OPT}'|$. Then after calling* $\mathrm{DC}(R, \kappa, T)$, $\textsc{Table}(V \cap R, \kappa, T)$ *contains a κ-cover for X whose cost is at most* $(1+\epsilon) \cdot \operatorname{cost}(\mathrm{OPT}')$.

Proof. The proof is similar to that of Lemma 3.2 in [GKK$^+$12] and is by induction on $|V \cap R|$. The proof of base cases is straightforward. When $|X| < 2$, the reader can verify that the algorithm returns an optimal solution in the lines 2 to 4.

Consider the case that $|X| \geq 2$, and recall that R' is a balanced rectangle over points of $V \cap R$. Without loss of generality, assume the separators of R' are vertical. First note that OPT' is an optimum solution for X' with $|\mathrm{OPT}'|$ clusters since each point in $(V \cap R) \setminus X'$ is covered by T'. Therefore, using Lemma 2.4.3, R' has a separator ℓ' that intersects at most 4 clusters of OPT'; let S be that set of clusters of OPT' and let $\tilde{\mathcal{Q}}_0 = \mathrm{Rep}(S)$. We can move ℓ' to be one of the separators in $L(R')$; so we consider the choice of $\ell \in L(R')$ in the algorithm that intersects only set S of those in OPT' and consequently, only $\tilde{\mathcal{Q}}_0$.

Consider the iterations of the loops in lines 7 and 8 where ℓ is chosen as above and $\mathcal{Q}_0$ is chosen to be $\tilde{\mathcal{Q}}_0$, and κ_1 and κ_2 are the numbers of clusters of OPT' to the left and right of ℓ, respectively. Similarly, we denote the subsets of OPT' to the left and right of ℓ by OPT'_1 and OPT'_2, respectively. We define $T'_1 = \{C \in T' \cup S : \ C \text{ intersects } R_1\}$ and $T'_2 = \{C \in T' \cup S : \ C \text{ intersects } R_2\}$. Note that from the definitions of T, T', S, $\mathcal{Q}_0$, and Corollary 2.4.1, it follows that $T_1 = \mathrm{Rep}(T'_1)$ and $T_2 = \mathrm{Rep}(T'_2)$. Observe that $T'_1 \subseteq \mathrm{OPT}$ and T'_1 contains every cluster of OPT that contains a point in both $V \cap R_1$ as well as a point in $V \setminus R_1$. Also, OPT'_1 is the set of clusters of OPT' that contain points in $X'_1 = \{x : x \in V \cap R_1, \ x \text{ is not covered by } T'_1\}$ and $|\mathrm{OPT}'_1| = \kappa_1$. Let us define $X_1 = \{x : x \in V \cap R_1, \ x \text{ is not covered by } T_1\}$. Thus, we can use induction hypothesis on the call to $\mathrm{DC}(R_1, \kappa_1, T_1)$ so that the entry $\textsc{Table}[V \cap R_1, \kappa_1, T_1]$ contains a κ_1-cover of X_1 of cost at most $(1+\epsilon)\mathrm{cost}(\mathrm{OPT}'_1)$. Similarly, after call $\mathrm{DC}(R_2, \kappa_2, T_2)$, the entry $\textsc{Table}[V \cap R_2, \kappa_2, T_2]$ contains a κ_2-cover of X_2 of cost at most $(1+\epsilon)\mathrm{cost}(\mathrm{OPT}'_2)$, where $X_2 = \{x : x \in V \cap R_2, \ x \text{ is not covered by } T_2\}$. Thus, the clusters in $\mathcal{Q}_0 \cup \textsc{Table}[V \cap R_1, \kappa_1, T_1] \cup \textsc{Table}[V \cap R_2, \kappa_2, T_2]$ form a κ-cover of X with cost at most $(1+\epsilon)\mathrm{cost}(S) + (1+\epsilon)(\mathrm{cost}(\mathrm{OPT}'_1) + \mathrm{cost}(\mathrm{OPT}'_2)) \leq (1+\epsilon) \cdot \mathrm{cost}(\mathrm{OPT}')$.

It only remains to show that $|T_1|, |T_2| \leq 83$ and as a result, we actually call $\mathrm{DC}(R_1, \kappa_1, T_1)$ and $\mathrm{DC}(R_2, \kappa_2, T_2)$. The argument for this is similar to that in Lemma 3.2 of [GKK$^+$12] whose outline was given at the end of Section 2.1.3. We prove this for $|T_1|$ and the case of $|T_2|$ is analogous. Suppose we have called $\mathrm{DC}(\tilde{R}_0, \cdot, \cdot), \ldots, \mathrm{DC}(\tilde{R}_t, \cdot, \cdot)$ before arriving at instance (R, κ, T), where $\tilde{R}_t = R$. Then it follows that $T_1 \subseteq \bigcup_{j=0}^{t} \mathcal{Q}_0(\tilde{R}_j)$. Assuming that R_1 in the algorithm has length a, we take $\bar{R}$ to be the square of side $3a$ with the same center as R_1. By Lemma 2.1.3, the number of vertical (horizontal) separators $\ell(\tilde{R}_0), \ldots, \ell(\tilde{R}_t)$ that intersect $\bar{R}$ is at most $(3a)/(a/3) + 1 = 10$ (as they are at least $a/3$ apart). Therefore, using Lemma 2.4.3, the number of clusters of T_1 that belong to $\mathcal{Q}_0(\tilde{R}_j)$ (for those j's) is at most $4 \times 20 = 80$, *i.e.*, 4 for each of 20 vertical and horizontal separators that intersect $\bar{R}$. Now we bound the number of clusters of T_1 that belong to a $\mathcal{Q}_0(\tilde{R}_j)$ where $\ell(\tilde{R}_j)$ does not intersect $\bar{R}$. These clusters must have diameter at least a as they intersect both R_1 as well as $\ell(\tilde{R}_j)$. From Corollary 2.4.2, the number of such clusters is bounded by 3. Thus, we have $|T_1| \leq 83$. ■

It follows from Lemma 2.4.5 that after the call to $\mathrm{DC}(R_0, k, \emptyset)$ the entry $\textsc{Table}[V, k, \emptyset]$ contains a $(1+\epsilon)$-approximate solution for V. This completes the proof of Theorem 2.2.3.

Remark 2.4.1 *We believe that this algorithm can be extended to three-dimensional Euclidean space or perhaps, any fixed dimension Euclidean space. The extension of all lemmas except Lemma 2.4.2 is similar to the extension of [GKK$^+$12] for the MSR problem. If one can prove a statement similar to Lemma 2.4.2 in higher dimensions (which we believe should be possible for three dimensions), our algorithm generalizes to higher dimensions.*

2.5 MSD with constant k

Recall that Hansen and Jaumard [HJ87] presented an exact algorithm for the MSD problem when $k = 2$, but the case of constant $k > 2$ has been open. The reader may ask why their algorithm fails to generalize to this case. Let us consider the case of $k = 3$. Recall the details of their algorithm from Section 2.1.3. The first step is to guess the diameter of the clusters, which we can do for the case of $k = 3$ in polynomial time, too. In case of $k = 2$, they use a boolean variable to represent the cluster of each vertex, but here the variable that shows the cluster of each vertex should have 3 values. Therefore, if we try to model the problem this way, we need to solve a constraint satisfaction problem which is hard in general case. Also, one may try to model the problem with boolean variables in the following way. We associate three variables x_{vi} to each vertex v for $1 \leq i \leq 3$. Here, only one of these variables must be true that shows the cluster of this vertex. Unfortunately, this modelling results in a 3-SAT instance, which is again **NP**-hard in general form.

Our approach is as follows. Define a *canonical optimal solution* as before, *i.e.,* among the optimal solutions, a *canonical optimal* solution is a solution with the minimum possible number of clusters. We denote the set of all the distinct distances between vertices in the input graph by $\mathcal{D}$. We have the following trivial observations:

Observation 2.5.1 *Let C_1^* and C_2^* be two clusters in a canonical optimal solution. There is a vertex v in C_1^* and a vertex u in C_2^* such that* $\operatorname{dist}(u,v) > \operatorname{diam}(C_1^*) + \operatorname{diam}(C_2^*)$.

Observation 2.5.2 *The size of $\mathcal{D}$ is at most n^2. In addition, the diameter of any cluster in an optimal solution is in $\mathcal{D}$.*

The general idea of the algorithm (Algorithm 2.5.1) is as follows. Since the size of $\mathcal{D}$ is polynomially bounded, we can guess the diameters of clusters in an optimal solution (done in Line 3 of Algorithm 2.5.1). Consider a canonical optimal solution and two clusters C_1^* and C_2^* in it with diameters D_1 and D_2 that we have already guessed. By Observation 2.5.1, there is a vertex v in C_1^* and a vertex u in C_2^* such that $\operatorname{dist}(u,v) > \operatorname{diam}(C_1^*) + \operatorname{diam}(C_2^*)$. Denote by V_1 and V_2 the set of vertices with distance at most D_1 from v and distance at most D_2 from u, respectively. The crucial observations are that C_1^* and C_2^* are subsets of V_1 and V_2, respectively and V_1 and V_2 are disjoint. As a result, if we guess the vertices u and v (done in Line 4) and compute V_1 and V_2, we have separated the set of vertices that can be in C_1^* and C_2^*. If we do the same process for all pairs of clusters, we can separate the vertices that can be in a specific optimal cluster from the vertices in other optimal clusters, which is equivalent to finding the clusters in an optimal solution.

Algorithm 2.5.1 BESTCOVER(G, k)

```
Require: A graph $G$ and an integer $k > 0$
Ensure: An optimal clustering $\mathcal{C}$
 1: Initialize $\mathcal{C}$ with the cluster containing all the vertices.
 2: for all $2 \le q \le k$ do
 3:   for all choices of $q$ values $D_1, \dots, D_q$ from $\mathcal{D}$ do
 4:     for all choices of $q$ groups $S_1, \dots S_q$ of at most $q-1$ vertices each do
 5:       for all $1 \le i \le q$ do
 6:         Compute $V_i$ the set of vertices with distance at most $D_i$ from all the vertices in $S_i$.
 7:         if there are two vertices in $V_i$ with distance more than $D_i$ then
 8:           Discard this choice of $S_1, \dots, S_q$ and go to Step 4.
 9:       if $\cup_{i=1}^{q} V_i \ne V$ then
10:         Discard this choice of $S_1, \dots, S_q$ and go to Step 4.
11:       if $\sum_{i=1}^{q} D_i$ is less than cost of $\mathcal{C}$ then
12:         Update $\mathcal{C}$ with the solution $V_1, \dots, V_q$.
13: return $\mathcal{C}$.
```

The correctness of this algorithm is stated in the following theorem, which implies Theorem 2.2.4.

Theorem 2.5.1 *Algorithm 2.5.1 returns an optimal solution for the MSD problem in polynomial time when k is a constant.*

Proof. We first show that the algorithm runs in polynomial time. Assume we know the distances between all the pairs of vertices and the numbers in set $\mathcal{D}$ by running an all-pairs-shortest-path algorithm in a preprocessing step. There are $O(k)$ choices in Step 2. In Step 3, we choose at most k values from the set $\mathcal{D}$. By Observation 2.5.2, there are $O(n^{2k})$ choices for this step. In Step 4, we choose at most $k(k-1)$ vertices from V. There are $O(n^{k(k-1)})$ choices for this step. It is not hard to see that the body of for-loop in Step 4 takes $O(kn^2)$. As a result, the final running time is in $O(k^2 n^{k^2+k+2})$, which is a polynomial for constant k.

One can verify that when the algorithm is in Step 11, $V_1, \dots, V_q$ is a feasible solution with cost $\sum_{i=1}^{q} D_i$. Therefore, we just need to prove that for some choices in the for-loops, we encounter an optimal solution in Step 11. Consider a canonical optimal solution $\mathcal{C}^* = \{C_1^*, \dots, C_{k^*}^*\}$. Consider the iteration in Step 2 that $q = k^*$. By Observation 2.5.2, we have $\mathrm{diam}(C_i^*) \in \mathcal{D}$ for all $1 \le i \le q$. Consider the iteration in Step 3 that $D_i = \mathrm{diam}(C_i^*)$ for all $1 \le i \le q$. By observation 2.5.1, for all $1 \le i \ne j \le q$, there are vertices in C_i^* and C_j^*, say $v_j^{(i)}$ and $v_i^{(j)}$, respectively, such that $\mathrm{dist}(v_j^{(i)}, v_i^{(j)}) > D_i + D_j$. Consider the iteration in Step 4 that $S_i = \{v_j^{(i)} : 1 \le j \le q, j \ne i\}$. We shortly prove that for each $1 \le i \le q$, the V_i computed from this choice of S_i is equal to C_i^* and as a result, for the above mentioned choices in the for-loops, we encounter an optimal solution in Step 11.

Clearly, we have $S_i \subseteq \mathcal{C}_i^*$ and hence, each vertex in $\mathcal{C}_i^*$ has distance at most D_i from all the vertices in S_i. Thus, we must have $\mathcal{C}_i^* \subseteq V_i$. In addition, for any $i \neq j$, we must have $V_i \cap V_j = \emptyset$. The reason is that if there is at least a vertex, say u, in the intersection of these sets, by definition of V_i and V_j, we have $\mathrm{dist}(v_j^{(i)}, u) \leq D_i$ and $\mathrm{dist}(v_i^{(j)}, u) \leq D_j$. Therefore, we must have $\mathrm{dist}(v_j^{(i)}, v_i^{(j)}) \leq D_i + D_j$, which is a contradiction. We claim that $V_i \setminus C_i^* = \emptyset$ and in other words, $V_i = C_i^*$. Assume there is at least one vertex in $V_i \setminus C_i^*$, say u. Let C_j^* be a cluster covering u where trivially $j \neq i$. Clearly, we have $u \in V_j$ that means $u \in V_i \cap V_j$, which is a contradiction. ■

2.6 Asymmetric metrics

We have two simple results for asymmetric metrics. First, in Proposition 2.6.1, we show that all the results for the MSD problem extend to the asymmetric version. Second, in Proposition 2.6.2, we show that the asymmetric MSR problem is not approximable within a factor better than $\ln(n-k)$. As a result, it is highly unlikely to improve upon the ratio H_{n-k} of Charikar and Panigrahy' algorithm by more than an additive constant, because $\lim_{n\to\infty}(H(n) - \ln(n)) = \gamma \approx 0.577$. These two simple results almost answer all remaining open questions related to the asymmetric MSD and MSR problems.

Proposition 2.6.1 *The asymmetric MSD (AMSD) problem reduces to the MSD problem.*

Proof. We modify an instance of AMSD to an instance of MSD. Let u and v be two arbitrary vertices such that $d(u,v) > d(v,u)$. We set $d(v,u)$ equal to $d(u,v)$. We do this for all such pairs of vertices. We claim that this modification keeps the triangle inequality intact. Let u, v and w be three arbitrary vertices and the distance function before the modification be d and after modification be d'. Also, assume that $d(u,v) > d(v,u)$. Now, in the new instance, we have $d'(u,v) = d(u,v) \leq d(u,w) + d(w,v) \leq d'(u,w) + d'(w,v)$, where we use the fact that $d(u,w) \leq d'(u,w)$ and $d(w,v) \leq d'(w,v)$. Since d' is symmetric and u, v and w are arbitrary vertices, we conclude that d' satisfies triangle inequality. In addition, it is easy to check that this modification does not change the diameter of a cluster. In other words, any clustering has the same cost on the AMSD and MSD instance. Therefore, any α-approximation for MSD gives an α-approximation for ASMD and vice versa. ■

Proposition 2.6.2 *The asymmetric MSR problem is* ***NP****-hard to approximate within factor* $(1 - \delta)\ln(n-k)$ *for any constant* $\delta > 0$*, unless* ***NP*** $\subseteq \mathrm{DTIME}(n^{\log\log(n)})$.

Proof. We get this hardness by a simple reduction from the set cover problem. In the set cover problem, given a set of elements $U = \{e_1, \ldots, e_p\}$ and a family $\mathcal{F} = \{S_1, \ldots, S_m\}$ of subsets of U, we want to choose the minimum number of subsets from $\mathcal{F}$ that cover U, *i.e.*, their union is equal to U.

Given an instance of the set cover problem, we create an instance of the asymmetric MSR problem as follows. There is a vertex v_{S_i} for each set S_i of $\mathcal{F}$ and a vertex v_{e_j} for each element e_j of U in the graph. Thus, we have $n = m + p$ vertices in the graph. There is a unit cost edge from each v_{S_i} to all vertices v_{e_j} such that $e_j \in S_i$. The cost of all other edges is ∞. To complete the instance we let k to be equal to m.

A solution for this instance of MSR having bounded cost must choose all v_{S_i} vertices as centers of clusters. The reason is that each v_{S_i} can be covered only by itself with bounded cost. Thus, in a solution having bounded cost, we have some clusters around the v_{S_i} vertices where some of them have radius 0 and some have radius 1. As a consequence, any solution having bounded cost corresponds to a solution for the set cover problem. Thus, if there is a $(1-\delta)\ln(n-k)$-approximation for the asymmetric MSR problem for any constant $\delta > 0$, it gives a $(1-\delta)\ln(p)$-approximation for the set cover problem. By Feige's hardness result for the set cover problem [Fei98], this is impossible unless **NP** $\subseteq \mathrm{DTIME}(n^{\log\log(n)})$ ■

2.7 Further Attempts

As we stated before, the QPTAS of Gibson *et al.* [GKK$^+$10] gives strong evidence for the existence of a PTAS for the MSR problem, while the current best approximation algorithm for this problem is the decade old $(3.504 + \epsilon)$-approximation algorithm of Charikar and Panigrahy [CP01]. It is an important open question to improve this approximation ratio. Below, we describe some possible natural approaches that one might try to tackle this question, and the difficulties in each approach. These difficulties suggest that substantially new ideas may be required to answer this open question.

2.7.1 Local Search Attempts

The natural local search algorithm to consider for the MSR problem is the p-swap local search algorithm that is proved to be successful for the k-median problem [AGK$^+$01]. Recall the definition of this problem from Section 1.2. The p-swap algorithm for the k-median problem is as follows. We start with k arbitrary medians. We assign each vertex to its closest median to find the best solution with respect to this set of medians. At each iteration, we consider swapping each subset of size p of the medians with another subset of p vertices and find the value of new solution. If the swap results in an improvement, we update the set of medians and proceed to the next iteration. If no swap operation improves the solution, we stop and return the current set of medians as our answer. The running time of this algorithm is in $n^{O(p)}$ which is a polynomial in the size of input for a constant p. Arya *et al.* [AGK$^+$01] showed that the approximation ratio of this algorithm is $3 + \frac{2}{p}$.

Unfortunately, a similar strategy does not work for the MSR problem. One problem is that given a set of k centers, we do not know how to assign a radius to these centers to cover all the vertices and minimize the total radii. One option is to use the 2-approximation algorithm that we presented in Proposition 2.3.2, but there is a more serious problem. Assume that we know how to optimally assign the radii to the guessed set of centers, so we could adapt a similar p-swap algorithm for the MSR problem. We show that the p-swap algorithm has an unbounded locality gap.

We choose $k = q(p+1)+1$ for some integer q. Consider the unweighted graph in Figure 2.4. This graph is formed by a spider graph (on the left side of the figure) connected with an edge to a vertex, say u, of a complete graph having $k-1$ vertices (on the right side of the figure). The spider graph is formed by $p+1$ paths x_iy_i for $1 \leq i \leq p+1$, which are each connected to the vertex v with an edge. The number of edges in each path is $q-1 = \frac{k-1}{p+1} - 1$. Therefore, this graph has $(p+1)(\frac{k-1}{p+1}) + 1 + k - 1 = 2k-1$ vertices in total.

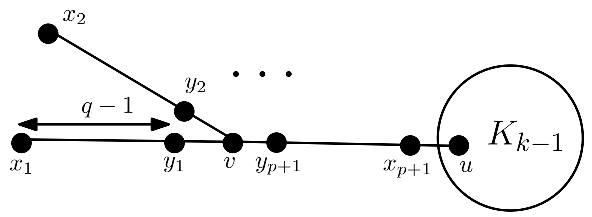


Figure 2.4: The graph of an instance with unbounded locality gap. This graph is formed by paths x_iy_i for $1 \leq i \leq p+1$ where each path has length $q-1$. All these paths are connected to the vertex v with an edge and x_{p+1} is connected to a complete graph of size $k-1$ at the vertex u.

Consider a solution for this instance in which all the vertices in the complete graph and the vertex v are the centers. The cost of the best solution with these centers is q. This is a local solution. The reason is that no matter how one swaps p vertices, we will not have a center on one of the x_iy_i paths and hence, we must pay at least q, which does not improve the solution. Now, the cost of the global solution is 1. To see this, we give the following solution. Place a center on all vertices of each path except x_{p+1} and a center on v ($k-1$ centers in total). Assign radius 0 to all these $k-1$ centers. Place a center on u and assign radius 1 to it so that it covers the whole complete graph of $k-1$ vertices and v_{p+1}. The cost of this solution is 1. Therefore, the locality gap is $q = \frac{k-1}{p+1} = \Theta(k)$.

Finally, we should mention that one may try to get around this problem by using Algorithm 2.1.1. In other words, we guess the number of zero balls and then, we only have a set of vertices for the location of zero balls. We determine the location of non-zero balls optimally by utilizing Algorithm 2.1.1. Thus, we can have a similar p-swap local search algorithm with the difference that now we only determine the location of zero balls by the local search operation. Unfortunately, one can verify that the same example as above has a bad locality gap. First, notice that there is only one optimal solution for this instance, which is the one we described above. This solution has exactly $k-1$ zero balls and this is the size of the set of vertices that we must use in the new local search algorithm that swaps the location of zero balls. It is not hard to see that the solution in which the zero balls are all on the K_{k-1} subgraph is a local solution and gives the same locality gap as above.

2.7.2 LP-based Attempts

Recall the natural LP relaxation of the problem that used by Charikar and Panigrahy [CP01]:

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{u\in V}\sum_{r} r.y_u^{(r)} & \\
\text{subject to} & \displaystyle\sum_{u\in V}\sum_{r:d(u,v)\le r} y_u^{(r)} = 1 & \forall v \in V \\
 & \displaystyle\sum_{u\in V}\sum_{r} y_u^{(r)} = k & \\
 & y_u^{(r)} \ge 0, & \forall u \in V, \forall r
\end{array}
$$

where $y_u^{(r)} = 1$ indicates a cluster of radius r is centered on vertex u. It is interesting that the LP has an unbounded integrality gap even for unweighted graphs. Consider a complete graph and $k = n-1$ as input. Clearly, the integral optimum value is 1. Now, we show a fractional solution of cost $\frac{1}{n-1}$, which gives an integrality gap of $n-1$. Consider this fractional solution. For all $v \in V$, set $y_v^{(0)} = \frac{n-2}{n-1}$ and for an arbitrary vertex u, set $y_u^{(1)} = \frac{1}{n-1}$. It is easy to verify the feasibility of this fractional solution.

Notice that in above bad example, the integral optimal value is 1. If in *all* the bad integrality gap instances, the integral optimal value is small, then there are a small number of non-zero balls in any optimal solution for these instances and we can guess the non-zero balls by enumeration. Hence, one might try to solve these bad instances by enumeration and solve the rest of instances with some LP rounding algorithm. Unfortunately, this strategy is not useful, because for any c, there is an instance having optimal value c, which yields an integrality gap of at least $\Omega(n^{1/c})$ even for unweighted graphs (we give this instance shortly). Notice that even if we go up to quasi-polynomial time and instances with $c = O(\log(n))$, the gap is still greater than 1, and hence, we cannot even get something similar to the exact quasi-polynomial time algorithm of Gibson *et al.* [GKK$^+$08b]. Therefore, the above LP does not seem promising even for unweighted graphs.

Let T_1 be a star with l leaves. For $i > 1$, we recursively define tree T_i. Let $u_1, \ldots, u_{l+1}$ be the roots of $l+1$ copies of T_{i-1}, denoted by $\tilde{T}_1, \ldots, \tilde{T}_{l+1}$. Build unweighted tree T_i by connecting its root u to these $l+1$ copies of T_{i-1} from $u_1, \ldots, u_{l+1}$ (see Figure 2.5). One can verify that T_i has height i, $l(l+1)^{i-1}$ leaves and

$$1 + (l+1) + \cdots + (l+1)^{i-1} + l(l+1)^{i-1} = \frac{(l+1)^i - 1}{l} + l(l+1)^{i-1} < 2(l+1)^i$$

vertices in total.

Our family of instances are T_i and $k = l(l+1)^{i-1}$ for $i \ge 1$. We claim that the cost of optimum for T_i with $k = l(l+1)^{i-1}$ is i. We prove this by induction. For the base case of $i = 1$, the value of optimum is 1 as we cannot cover every vertex with a zero ball. Assume that the statement is true for $i = p-1$. Now, consider T_p and $k = l(l+1)^{p-1}$ as input where $p > 1$. Let u be the root of T_p and $\tilde{T}_1, \ldots, \tilde{T}_{l+1}$ be the $l+1$ subtrees of u. If we place a ball of radius p on u, it covers all the vertices

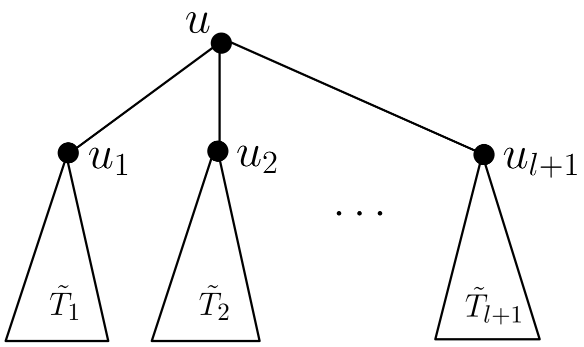


Figure 2.5: Recursive construction of tree T_i.

and as a result, the cost of optimum is at most p. Now, we prove the cost of optimum is not less than p by utilizing the induction hypothesis.

We call a solution *non-intersecting* if it does not contain intersecting balls. By Lemma 2.3.1, there is a non-intersecting optimal solution. Fix one such solution. Let S be the set of centers of balls in this solution. Without loss of generality, assume that $\tilde{T}_1$ is a subtree with the minimum number of centers from S. Let S_1 be the set of centers from S in $\tilde{T}_1$ and let u_1 be the root of $\tilde{T}_1$. Let $\mathcal{B}_1^*$ be the balls of the optimal solution, which have a center in S_1. The ball that covers u_1 is either inside or outside $\mathcal{B}_1^*$. By a case by case analysis, we prove that the cost of optimal solution is at least p.

Case 1: Assume the ball that covers u_1 is inside $\mathcal{B}_1^*$. Thus, the balls in $\mathcal{B}_1^*$ cover all the vertices in $\tilde{T}_1$. The reason is that the optimal solution is non-intersecting and no ball with a center outside of S_1 can cover a vertex in $\tilde{T}_1$ without covering u_1 and intersecting some other ball. Since the number of balls in $\mathcal{B}_1^*$ is at most $l(l+1)^{p-2}$ by the choice of $\tilde{T}_1$ and the induction hypothesis, the cost of these balls is at least $p-1$. If their cost is more than $p-1$, the proof of this case is complete. If the cost of these balls is exactly $p-1$, the radius of the largest ball in $\mathcal{B}'$ is at most $p-1$. Therefore, these balls can cover the vertices with distance at most $p-1$ from u_1 and consequently, distance at most $p-2$ from u. Therefore, they do not cover at least $l(l+1)^{p-1}$ vertices, which are located at the last two levels of $\tilde{T}_2, \dots, \tilde{T}_{l+1}$. As the size of $\mathcal{B}_1^*$ is at least one, all these $l(l+1)^{p-1}$ vertices are not covered by zero balls and there is at least one non-zero ball covering them, which shows the cost of optimal solution is at least p.

Case 2: Assume the ball covering u_1, say $B(x,r)$, is outside $\mathcal{B}_1^*$. Thus, x has distance at least 1 from u_1. Let d be this distance. One can easily verify that the balls in $\mathcal{B}_1^*$ and the ball $B(u_1, r-d)$ cover the vertices in $\tilde{T}_1$. We consider two subcases.

(a) Assume that $x = u$. In this case, the size of $\mathcal{B}^*$ is at most $l(l+1)^{p-2} - 1$. Therefore, the cost of the balls $B(u_1, r-d) \cup \mathcal{B}_1^*$ is at least $p-1$ by induction hypothesis. Since $B(x,r) \cup \mathcal{B}_1^*$ are the balls of the optimal solution with total radii at least $d + p - 1$, the cost of optimum is at least p.

(b) Assume that $x \neq u$. Then, we must have $d \geq 2$ and $|\mathcal{B}_1^*|$ is at most $l(l+1)^{p-2}$. Therefore, the cost of the balls $B(u_1, r-d) \cup \mathcal{B}_1^*$ is at least $p-2$. The reason is that the optimal cover for $\tilde{T}_1$ with at most $l(l+1)^{p-2}$ balls has cost at least $p-1$ and increasing the number of balls by one improves the optimal cost for $\tilde{T}_1$ by at most one by Remark 2.3.1. Since $B(x,r) \cup \mathcal{B}_1^*$ are the balls of an optimum with total radii at least $d+p-2$, the cost of optimum is at least p. This completes the proof.

In the fractional solution, we put a ball of radius 0 to the fraction $\frac{k-1}{n_i-1}$ on each vertex where n_i is the number of vertices in T_i. In addition, we put a ball of radius i to the fraction $\frac{n_i-k}{n_i-1}$ on the root vertex. It is easy to check that this is a feasible fractional solution. Therefore, the cost of optimal fractional solution is at most $i\frac{n_i-k}{n_i-1}$ and the integrality gap is at least

$$\frac{n_i-1}{n_i-k} = \frac{\frac{(l+1)^i-1}{l} + l(l+1)^{i-1} - 1}{\frac{(l+1)^i-1}{l}} = \frac{(l+1)^i - 1 + l^2(l+1)^{i-1} - l}{(l+1)^i - 1} > l,$$

for any $i \geq 2$. Thus, the integrality gap is at least $l = \Omega(n_i^{1/i})$.

We can strengthen the LP relaxation by adding this constraint:

$$\sum_{i \in V} y_i^{(0)} = k_0, \tag{2.4}$$

where k_0 is the minimum number of zero balls used in an optimum solution. We can determine k_0 by guessing (*i.e.*, having $k+1$ LPs for $0 \leq k_0 \leq k$). This constraint has the following intuition: in all the bad examples, the fractional solution exploits the zero balls to pay less on non-zero balls. For example, consider the first bad example, which was a complete graph with $k = n-1$. In this instance, the minimum number of zero balls used in an optimum solution is 0 and as a result, the fractional solution cannot use the zero balls to cheat any more and the integrality gap decreases to 1 for this instance. Our experimental results show that for the unweighted graphs with at most 20 vertices the integrality gap is less than 2. We believe that the integrality gap for all instances should not be more than 2, but we could not find a rounding algorithm for this LP.

The unweighted MSR without singletons

As the main difficulty in the above examples was the existence of zero balls, one may wonder what an LP-based approach can achieve in absence of zero balls. It is interesting that the LP based approach cannot beat even ratio $3/2$ that we achieved with the simple algorithm of Theorem 2.2.2.

Consider a special case of the natural LP relaxation for the MSR problem used by Charikar and

Panigrahy [CP04] that no variable is allocated to zero balls:

$$\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{u\in V}\sum_{r=1}^{n-1} r.y_u^{(r)} & \\
\text{subject to} & \displaystyle\sum_{u\in V}\sum_{r:d(u,v)\le r} y_u^{(r)} = 1 & \forall v \in V \\
& \displaystyle\sum_{u\in V}\sum_{r=1}^{n-1} y_u^{(r)} = k & \\
& y_u^{(r)} \ge 0, & \forall u \in V, \forall r \in \{1,\dots,n-1\}
\end{array}$$

where $y_u^{(r)} = 1$ indicates a cluster of radius r is centered on vertex u. The graph of an instance that has a integrality gap of $\frac{3}{2}$ is shown in Figure 2.6. This graph has $2l + 4$ vertices. The part in the oval is a complete graph with $2l$ vertices missing a perfect matching. In this instance, the parameter k is 2. It is not hard to argue that optimal value is 3. The fractional optimal value is at most $2 + \frac{1}{l}$, because we have the following fractional solution: $2l$ balls of radius 1, one on each vertex in the oval with fraction $\frac{1}{2l}$, a ball of radius 3 on x with fraction $1/2l$, a ball of radius 1 on y with fraction $(2l-1)/2l$. The summation of y-values is clearly equal to 2. Each vertex in the oval is covered by $2l-1$ vertices in the oval to the fraction $\frac{1}{2l}$ and by the ball of radius 3 to the fraction $\frac{1}{2l}$. Thus, its total coverage is 1. The vertices x, y, z_1, and z_2 are covered to the fraction $\frac{1}{2l}$ by the ball of radius 3 on x and to the fraction $\frac{2l-1}{2l}$ by the ball of radius 1 on y, which makes their total coverage equal to 1. Therefore, this is a feasible fractional solution.

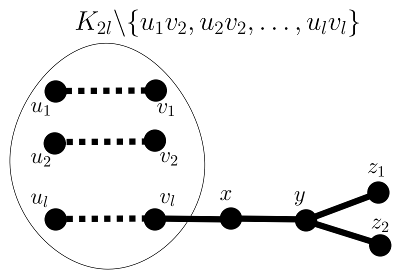


Figure 2.6: The graph of instance with integrality gap essentially $3/2$. In this figure, solid circles show vertices, solid lines show the edges and dashed lines show that there is not an edge between a pair of vertices.

One may hope to get around this integrality gap with a Lagrangian relaxation technique similar to Charikar and Panigrahy [CP04]. The following LP is the Lagrangian relaxation of above LP where the Lagrange multiplier λ can be considered as the cost of creating a cluster. Notice that the number of clusters is not fixed here.

$$\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{u\in V}\sum_{r=1}^{n-1} (r+\lambda).y_u^{(r)} & \\
\text{subject to} & \displaystyle\sum_{u\in V}\sum_{r:d(u,v)\le r} y_u^{(r)} \ge 1 & \forall v \in V \\
& y_u^{(r)} \ge 0, & \forall u \in V, \forall r \in \{1,\dots,n-1\}
\end{array}$$

The first step in this technique is to find an α-approximation algorithm for the new LP. Then, by using this algorithm, we may be able to get a β-approximation algorithm for the original problem where $\beta \geq \alpha$. Unfortunately, the gap of the Lagrangian relaxation LP is at least essentially $3/2$ as we show shortly, and hence, we cannot get a better approximation ratio by a straightforward application of this technique. Consider the instance consisting of a complete graph with $2l$ vertices missing a perfect matching and $\lambda = 1$. It is easy to see that one cluster of radius 2 is an optimal solution for this instance. In the fractional solution, we can pick one clusters of radius 1 around each vertex to the fraction of $1/(2l-1)$. It is easy to see this is a feasible solution having cost $\frac{2l}{2l-1}(1+1)$, which gives an integrality gap of at least essentially $3/2$.

Chapter 3

The Unsplittable Capacitated Facility Location Problem

In this chapter, we consider the Unsplittable Capacitated Facility Location Problem (UCFLP). The formal definition of the problem is as follows. Let G be a bipartite graph with bipartition (F, C), where F is a set of facilities and C is a set clients. Let f_i be the cost of opening facility i, u be the maximum demand that can be served by a facility (*i.e.*, capacity of each facility), d_j be the demand of client j, and c_{ij} be the cost of serving one unit of demand of client j by (opened) facility i. All these input parameters are non-negative rational numbers. We assume that c_{ij} satisfies the triangle inequality. The UCFLP is to find a subset I of facilities to open and a function $\phi : C \rightarrow I$ assigning each client to one open facility (to serve the entire demand of that client) in such a way that the sum of demands assigned to each facility is no more than u while the total cost of opening facilities (facility cost or opening cost), and serving demands (service or connection cost) is minimized, i.e. minimizing $\sum_{i \in I} f_i + \sum_{j \in C} d_j c_{\phi(j)j}$.

There are some variants of the UCFLP that we discuss in this chapter. When we do not have any capacity constraints , *i.e.* $u = \infty$, and we can split the demand of a client the problem is called the *Uncapacitated* Facility Location Problem (UFLP). When we can split the demand of a client among multiple facilities, the problem is called the *splittable* Capacitated Facility Location Problem (splittable CFLP). In the splittable CFLP, when each facility can be opened multiple times (*i.e.*, we must pay the opening cost for each time that we open them), we have the so-called *soft* Capacitated Facility Location Problem (soft CFLP). Sometimes, to contrast the version with soft capacities with the version that we cannot open a facility multiple times, the term *hard* capacities are used in the literature for the latter case. We adopt the same terminology and whenever we say hard capacities we mean the version that one cannot open a facility multiple times. Each of these relaxations (e.g. allowing splitting the demands of clients and/or having multiple copies of each facility) makes the problem significantly easier as discussed below.

All of the above capacitated problems can be defined in the more general setting in which each facility i has a given capacity u_i. When we are discussing a problem in this more general version, we call it the *non-uniform* version. The non-uniform version turns out to be more difficult than the uniform case, particularly in the UCFLP. For this reason, most previous works have focused on the uniform capacity case of the UCFLP and our work does the same.

In the following discussions, for a solution (I, ϕ), where I is the set of open facilities and $\phi : C \to I$ is the assignment of clients to open facilities, we use $c_f(\phi)$ to denote the facility opening cost and $c_s(\phi)$ to denote the service cost, and $c(\phi)$ to denote the total cost of the solution. Thus, we have $c(\phi) = c_f(\phi) + c_s(\phi)$. In the splittable versions, the assignment function is $\phi : C \times F \to \mathbb{Q}_{\geq 0}$, where $\phi(i, j)$ shows the amount of demand of client j served by facility i.

3.1 Overview of Related Works

In this section, we briefly review the related works to see how the new results fit in.

3.1.1 The Uncapacitated Facility Location Problem

Perhaps, the UFLP is the most well studied facility location type problem. In this problem, we only need to decide which facilities to open. Then, the assignment of clients to facilities becomes trivial: we should assign each client to its nearest open facility, because there is no capacity constraint. This fact makes the problem much easier.

The first constant factor approximation for the UFLP was a 3.16-approximation algorithm by Shmoys, Tardos, and Aardal [STA97]. This algorithm was based on rounding an LP relaxation of the problem. They used the following natural LP relaxation of this problem:

$$\begin{array}{lll}
\text{minimize} & \sum_{i \in F} f_i y_i + \sum_{i \in F} \sum_{j \in C} d_j c_{ij} x_{ij} & \\
\text{subject to} & \sum_{i} x_{ij} \geq 1 & \forall j \in C \\
& x_{ij} \leq y_i & \forall i \in F, j \in C \\
& x_{ij} \geq 0, y_i \geq 0 & \forall i \in F, j \in C,
\end{array}$$

where $x_{ij} = 1$ in the integral program indicates client j is assigned to facility i and $y_i = 1$ indicates facility i is open. It should be noted that most of the succeeding improvements for this problem were using the value of this natural relaxation as their lower bound of the optimum.

To round a fractional solution for this LP, they use a filtering method due to Lin and Vitter [LV92]. Using this filtering, they show that with a small increase in cost, one can change this solution to a new feasible fractional solution such that each client is fractionally assigned only to facilities that are relatively close to it. Then, utilizing the metric property, they show how one can open some of the fractionally open facilities and assign each client to its closest facility with an additional small increase in cost. Finally, by randomly choosing the parameter for the filtering stage, they get the 3.16 approximation ratio.

There is a long series of works that improve this constant approximation ratio for the UFLP. Guha and Khuller [GK98] improved the ratio to 2.41 by combining a simple greedy heuristic with the algorithm of [STA97]. This greedy heuristic adds unused facilities one by one greedily based on some measure of effectiveness for each facility. Later, the factor improved to $1 + 2/e \approx 1.74$ by Chudak [Chu98] using some generalized techniques of [STA97] for the LP rounding. A key element to this improvement is the use of randomized rounding of y_i-values in conjunction with the approach of Shmoys, Tardos, and Aardal. Meanwhile, Jain and Vazirani [JV99] gave a 3-approximation primal-dual algorithm with a better running time and Korupolu *et al.* [KPR98] gave a surprisingly simple local search algorithm with factor $5 + \epsilon$ for any $\epsilon > 0$.

Charikar and Guha [CG99] slightly improved the ratio to 1.73 by combining primal dual algorithm of [JV99] with cost scaling and greedy augmentation. The scaling technique exploits the difference between approximation guarantees for the facility cost and the service cost. This can be used by producing a new instance where the facility costs are multiplied by δ, then apply the algorithm to the scaled instance, and then scale back to get a solution for the original instance. Mahdian *et al.* [JMS02] used a variant of the primal-dual method, called dual fitting, and a new analysis technique, called factor revealing LP, to bring down the factor to 1.61. Later, the authors of [MYZ02] combined this algorithm with greedy augmentation of [CG99] to decrease the factor to 1.52. Afterwards, Byrka [Byr07] combined this new algorithm of [MYZ02] with the algorithm of Chudak [Chu98] to get a 1.5-approximation. Finally, Li [Li11] showed that by choosing a parameter of Byrka's algorithm from some specific distribution, one can get a 1.488 factor approximation algorithm, which is the current best known factor.

On the negative side, a result of Guha and Khuller [GK98], combined with an observation of Sviridenko (personal communication cited in [CW99]), implies 1.463-hardness for the UFLP. As a result, unless **P**$=$ **NP**, there is very little room to improve the best known approximation algorithm for the UFLP.

Unsplittable Version of the UFLP: If we add the constraint that the demand of a client cannot be split, it does not make the problem more difficult. First, observe that the optimum value for the splittable version is a lower bound for unsplittable version, because any feasible solution for the unsplittable version is a feasible solution for the splittable version. Second, we can change any solution for the splittable version to a solution for the unsplittable version without increasing cost: assign each client to its closest open facility. As a result, the optimum value of these two versions is the same. Thus, any α-approximation for the standard splittable version gives an α-approximation for the unsplittable version and vice versa.

3.1.2 The Soft Capacitated Facility Location Problem

In the soft CFLP, despite having capacities, the fact that we can open a facility multiple times makes the problem easier in comparison to the case of hard capacities. Intuitively, if you violate the capacity of a facility, we have the option to open enough copies of it to serve the demands assigned to it. This option just increases the facility cost and this increase in cost is not much more than the cost we paid to open the facility in the first place.

Shmoys, Tardos, and Aardal [STA97] designed the first constant factor, a 5.69 factor, approximation algorithm for this problem by a similar LP rounding technique they used for the first constant factor approximation for the UFLP. Then, Chudak and Shmoys [CS99] used the same LP and the randomized LP-rounding technique to get a 3-approximation algorithm for this problem. For non-uniform capacities, Jain and Vazirani [JV99] reduced this problem to the UFLP, and by solving the UFLP, they obtained a 4-approximation algorithm. Arya *et al.* [AGK$^+$01] proposed a simple local search algorithm with an approximation ratio of 3.72 for the non-uniform version. Following the reduction of Jain and Vazirani [JV99] to the UFLP, Jain *et al.* [JMS02] showed that the soft CFLP with non-uniform capacities can be solved within a factor 3 of optimum. This result was improved to a 2.89-approximation algorithm for the non-uniform soft CFLP in [MYZ02]. Finally, Mahdian *et al.* [MYZ03] improved this factor to 2, achieving the integrality gap of the natural LP relaxation of the problem. To the best of our knowledge, this is the current best ratio for this problem.

The main idea of Mahdian *et al.*'s [MYZ03] algorithm is to consider the approximation factor for the facility and connection costs separately. This idea helped them to analyse the performance of the algorithm more carefully and get the current best ratio. At first, they map any instance, I, of the non-uniform soft CFLP to an instance, $R(I)$, of a problem called the linear cost facility location problem, which is equivalent to the UFLP. In this process, they at most double the facility cost of any solution for I and they can translate any solution for $R(I)$ to a solution of at most the same value for I. Then, they solve the new instance $R(I)$ by a UFLP algorithm described in [JMS02]. This algorithm returns a solution that only pays at most double of the connection cost in an optimum and pays at most the same as the optimum for the facility cost. Thus, overall, they get a solution for $R(I)$ that pays at most double of facility cost and double of connection cost. They translate back this solution to a solution for I. As a result, they have a solution with cost within factor 2 of optimum.

We should point out that all of the above algorithms except Arya *et al.*'s local search algorithm [AGK$^+$01] use the optimal value of a natural LP relaxation of the soft CFLP as a lower bound in their analysis. Therefore, they cannot obtain a better ratio than the integrality gap of this relaxation. Mahdian *et al.* [MYZ03] also showed that the integrability gap of this LP is 2 and hence, their analysis is tight. To beat this ratio, one needs to use the optimum value of a stronger LP as the lower bound or need to utilize a technique which is not LP-based.

Unsplittable Version: If we add the constraint that the demand of a client cannot be split, it does not make the problem much more difficult than its splittable counterpart in the soft setting. One can show that any α-approximation algorithm for the unsplittable version of the soft CFLP yields a 2α-approximation algorithm for the splittable version and vice versa (see Proposition 3.6.1 in Section 3.6). As a result, Mahdian *et al.*'s 2-approximation algorithm yields a 4-approximation for unsplittable version of the soft CFLP. In fact, it is not difficult to observe that this algorithm is a 2-approximation for this version. Recall that their algorithm assigns each client by utilizing a UFLP algorithm and hence, to a *single* facility, and its cost is within factor 2 of the optimum value of soft CFLP. Clearly, the optimum value of the unsplittable version is not less than the optimum value of the splittable version, because all feasible solutions for the unsplittable version are feasible solutions with the same cost for the splittable version, too. Thus, their algorithm gives a feasible solution for the unsplittable version of the soft CFLP which is within factor 2 of the optimum value for the unsplittable version.

3.1.3 The Splittable Capacitated Facility Location Problem

The splittable CFLP has also received a lot of attention. In contrast to the UFLP and soft CFLP, there is an important distinction between the splittable and unsplittable case in the presence of hard capacities, because in the unsplittable case, even checking whether there exists a feasible solution becomes **NP**-hard and we can only hope for a bicriteria algorithm (see discussions in Section 3.1.4). In contrast, in the splittale case, if we decide on the set of open facilities, the best way of serving the clients can be determined by building a flow network and using a minimum cost flow algorithm.

For the splittable case, Korupolu, Plaxton, and Rajaraman [KPR98] gave a simple factor $8 + \epsilon$ local search algorithm. This was the first constant factor approximation algorithm for the hard capacitated facility location problem. In this algorithm, they start with an arbitrary set of open facilities which have enough capacity to serve all the demands. Then, they assign the clients optimally to these open facilities. They have three local operations to change the set of open facilities and potentially improve the solution cost: opening a close facility, closing an open facility, and swapping a facility in the set with a facility outside of the set of open facilities. If any of these operations improves the solution more than some small value dependent on the value of ϵ, they change the set of open facilities and proceed to the next iteration. Otherwise, they stop and return the optimal assignment corresponding to current open facilities. Although, the algorithm itself is easy to describe, the analysis was involved.

Later, Chudak and Williamson [CW99] simplified their analysis and showed the actual ratio of Korupolu *et al.*'s algorithm is at most $6+\epsilon$. Pal, Tardos, and Wexler [PTW01] gave a more powerful local search with factor $8.54 + \epsilon$ for the case of non-unifrom capacities. Later, with a series of more powerful local search algorithms, the ratio for the case of non-uniform capacities decreased to $7.46 + \epsilon$ [MP03], and $5.83 + \epsilon$ [ZCY04]. Later, Aggarwal *et al.* [AAB$^+$10] showed that Korupolu *et al.*'s algorithm is actually a 3-approximation for the uniform splittable CFLP and this ratio is tight. Recently, Bansal *et al.* [BGG12] showed that slightly more powerful versions of Zhang *et al.*'s [ZCY04] local search operations give a 5-approximation for the non-uniform splittable CFLP and this ratio is tight.

It should be noted that in contrast to the UFLP and soft CFLP, all the known LP relaxations for this problem have super-constant integrality gap in the general case. The only LP-based result is a 5-approximation algorithm by Levi *et al.* [LSS04] for the non-uniform version in the special case that all facility opening costs are equal.

3.1.4 The Unsplittable Capacitated Facility Location Problem

The UCFLP has a major difference from the problems we have already mentioned. In this problem, it is **NP**-hard to even determine whether a feasible solution exits. In other words, even if we open all the facilities and ignore the cost of the solutions, it is **NP**-hard to find an assignment of demands to the facilities without violation of capacities, which comes from a reduction to the partition problem. In fact, we can show that it is **NP**-hard to even find a solution that does not violate the capacity of less than $\lfloor \frac{n}{2} \rfloor$ facilities (see Proposition 3.6.2 of Section 3.6).

Recall that an (α, β)-approximation algorithm for the UCFLP returns a solution whose cost is within factor α of the optimum and violates the capacity constraints within factor β. Here, we compare the value of an approximate solution with the value of an optimal solution that does *not* violate any capacities. In addition, it should be noted that if we violate the capacity of a facility within factor β, we must pay β times its opening cost. In the context of approximation algorithms, Shmoys, Tardos, and Aardal [STA97] were the first to consider this problem and presented a $(9, 4)$-approximation algorithm. First, they designed a $(9, 3)$-approximation algorithm for the splittable CFLP through the same ideas they used for the first constant approximation algorithm for UFLP, *i.e.*, by filtering and rounding technique. Then, they used a rounding for the generalized assignment problem (GAP) [ST93], which is a generalized version of the matching problem (we define it shortly), to obtain a $(9, 4)$-approximation algorithm for the UCFLP. In addition, by a randomized version of the filtering technique, they got a new algorithm with the ratio $(7.62, 4.29)$.

One can show that by the same technique as the one used by Shmoys, Tardos, and Aardal, any (α, β)-approximation algorithm for the splittable CFLP can be converted to a $(2\alpha, \beta + 1)$-approximation algorithm for the UCFLP. This technique of reducing the unsplittable version using the rounding for the GAP to the splittable version was a cornerstone of the subsequent approximation algorithms. This reduction works as follows.

The GAP is a generalization of the matching problem that can be described as a scheduling problem which has some similarities to the UCFLP. In the GAP, we have a collection of jobs J and a set M of machines. Each job must be assigned to exactly one machine in M. If job $j \in J$ is assigned to machine $i \in M$, then it requires p_{ij} units of processing time and incurs a cost r_{ij}. Each machine $i \in M$ can be assigned jobs of total processing time at most P_i. We want to find an assignment of jobs to machines to minimize the total assignment cost. We should point out that r_{ij} values do not necessarily satisfy the triangle inequality. Shmoys and Tardos [ST93] considered an LP relaxation and showed that a feasible solution of this LP can be rounded, in polynomial time, to an integral solution with the same cost that violates processing time limit P_i within additive factor

$\max_{j \in J} p_{ij}$.

We can model (view) the UCFLP as an instance of the GAP in the following way: jobs are clients, machines are facilities, $p_{ij} = d_j$ for all i, $r_{ij} = d_j c_{ij}$ for all i and j, and $P_i = u$, and all facilities are already open (machines are available). Therefore, if we have a fractional assignment of clients to facilities (i.e. a splittable assignment) which pays α times optimum and violates the capacities within factor β, then using the rounding algorithm of [ST93], we can round this fractional assignment to an integral assignment (*i.e.*, one that assigns each client to one facility) without increasing the connection cost such that the capacity constraints are violated by at most an additive factor of $\max d_j \leq u$. Here, we have assumed that the demand values are at most u, because otherwise; the instance is infeasible and we can rule out these instances in a preprocessing step. As a result, in the unsplittable solution, the capacity of each facility is violated within $\beta + 1$ factor, which at most doubles the facility cost. Consequently, we have a $(2\alpha, \beta + 1)$-approximation algorithm for the UCFLP. Note that when we have separate bounds on the connection cost and facility cost, we may get a better factor than 2α. The reader may verify that if the algorithm for the splittable CFLP fractionally assigns a client only to a facility with capacity not less than its demand, the above approach works for the non-unifrom version as well.

Korupolu, Plaxton, and Rajaraman [KPR98] gave the first constant factor approximation algorithm for the splittable hard capacitated version, and applied the GAP rounding technique of [STA97] to get a $(O(1), 2)$-approximation algorithm for the UCFLP. All subsequent constant factor approximation algorithms for the uniform splittable CFLP combined with the above GAP reduction technique give a $(O(1), 2)$-approximation algorithm for the UCFLP, even though this may not be explicitly stated by their authors. Applying the current best approximation algorithms for the splittable CFLP [AAB$^+$10], one can get a $(5, 2)$-approximation algorithm for the UCFLP. In addition, since the current best approximation algorithm for the non-uniform splittable CFLP does not assign a client to a facility with capacity less than its demand, one can get a $(9, 2)$-approximation algorithm for the UCFLP with non-uniform capacities (see Proposition 3.6.3 of Section 3.6).

Recently, Bateni and Hajiaghayi [BH09] modeled an assignment problem in content distribution networks by the UCFLP. This assignment problem has been first considered by Alzoubi *et al.* [ALR$^+$08] and is basically the assignment of downloadable objects, such as media files or softwares, to some servers. We cannot split a downloadable object and we need to store it in a single server. As Alzoubi *et al.* mention, the server capacities is very crucial in practice and a high overload amount on a server can disrupt a large number of connections. Motivated by this strict requirement on capacities, the authors of [BH09] designed a $(1+\epsilon, 1+\epsilon)$-approximation algorithm for *tree metrics* (for any constant $\epsilon > 0$) using a dynamic programming approach. They also presented a quasi-polynomial time $(1 + \epsilon, 1 + \epsilon)$-approximation algorithm (again for trees) for the non-uniform version. As we explain shortly, these results can be used to get a polynomial time $(O(\log n), 1 + \epsilon)$-approximation algorithm for the uniform capacities and a quasi-polynomial time $(O(\log n), 1 + \epsilon)$-approximation algorithm for the non-uniform version for any constant $\epsilon > 0$.

Fakcharoenphol *et al.* [FRT03] improved the well known machinery of Bartal [Bar98] and showed that any n point metric space can be embedded into a distribution over tree metrics such that the expected stretch of any edge is $O(\log n)$ and we can sample from this distribution efficiently. Therefore, to solve the UCFLP in general metrics, Bateni and Hajiaghayi first embed the general metric into the above distribution of tree metrics and sample a tree metric from it. By Fakcharoenphol *et al.*'s result, the cost of an optimal solution for the general metric increases by an expected factor of $O(\log n)$ on this tree metric. Then, they run their approximation algorithms for trees on this tree metric and obtain a solution within factor $(1 + \epsilon)$ of the optimum of this tree, which is within expected factor $O(\log n)$ of the optimum of the general metric.

Non-metric version: It should be noted that here we assumed the cost function, c, satisfies triangle inequality. For non-metric cost functions, unless **P**= **NP**, the UCFLP does not admit any $(f(x), \frac{3}{2} - \epsilon)$-approximation algorithm for any polynomial time computable function $f(x)$ and any constant $\epsilon > 0$, where x is the size of input [BH09]. This result comes from a simple reduction from a special case of the makespan problem that $p_{ij} \in \{p_j, \infty\}$, where p_{ij} is the processing time of job j on machine i . This special case is $\frac{3}{2}$-hard [LST90]. In addition, this problem does not admit any $(c \log n, \beta)$-approximation algorithm for some constant $c > 0$ and any $\beta \geq 1$ unless **P**= **NP** (see Proposition 3.6.4 in Section 3.6).

3.2 New Results

Recall that given an instance (F, C) of the UCFLP with opening costs f_i, demands d_j, and connection costs c_{ij}, a solution is a subset I of facilities to open along with assignment function $\phi : C \to I$. Since all capacities are uniform, by a simple scaling, we can assume that all of them are 1 and all the client demands are at most 1.

All the known constant factor algorithms for the UCFLP violate the capacity constraints by a factor of at least 2 which is mainly due to using the rounding algorithm for GAP [ST93]; and the algorithm of [BH09] (although has $1 + \epsilon$ violation) is not a constant factor approximation. We are interested in $(O(1), \beta)$-approximation algorithms for some $\beta < 2$. We define a restricted version of the problem and show that finding a good approximation algorithm for this restricted version would imply a good approximation for the general version. The definition of similar restricted versions has been a common practice in solving bin packing type problems (*e.g.*, see [dlVL81, KK82]).

Definition 3.2.1 *An ϵ-restricted UCFLP instance, denoted by RUCFLP(ϵ), is an instance of the UCFLP in which each demand has size more than ϵ, i.e., $\epsilon < d_j \leq 1$ for all $j \in C$.*

The following theorem establishes the reduction from the general instances of the UCFLP to the restricted version. Here, the general idea is that if we assign the large clients oblivious to small clients, we can fractionally assign the small clients without paying too much. We use the maximum-flow minimum-cut theorem to show this. Then we can round this fractional assignment of small clients with the GAP rounding technique [ST93].

Theorem 3.2.1 *If $\mathcal{A}$ is an $(\alpha(\epsilon), \beta(\epsilon))$-approximation algorithm for the RUCFLP(ϵ) with running time $\tau(\mathcal{A})$ then there is an algorithm $\mathcal{A}_C$ which is a $(g(\epsilon, \alpha(\epsilon)), \max\{\beta(\epsilon), 1+\epsilon\})$-approximation algorithm for the UCFLP whose running time is polynomial in $\tau(\mathcal{A})$ and the instance size, where $g(\epsilon, \alpha(\epsilon)) \leq c(1+\epsilon)\alpha(\epsilon)$ for some constant c.*

Corollary 3.2.1 *For any constant $\epsilon > 0$, an $(\alpha(\epsilon), 1+\epsilon)$-approximation algorithm for the RUCFLP(ϵ) yields an $(O(\alpha(\epsilon)), 1+\epsilon)$-approximation for the UCFLP. Particularly, when $\alpha(\epsilon)$ is a constant, we have a constant approximation for the UCFLP with a $(1+\epsilon)$ violation of capacities in polynomial time.*

This reduction shows that to get a $(O(1), 1+\epsilon)$-approximation, it is sufficient to consider large clients only, which may open the possibility of designing algorithms using some of the techniques used in the bin packing type problems. If one finds such an algorithm for large clients, the above corollary shows that we have an $(O(1), (1+\epsilon))$-approximation for the UCFLP. As an evidence for this, we find approximation algorithms for the RUCFLP($\frac{1}{2}$) and the RUCFLP($\frac{1}{3}$). For the RUCFLP($\frac{1}{2}$), we present an exact algorithm and for the RUCFLP($\frac{1}{3}$), we present a $(21, 1)$-approximation algorithm. These, together with Theorem 3.2.1 imply:

Theorem 3.2.2 *There is a polynomial time $(9, \frac{3}{2})$-approximation algorithm for the UCFLP.*

Theorem 3.2.3 *There is a polynomial time $(29.315, \frac{4}{3})$-approximation algorithm for the UCFLP.*

Finally, we give a QPTAS for Euclidean metrics. Here, we employ a dynamic programming technique and combine the shifted quad-tree dissection of Arora [Aro96], some ideas from [BH09], and some new ideas to design a dynamic programming.

Theorem 3.2.4 *There exists a $(1+\epsilon, 1+\epsilon)$-approximation algorithm for the Euclidean UCFLP in $\mathbb{Q}^2$ with quasi-polynomial running time for any constant $\epsilon > 0$.*

Although this theorem is presented for $\mathbb{Q}^2$, it can be generalized to $\mathbb{Q}^d$ for any constant $d > 2$.

3.3 Reduction to the Restricted UCFLP

In this section, we prove Theorem 3.2.1. Let $L = \{j \in C : d_j > \epsilon\}$ be the set of large clients and $S = C \backslash L$ be the set of small clients[1]. We call two assignments $\phi_1 : C_1 \to F_1$ and $\phi_2 : C_2 \to F_2$ consistent if $\phi_1(j) = \phi_2(j)$ for all $j \in C_1 \cap C_2$. The high level idea of the algorithm is as follows.

[1]We should point out that the definitions of L and S are with respect to a given parameter ϵ. Since throughout the following sections, this parameter is the same for all statements, in the interest of brevity, we use this notation instead of $L(\epsilon)$ and $S(\epsilon)$.

We first ignore the small clients and solve the problem restricted to only the large clients by running algorithm $\mathcal{A}$ of Theorem 3.2.1. We can show that given a good assignment of large clients, there exists a good assignment of all the clients (large and small) which is consistent with this assignment of large clients, i.e. a solution which assigns the large clients the same way that $\mathcal{A}$ does, whose cost is not far from the optimum cost. More specifically, we show there is a *fractional* (i.e. splittable) assignment of small clients that together with the assignment of large clients obtained from $\mathcal{A}$ gives an approximately good solution. Having this property, we try to find a fractional assignment of small clients. To do this, we update the capacities and the opening costs of facilities with respect to the assignment of large clients (according to the solution of $\mathcal{A}$). Then, we fractionally assign small clients and round this fractional assignment at the cost of violating the capacities with additive factor ϵ. Algorithm 3.3.1 shows the details. Here, $\phi^{-1}(i)$ is the set of clients assigned to facility i by the assignment ϕ and for a set $F' \subseteq F$, $\phi^{-1}(F') = \cup_{i \in F'} \phi^{-1}(i)$.

Algorithm 3.3.1 Algorithm for the UCFLP by reduction to the RUCFLP(ϵ)

```
Require: An instance of the UCFLP, a parameter $\epsilon > 0$, and an algorithm $\mathcal{A}$ for the RUCFLP($\epsilon$)
Ensure: A subset $I \subseteq F$ to open and an assignment of clients $\phi : C \to I$
1: Let $L = \{j \in C : d_j > \epsilon\}$ and $S = C \backslash L$. Assign the clients in $L$ by running $\mathcal{A}$. Let $I_L$ be the
   opened facilities and $\phi_L : L \to I_L$ be the assignment found by $\mathcal{A}$.
2: For $i \in I_L$, set $f_i = 0$, and set $u'_i = \max\{0, 1 - \sum_{j \in \phi_L^{-1}(i)} d_j\}$ as the new capacity of facility $i$.
   Assign the clients in $S$ with respect to updated opening costs and capacities by an approximation
   algorithm for the splittable CFLP with non-uniform capacities. Let $I_S$ be the new set of opened
   facilities and $\phi'_S : S \to I'_S$ be the assignment function, where $I'_S \subseteq I_S \cup I_L$.
3: Round the splittable assignment $\phi'_S$ using algorithm of [ST93] to find an unsplittable assignment
   $\phi_S : S \to I'_S$.
4: Let $I = I'_S \cup I_L$ and define $\phi : C \to I$ as $\phi(j) = \phi_S(j)$ if $j \in S$ and $\phi(j) = \phi_L(j)$, otherwise.
   Return $\phi$ and $I$.
```

First, we formally prove the property that given assignment of large clients, there is a feasible *fractional* assignment of small clients with an acceptable cost. Note that we do not open facilities fractionally and a fractional assignment of demands of (small) clients is essentially equivalent to splitting their demands between multiple open facilities. We should point out that the proof of this property is only an existential result and we do not actually find the assignment in the proof. We only use this lemma to bound the cost of our solution. Let OPT be an optimum solution which opens set I^* of facilities and with assignment of clients $\phi^* : C \to I^*$. We use $\phi^*_L : L \to I^*$ and $\phi^*_S : S \to I^*$ to denote the restriction of ϕ^* to large and small clients, respectively.

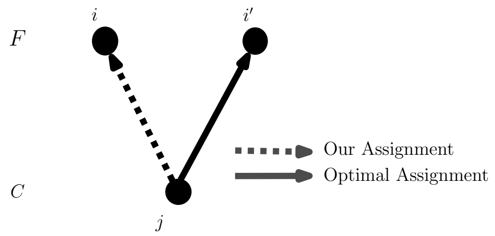
Lemma 3.3.1 *Suppose I_L is a set of open facilities and $\phi_L : L \to I_L$ is an arbitrary (not necessarily capacity respecting) assignment of large clients. Given the assignment ϕ_L, there exists a feasible fractional assignment of small clients, $\phi''_S \to I''_S$ such that $c_s(\phi''_S) \leq c_s(\phi^*) + c_s(\phi_L)$ and $c_f(\phi''_S) \leq c_f(\phi^*)$.*

Proof. Recall u'_i is equal to $\max\{0, 1 - \sum_{j \in \phi_L^{-1}(i)} d_j\}$, i.e., the amount of capacity left for facility i after the assignment of large clients based on ϕ_L. We open all the open facilities in the optimum solution, *i.e.*, all facilities in I^* (if not already open in I_L). Let $I''_S = I_L \cup I^*$. To show the existence of ϕ''_S, first we move the demands of small clients to the facilities in I^* based on ϕ^*_S and we pay $c_s(\phi^*_S)$ for this. So now the demands of small clients are located at facilities in I^*. However, a facility $i \in I''_S$ has only u'_i residual capacity left (after commiting parts of its capacity for the large clients assigned to it by ϕ_L) and this capacity may not be enough to serve the demands of small clients moved to that location. In order to rectify this, we will fractionally redistribute the demands of these small clients between facilities (in I''_S) in such a way that we do not violate capacities u'_i. In this redistribution, we only use the edges used in ϕ_L or ϕ^*_L and if an edge is used to assign large client j to facility i (in ϕ_L or ϕ^*_L), we move at most d_j units of demands of small clients along this edge. Therefore, we pay at most $c_s(\phi_L) + c_s(\phi^*_L)$ in this redistribution. Thus, by the triangle inequality, the connection cost of the fractional assignment of small clients obtained at the end is bounded by $c_s(\phi^*_S) + c_s(\phi^*_L) + c_s(\phi_L) = c_s(\phi^*) + c_s(\phi_L)$. Since we only open facilities in the optimum solution (on top of what is already open in I_L) the extra facility cost (for assignment ϕ''_S) is bounded by the facility cost of the optimum.

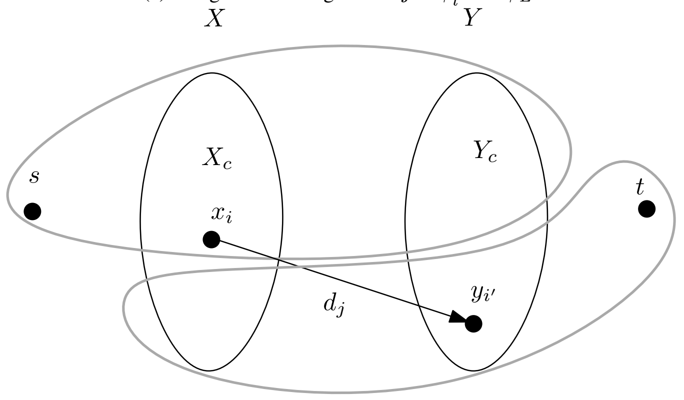
This process of moving the small client demands can be alternatively thought in the following way. We start from the optimum assignment ϕ^* and change the assignment of large clients to get an assignment identical to ϕ_L for those in L. Specifically, we change the assignment of a large client j from $i' = \phi^*(j)$ to $i = \phi_L(j)$. This switch increases the amount of demands served at i by d_j and decreases the amount of demand served at i' by d_j. After doing all these switches, we might have more demand at some facilities than their capacities, while the total demands assigned to some facilities might be less than 1. To resolve this problem, we try to redistribute (fractionally) the demands of small clients so that there is no capacity violation and we use the max-flow min-cut theorem to show that this redistribution is possible.

Let s^*_i be the total demand of small clients served by facility i in ϕ^*. Define a flow network H as follows. H has set of vertices $X \cup Y \cup \{s, t\}$ where $X = \{x_i | i \in F\}$ and $Y = \{y_i | i \in F\}$, and s is called the source and t is the sink. We add an edge from s to all the vertices in X and set the capacity of edge sx_i to s^*_i (this represents the total demand of small clients that can be moved *from* facility i). We connect each $y_i \in Y$ to t and set the capacity of edge $y_i t$ to u'_i (this represents the residual capacity of facility i after the assignment of large clients according to ϕ_L). We connect x_i and y_i bidirectionally with edges of unlimited capacity (because x_i and y_i represent the same facility). Finally, for each large client j with $\phi_L(j) = i$ and $\phi^*(j) = i'$ we add an edge from x_i to $y_{i'}$ with capacity d_j (see Figure 3.1). This means that we can transport d_j units of (small clients) demands from facility i to facility i' since we switch the demand of large client j from being served at i' in the optimum to be served at i.

A flow of value $\sum_i s^*_i$ in H represents a fractional redistribution of demands of small clients between facilities in such a way that we do not violate capacities u'_i. It follows that if there is a flow of value $\sum_i s^*_i$ in H then we can redistribute the demands of small clients among facilities so that



(a) Assignment of large client j in ϕ_l^* and ϕ_L.



(b) The edge $x_i y_{i'}$ corresponding to the large client j shown in part (a). Also, an arbitrary cut of bounded value in H shown with gray ovals. The top oval shows the partition $s \cup X_c \cup Y_c$.

Figure 3.1: The flow network used in the proof of Lemma 3.3.1.

large clients are served according to ϕ_L and no facility capacity is violated because of small clients. We show that a maximum s-t flow in H has value at least $\sum_i s_i^*$.

We show that the capacity of any s-t cut in H is at least $\sum_i s_i^*$. Therefore, the capacity of minimum cut is at least $\sum_i s_i^*$ and by the max-flow min-cut theorem, the value of max flow is at least $\sum_i s_i^*$. If for any $i \in F$, the vertices x_i and y_i are in different partitions of a cut, the cut has unbounded capacity (because of the unlimited capacity of bidirectional edge $x_i y_i$) and clearly has capacity at least $\sum_i s_i^*$.

Now, consider an arbitrary cut in which for all $i \in F$, the vertices x_i and y_i are in the same partition. Consider the partition containing s in this cut and let $X_c \subseteq X$ and $Y_c \subseteq Y$ be the rest of vertices in this partition, *i.e.*, the partition is $s \cup X_c \cup Y_c$. Let F' be the facilities corresponding to the vertices in X_c (or equivalently, in Y_c). Let C_l^* be the large clients assigned to the facilities in F' by ϕ^* and C_l be the large clients assigned to the facilities in F' by ϕ_L. Since ϕ^* does not violate capacities, the total demand of clients assigned to facilities in F' is at most their total capacity, *i.e.* $|F'|$. In other words, we have

$$\sum_{i \in F'} s_i^* + \sum_{j \in C_l^*} d_j \leq |F'| \leq \sum_{i \in F'} u_i' + \sum_{j \in C_l} d_j,$$

where the second inequality follows from the way that we defined u_i' and C_l. Adding the term

$\sum_{i\in F\setminus F'} s_i^* - \sum_{j\in C_l^*} d_j$ to both sides:

$$\sum_{i\in F'} s_i^* + \sum_{i\in F\setminus F'} s_i^* \leq \sum_{i\in F\setminus F'} s_i^* + \sum_{i\in F'} u_i' + \sum_{j\in C_l} d_j - \sum_{j\in C_l^*} d_j,$$

and by the simple fact that $\sum_{j\in C_l} d_j - \sum_{j\in C_l^*} d_j \leq \sum_{j\in C_l\setminus C_l^*} d_j$, the above inequality implies

$$\sum_{i\in F} s_i^* \leq \sum_{i\in F\setminus F'} s_i^* + \sum_{i\in F'} u_i' + \sum_{j\in C_l\setminus C_l^*} d_j.$$

Notice that the right hand side of the above inequality is exactly the capacity of the cut (see Figure 3.1). The first term of the right hand side is the capacity of the edges leaving the cut from s to $X \setminus X_c$. The second term is the capacity of the edges leaving the cut from the vertices in Y_c to t. The third term is the capacity of the edges leaving the cut from X_c to $Y \setminus Y_c$, because a client j is in $C_l\setminus C_l^*$, if and only if it is assigned to a facility in F' by ϕ_L and is assigned to a facility outside of F' by ϕ^*, if and only if we have an edge of capacity d_j from $\phi_L(j) \in X_c$ to $\phi^*(j) \in Y\setminus Y_c$. ■

Remark 3.3.1 *The above lemma can be generalized in the following way. Assume facility i has capacity u_i,* i.e.*, we are in the non-uniform case. Let $\phi_{C'} : C' \to I_{C'}$ and $\phi_C : C \to I_C$ be two arbitrary assignments, where C' can be any subset of C, and $I_{C'}$ and I_C are subsets of F. Let $C'' = C\setminus C'$. Almost the same proof as above shows that given the assignment $\phi_{C'}$, there exists a feasible fractional assignment $\phi_{C''} : C'' \to I_C''$ such that $c_s(\phi_{C''}) \leq c_s(\phi_C) + c_s(\phi_{C'})$ and $c_f(\phi_{C''}) \leq c_f(\phi_C)$, where $I_C'' \subseteq I_C \cup I_{C'}$. If we set $C' = L$ and $\phi_C = \phi^*$, then $C'' = S$ and we get the above lemma.*

We point out that the above lemma is tight. Consider a path $P = v_1v_2v_3$ where the edges have unit cost. Assume $1/2\epsilon$ is an integer and denoted this integer by q. There are q small clients of demand ϵ on v_1, there is a facility with opening cost 0 on v_1, there is a large client with demand $1/2$ on v_2, there is another facility with opening cost 0 on v_3 and there are $2q$ small clients of demand ϵ on v_3. In the optimal solution, we open both facilities and assign the large client on v_2 to the facility on v_1. Thus, we have $c_s(\phi^*) = 1/2$ and $c_f(\phi^*) = 0$. An algorithm which decides the assignment of large clients oblivious to small clients (including Algorithm 3.3.1) may assign the large client on v_2 to the facility on v_3. After this assignment, any feasible fractional assignment of small clients, ϕ_S'', must assign at least $1/2$ units of demands of small clients on v_3 to the facility on v_1. Thus, we have $c_s(\phi_S'') \geq 1/2 + 1/2 = 1 = c_s(\phi^*) + c_s(\phi_L)$, because $c_s(\phi_L) = 1/2$ and $c_f(\phi_S'') = c_f(\phi^*) = 0$. This shows that Lemma 3.3.1 is tight. Note that this example shows that any algorithm which decides the assignment of large clients oblivious to small clients is at least a factor 3 away from the optimum even for simple metrics such as Euclidean, planar, and tree metrics.

One should note that when all facility costs and at least one of $c_s(\phi_L^*)$ or $c_s(\phi_L)$ are non-zero, Lemma 3.3.1 is not tight: remember that I^* is the set of facilities opened by ϕ^*. Assume the facility cost inequality of Lemma 3.3.1 is tight, *i.e.*, for any feasible fractional assignment of small clients

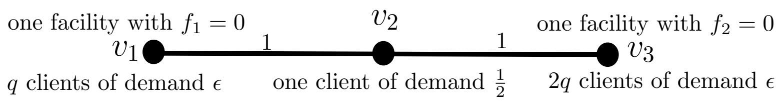

one facility with $f_1 = 0$
v_2
one facility with $f_2 = 0$
v_1
1
1
v_3
q clients of demand ϵ
one client of demand $\frac{1}{2}$
$2q$ clients of demand ϵ

Figure 3.2: A tight example for Lemma 3.3.1.

ϕ''_S, we have $c_f(\phi''_S) = c_f(\phi^*)$. This means that none of the large clients are assigned to I^* by ϕ_L or equivalently, no small client assigned to a facility opened by ϕ_L. Therefore, when we switch the assignment of large clients from the one in the optimal solution to the one assigned by ϕ_L, we do not need to send back anything and change the assignment of small clients. Since at least one of $c_s(\phi^*_L)$ or $c_s(\phi_L)$ is non-zero, we have $c_s(\phi''_S) < c_s(\phi^*_S) + c_s(\phi^*_L) + c_s(\phi_L)$ for a ϕ''_S that assigns small clients the same way as ϕ^*_S.

Now, we prove our main theorem using Lemma 3.3.1:

Proof of Theorem 3.2.1. Since the cost of the optimum solution for the instance consisting of only the large clients is clearly no more than that of the original instance, after Step 1 of Algorithm 1, we have an assignment ϕ_L such that $c(\phi_L) \leq \alpha(\epsilon)c(\phi^*_L)$ and it violates the capacities by a factor of at most $\beta(\epsilon)$. By Lemma 3.3.1, given ϕ_L, there is a feasible fractional assignment ϕ''_S for small clients such that $c_s(\phi''_S) \leq c_s(\phi^*) + c_s(\phi_L)$ and $c_f(\phi''_S) \leq c_f(\phi^*)$.

In Step 2, consider the instance of the splittable CFLP consisting of the small clients and the residual facility opening costs and capacities as defined. We use an approximation algorithm for the splittable CFLP to find an approximate splittable (i.e. fractional) assignment ϕ'_S for small clients. Suppose that the approximation algorithm used for the splittable CFLP has separate factors λ_{ss}, λ_{sf}, λ_{fs}, λ_{ff} such that it returns an assignment with service cost at most $\lambda_{ss}c_s(\tilde{\phi}_S)+\lambda_{sf}c_f(\tilde{\phi}_S)$ and with opening cost $\lambda_{fs}c_s(\tilde{\phi}_S) + \lambda_{ff}c_f(\tilde{\phi}_S)$ for any feasible solution $\tilde{\phi}_S$. Therefore, using Lemma 3.3.1:

$$c_s(\phi'_S) \leq \lambda_{ss}c_s(\phi''_S) + \lambda_{sf}c_f(\phi''_S), \tag{3.1}$$

and

$$c_f(\phi'_S) \leq \lambda_{fs}c_s(\phi''_S) + \lambda_{ff}c_f(\phi''_S). \tag{3.2}$$

The current best approximation for the non-uniform splittable CFLP is due to Bansal *et al.* [BGG12] with parameters $\lambda_{ss} = 1$, $\lambda_{sf} = 1$, $\lambda_{fs} = 4$, and $\lambda_{ff} = 4$.

In Step 3, we round the splittable assignment ϕ'_S using the algorithm of Shmoys and Tardos [ST93] for the Generalized Assignment Problem (GAP) to find an integer assignment ϕ_S (recall the discussion in Section 3.1.4). Given the fractional assignment of clients to facilities ϕ'_S, using the rounding algorithm of [ST93], we can round ϕ'_S to ϕ_S without increasing the connection cost, i.e. $c_s(\phi_S) = c_s(\phi'_S)$, such that the capacity constraints are violated by at most an additive factor of $\max_{j \in S} d_j$. Since all the jobs in S have demand at most ϵ, the capacity constraints are violated by at most a factor of $1 + \epsilon$.

After combining ϕ_S and ϕ_L in Step 4, the violation of capacities is within a factor of at most $\max\{\beta(\epsilon),(1+\epsilon)\}$, because the facilities with violated capacities in Step 1 will be removed in Step 2 and will not be used in Step 3. So it only remains to bound the cost of this assignment:

$$\begin{array}{rcll}
c_s(\phi_S) & = & c_s(\phi'_S) & \text{by rounding of the GAP [ST93]}\\
& \leq & \lambda_{ss}c_s(\phi''_S)+\lambda_{sf}c_f(\phi''_s) & \text{by Equation (3.1)}\\
& \leq & \lambda_{ss}(c_s(\phi^*)+c_s(\phi_L))+\lambda_{sf}c_f(\phi^*), & \text{by Lemma 3.3.1}\\
c_f(\phi_S) & \leq & (1+\epsilon)c_f(\phi'_S) & \text{by rounding of the GAP [ST93]}\\
& \leq & (1+\epsilon)\lambda_{fs}c_s(\phi''_S)+(1+\epsilon)\lambda_{ff}c_f(\phi''_S) & \text{by Equation (3.2)}\\
& \leq & (1+\epsilon)\lambda_{fs}(c_s(\phi^*)+c_s(\phi_L))+(1+\epsilon)\lambda_{ff}c_f(\phi^*). & \text{by Lemma 3.3.1}
\end{array}$$

Therefore:

$$\begin{array}{rcl}
c(\phi) & = & c(\phi_S)+c(\phi_L)\\
& = & c_s(\phi_S)+c_f(\phi_S)+c_s(\phi_L)+c_f(\phi_L)\\
& \leq & h_1(\epsilon)c_s(\phi^*)+h_2(\epsilon)c_f(\phi^*)+(h_1(\epsilon)+1)c_s(\phi_L)+c_f(\phi_L),
\end{array}\tag{3.3}$$

where $h_1(\epsilon)=\lambda_{ss}+(1+\epsilon)\lambda_{fs}$ and $h_2(\epsilon)=\lambda_{sf}+(1+\epsilon)\lambda_{ff}$. Since $h_1(\epsilon)\geq 0$ for any $\epsilon>0$: $(h_1(\epsilon)+1)c_s(\phi_L)+c_f(\phi_L)\leq(h_1(\epsilon)+1)c(\phi_L)\leq\alpha(\epsilon)(h_1(\epsilon)+1)c(\phi^*_L)\leq\alpha(\epsilon)(h_1(\epsilon)+1)c(\phi^*)$. Combining this with Inequality (3.3), we obtain that the cost of ϕ is within factor:

$$g(\epsilon,\alpha(\epsilon))=\max(h_1(\epsilon),h_2(\epsilon))+\alpha(\epsilon)(h_1(\epsilon)+1)\tag{3.4}$$

of the optimum, where $h_1(\epsilon)=h_2(\epsilon)=5+4\epsilon$ using the current best ratio of [BGG12]. ■

3.4 The RUCFLP($\frac{1}{2}$) and RUCFLP($\frac{1}{3}$)

In this section, we give two approximation algorithms for the RUCFLP($\frac{1}{2}$) and RUCFLP($\frac{1}{3}$). Combined with Theorem 3.2.1 (and using Algorithm 1) these imply two approximation algorithms for the UCFLP. We start with the simpler of the two, namely the RUCFLP($\frac{1}{2}$).

Theorem 3.4.1 *There is a polynomial time exact algorithm for the RUCFLP($\frac{1}{2}$).*

Proof. Consider an optimal solution for a given instance of this problem with value OPT$_L$. Because $d_j>\frac{1}{2}$ for all $j\in C$, each facility can serve at most one client in the optimal solution. Therefore, the optimal assignment function, ϕ^*_L, induces a matching $M=\{j\phi^*_L(j):j\in C\}$. Let $w_{ij}=c_{ij}.d_j+f_i$ and let $w(H)=\sum_{e\in H}w_e$ for any subset of edges $H\subseteq E$. It follows that $w(M)=$ OPT$_L$.

Let M^* be a minimum weight perfect matching with respect to weights w_{ij}. Clearly, $w(M^*)\leq w(M)=$ OPT$_L$. In addition, M^* induces a feasible assignment of clients to facilities with cost $w(M^*)$. Thus, M^* induces an optimal solution for the RUCFLP($\frac{1}{2}$). Since we can find a minimum weight perfect matching in polynomial time, there is an exact algorithm for the RUCFLP($\frac{1}{2}$). ■

Corollary 3.4.1 *There is a polynomial time $(15, 3/2)$-approximation algorithm for the UCFLP.*

Proof. We run Algorithm 3.3.1, where we use the algorithm of Theorem 3.4.1 in the first step. Substituting $\alpha(\epsilon) = 1$ and $\epsilon = 1/2$, we have $h_1(\frac{1}{2}) = 7$, $h_2(\frac{1}{2}) = 7$, and $g(\epsilon, \alpha(\epsilon)) = 15$. Since $\beta(\epsilon) = 1$, the overall ratio is $(15, 3/2)$. ∎

Remark 3.4.1 *We can generalize the above lemma. Let ϕ_L be the assignment induced by M^* and $\tilde{\phi_L}$ be any feasible assignment for the RUCFLP($\frac{1}{2}$) instance. The reader may verify that a similar proof shows that $c_s(\phi_L) + c_f(\phi_L) \leq c_s(\tilde{\phi_L}) + c_f(\tilde{\phi_L})$.*

The algorithm for the RUCFLP($\frac{1}{3}$) is more involved. First, we show how finding an approximation algorithm for the RUCFLP(ϵ) with zero facility opening costs leads to an approximation algorithm for the general RUCFLP(ϵ). Then, we give an approximation algorithm for the RUCFLP($\frac{1}{3}$) with zero opening costs.

Lemma 3.4.1 *Given an $(\alpha'(\epsilon), \beta(\epsilon))$-approximation algorithm $\mathcal{A}'$ for the RUCFLP(ϵ) with zero facility opening costs, we can find a $(\alpha'(\epsilon)\frac{1}{\epsilon}, \beta(\epsilon))$-approximation $\mathcal{A}$ for the general RUCFLP(ϵ).*

Proof. Define a new connection cost $c'_{ij} = c_{ij} + f_i$ and opening cost $f'_i = 0$ for all $i \in F$ and $j \in C$. Note that the new cost function is still metric. Then, we run $\mathcal{A}'$ on this new modified instance and let the solution returned by it be assignment ϕ_L. We use ϕ_L to assign the clients for the original instance and we claim this a $(\alpha'(\epsilon)\frac{1}{\epsilon}, \beta(\epsilon))$-approximation. In the following, the costs of all assignments are based on c_{ij} and f_i values.

Let ϕ^*_L be an optimal assignment for the original instance of the RUCFLP(ϵ) and OPT$_L$ be the cost of ϕ^*_L (including opening costs). Let $C_i = {\phi^*_L}^{-1}(i)$. The cost of ϕ^*_L in the new instance will be

$$\sum_i \sum_{j \in C_i} d_j c'_{ij} = \sum_i \left(\sum_{j \in C_i} d_j c_{ij} + (\sum_{j \in C_i} d_j).f_i \right) \leq \sum_i \left(\sum_{j \in C_i} d_j c_{ij} + f_i \right) \leq \text{OPT}_L,$$

where we used the fact that $\sum_{j \in C_i} d_j \leq 1$. Thus, there is a solution in the modified instance with cost at most OPT$_L$. Therefore, the value of the assignment found by $\mathcal{A}'$ in the new graph is at most $\alpha'(\epsilon) \cdot \text{OPT}_L$. Let γ be the portion of this value that comes from the facility costs in c' costs, i.e., the cost of solution ϕ_L in the new graph is $c_s(\phi_L) + \gamma \leq \alpha'(\epsilon)\text{OPT}_L$. Since $d_j > \epsilon$ for all $j \in C$, each client j pays at least ϵ fraction of the opening cost of $\phi_L(j)$ embedded in costs c' and hence, $c_f(\phi_L) \leq \frac{1}{\epsilon}\gamma$. Therefore, we have

$$c(\phi_L) = c_s(\phi_L) + c_f(\phi_L) \leq c_s(\phi_L) + \frac{1}{\epsilon}\gamma \leq \frac{1}{\epsilon}c_s(\phi_L) + \frac{1}{\epsilon}\gamma \leq \frac{1}{\epsilon}\alpha'(\epsilon)\text{OPT}_L$$

for $\epsilon \leq 1$. ∎

Remark 3.4.2 *Let $\tilde{\phi_L}$ be any feasible assignment for the original instance of the RUCFLP(ϵ). A similar proof shows that $c_s(\phi_L) + \gamma \leq \alpha'(\epsilon)(c_s(\tilde{\phi_L}) + c_f(\tilde{\phi_L}))$. We use this fact in Section 3.4.1 to get a better ratio for the RUCFLP($\frac{1}{3}$).*

In the following, we present a $(7,1)$-approximation algorithm for the RUCFLP($\frac{1}{3}$) with zero opening costs (see Algorithm 3.4.1), which coupled with Lemma 3.4.1 yields a $(21,1)$-approximation algorithm for the RUCFLP($\frac{1}{3}$).

Algorithm 3.4.1 Algorithm for solving the RUCFLP($\frac{1}{3}$) with zero opening costs

```
Require: An instance of the RUCFLP($\frac{1}{3}$) with zero opening costs
Ensure: A subset $I \subseteq F$ and a function $\phi : C \to I$
1: Let $L' = \{j \in C : d_j > \frac{1}{2}\}$ and $L'' = C \backslash L'$. Assign the clients in $L'$ by running a minimum
   weight perfect matching algorithm with edge weights $w_{ij} = d_j c_{ij}$. Let $I_{L'}$ be the opened
   facilities and $\phi_{L'} : L' \to I_{L'}$ be the assignment function.
2: Build the flow network $H$ as described in Theorem 3.4.2.
3: Find a minimum cost maximum flow in $H$. If the value of the flow is smaller than $|L''|$ then
   return "Infeasible". Else, let $I_{L''}$ be the subset of facilities in $F \backslash I_{L'}$ whose corresponding nodes
   in $Y$ (in $H$) have non-zero flow through them and $\phi_{L''}$ be the assignment function defined as: if
   there is a unit flow from $x_j$ to $y_i$ in $H$ then $\phi_{L''}(j) = i$.
4: Let $I = I_{L''} \cup I_{L'}$. Combine $\phi_{L''}$ and $\phi_{L'}$ to form assignment function $\phi_L : C \to I$ where
   $\phi(j) = \phi_{L''}(j)$ if $j \in L''$, otherwise $\phi(j) = \phi_{L'}(j)$. Return $\phi$ and $I$.
```

Theorem 3.4.2 *There is a $(7,1)$-approximation algorithm for the RUCFLP($\frac{1}{3}$) with zero opening costs.*

Proof. Note that all the clients in the given instance have size $> \frac{1}{3}$. We break them into two groups: $L' = \{j \in C : d_j > \frac{1}{2}\}$ and $L'' = C \backslash L'$ are those which have size in $(\frac{1}{3}, \frac{1}{2}]$. In this proof (and of Lemma 3.4.2), we call clients in L', *huge* clients and those in L'', *moderately-large* clients. The algorithm assigns the huge clients by running a minimum weight perfect matching algorithm with edge weights $w_{ij} = d_j c_{ij}$. Let $I_{L'}$ be the opened facilities and $\phi_{L'} : L' \to I_{L'}$ be the assignment function. For moderately-large clients (i.e. those in L''), we define a flow-network H and show that minimum cost maximum flows in H correspond to minimum cost feasible assignment of clients in L'' to facilities (given the assignment $\phi_{L'}$).

Directed network H has node set $X \cup Y \cup \{s, t\}$ where there is a node $x_j \in X$ for every client $j \in L''$ and a node $y_i \in Y$ for every facility $i \in F$; s is the source and t is the sink. The source is connected to each node $x_j \in X$, and all $y_i \in Y$ are connected to the sink. Each $x_j \in X$ is connected to a node $y_i \in Y$ if either: the corresponding facility i is in $F \backslash I_{L'}$, i.e. unopened yet, or i is in $I_{L'}$ and the remaining capacity of i is enough to serve the demand of client j. Set the capacity of the edges between the source and the nodes in X to 1, set the capacity of the edges between X and Y to 1, set the capacity of the edges between the nodes $y_i \in Y$ whose corresponding facility i is unopened (i.e. not in $I_{L'}$) and the sink to 2, and set the capacity of the edges between the nodes $y_i \in Y$ whose corresponding facility is in $I_{L'}$ and the sink to 1. The cost of an edge connecting $x_j y_i$ is $d_j \cdot c_{ij}$ and all the other costs are 0. Algorithm 3.4.1 summarizes the algorithm for the RUCFLP($\frac{1}{3}$) with zero opening costs.

Let ϕ_L^* be an optimal assignment for the given instance of the RUCFLP($\frac{1}{3}$) with cost OPT_L. In Lemma 3.4.2, we will prove that there exists an assignment ϕ' of clients consistent with assignment

$\phi_{L'}$ found in Step 1, with cost at most 7OPT_L. Below we prove that in Steps 2 and 3 the algorithm finds the best possible feasible assignment of clients in L'' (given $\phi_{L'}$). Therefore, the cost of ϕ formed in Step 4 is at most $c(\phi')$ and hence, is at most 7OPT_L.

Since for any $j \in L''$: $\frac{1}{3} < d_j \leq \frac{1}{2}$, each unopened facility after Step 1 can serve any two clients of L'' (and no more than two). This fact is reflected in that we connect all the nodes in X (corresponding to moderately-large clients) to the nodes in Y corresponding to unopened facilities $F \backslash I_{L'}$ and we set the capacity of the edges between those nodes in Y and the sink to 2. In addition, each facility in $I_{L'}$ can serve at most one moderately-large client, because more than $\frac{1}{2}$ of its capacity is already used by a huge client; accordingly we set the capacity of the edges from those nodes in Y to the sink to 1 and we only connect to them the nodes of X whose corresponding client can be served by them. Considering these two simple facts:

Claim 3.4.1 *The maximum flow in H has value $|L''|$ if and only if the given instance is feasible and there is a one to one correspondence between maximum flows in H and feasible assignment of moderately-large clients (i.e. in L'') given $\phi_{L'}$. Furthermore, a pair of corresponding maximum flow in H and assignment of clients of L'' to F have the same cost.*

Proof. Observe that all the edges of H have integer capacities; so in any maximum flow edges have only integer flows. If a node $x_j \in X$ has a flow of one to a node $y_i \in Y$ we assume client j is assigned to facility i, and vice-versa. First, suppose that the instance is feasible and let ϕ be an arbitrary feasible assignment. We show that there is a feasible assignment consistent with $\phi_{L'}$. Let $I_{L'}^{(\phi)}$ be the set of open facilities in ϕ to which a client of L' is assigned. Clearly, $|I_{L'}^{(\phi)}| = |L'|$. Since opening costs are zero and all facilities have the same capacity, we can easily swap the facilities in $I_{L'}^{(\phi)}$ with ones to which a client of L' is assigned to in $\phi_{L'}$ so that we get a feasible assignment consistent with $\phi_{L'}$. Then it is easy to see that H has a flow of value $|L''|$ (basically the edges between X and Y in H with non-zero flow correspond to the assignment of clients of L'' in the feasible solution consistent with $\phi_{L'}$). Conversely, if H has a flow of $|L''|$ then that corresponds to a feasible assignment of moderately-large clients to facilities (consistent with $\phi_{L'}$). The correspondence between cost of a maximum flow and an assignment of clients (of L'') is immediate from the definition of costs of edges. ■

Therefore, the assignment $\phi_{L''}$ obtained from a minimum cost maximum flow in H has the minimum cost among the assignments consistent with $\phi_{L'}$. This together with Lemma 3.4.2 implies that ϕ_B as defined has cost at most 7OPT_L. ■

Lemma 3.4.2 *There exists an assignment ϕ' of clients consistent with $\phi_{L'}$ with cost at most 7OPT_L where OPT_L is the cost of an optimum assignment ϕ^*_L for the given instance of the RUCFLP($\frac{1}{3}$).*

Proof. Let $\phi^*_{L'}$ be the restriction of ϕ^*_L to clients in L', and $G_{L'}$ be the graph induced by the edges used in $\phi_{L'}$ or $\phi^*_{L'}$. Since at most one client in L' is assigned to each facility by any feasible solution, the degree of vertices in $G_{L'}$ is at most 2; so $G_{L'}$ is a collection of cycles and paths. For an arbitrary path, we show how the assignment of huge clients in ϕ^*_L (i.e. those in L') can be changed to the ones in $\phi_{L'}$ without violating any capacity, while the cost increases by at most a factor of 7. Similarly, it can be shown that each cycle can be fixed with at most a factor of 3 increase in its cost, which completes the proof.

Let $P = i_1 j_1 i_2 j_2 \dots i_k j_k i_{k+1}$ be an arbitrary path in $G_{L'}$ where $\phi^*_{L'}(j_l) = i_l$ and $\phi_{L'}(j_l) = i_{l+1}$ for $1 \le l \le k$ (see Figure 3.3). Let S_l be the set of moderately-large clients (i.e. in L'') assigned to facility i_l by ϕ^*_L. In ϕ', we assign j_l and the clients in S_l to i_{l+1} for $1 \le l \le k$ and we assign the clients in S_{k+1} to j_1. Since we used a min-cost maximum matching algorithm to find $\phi_{L'}$, changing the assignment of clients in L' does not increase the cost. Therefore, we only need to analyze the increase in cost because of the change in assignment of moderately large clients.

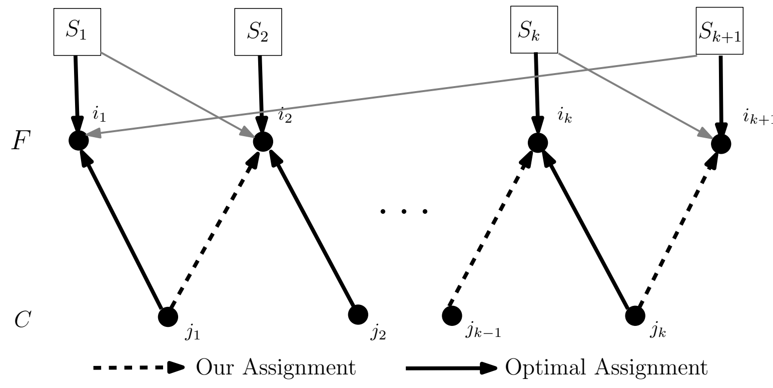


Figure 3.3: An arbitrary path in G'_L. In ϕ', the clients in S_l are assigned to the facility pointed to by the gray arrow adjacent to S_l.

Consider an arbitrary l where $1 \le l \le k$. Since $d_l > \frac{1}{2}$, the total demand of clients in S_l is less than d_l. Therefore, if we send the clients in S_l to i_{l+1} over the edges $i_l j_l$ and $j_l i_{l+1}$, we increase the cost by at most $d_l(c_{i_l j_l} + c_{j_l i_{l+1}})$. Thus, changing the assignment of clients in $\cup_{1 \le l \le k} S_l$ increases the cost by at most a factor of 2. Finally, the total demand of clients in S_{k+1} is at most $1 < 2d_l$ for any $1 \le l \le k$. Therefore, sending back these clients to i_1 over P increases the cost by at most a factor of 4. Hence, the overall cost of ϕ' is at most 7OPT_L. ■

The above lemma is essentially tight: there are instances of the RUCFLP($\frac{1}{3}$) with zero costs that after deciding the assignment of huge clients by a minimum weight perfect matching algorithm, any feasible solution consistent with this assignment has cost at least 7OPT_L (see Figure 3.4). Consider a path $P = v_1 v_2 v_3$ where the edges have unit cost. There are two facilities with zero opening cost on v_1 and v_3. In addition, there is a moderately-large client of demand $1/2 - \delta$ on v_1, a huge client of demand $1/2 + \delta$ on v_2, and two moderately-large clients of demand $1/2$ on v_3, where

δ is a small positive number. In the optimal solution, the client on v_2 is assigned to the facility on v_1 and $OPT_L = 1/2 + \delta$. In our solution, the minimum weight perfect matching algorithm may assign the huge client on v_2 to the facility on v_3. Then, in any feasible solution, we must move the moderately-large client on v_1 to v_3 and move the two moderately-large clients on v_3 to v_1, so the total cost of any feasible solution given the assignment of the huge client, is at least $(1/2 - \delta) \times 2 + 1 \times 2 + (1/2 + \delta) \times 1 = 7/2 - \delta$. This shows any feasible solution is essentially a factor of 7 away from the optimum.

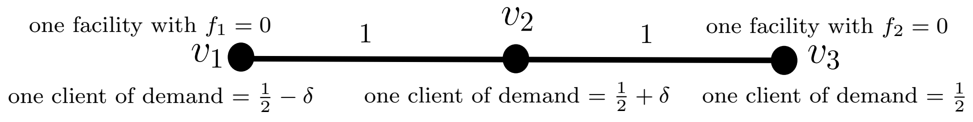

one facility with $f_1 = 0$
v_1
1
v_2
1
one facility with $f_2 = 0$
v_3
one client of demand $= \frac{1}{2} - \delta$
one client of demand $= \frac{1}{2} + \delta$
one client of demand $= \frac{1}{2}$

Figure 3.4: A tight example for Lemma 3.4.2.

Combining Lemma 3.4.1 and Theorem 3.4.2:

Corollary 3.4.2 *There is a polynomial time* $(21, 1)$*-approximation algorithm for the RUCFLP(*$\frac{1}{3}$*).*

Corollary 3.4.3 *There is a (160.334,4/3)-approximation algorithm for the UCFLP.*

Proof. We run Algorithm 3.3.1, where we use the algorithm of Corollary 3.4.2 for $\mathcal{A}$. That is, we first run the $(7, 1)$-approximation algorithm of Theorem 3.4.2 as algorithm $\mathcal{A}'$ in Lemma 3.4.1 to obtain $\mathcal{A}$ with $\alpha(\epsilon) = 21$ and $\epsilon = 1/3$. Thus $h_1(\frac{1}{3}) = 19/3$, $h_2(\frac{1}{3}) = 19/3$, and $g(\epsilon, \alpha(\epsilon)) = 19/3 + 21(22/3) < 160.334$. Since $\beta(\epsilon) = 1$, the overall ratio is $(160.334, 4/3)$. ■

Notice that we solved the RUCFLP($\frac{1}{2}$) and ($\frac{1}{3}$) without violation of capacities, but this is not possible for smaller values of ϵ as shown below.

Theorem 3.4.3 *The RUCFLP(*ϵ*) does not admit any* $(\alpha(\epsilon), 1)$*-approximation algorithm for* $\epsilon < \frac{1}{3}$ *unless* ***P**= **NP***.

Proof. This can be shown by a simple reduction from the **NP**-hard problem 3-partition. In the 3-partition problem, we are given a set of $3m$ integers $a_1, \ldots, a_{3m}$, a positive integer bound B where $\frac{B}{4} < a_j < \frac{B}{2}$ for all $1 \leq j \leq 3m$ and $mB = \sum_{1 \leq j \leq 3n} a_j$. The question is if there is a way to partition these numbers into m sets of size 3 each such that the sum of the numbers in each set is exactly B. This problem is **NP**-hard [GJ79].

Starting from a given instance $\mathcal{I}_p$ of the 3-partition problem, we build an instance $\mathcal{I}_R$ of the RUCFLP(ϵ) in the following way. Let c be a constant dependent on ϵ, which we define soon. For each $1 \leq j \leq 3m$, let $d_j = \frac{a_j + cB}{B(3c+1)}$ and create a client j with demand d_j. Also, create m facilities with zero opening cost and capacity 1. We set all the connection costs $c_{ij} = 0$. We choose constant c large enough such that $\epsilon < \frac{1}{3} - \frac{1}{12(3c+1)}$. Since $\frac{B}{4} < a_j < \frac{B}{2}$ for all $1 \leq j \leq 3m$, the value of the demands are between $\frac{1}{3} - \frac{1}{12(3c+1)}$ and $\frac{1}{3} + \frac{1}{6(3c+1)}$ and clearly, are greater than ϵ by the choice of c. This completes the description of instance $\mathcal{I}_R$.

First, note that if we define $a'_j = \delta_1 a_j + \delta_2$ for all $1 \leq j \leq 3m$ and $B' = \delta_1 B + 3\delta_2$ for two positive constants δ_1 and δ_2, then this new instance is a yes instance of the 3-partition problem if and only if $\mathcal{I}_p$ is a yes instance. In the above reduction, we used $\delta_1 = \frac{1}{B(3c+1)}$ and $\delta_2 = \frac{c}{3c+1}$ to define d_j values. Thus, $\mathcal{I}_p$ is a yes instance if and only if we can partition d_j values to m sets of size 3 each such that the sum of numbers in each set is exactly $B' = 1$. Since any solution for $\mathcal{I}_R$, that does not violate the capacity constraints, provides such a partition of d_j values, an $(\alpha(\epsilon), 1)$-approximation algorithm for the RUCFLP(ϵ) can be used to solve the 3-partition problem in polynomial time. Therefore, unless **P**= **NP**, there is no such approximation algorithm. ■

It should be noted that to find an algorithm for the UCFLP that violates capacities within factor $1 + \epsilon$, we do not need to find an algorithm that does not violate capacities in the RUCFLP(ϵ). Even if we violate the capacities within factor $1 + \epsilon$ in the RUCFLP(ϵ), using Theorem 3.2.1 we can get an algorithm for the UCFLP that violates the capacities within factor $1 + \epsilon$. We think it is possible to find an $(\alpha(\epsilon), 1 + \epsilon)$-approximation for the RUCFLP(ϵ) for any constant $\epsilon > 0$. This, together with Theorem 3.2.1 would imply an $(f(\epsilon), 1 + \epsilon)$-approximation for the UCFLP, for any constant $\epsilon > 0$.

3.4.1 Improving the ratios

With a more careful analysis and a simple scaling to balance the bi-factors of connection and facility costs, we can bring down the factors of our algorithms. A similar scaling has been used to obtain better ratios for several variations of facility location problems (for example see [CG99]). For some parameters δ_1 and δ_2 to be defined, we change Algorithm 3.3.1 as follows:

1. We multiply the original connection costs by δ_1 to get a new cost function $c^{(1)}$, *i.e.*, $c^{(1)}_{ij} = \delta_1 c_{ij}$ for all $i \in F$ and $j \in C$. Then, we perform Step 1 with cost $c^{(1)}$ to find ϕ_L.
2. After step 1, we multiply the *original* connection costs by δ_2 to get a new cost function $c^{(2)}$ *i.e.*, $c^{(2)}_{ij} = \delta_1 c_{ij}$ for all $i \in F$ and $j \in C$. Then, we do Steps 2 and 3 with cost $c^{(2)}$ to find ϕ_S.

In the following, for an assignment ϕ, we use $c_s(\phi)$, $c^{(1)}_s(\phi)$, and $c^{(2)}_s(\phi)$ to indicate the service cost of this assignment with respect to cost functions c, $c^{(1)}$, and $c^{(2)}$, respectively. In addition, assignments ϕ^* and ϕ^*_L have the same definition as before (i.e. for the optimal solution) and are defined with respect to the original costs. As a result, they are not necessarily optimal with respect to cost functions $c^{(1)}$ and $c^{(2)}$.

By Remark 3.3.1, although ϕ^* is not necessarily optimal with respect to $c^{(1)}$, there exist an assignment ϕ''_S such that $c^{(1)}_s(\phi''_S) \leq c^{(1)}_s(\phi^*) + c^{(1)}_s(\phi_L)$. Thus:

$$\begin{aligned} c_s^{(2)}(\phi''_S) &= \frac{\delta_2}{\delta_1} c_s^{(1)}(\phi''_S) \\ &\leq \frac{\delta_2}{\delta_1} c_s^{(1)}(\phi^*) + \frac{\delta_2}{\delta_1} c_s^{(1)}(\phi_L) \\ &= c_s^{(2)}(\phi^*) + c_s^{(2)}(\phi_L). \qquad (3.5) \end{aligned}$$

In addition, the inequalities for $c_s(\phi_S)$ and $c_f(\phi_S)$ at the end of proof of Theorem 3.2.1 change to:

$$\begin{aligned} c_s^{(2)}(\phi_S) &= c_s^{(2)}(\phi'_S) \\ &\leq \lambda_{ss} c_s^{(2)}(\phi''_S) + \lambda_{sf} c_f(\phi''_s) \\ &\leq \lambda_{ss}(c_s^{(2)}(\phi^*) + c_s^{(2)}(\phi_L)) + \lambda_{sf} c_f(\phi^*), \qquad \text{by Eq. (3.5)} \qquad (3.6) \\ c_f(\phi_S) &\leq (1+\epsilon) c_f(\phi'_S) \\ &\leq (1+\epsilon)\lambda_{fs} c_s^{(2)}(\phi''_S) + (1+\epsilon)\lambda_{ff} c_f(\phi''_S) \\ &\leq (1+\epsilon)\lambda_{fs}(c_s^{(2)}(\phi^*) + c_s^{(2)}(\phi_L)) + (1+\epsilon)\lambda_{ff} c_f(\phi^*). \qquad \text{by Eq. (3.5)} \quad (3.7) \end{aligned}$$

Therefore, scaling down Equation (3.6) by δ_2 and using definition of $c^{(2)}$:

$$c_s(\phi_S) \leq \lambda_{ss}(c_s(\phi^*) + c_s(\phi_L)) + \frac{\lambda_{sf}}{\delta_2} c_f(\phi^*).$$

Also, using definition of $c^{(2)}$ and Equation (3.7):

$$c_f(\phi_S) \leq \delta_2(1+\epsilon)\lambda_{fs}(c_s(\phi^*) + c_s(\phi_L)) + (1+\epsilon)\lambda_{ff} c_f(\phi^*).$$

Adding these inequalities together and then, adding $c_s(\phi_L) + c_f(\phi_L)$ to the result, we obtain:

$$\begin{aligned} c(\phi) &= c_s(\phi_S) + c_f(\phi_S) + c_s(\phi_L) + c_f(\phi_L) \\ &\leq \hat{h}_1(\epsilon) c_s(\phi^*) + \hat{h}_2(\epsilon) c_f(\phi^*) + (\hat{h}_1(\epsilon) + 1) c_s(\phi_L) + c_f(\phi_L) \qquad (3.8) \end{aligned}$$

where $\hat{h}_1(\epsilon) = \lambda_{ss} + \delta_2(1+\epsilon)\lambda_{fs}$ and $\hat{h}_2(\epsilon) = \frac{\lambda_{sf}}{\delta_2} + (1+\epsilon)\lambda_{ff}$. Using algorithm of Aggarwal et al. [AAB$^+$10] for splittable CFLP, we have $\lambda_{ss} = \lambda_{sf} = 1$ and $\lambda_{fs} = \lambda_{ff} = 4$. Therefore, $\hat{h}_1(\epsilon) = 1 + 4\delta_2(1+\epsilon)$ and $\hat{h}_2(\epsilon) = \frac{1}{\delta_2} + 4(1+\epsilon)$. We use these equalities to get the improved results.

Proof of Theorem 3.2.2. We run the modified Algorithm 3.3.1 with $\delta_1 = \hat{h}_1(\frac{1}{2}) + 1$ and value of δ_2 to be defined soon. We have:

$$\begin{aligned} (\hat{h}_1(\frac{1}{2}) + 1) c_s(\phi_L) + c_f(\phi_L) &= c_s^{(1)}(\phi_L) + c_f(\phi_L) \\ &\leq c_s^{(1)}(\phi_L^*) + c_f(\phi_L^*) \\ &= (\hat{h}_1(\frac{1}{2}) + 1) c_s(\phi_L^*) + c_f(\phi_L^*) \\ &\leq (\hat{h}_1(\frac{1}{2}) + 1) c_s(\phi^*) + c_f(\phi^*), \end{aligned}$$

where the first inequality follows from Remark 3.4.1. Combining this with Inequality (3.8):

$$c(\phi) \le (2\hat{h}_1(\frac{1}{2}) + 1)c_s(\phi^*) + (\hat{h}_2(\frac{1}{2}) + 1)c_f(\phi^*) = (12\delta_2 + 3)c_s(\phi^*) + (1/\delta_2 + 7)c_f(\phi^*).$$

Solving the equation $12\delta_2 + 3 = 1/\delta_2 + 7$ for δ_2, we find $\delta_2 = (4 + \sqrt{64})/24 = \frac{1}{2}$, which gives the claimed ratio. ■

Proof of Theorem 3.2.3. Recall that we use Lemma 3.4.1 to obtain algorithm $\mathcal{A}$ used in Algorithm 1 for large clients. We first start with a more careful analysis in Lemma 3.4.1. By Remark 3.4.2, the proof of this lemma implies:

$$c_f(\phi_L) \le \frac{1}{\epsilon}\gamma \text{ and } c_s^{(1)}(\phi_L) + \gamma \le \alpha'(\epsilon)(c_s^{(1)}(\phi_L^*) + c_f(\phi_L^*)).$$

In our case, we have $\epsilon = 1/3$ and $\alpha'(\epsilon) = 7$. Thus: $c_f(\phi_L) \le 3\gamma \le 3(7(c_s^{(1)}(\phi_L^*) + c_f(\phi_L^*)) - c_s^{(1)}(\phi_L))$ which implies $3c_s^{(1)}(\phi_L) + c_f(\phi_L) \le 21(c_s^{(1)}(\phi_L^*) + c_f(\phi_L^*))$. If we set $\delta_1 = (\hat{h}_1(\frac{1}{3}) + 1)/3$, we have

$$\begin{aligned}(\hat{h}_1(\frac{1}{3}) + 1)c_s(\phi_L) + c_f(\phi_L) &\le 7(\hat{h}_1(\frac{1}{3}) + 1)c_s(\phi_L^*) + 21c_f(\phi_L^*) \\ &\le 7(\hat{h}_1(\frac{1}{3}) + 1)c_s(\phi^*) + 21c_f(\phi^*).\end{aligned}$$

Combining this with Inequality (3.8):

$$\begin{aligned}c(\phi) &\le (\hat{h}_1(1/3) + 7(\hat{h}_1(1/3) + 1))c_s(\phi^*) + (\hat{h}_2(1/3) + 21)c_f(\phi^*) \\ &= (8(16\delta_2/3 + 1) + 7)c_s(\phi^*) + (1/\delta_2 + 16/3 + 21)c_f(\phi^*) \\ &= (128\delta_2/3 + 15)c_s(\phi^*) + (1/\delta_2 + 79/3)c_f(\phi^*)\end{aligned}$$

By solving the equation $128\delta_2/3 + 15 = 1/\delta_2 + 79/3$ for δ_2, we find $\delta_2 = (17 + \sqrt{673})/128$, which gives the claimed ratio. ■

3.5 The Euclidean UCFLP

In this section, we present a quasi-polynomial time $(1 + \epsilon, 1 + \epsilon)$-approximation algorithm for the UCFLP in Euclidean metrics. For these instances, we do not reduce our problem to the RUCFLP(ϵ) as we did in the previous sections. The reason is that one cannot get a ratio better than $(3, 1 + \epsilon)$ by applying this reduction as shown in the tight example of Lemma 3.3.1 (note that the cost function in that example is Euclidean).

Our algorithm for Euclidean metrics assigns the large clients integrally using a dynamic programming approach, while assigns the small clients fractionally at the same time at low cost. Then we can round the fractional assignment of the small clients by the algorithm of Shmoys and Tardos [ST93] for the GAP, as we did in Section 3.3. This rounding will not increase the cost and yields a blowup of factor at most $1+\epsilon$ in facility capacities. We should note that Bateni and Hajiaghayi [BH09] used this approach in the design of their $(1+\epsilon, 1+\epsilon)$-approximation algorithm for the UCFLP in the metrics induced by trees. We combine some of their ideas with techniques developed for designing PTASs for some Euclidean optimization problems; most notably the algorithms of [Aro96, ARR98]. While our algorithm has some similarities to that of [BH09] (such as the way we define client types), there are some technical differences. In particular, our table entries in the dynamic programming are defined based on shifted quad-tree that is obtained from a dissection of the plane and the way we combine solutions for the subproblems is more complicated.

We assume that the input points are on a unit grid, the minimum inter-node distance is at least 4 (which implies no two nodes are located on the same grid point), and the maximum inter-node distance is at most $O(n^4)$. We can enforce all these assumptions by a preprocessing step similar to the one used by Arora [Aro97] to make the instances *well-rounded* for k-TSP and k-MST. For completeness of exposition, we describe this perturbation step in Subsection 3.5.4.

We use the randomly shifted quad-tree dissection method due to Arora to partition Euclidean plane [Aro96]. We briefly explain it here, a reader familiar with this can skip this paragraph. We start with the bounding square of the input points and recursively partition it into smaller sub-squares. Suppose the length of this bounding square (box) is L and without loss of generality, assume L is a power of 2. Each square is partitioned into four equal sub-squares and we recursively proceed for each sub-square and stop partitioning a sub-square if it has unit length (and therefore has at most one input point in it). There is an immediate 4-ary tree structure corresponding to this dissection where each square in the dissection is a node of the tree. For two integers $a, b \in [0, L)$, the (a, b)-shift of dissections is obtained by shifting the x- and y- coordinates of all the dissecting lines by a and b modulo L, respectively. We obtain an (a, b)-shifted quad-tree from the 4-ary tree representing the corresponding shifted dissection by simply removing (from the tree) the partitioning of squares without any input point.

For each square in a shifted dissection, we place a portal at each of its four corners and place m evenly spaced points on each edge, where m is a power of 2 (to be defined later). In other words, we put $4(m+1)$ portals on each square and the portals of higher level squares are co-located on some portals of lower level squares. Note however that each portal is owned by a distinct square (so even if several portals are co-located at the same point each belongs to one unique square). Let a portal-respecting path between two points be a path that crosses the squares only at portals. Let S be a collection of pairs of input points in the plane and let $c(S)$ be the total Euclidean distance of pairs. Arora, Raghavan, and Rao [ARR98] show that in presence of our assumptions, if we pick $0 \leq a, b < L$ uniformly at random and use $m = O(\log n/\epsilon)$ portals on each side of each square, then with probability at least $\frac{1}{2}$, there exist some portal-respecting paths between the pairs in S with total length (cost) at most $(1+\epsilon)c(S)$. We present a dynamic programming algorithm to find a $(1+\epsilon, 1+\epsilon)$-approximate portal-respecting solution for the UCFLP instances. Then, we run this algorithm for all possible values of a and b and return the best solution of these runs. Based on the above result regarding portal respecting paths, this solution is $(1+\epsilon', 1+\epsilon')$-approximate solution for the Euclidean UCFLP for some ϵ' depending on ϵ.

3.5.1 Grouping Clients and Rounding Demand Sizes

For the simplicity of description, we assume $\frac{1}{\epsilon}$ is an integer and denote it by p. First, we apply a grouping and rounding step which is essentially the same as the one in [BH09]. We partition the clients into three groups of *large*, *small*, and *tiny* clients according to their demand sizes. We round down the demands of large and small clients and show that the precision error (i.e. violation of facility capacities) resulted from this rounding does not exceed 2ϵ. Also, this rounding does not increase the cost by more than a $(1+\epsilon)$ factor when we restore the original demands. As before, the clients with demands greater than ϵ are called *large* but the definition of small clients are different from before. We round down the demands of large clients to the nearest multiple of ϵ^2. This yields $q = \frac{1-\epsilon}{\epsilon^2} + 1 = p^2 - p + 1$ distinct demand values. Since each facility can serve at most q large clients, this rounding yields a blowup of at most $p\epsilon^2 = \epsilon$ of facility capacities. Therefore, if we work with these rounded down demand values from now on, we violate the facility capacities by at most an additive value of ϵ. We define a client type corresponding to each of these q distinct demand value; so far we have defined q client types.

A client j with demand $\epsilon/n < d_j \leq \epsilon$ is called *small*. We round down the demands of small clients to the nearest multiple of ϵ^2/n. Similar to large clients, this yields an additive blowup of at most ϵ of facility capacities (as there can be at most n/ϵ small clients adding a total of ϵ to the demand of a facility). We intend to assign these clients fractionally. Therefore, we just need to keep track of their total demand which is a multiple of ϵ^2/n. For this purpose, we divide each small client with rounded demand d into dn/ϵ^2 clients each with demand ϵ^2/n. Since $d \leq \epsilon$ and is a multiple of ϵ^2/n, we break each small client into at most n/ϵ clients. These clients of demand ϵ^2/n form type $q+1$.

Finally, we call all the other clients *tiny*. The demand of these clients are negligible with respect to capacities, *i.e.* even if we assign all of them to an already full facility, it yields an additive blowup of at most ϵ. We leave their demands intact. It should be noted that their assignment cost may still be significant and we must be careful in their assignment. Now, we are ready to define the dynamic programming tables with respect to the $p+1$ client types and tiny clients. Note that $q+1$ is a constant for a fixed $\epsilon > 0$ and this is crucial in our algorithm.

3.5.2 The Dynamic Programming Tables

We define a set of subproblems for each square in the shifted quad-tree and solve it by combining the solutions of its children. We treat the clients of type 1 through $q+1$ separately from tiny clients. First, we explain how we handle the clients of type 1 through $q+1$. Consider a shortest portal-respecting path from a client to a facility. It passes through some nodes of the quad-tree (i.e. squares in the dissection): The client moves up to its least common ancestor in the quad-tree and moves downward to the node (i.e. square) containing the facility. Equivalently, we can think of moving up the facility to meet the client in their least common ancestor node in the tree and each one pays its own movement cost. Looking at the problem this way, the portion of a facility capacity serving a client must move to their least common ancestor. In fact, we are breaking each facility i into some virtual facilities, one for each client j it must serve, and move that virtual facility (corresponding to the portion of capacity for serving j) to meet j at the least common ancestor of the two squares containing i and j.

Assume we have decided (in the dynamic programming) that a facility must serve n_ℓ clients of demand type ℓ for each $1 \leq \ell \leq q + 1$ (we make this decision by enumerating all possible ways of choosing n_ℓ values when we solve the base case corresponding to the leaf of quad-tree containing this facility). We break this facility into n_ℓ virtual facilities of type ℓ for all $1 \leq \ell \leq q+1$. A virtual facility of type ℓ can only serve a client of type ℓ. Considering the problem this way, in each square some clients will be served by facilities inside, some clients will be shipped outside through the portals of the square, some virtual facilities inside will serve the clients inside and some will be sent outside through the portals to serve clients from outside. Thus, for each portal of a square, we just need to keep track of how many of each demand type is sent to this portal to be served outside and how many of each virtual facility type is sent to this portal to service clients from outside. One nice feature of breaking the facilities into virtual facilities is that once we have decided how a facility is going to be broken into virtual facilities (i.e. each n_ℓ is guessed for $1 \leq \ell \leq q+1$), we can consider the subproblem of each demand type independent of other types since no client or virtual facility of a type can interfere with other types.

To handle tiny clients, we first need to point out a simple observation. Assume some tiny clients are sent to a portal of a square they belong to, to be sent to their facility. As we stated before, tiny clients can be assigned to any open facility without a significant blowup in capacities. Thus, we can send all the tiny clients sent to a portal to the closest open facility, without increasing the connection cost. In other words, there is a solution with cost no more than optimal cost in which all the tiny clients sent to the same portal, head to the same facility (i.e. nearest open one) and we seek such a solution in our algorithm. To find such a solution, we fix (by enumerating all possibilities) the facility that should be used by tiny clients sent to each portal (of each square) in our dynamic programming. Then, instead of assigning each tiny client to a facility, we assign it to one of the portals around the square containing it and automatically all the tiny clients assigned to that portal will be served at the ”guessed” nearest open facility for that portal. After choosing a portal for a tiny client, the client pays the cost of connection to the facility of that portal with a portal-respecting path fully upfront.

Formally, an instance of a subproblem is defined as a tuple (s, D, F) where s is a square in the quad-tree (i.e. the dissection) and D and F are two matrices containing information about the demands and (virtual) facilities moved to portals of s: D is a $4(m+1) \times (q+2)$ matrix (for demands) where the ith row keeps track of the information regarding the ith portal of S. The first $q+1$ elements of this row show the number of clients of each client type (from 1 to $q+1$) moved to this portal. The $(q+2)$th element shows the index of the facility that the tiny clients (inside of s) that are moved to this portal plan to use. F is defined similarly for facilities, namely it is a $4(m+1) \times (q+2)$ matrix where the ith row keeps track of the information regarding the ith portal for facilities. The first $q+1$ elements of this row show the number of virtual facilities of each of $q+1$ virtual facility types moved to this portal. The $(q+2)$th element shows the index of the closest open facility inside s to this portal (to be used for tiny clients moved to this portal). A value of 0 for facility index indicates that there is no open facility available inside. We store in $A[s, D, F]$ the value of optimal solution of the subproblem (s, D, F), *i.e.* the minimum total cost to service the clients in square s by opening facilities inside and sending clients and virtual facilities inside s to outside of s through its portals according to matrices D and F. The tiny clients inside s pay their connection cost fully upfront i.e. if they are to be served at a facility outside of s they contribute their connection cost to those facilities to the total cost, while the other clients (large and small) just pay the cost to move to their portal or to be served inside. The solution that we are seeking to find is $A[s_r, \mathbf{0}, \mathbf{0}]$ with s_r being the bounding box of all the input points. In the last step, we should round the "fractional" assignment of small clients to an integer one, again using the algorithm of Shmoys and Tardos [ST93] for the GAP. This rounding does not increase the cost but may violate the capacity constraints by the maximum demand value. Since each small client has demand at most ϵ, this will incur another additive of at most ϵ blow up in capacity constraints.

3.5.3 Computing the table

We now show how we can compute $A[s, D, F]$ recursively. The leaves of the quad-tree have either one client or one facility in them. We have three cases:

1. There is one client of type j ($1 \leq j \leq p+1$) inside s: We must ship this client to one of the portals. For the entries $A[s, D, F]$ where F is the zero matrix and D has one non-zero value equal to 1 in the jth column of exactly one of its rows, say row ℓ, we set $A[s, D, F]$ to the cost of shipping the client to the ℓth portal. For all other entries, we set $A[s, D, F]$ to undefined (or infinity).

2. There is one tiny client inside s: For the entries $A[s, D, F]$ where F is zero and D has exactly one non-zero value, say i (corresponding to facility i), in the $(p+2)$th column of exactly one of its rows, say ℓ, we set $A[s, D, F]$ to the cost of moving the client to the ℓth portal and then moving it through the shortest portal-respecting path between the portal and facility i. For all other entries, we set $A[s, D, F]$ to undefined (or infinity).

3. There is a facility, say i, in the square. We set $A[s, \mathbf{0}, \mathbf{0}] = 0$. For the entries $A[s, D, F]$ in which D is the zero matrix, and the total capacity (i.e. virtual facilities) sent to the portals of s according to F is not more than the capacity of facility i, and the $(p+2)$th column of F has value i in its entries, we set $A[s, D, F]$ to the opening cost of i plus the cost of moving virtual facilities according to F to the portals. For all other entries, we set $A[s, D, F]$ to undefined (or infinity).

We update each non-leaf node as follows. Let s be a non-leaf node and s_1, s_2, s_3, and s_4 be its four children. We enumerate all the combinations of matrices D and F for s and matrices D_i and F_i for s_i for all $1 \le i \le 4$ where the subproblems are consistent (to be explained below). We update $A[s, D, F]$ with respect to subproblems (s_i, D_i, F_i). As mentioned before, we can solve the problem for each demand type and the corresponding virtual facility type separately, because there is no dependencies between two different types. Let Ψ_ℓ be the total cost for demand types ℓ in subproblem (s, D, F) assuming the subproblems (s_i, D_i, F_i) $(1 \le i \le 4)$. We show how to compute Ψ_ℓ below.

Let $m^{(\ell)}$ and $m_i^{(\ell)}$ (for $1 \le i \le 4$) be the number of virtual facilities of type ℓ in F and F_i, respectively. We define $n^{(\ell)}$ and $n_i^{(\ell)}$ (for $1 \le i \le 4$) for the number of clients of type ℓ in D and D_i, respectively. Note that $n_1^{(\ell)} + n_2^{(\ell)} + n_3^{(\ell)} + n_4^{(\ell)} - n^{(\ell)}$ is the total number of clients of type ℓ inside s that are to be serviced inside s since only $n^{(\ell)}$ many demands of type ℓ are shipped outside from a total of $n_1^{(\ell)} + n_2^{(\ell)} + n_3^{(\ell)} + n_4^{(\ell)}$. Similarly, of a total of $m_1^{(\ell)} + m_2^{(\ell)} + m_3^{(\ell)} + m_4^{(\ell)}$ virtual facility of type ℓ inside s, only $m^{(\ell)}$ is sent to portals and therefore $m_1^{(\ell)} + m_2^{(\ell)} + m_3^{(\ell)} + m_4^{(\ell)} - m^{(\ell)}$ virtual facilities of type ℓ must be used inside s. Therefore, we must have

$$n_1^{(\ell)} + n_2^{(\ell)} + n_3^{(\ell)} + n_4^{(\ell)} - n^{(\ell)} = m_1^{(\ell)} + m_2^{(\ell)} + m_3^{(\ell)} + m_4^{(\ell)} - m^{(\ell)}, \tag{3.9}$$

or else the subproblems considered are inconsistent. Denote this quantity by $r^{(\ell)}$; so we must assign exactly $r^{(\ell)}$ clients of type ℓ to $r^{(\ell)}$ facilities of this type inside s; otherwise this combination of sub-problems is impossible and the solutions to the sub-problems are inconsistent. We can assign these $r^{(\ell)}$ clients and facilities of type ℓ optimally in polynomial time by a running a minimum cost perfect matching algorithm as described below. Note that by Equation (3.9), we must have $n_1^{(\ell)} + n_2^{(\ell)} + n_3^{(\ell)} + n_4^{(\ell)} + m^{(\ell)} = m_1^{(\ell)} + m_2^{(\ell)} + m_3^{(\ell)} + m_4^{(\ell)} + n^{(\ell)}$. Since out of $n_1^{(\ell)} + n_2^{(\ell)} + n_3^{(\ell)} + n_4^{(\ell)}$ clients of type ℓ, $n^{(\ell)}$ are shipped outside, therefore, we must match each of $n_1^{(\ell)} + n_2^{(\ell)} + n_3^{(\ell)} + n_4^{(\ell)}$ to one of $n^{(\ell)}$ or one of $m_1^{(\ell)} + m_2^{(\ell)} + m_3^{(\ell)} + m_4^{(\ell)}$ virtual facilities of type ℓ. Similarly, out of $m_1^{(\ell)} + m_2^{(\ell)} + m_3^{(\ell)} + m_4^{(\ell)}$ facilities of type ℓ, only $m^{(\ell)}$ are sent to portals of s, thus each is either matched to one of $m^{(\ell)}$ portals or is used to serve one of $n_1^{(\ell)} + n_2^{(\ell)} + n_3^{(\ell)} + n_4^{(\ell)}$ clients. This suggests to form a bipartite graph H with $n_1^{(\ell)} + n_2^{(\ell)} + n_3^{(\ell)} + n_4^{(\ell)}$ clients of the sub-squares $s_1, \ldots, s_4$ and $m^{(\ell)}$ facilities of s on one side and $m_1^{(\ell)} + m_2^{(\ell)} + m_3^{(\ell)} + m_4^{(\ell)}$ facilities of the sub-squares and $n^{(\ell)}$ clients of s on the other side. Put an edge between two vertices of the two partitions of H unless one of them is from the $m^{(\ell)}$ facilities and the other is from the $n^{(\ell)}$ clients. In other words, H has

$(n_1^{(\ell)} + n_2^{(\ell)} + n_3^{(\ell)} + n_4^{(\ell)} + m^{(\ell)})(m_1^{(\ell)} + m_2^{(\ell)} + m_3^{(\ell)} + m_4^{(\ell)} + n^{(\ell)}) - m^{(\ell)}n^{(\ell)}$ edges. The cost of an edge of H is the cost of the shortest portal-respecting path between its endpoints times d_ℓ (demand of type ℓ). It is not hard to see that this gives the best possible assignment for the demand type ℓ given the sub-problems, i.e. Ψ_ℓ is the weight of the minimum cost matching of H. We update $A[s, D, F]$ to:

$$\min(A[s, D, F], A[s_1, D_1, F_1] + A[s_2, D_2, F_2] + A[s_3, D_3, F_3] + A[s_4, D_4, F_4] + \sum_{\ell=1}^{p+1} \Psi_\ell).$$

If for one of the types $1 \leq \ell \leq p+1$, one of the sub-problems does not have a solution, then the combinations of matrices that we have guessed are inconsistent (and we do not update $A[s, D, F]$).

One other consistency condition of the matrices D, F, and D_i, F_i (for $1 \leq i \leq 4$) that we need to check is that for co-located portal points of s, s_1, s_2, s_3, and s_4, we must have consistency of facilities devoted to tiny clients in those portal points. For instance, if t_1 is a portal of s_1 and t is a portal point of s and the tiny clients of s_1 shipped to t_1 are to be served at facility i_1 and the tiny clients of s shipped to t are to be served at i and the shortest portal-respecting path from t_1 to i_1 goes through t then we must have $i_1 = i$, or else the sub-problems are not consistent. Here is how we check for this type of consistency. Let T be the set of all portals in s, s_1, s_2, s_3, s_4. Recall that each portal point is "owned" by a square, so there might be several portal points in T that are co-located. For each portal $t \in T$ (where t is the portal of exactly one of s, s_1, s_2, s_3, or s_4) let $f(t)$ be the index of the facility that the tiny clients sent to t are assigned to; this index can be found in row t and column $(p+2)$ of D or D_1, D_2, D_3, or D_4 (whichever sub-problem the portal t belongs to). For each portal $t \in T$, let $f'(t)$ be the index of the closest facility to this portal inside the square of this portal, which can be found in column $(p+2)$ of F or F_1, F_2, F_3, or F_4 (again depending on which square owns portal t). For each portal $t \in T$, we check a shortest portal-respecting path from t to $f(t)$. We consider two cases.

Case 1: if $f(t)$ is outside s then let $t_s \in T$ be the the last portal of T on this shortest path from t to $f(t)$ (note that t_s must be a portal of s). Then we must have that $f(t_s) = f(t)$.

Case 2: if $f(t)$ is inside s and belongs to one of the 4 sub-squares of it, say square s_i, then let t_i be the last portal of T on this shortest path from t to $f(t)$ (note that t_i must belong to s_i). Then we must have that $f'(t_i) = f(t)$.

If either of these conditions are not satisfied we have a inconsistency of facilities of tiny clients and we skip the guessed matrices for the sub-problems.

3.5.4 Preprocessing Step

In this subsection we describe a perturbation step which enforces the assumptions we made about the size of the bounding box containing the points. Recall that we want: all the points are on a unit grid, the minimum inter-node distance is at least 4, and the maximum inter-node distance is at most $O(n^4)$. Observe that if we scale the connection and opening costs, then the optimum solution of the problem remains the same. Thus, we scale the costs by $4/c_{min}$, where c_{min} is the minimum non-zero inter-node distance. In the new instance with scaled costs, c_{min} is 4. From now on, we work with this instance. Let OPT be the value of an optimal solution. We first compute a crude approximation, A, of OPT. For instance, this can be done by using the $O(\log(n), 1+\epsilon)$-approximation algorithm of [BH09]. Therefore, we have $A = \text{OPT}O(\log n)$. We pick random a and b and construct a shifted quad-tree. This time, we stop as soon as the size of squares become less than nA. We claim that the shifted quad-tree does not cut any edge of an optimal solution with probability at least $1 - 4/n$. Hence, we can treat each leaf of this quad-tree as an independent instance where the length of the bounding box of each instance is at most nA.

To prove the claim, consider that the optimal solution consisting of a collection of line segments with length at least 4. By Lemma 4 of [Aro97], each line segment of the optimal solution with length s crosses at most $2s$ lines of unit grid. Therefore, at most $2OPT \leq 2A$ lines of gird are crossed by any segment in the solution. As a result, the probability that the shifted quad-tree cuts one of these segments is at most $2A$ over the length of the leaf squares which is at least $nA/2$. Hence, the probability is at most $4/n$.

Let L be the size of the bounding box of the points. We lay a grid of granularity $\epsilon A/(4n^2 \log(n))$. We move the input nodes one by one to its nearest grid point whose neighborhood of gird-distance 4 is empty of any other node. It is not hard to see that each original node can be assigned to a grid point of grid-distance at most $4n$. Therefore, we move each point by at most $4n\epsilon A/(4n^2 \log(n))$, and by the triangle equality, the cost of any solution increases by at most $n\epsilon A/(n \log(n)) \leq \epsilon OPT$. Since $L < nA$, the size of the bounding box after scaling is at most $2(4n) + L/(\epsilon A/(4n^2 \log(n))) < 8n + 4\frac{1}{\epsilon}n^3 \log(n) = O(n^4)$.

3.5.5 Wrap-up

Proof of Theorem 3.2.4. Recall that in the preprocessing step, the cost of the solution increases by a factor of at most $(1+\epsilon)$. When we restrict our solution to be portal-respecting, we lose another $1+\epsilon$ factor. In the dynamic programming, we find the best solution under these restrictions with respect to rounded demand values. When we restore the demands to their original values, we increase them by a factor of at most $1+\epsilon$, which increases the cost by at most another $1+\epsilon$ factor. As a result, we find a solution whose cost is at most $(1+\epsilon)^3$ factor away from the optimal value. As we stated before, the total blow-up in facility capacities incurred due to rounding of large and small clients is at most 2ϵ. In addition, the assignment of tiny clients may result in another ϵ blowup of capacity for each facility. Finally, when we round the fractional assignment of small clients using the Shmoys-Tardos algorithm, we may have another blowup of at most ϵ. Therefore, in total, we may have a violation of capacities by at most 4ϵ. This shows that the bicriteria factor of our algorithm is $((1+\epsilon)^3, 1+4\epsilon)$. By setting $\epsilon < \epsilon'/4$, we have the claimed performance, because $(1+\epsilon'/4)^3 \leq 1+\epsilon'$ for $0 < \epsilon' \leq 1$.

Now, we analyze the time complexity. The preprocessing step and construction of the shifted quad-tree can be done in time $O(n \log n)$ [Aro97]. Let T be the number of table entries for each square s. It can be seen that $T = O(n^{8(m+1)(q+2)})$. We compute the table entries corresponding to each square s in time $O(T^5)$ which is to enumerate all combinations of possible entries for s and its four children. Therefore, for any fixed $\epsilon > 0$ the overall running time of the dynamic programming algorithm is in $n^{O(m)} = n^{O(\log n/\epsilon)}$. ■

Remark 3.5.1 *We presented our algorithm and the proof for $\mathbb{R}^2$. One can generalize these to d dimensions where the performance ratio remains the same and the running time increases to $n^{O((\log n/\epsilon)^{d-1})}$ which is still quasi-polynomial time for constant $d > 0$.*

3.6 Simple Observations

Here, we present some simple observations.

Proposition 3.6.1 *Any α-approximation algorithm for the soft CFLP yields a 2α-approximation algorithm for the soft CFLP with unsplittable demands and vice verca (both in non-uniform and uniform demand case).*

Proof. First, we show that given a solution for the soft CFLP, we can change it to a solution with unsplittable demands while we at most double the cost. Let $(\mathbf{x}, \mathbf{y})$ be a solution for the soft CFLP. Here, x_{ij} shows the units of demand of clients j assigned to facility i in the solution, *i.e.*, $x_{ij} = \phi(i, j)$ and y_i shows the number of copies of facility i which is open in the solution. Clearly, we must have $\sum_{j \in C} x_{ij} \le y_i u_i$. Let t_i be the total units of demand served by copies of facility i, *i.e.*, $t_i = \sum_{j \in C} x_{ij}$.

We describe a charging scheme to determine the share of each client from the total cost. Later, we use these shares to bound the cost of an unsplittable solution we build from the solution $(\mathbf{x}, \mathbf{y})$. Each client pays for its own connection cost and it pays for the facility cost proportional to its usage. In other words, client i pays $\frac{x_{ij}}{t_i} f_i y_i$ for the opening cost of copies of facility i. Thus, the total share of client j is $\sum_i c_{ij} x_{ij} + \sum_i \frac{x_{ij}}{t_i} f_i y_i = \sum_i x_{ij}(c_{ij} + \frac{f_i y_i}{t_i})$.

Now, we construct an unsplittable solution from $(\mathbf{x}, \mathbf{y})$. Let F' be the set of all facilities such that $y_i > 0$. Assign the demand of client j to a facility i^* such that $i^* = \arg\min_i \{c_{ij} + \frac{f_i}{u_i}\}$. Denote this assignment function by ϕ_u and let t'_i be the total demand assigned to copies of facility i by ϕ_u. Then, the total cost of the unsplittable solution is:

$$\sum_j d_j c_{\phi_u(j)j} + \sum_{i \in F'} \left\lceil \frac{t'_i}{u_i} \right\rceil f_i \le \sum_j d_j c_{\phi_u(j)j} + \sum_i \frac{t'_i}{u_i} f_i + \sum_{i \in F'} f_i$$

where we used the simple facts that $\lceil \frac{t'_i}{u_i} \rceil \le \frac{t'_i}{u_i} + 1$ and $t'_i = 0$ for $i \notin F'$. Clearly, the value of the third term of the right hand side is at most the cost of $(\mathbf{x}, \mathbf{y})$. We only need to prove that the sum of the first two terms is at most the cost of $(\mathbf{x}, \mathbf{y})$:

$$\begin{aligned}
\sum_j d_j c_{\phi_u(j)j} + \sum_i \frac{t_i'}{u_i} f_i &\leq \sum_j d_j c_{\phi_u(j)j} + \sum_i \frac{\sum_{j:\phi_u(j)=i} d_j}{u_i} f_i \\
&\leq \sum_j d_j c_{\phi_u(j)j} + \sum_j d_j \frac{f_{\phi_u(j)}}{u_{\phi_u(j)}} \\
&= \sum_j d_j (c_{\phi_u(j)j} + \frac{f_{\phi_u(j)}}{u_{\phi_u(j)}}) \\
&= \sum_j (\sum_i x_{ij})(c_{\phi_u(j)j} + \frac{f_{\phi_u(j)}}{u_{\phi_u(j)}}) \\
&= \sum_j \sum_i x_{ij} (c_{\phi_u(j)j} + \frac{f_{\phi_u(j)}}{u_{\phi_u(j)}}) \\
&\leq \sum_j \sum_i x_{ij} (c_{ij} + \frac{f_i}{u_i}) \\
&\leq \sum_j \sum_i x_{ij} (c_{ij} + \frac{f_i}{t_i/y_i}),
\end{aligned}$$

where in the second line, we used the fact that each client assigned to exactly one facility, in the fourth line, we utilized $d_j = \sum_i x_{ij}$, in the sixth line, we used the facts that $\phi_u(j) = \arg\min_i\{c_{ij} + \frac{f_i}{u_i}\}$ and in the seventh line, we used $t_i/y_i \leq u_i$. As the final upper-bound is the summation of share of each client from cost of (x, y), we showed that the cost of our unsplittable solution is at most twice of cost of the original solution.

Finally, let opt be the optimum value of the soft CFLP and let opt_u be the optimum with additional constraint of unsplittable demands. Since any unsplittable solution is a feasible solution for the soft CFLP and because of the above construction, we have $\mathrm{opt} \leq \mathrm{opt}_u \leq 2\,\mathrm{opt}$.

Given any α-approximation algorithm for the soft CFLP, we run this algorithm and we transform its solution, which has cost at most $\alpha\,\mathrm{opt}$, to an unsplittable solution with cost at most $2\alpha\,\mathrm{opt} \leq 2\alpha\,\mathrm{opt}_u$. Conversely, given any α-approximation algorithm for the unsplittable soft CFLP, we run this algorithm and return its solution as our answer. The cost of its solution is at most $\alpha\,\mathrm{opt}_u \leq 2\alpha\,\mathrm{opt}$. ■

Proposition 3.6.2 *Unless **P= NP**, any approximation algorithm with bounded approximation ratio for the UCFLP violates the capacities of at least $\lfloor \frac{n}{2} \rfloor$ facilities.*

Proof. The proof is via a reduction from the partition problem. The partition problem is the problem of whether a given multiset of positive integers can be partitioned into two subsets having equal sum. Consider an instance of the partition problem $\mathcal{I}_p$ and let $a_1, \ldots, a_k$ be the integers in this instance, where k is the size of the multiset.

For an arbitrary n, we construct an instance of the UCFLP, denoted by $\mathcal{I}_U$, with n facilities. Construct a UCFLP instance with $\lfloor \frac{n}{2} \rfloor$ pair of facilities and if n is odd, an additional separate facility.

The capacity of each facility is $\frac{1}{2}\sum_{i=1}^{k} a_i$. Distance of two facilities is 0 if they are in the same pair and is 1 otherwise. Place k clients at distance 0 from each pair of facilities ($k\lfloor\frac{n}{2}\rfloor$ clients in total). Set the demand of ith client of each pair to be a_i.

If $\mathcal{I}_p$ is a yes instance, then there is a solution with cost 0 for $\mathcal{I}_U$. Thus, any algorithm with bounded approximation ratio must assign the clients at distance 0 of each pair of facilities to these facilities. Therefore, if an algorithm with bounded approximation ratio violates the capacities of less than $\lfloor\frac{n}{2}\rfloor$ facilities, it does not violate the capacities of at least a pair of facilities in $\mathcal{I}_U$. The assignment of clients at distance 0 from this pair to the facilities of this pair gives a valid partition of the integers. If $\mathcal{I}_p$ is a no instance, in any solution for $\mathcal{I}_U$, the capacity of at least one facility of each pair must be violated and we can recognize this case. As a result, we can solve the partition problem in polynomial time. Thus, Unless **P**= **NP**, we cannot have such an approximation algorithm. ■

Proposition 3.6.3 *There is a* $(5,2)$*-approximation algorithm for the uniform UCFLP and a factor* $(9,2)$ *algorithm for the non-uniform version.*

Proof. We first run the algorithm of [AAB+10] for the uniform splittable CFLP. This algorithm returns an assignment ϕ such that $c_f(\phi) \le 2c(\phi^*)$ and $c_s(\phi) \le c(\phi^*)$ where ϕ^* is an optimal assignment. Then, we change this splittable assignment to an unsplittable assignment ϕ' using GAP rounding algorithm (recall GAP reduction technique from Section 3.1.4). This rounded solution violates the facility capacities within factor 2 and only doubles the facility cost and keeps the connection cost intact. Therefore, we get $c(\phi') = c_s(\phi') + c_f(\phi') \le c(\phi^*) + 4c(\phi^*) \le 5c(\phi^*)$.

We do the same for the non-uniform version. The only difference is that we run the algorithm of [BGG12] for the non-uniform splittable CFLP. This algorithm returns an assignment ϕ such that $c_f(\phi) \le 4c(\phi^*)$ and $c_s(\phi) \le c(\phi^*)$ where ϕ^* is an optimal assignment. The rest of analysis is similar to the uniform case. ■

Proposition 3.6.4 *The non-metric UCFLP does not admit any* $(c\log n, \beta)$*-approximation algorithm for some constant* $c > 0$ *and any* $\beta \ge 1$ *unless* ***P****=* ***NP****.*

Proof. There is a simple factor-preserving reduction from the Set Cover problem to the non-metric UCFLP (sets are facilities, elements are clients and costs are 0 or ∞ based on membership relations). Since we cannot approximate set cover problem within factor $c\log n$ for some constant $c > 0$ unless **P**= **NP** [AMS06], we have the above hardness result for non-metric UCFL problem. ■

3.7 Towards solving RUCFLP(ϵ)

Recall that an $(f(\epsilon), 1+\epsilon)$-approximation algorithm for the RUCFLP(ϵ) yields an $(O(1), 1+\epsilon)$-approximation for the UCFLP for a fixed $\epsilon > 0$. In this section, we try to simplify RUCFLP(ϵ) with the hope of getting such factor $(f(\epsilon), 1+\epsilon)$ algorithm.

There are two simplifications that we can make. First, note that by Lemma 3.4.1, we can assume that the facility costs are zero. The reason is that an $(f'(\epsilon), 1+\epsilon)$-approximation algorithm for the RUCFLP(ϵ) with zero facility opening costs yields a $(f(\epsilon), 1+\epsilon)$-approximation algorithm for the general RUCFLP(ϵ), where $f(\epsilon) = f'(\epsilon)\frac{1}{\epsilon}$. Second, as we show below, any instance of RUCFLP(ϵ), say I, can be changed to a new instance, say I', in which all demand values are a multiple of ϵ^2 between ϵ and 1 (this is the same as what we did in Section 3.5). Any $(f(\epsilon'), 1+\epsilon')$-approximate solution for instance I' is an $(f(\epsilon')(1+\epsilon'), 1+3\epsilon')$-approximate solution for instance I for all $\epsilon \leq 1$. Therefore, by choosing the $\epsilon' = \epsilon/3$, we can have a $(f(\epsilon), 1+\epsilon)$-approximation algorithm for the general RUCFLP(ϵ).

To do the second simplification, we simply round down the demand values to the closest multiple of ϵ^2 to get the new instance I'. We have the following simple observation:

Observation 3.7.1 *An $(f(\epsilon), 1+\epsilon)$-approximate solution for instance I' is an $(f(\epsilon)(1+\epsilon), 1+3\epsilon)$-approximate solution for instance I for all $\epsilon \leq 1$.*

Proof. Note that the cost of optimal solution for instance I' is at most the cost of optimum solution for instance I, because any feasible solution for I is also a feasible solution for I'. Therefore, a solution having cost within factor α of the optimum of I' is within factor α of the optimum of I as well.

First, notice that the demand of a client in I', say j, increases in I by a factor of at most $\frac{d_j+\epsilon^2}{d_j} = 1 + \frac{\epsilon^2}{d_j} \leq 1+\epsilon$, because $d_j \geq \epsilon$. Therefore, the connection cost of the solution increases by a factor of $1+\epsilon$ in I'. Second, since each demand value increases by a factor of at most $1+\epsilon$, the total demand assigned to each facility in I increases by at most a factor $(1+\epsilon)$, which increases the facility cost by at most a factor of $1+\epsilon$. In addition, the total demand assigned to each facility is at most $(1+\epsilon)^2 \leq 1+3\epsilon$ for $\epsilon \leq 1$. Therefore, the solution is an $(f(\epsilon)(1+\epsilon), 1+3\epsilon)$-approximate solution for I. ■

In summary, if we find an $(f(\epsilon), 1+\epsilon)$-approximation algorithm for instances with the following restrictions, we get our desired approximation algorithm for the RUCFLP(ϵ):

1. All facility costs are zero.
2. All capacities are 1.
3. All demand values are a multiple of ϵ^2 between ϵ and 1.

We call a problem with these restricted instances the *simplified RUCFLP(ϵ)*. The third restriction has an important implication: for any fixed $0 < \epsilon < 1$, the number of distinct demand values, q, is a constant, which is at most $(1-\epsilon)/\epsilon^2 + 1 \leq p^2 - p + 1$ where $p = \lceil \frac{1}{\epsilon} \rceil$. Having a constant number of distinct demand values can be helpful in guessing some structures in a good solution. For instance, we can decide if the instance has a feasible solution in polynomial time. This is shown in the following discussion and the idea is similar to the idea used in the first asymptotic PTAS algorithm for the bin packing problem [dlVL81].

Let $d_1 \leq d_2 \leq \cdots \leq d_q$ be the q different demand values. Define q *client types* corresponding to the q distinct demand values. In other words, a client with demand d_l is of client type l. Let C_l be the set of type l clients. Similarly, we define different types of facilities based on how many of each client type they serve. A *facility type* is a vector with q elements where lth element shows how many type l clients can be served by this type of facility. In other words, in a facility type $\mathbf{t} = [t_1, t_2, \ldots, t_q]$, t_l shows the number of type l clients and we have $\sum_l d_l t_l \leq 1$. Since each facility can serve at most p clients, we must have $t_1 + t_2 + \cdots + t_l \leq p$. There are at most $\binom{p+q}{p}$ solutions for this inequality which shows the number of distinct facility types, denoted by r, is at most $\binom{p+q}{p}$. Note that for a fixed ϵ, r is a constant. Finally, we can characterize any solution based on the type of its facilities, *i.e.*, how many of each facility type is present in the solution. A *facility configuration* is a vector with r elements where the kth element shows how many type k facilities are present in a solution. Since we have at most $O(m^r)$ different configurations (recall that m denotes the number of facilities), we can enumerate all of them in polynomial time.

By utilizing the observation above, we can decide if an instance has a feasible solution in polynomial time as follows. For each facility configuration, we count the total number of type l clients present in it. If for all client types, this number is equal to the number of type l clients present in the instance, then there is feasible solution with this facility configuration. Given a feasible configuration, the difficult task is to find a feasible solution with minimum cost among all the solutions having this configuration.

Fix a configuration and let $\mathbf{t_1}, \mathbf{t_2}, \ldots, \mathbf{t_m}$ be the m facility types in this configuration (not necessarily distinct), where $\mathbf{t_k} = [t_{k1}, t_{k2}, \ldots, t_{kq}]$ and t_{kl} shows that number of type l clients in the kth facility type. If we can get a constant factor approximation algorithm for finding the minimum cost solution that has this configuration, we can get a constant factor approximation for the RUCFLP(ϵ). We only need to run this approximation algorithm on each of the polynomially many different configurations and return the best solution among the solutions we found for each configuration. We call this problem the *configuration realization problem.*

In the followings, we consider the viable options to attack the simplified RUCFLP(ϵ) and configuration realization problem: the local search techniques that are extensively used for the splittable CFLP problem and LP-based techniques that are widely used for the UFLP.

3.7.1 Local search attempts

Each solution for the configuration realization problem can be decomposed into two parts: assignment of clients to facility types and assignment of facility types to facilities. In the former part, we want to know, out of all the clients of type l, which t_{kl} of them must be assigned to the facility type $\mathbf{t_k}$. In the latter part, we want to know that the facility type $\mathbf{t_k}$ must be assigned to which facility. It is interesting to know that knowing the answer to one part, we can determine the other one optimally:

Observation 3.7.2 *We can solve the configuration realization problem in polynomial time in the following cases:*

- *We know the assignment of facility types to facilities.*
- *We know the assignment of clients to facility types.*

Proof. In the first case, we can solve the problem by running a separate min-cost max-flow algorithm for each client type. Consider type l clients. Since we know the assignment of facility types to facilities, for each facility, we know exactly how many type l clients must be assigned to this facility. As a result, we can build a flow network in the following standard way. Create a source vertex s and a sink vertex t. Connect the source vertex to type l clients with edges of cost 0 and capacity 1. Connect type l client j to facility i with an edge of capacity 1 and cost c_{ij} for all $j \in C_l$ and $i \in F$. Connect facility i to the sink vertex t with an edge of capacity t_{il} and cost 0 for all $i \in F$, where t_{il} is the number type l clients that must be served by facility i. By a min-cost max-flow algorithm on this flow network, we can find the best assignment of type l clients to the facilities. We do this for all client types.

In the second case, we know the assignment of clients to facility types. Therefore, we know the exact cost of assigning each facility type to each facility. To do this assignment with minimum total cost, we can solve the problem by running a min-weight perfect matching algorithm, where the weight of an edge between a facility type and a facility is the cost of assigning this facility type to the facility. ■

The above observation indicates that knowing the type of each facility is enough to solve the problem. This shows that the local operations that modify this assignment may improve the solution. In addition, knowing which clients must be assigned to each facility type is also enough to solve the problem. This leads to the following natural local operations:

1. Swap the facility type of two facilities and then find the optimal assignment of clients to the facilities.
2. For the current assignment of facility types to facilities, find the best assignment of clients to facility types.

3. For the current assignment of the clients to the facility types, find the best assignment of facility types to facilities.

Unfortunately, a local search algorithm based on the above operations has a bad locality gap. This example is shown in Figure 3.5. In this figure, solid squares and circles are the facilities and clients, respectively. There are $2n$ facilities and $2n$ clients. Assume that we only have one client type. In the configuration that we want to realize, there are n facility types containing two clients and n facility types containing zero clients.

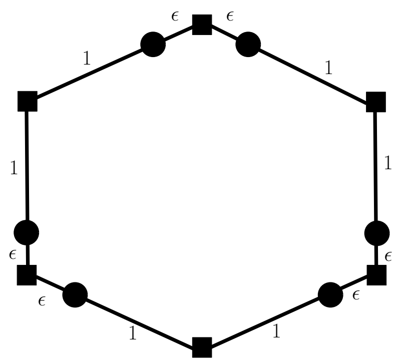


Figure 3.5: The bad example for the proposed local operations when $n = 3$. Solid squares show the facilities and solid circles show the clients. For $n > 3$, the structure of bad examples are similar to the above example.

In the local solution, all facilities incident to a unit length edge are chosen to serve two clients (assigned the facility type containing two clients) and in the optimal solution, all facilities incident to ϵ length edges are assigned the facility type containing two clients. We set $\epsilon = 1/n$. Now , the value of local solution is $2n$ while the value of global optimum is $2n\epsilon$, which gives a gap of $1/\epsilon = n$. This is a local optimum solution for the first operation, because with any single swap the cost of solution becomes at least $2\epsilon + (2 + 2\epsilon)(n - 1) = 2n\epsilon + 2(n - 1) = 2n$. This is a local optimum solution for the second operation, because for the current assignment of facility types to facilities, the optimal assignment of clients is the same as the local solution. The reason is that in the local solution, each client is assigned to the closest facility that can serve anything and one cannot pay less than this for any client. Finally, this is a local optimum solution for the third operation as well. In the current solution, the two clients which are assigned to facilities incident to a unit length edge, are grouped together and assigned to a facility type containing two clients. Based on this grouping, the best assignment of groups to facilities, is the same as the one in the local solution. The reason is that the distance of each group from a facility is at least two and in the local solution, each group assigned to a facility at total distance two. Thus, we cannot have a better assignment.

The reader may wonder what happens if we use the p-swap local search operation that used for the k-median problem. Unfortunately, by setting $\epsilon = \frac{p}{n}$ in the example of Figure 3.5, we get an unbounded locality gap of n/p. In the local solution, all facilities incident to a unit length edge are chosen to serve two clients (assigned the facility type containing two clients) and in the optimal solution, all facilities incident to ϵ length edges are assigned the facility type containing two clients. The value of local solution is $2n$ while the value of global optimum is $2n\epsilon$, which gives a gap of $1/\epsilon = n/p$. This is a local optimum solution, because with any p-swap the cost of solution becomes at least $2p\epsilon + (2+2\epsilon)(n-p) = 2n\epsilon + 2(n-p) = 2n$.

3.7.2 LP-based attempts

We can try to solve the configuration realization problem with an LP-based approach. We have the following LP relaxation:

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{i\in F}\sum_{j\in C}\sum_{1\le k\le m} c_{ij}x_{ijk} & \\
\text{subject to} & \displaystyle\sum_{1\le k\le m} y_{ik} = 1 & \forall i\in F \\
& \displaystyle\sum_{i\in F} y_{ik} = 1 & \forall 1\le k\le m \\
& \displaystyle\sum_{i\in F}\sum_{1\le k\le m} x_{ijk} = 1 & \forall j\in C \\
& x_{ijk} \le y_{ik} & \forall i\in F, j\in C, 1\le k\le m \\
& \displaystyle\sum_{j\in C_l} x_{ijk} = t_{kl}y_{ik}, & \forall i\in F, 1\le k\le m, 1\le l\le q
\end{array}
$$

where $y_{ik}=1$ shows that the facility type $\mathbf{t_k}$ is assigned to facility i and $x_{ijk}=1$ shows that client j is assigned to facility i, which has type $\mathbf{t_k}$.

Unfortunately, this LP has an unbounded integrality gap. Figure 3.6 shows an instance with an unbounded gap. In this instance, there are $2s$ facilities and $2s$ clients where s is an odd number. All clients have the same type. The solid squares show the facilities and the solid circles show the clients. The facilities and clients in the ovals are all at distance zero from each other. The distance between any facility or client from one oval to any facility or client from the other oval is 1.

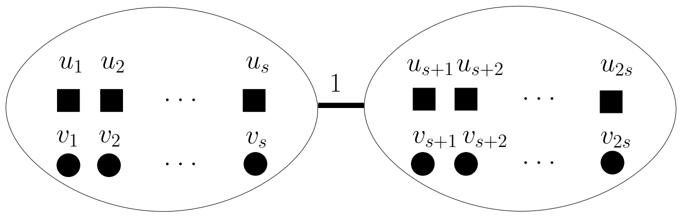


Figure 3.6: An instance with unbounded integrality gap for the proposed LP relaxation.

The configuration that we want to realize has s facility types that each contains two clients and s facility types that each contains zero clients. More precisely, let $\mathbf{t_1} = \mathbf{t_2} = \cdots = \mathbf{t_s} = [2]$ and

$\mathbf{t_{s+1}} = \mathbf{t_{s+2}} = \cdots = \mathbf{t_{2s}} = [0]$. Since s is odd, we have $\mathrm{opt} = 1$, but there is a fractional solution with value 0, which yields an unbounded integrality gap. This fractional solution is as follows. Set y_{ii}, $y_{i,s+i}$, $y_{s+i,i}$, and $y_{s+i,s+i}$ equal to $1/2$ for all $1 \le i \le s$ and set all other y-values to zero. One can check that these y-values satisfy the first and second set of constraints in the above LP. Also, set x_{iii} and $x_{i+1,i,i+1}$ to $1/2$ for all $1 \le i \le s$ where $i+1$ is module s to stay between 1 and s (*i.e.*, $x_{1,s,1} = 1/2$). Set all other x-values of the the first s clients to zero. It is easy to check that these x values for the first s clients satisfy the third, fourth, and fifth set of constraints. Similarly, set $x_{s+i,s+i,i}$ and $x_{s+i+1,s+i,i+1}$ to $1/2$ for all $1 \le i \le s$ where $i+1$ is module s to stay between 1 and s (*i.e.*, $x_{s+1,2s,1} = 1/2$).

Chapter 4

Conclusion

In this chapter, we present the final thoughts on the subjects of last two chapters.

4.1 The MSR and MSD problems

We presented an exact algorithm for the unweighted MSR problem when no singleton cluster is permitted. To get the exact algorithm, we carefully used the structure of a class of optimal solutions that we called canonical optimal solutions. These optimal solutions have the minimum number of possible balls in an optimal solution.

We proved that in a canonical optimal solution, each ball has at most two non-zero adjacent balls and as a result, in the absence of zero balls, each connected component of the input graph is covered by a path or cycle of balls in a canonical optimal solution. By using this structural property, we designed an exact dynamic programming algorithm that runs in polynomial time. In the presence of zero balls, the balls of a canonical optimal solution do not necessarily form a path or cycle. As a consequence, we cannot bound the size of dynamic programming table to a polynomial of input size. It should be noted that we still can infer some interesting structural properties, but we do not know how useful these properties are in designing an algorithm.

The difficult core of the MSR problem is to find the location of zero balls. If we can find these locations even approximately, our exact algorithm solves the rest of the problem and at least, we can hope for a good approximation algorithm for this case. The current best ratio even for unweighted graphs is 3.504, which is the same for the general metrics. To solve the difficulty with zero balls, we tried the widely used LP-based and local search techniques. Unfortunately, as we showed, the natural ways to utilize these techniques were not promising. It seems that to overcome this difficulty, one needs to introduce substantially new techniques.

We tried to obtain a similar result, *i.e.*, an exact algorithm, for the MSD problem when no singleton is allowed. Unfortunately, in this problem, the strongest property that we can prove is that each ball has at most three non-zero balls in a canonical optimal solution. We could not use this property to bound the size of dynamic programming table as we did for the MSR problem without singletons.

We also presented a PTAS for the Euclidean MSD problem that partially answers the open question asked by Doddi *et al.* [DMR$^+$00a]. They asked about the complexity of the Euclidean MSD problem. Before our algorithm, the best approximation ratio for this problem was 2 and it was not clear if one can beat this approximation factor. Our result shows that this problem is not **APX**-hard, where it remains open whether this problem is **NP**-hard.

Finally, we settled another open question of Doddi *et al.* [DMR$^+$00a] regarding the complexity of the MSR problem when k is constant. We presented a polynomial time exact algorithm for this case, while the best previous result was a 2-approximation algorithm.

4.2 The UCFLP

We presented a reduction from the UCFLP to a restricted version in which all the demand values are large (*i.e.*, are larger than ϵ) and presented two algorithms for the cases of $\epsilon = \frac{1}{2}$ and $\epsilon = \frac{1}{3}$. These implied two constant factor approximation algorithms for the UCFLP with capacity bounds within factor $3/2$ and $4/3$. These are the first constant factor approximation algorithms with capacity violation within a factor smaller than 2.

We believe similar results can be found with capacity violations bounded within factor $1 + \epsilon$ for any $\epsilon > 0$. Recall that an $(f(\epsilon), 1 + \epsilon)$-approximation algorithm for the RUCFLP(ϵ) yields an $(O(1), 1 + \epsilon)$-approximation for the UCFLP for a fixed $\epsilon > 0$. We simplified RUCFLP(ϵ) with the hope of getting such factor $(f(\epsilon), 1 + \epsilon)$ algorithm. In particular, we showed that if we find an $(f(\epsilon), 1 + \epsilon)$-approximation algorithm for instances having zero facility costs, unit capacities, and demands values which are a multiple of ϵ^2 between ϵ and 1, we get our desired approximation algorithm for the RUCFLP(ϵ). The third restriction has an important implication: for any fixed $0 < \epsilon < 1$, the number of distinct demand values is a constant. As a result, one can guess the facility configuration in an optimal solution by enumerating all of the configurations in polynomial time. Given the facility configuration of an optimal solution, we tried LP-based and local search techniques to find a feasible solution with minimum cost among all the solutions having this configuration, but our attempts were not promising.

Perhaps, the idea of fixing a configuration first and then trying to find the best solution among all the solutions having this fixed configuration is not a good approach to tackle the problem and we should let the configuration and a solution realizing this configuration be selected simultaneously. For example, consider the instances in which the demand of all clients is d. In this case, each facility can serve at most $\lfloor \frac{1}{d} \rfloor$ clients and we can solve the problem by building a standard flow network and using a min-cost max-flow algorithm. In contrast, when we fix a facility type for this case (which just indicates how many facilities serve zero clients, how may serve one client, etc.), we do not know how to solve the problem.

For the case of Euclidean metrics, we presented a $(1 + \epsilon, 1 + \epsilon)$-approximation algorithm that runs in quasi-polynomial time. One might ask if the stronger structural theorem of Kolliopoulos and Rao [KR99] for the standard facility location problem which only needs $O(1/\epsilon)$ portals instead of $O(\log n/\epsilon)$ portals for each square could be used to improve the running time of our algorithm from quasi-polynomial to a true polynomial. Recall that the running time of our algorithm is $n^{O(p)}$, where p is the number of portals and hence, by decreasing the number of portals to $O(1/\epsilon)$, we get a polynomial time algorithm. The difficulty is that Kolliopoulos and Rao proved their theorem for the uncapacitated facility location problem and in the proof of their structural theorem, they critically used the fact that we can assign a client to any open facility in the solution of the uncapacitated facility location problem. Unfortunately, in the UCFLP, because of the capacities, we cannot freely change the assignment of a client from one facility to another facility.

4.3 Future Directions

There are still several interesting questions left open related to the considered problems:

- Perhaps the most interesting question is to obtain a PTAS for general MSR. The existence of a QPTAS for this problem [GKK$^+$12] is strong evidence that there could be a PTAS.

- Given a fixed set of centers, we gave a 2-approximation algorithm for the problem of assigning a radius to each center to minimize the total radii. It is an interesting question to find an algorithm with an improved ratio for this case.

- For MSD on Euclidean metrics, the complexity of the problem is open. To settle this questions, one needs to either give an **NP**-hardness proof or find an exact algorithm for this problem. We suspect that unlike MSR (which has an exact algorithm on Euclidean metrics), the Euclidean MSD problem is **NP**-hard.

- We presented an exact algorithm for the MSD problem with constant k, but the running time is too high even for small k. It will be useful if someone finds a more efficient algorithm for this case.

- In the UCFLP, the most interesting questions is to either find a factor $(O(1), 1+\epsilon)$ or even a factor $(f(\epsilon), 1+\epsilon)$ approximation algorithm or show that for some ϵ, the problem is not approximable within factor $(O(1), 1+\epsilon)$.

- In the Euclidean UCFLP, the main question is to find a PTAS for it. It is an interesting question whether it is possible to find a similar structural theorem to [KR99] for the UCFLP with $O(1/\epsilon)$ portal points for each square; that would imply a PTAS for the Euclidean UCFLP. As a first step in this direction, one may try to find a PTAS for the splittable CFLP, because even for this simpler problem, the best known is a QPTAS by Arora *et al.* [ARR98].

Bibliography

[AAB$^+$10] Ankit Aggarwal, L Anand, Manisha Bansal, Naveen Garg, Neelima Gupta, Shubham Gupta, and Surabhi Jain. A 3-Approximation for Facility Location with Uniform Capacities. In *Integer Programming and Combinatorial Optimization*, volume 6080 of *Lecture Notes in Computer Science*, pages 149–162. Springer-Verlag, 2010.

[AGK$^+$01] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristic for k-median and facility location problems. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 21–29, New York, NY, USA, 2001. ACM.

[ALR$^+$08] Hussein A Alzoubi, Seungjoon Lee, Michael Rabinovich, Oliver Spatscheck, and Jacobus der Merwe. Anycast CDNS revisited. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 277–286, New York, NY, USA, 2008. ACM.

[AMS06] Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k-restrictions. *ACM Trans. Algorithms*, 2(2):153–177, 2006.

[Aro96] S Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 2–12, Washington, DC, USA, 1996. IEEE Computer Society.

[Aro97] S Arora. Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 554–563, Washington, DC, USA, 1997. IEEE Computer Society.

[ARR98] Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for Euclidean k-medians and related problems. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 106–113, New York, NY, USA, 1998. ACM.

[AS11] Sara Ahmadian and Chaitanya Swamy. Improved Approximation Guarantees for Lower-Bounded Facility Location. Technical report, University of Waterloo, 2011.

[Ass83] Patrice; Assouad. Plongements lipschitziens dans $\mathbb{R}^n$. *Société mathématique de France*, 111(4):429–448, 1983.

[Bar98] Yair Bartal. On approximating arbitrary metrices by tree metrics. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 161–168, New York, NY, USA, 1998. ACM.

[BGG12] Manisha Bansal, Naveen Garg, and Neelimam Gupta. A 5-approximation for capacitated facility location. In *20th European Symposium on Algorithms*, pages 133–144, 2012.

[BH09] M H Bateni and M T Hajiaghayi. Assignment problem in content distribution networks: unsplittable hard-capacitated facility location. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 805–814, 2009.

[Byr07] Jaroslaw Byrka. An Optimal Bifactor Approximation Algorithm for the Metric Uncapacitated Facility Location Problem. In *APPROX '07/RANDOM '07: Proceedings of the 10th International Workshop on Approximation and the 11th International Workshop on Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 29–43, Berlin, Heidelberg, 2007. Springer-Verlag.

[CG99] Moses Charikar and Sudipto Guha. Improved Combinatorial Algorithms for the Facility Location and k-Median Problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 378–388, Washington, DC, USA, 1999. IEEE Computer Society.

[Chu98] Fabián Chudak. Improved Approximation Algorithms for Uncapacitated Facility Location. In *Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 180–194. Springer Berlin / Heidelberg, 1998.

[CP01] M Charikar and R Panigrahy. Clustering to Minimize the Sum of Cluster Diameters. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 1–10, 2001.

[CP04] Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. *Journal of Computer and System Sciences*, 68(2):417–441, 2004.

[CRW91] Vasilis Capoyleas, Gunter Rote, and Gerhard Woeginger. Geometric Clusterings. *Journal of Algorithms*, 12(2):341–356, 1991.

[CS99] Fabián A Chudak and David B Shmoys. Improved approximation algorithms for a capacitated facility location problem. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 875–876, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[CW99] Fabián A Chudak and David P Williamson. Improved Approximation Algorithms for Capacitated Facility Location Problems. In *Proceedings of the 7th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pages 99–113, London, UK, 1999. Springer-Verlag.

[DH04] Zvi Drezner and Horst W Hamacher. *Facility Location: Applications and Theory*. Springer, 2004.

[dlVL81] W de la Vega and G S Lueker. Bin Packing can be solved within 1+ϵ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[DMR$^+$00a] Srinivas Doddi, Madhav V Marathe, S S Ravi, David Scot Taylor, and Peter Widmayer. Approximation algorithms for clustering to minimize the sum of diameters. *Nordic J. of Computing*, 7(3):185–203, 2000.

[DMR$^+$00b] Srinivas Doddi, Madhav V Marathe, S S Ravi, David Scot Taylor, and Peter Widmayer. Approximation algorithms for clustering to minimize the sum of diameters. In *SWAT '00: Proceedings of the 3rd Scandinavian workshop on Algorithm Theory*, pages 237–250, 2000.

[ELLS11] Brian S. Everitt, Sabine Landau, Morven Leese, and Daniel Stahl. *Cluster Analysis*. Wiley, 5 edition, 2011.

[Fei98] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[FRT03] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 448–455, New York, NY, USA, 2003. ACM.

[GJ79] 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.

[GK98] Sudipto Guha and Samir Khuller. Greedy strikes back: improved facility location algorithms. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 649–657, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[GKK$^+$08a] Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A Pirwani, and Kasturi Varadarajan. On clustering to minimize the sum of radii. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 819–825, 2008.

[GKK$^+$08b] Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A Pirwani, and Kasturi Varadarajan. On Metric Clustering to Minimize the Sum of Radii. In *SWAT '08: Proceedings of the 11th Scandinavian workshop on Algorithm Theory*, pages 282–293, Berlin, Heidelberg, 2008. Springer-Verlag.

[GKK$^+$10] Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A Pirwani, and Kasturi Varadarajan. On Metric Clustering to Minimize the Sum of Radii. *Algorithmica*, 2010.

[GKK$^+$12] Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A Pirwani, and Kasturi Varadarajan. On Clustering to Minimize the Sum of Radii. *SIAM Journal on Computing*, 41(1):47–60, 2012.

[Gon85] T Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[HJ87] P Hansen and B Jaumard. Minimum sum of diameters clustering. *Journal of Classification*, 4(2):215–226, 1987.

[HS85] 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.

[JMS02] Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 731–740, New York, NY, USA, 2002. ACM.

[Jun01] Heinrich W E Jung. {Ü}ber die kleinste Kugel, die eine r{ä}umliche Figur einschliesst. *J. reine angew. Math.*, 123:241–257, 1901.

[JV99] Kamal Jain and Vijay V Vazirani. Primal-Dual Approximation Algorithms for Metric Facility Location and k-Median Problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 2–13, Washington, DC, USA, 1999. IEEE Computer Society.

[KK82] Narendra Karmarkar and Richard M Karp. An Efficient Approximation Scheme for the One-Dimensional Bin-Packing Problem. In *IEEE Symposium on Foundations of Computer Science*, pages 312–320, 1982.

[KPR98] Madhukar R Korupolu, C Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, SODA '98, pages 1–10, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[KR99] Stavros G Kolliopoulos and Satish Rao. A Nearly Linear-Time Approximation Scheme for the Euclidean kappa-median Problem. In *Proceedings of the 7th Annual European Symposium on Algorithms*, ESA '99, pages 378–389, London, UK, 1999. Springer-Verlag.

[Li11] Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. In *Proceedings of the 38th international conference on Automata, languages and programming - Volume Part II*, ICALP'11, pages 77–88, Berlin, Heidelberg, 2011. Springer-Verlag.

[LMW88] Robert F Love, James G Morris, and George O Wesolowsky. *Facilities location: Facilities location: models and methods*. North-Holland, 1988.

[LSS04] Retsef Levi, David Shmoys, and Chaitanya Swamy. LP-based Approximation Algorithms for Capacitated Facility Location. In Daniel Bienstock and George Nemhauser, editors, *Integer Programming and Combinatorial Optimization*, volume 3064 of *Lecture Notes in Computer Science*, pages 21–27. Springer Berlin / Heidelberg, 2004.

[LST90] J K Lenstra, D B Shmoys, and E Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:271–459, 1990.

[LV92] Jyh-Han Lin and Jeffrey Scott Vitter. e-approximations with minimum packing constraint violation (extended abstract). In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, STOC '92, pages 771–782, New York, NY, USA, 1992. ACM.

[MP03] Mohammad Mahdian and Martin Pál. Universal Facility Location. In Giuseppe Di Battista and Uri Zwick, editors, *Algorithms - ESA 2003*, volume 2832 of *Lecture Notes in Computer Science*, pages 409–421. Springer Berlin / Heidelberg, 2003.

[MRS08] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[MYZ02] Mohammad Mahdian, Yinyu Ye, and Jiawei Zhang. Improved Approximation Algorithms for Metric Facility Location Problems. In *APPROX*, pages 229–242, 2002.

[MYZ03] Mohammad Mahdian, Yingyu Ye, and Jiawei Zhang. A 2-Approximation Algorithm for the Soft-Capacitated Facility Location Problem. In *RANDOM-APPROX*, pages 129–140, 2003.

[PTW01] M Pál, É Tardos, and T Wexler. Facility Location with Nonuniform Hard Capacities. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, FOCS '01, pages 329–338, Washington, DC, USA, 2001. IEEE Computer Society.

[ST93] D B Shmoys and E Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(3):461–474, 1993.

[STA97] D B Shmoys, E Tardos, and K Aardal. Approximation algorithms for facility location problems. In *Proceedings of the twenty-ninth annual ACM symposium on theory of computing*, pages 265–274, 1997.

[Svi10] Zoya Svitkina. Lower-bounded facility location. *ACM Trans. Algorithms*, 6(4):69:1—-69:16, September 2010.

[Ver11] Vedat Verter. *Foundations of Location Analysis*, chapter 2. International Series in Operations Research and Management Science. Springer, 2011.

[Wes00] Douglas B West. *Introduction to Graph Theory*. Prentice Hall, 2nd edition, August 2000.

[ZCY04] Jiawei Zhang, Bo Chen, and Yinyu Ye. A Multi-exchange Local Search Algorithm for the Capacitated Facility Location Problem. In *Integer Programming and Combinatorial Optimization*, pages 1–4. Springer Berlin / Heidelberg, 2004.