APPROXIMATION ALGORITHMS FOR FACILITY LOCATION PROBLEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Sudipto Guha
August 2000

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Rajeev Motwani
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Dan Boneh

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Serge Plotkin

Approved for the University Committee on Graduate Studies:

Abstract

The facility location problem is one of the most well–studied problems in optimization literature over the last four decades. Given a set of client locations, the problem seeks to find an optimal set of locations to build facilities (warehouses) minimizing the sum of the cost of building the facilities along with the cost of assigning clients (stores) to facilities. The assignment cost is often modeled by a weight times the metric distance. The facility location problem and its variant the k-median problem, which minimizes the assignment cost allowing at most k facilities, have been used in a variety of applications, for example network design, web-caching, and clustering. These problems are NP-hard, and have been studied from perspectives of empirical investigation of different heuristics, average-case analysis, polyhedral characterization and approximate heuristics with worst-case performance guarantees, among others.

These problems showcase some of the most interesting techniques in approximation algorithms, including local search, linear programming rounding, and primal-dual algorithms. We will present the results according to the various techniques.

The highlights of the results include:

- The k-Median Problem: We prove the first constant factor approximation algorithm for the k-median problem. We show that a natural linear programming formulation for this problem has an integrality gap $6\frac{2}{3}$. Subsequently using primal-dual algorithms, we show that the k-median problem can be approximated within factor 4. This is the best known upper bound. We also prove a lower bound of factor 1.36.

- The Facility Location Problem: We present a local search algorithm which along

with previous known linear programming relaxation based algorithms gives the best known constant approximation of 1.728. We obtain a 1.47 lower bound for this problem. We show the first constant factor approximation for fault-tolerant facility location, hierarchical placement and other variants. We provide the first combinatorial approximation for multi-level facility location.

- Single-Sink Edge-Installation: We use the variants of the facility location problem to provide the first constant factor approximation algorithm for the single-sink edge-installation problem. This problem models a distribution network of utility companies and service providers. The problem minimizes the cost of designing a network which routes all the demand to a special sink node, where the cost per unit length of any edge obeys economies of scale and is a concave function of traffic through the edge.

- The k-Median Problem on Data Streams: Clustering large datasets, such as click streams, web pages, multimedia data, retail transactions, and telecommunication records, necessitate scalable algorithms that revisit data sparingly, and consume small space. We give a constant factor approximation for k-Median problem with $O(n^{\epsilon})$ space, in the data streams model, showing the feasibility of divide-and-conquer approaches.

- Robust Hierarchical Agglomerative Clustering: We use some of the approximation techniques along with sampling and divide-and-conquer approach to develop a robust hierarchical agglomerative clustering algorithm.

Acknowledgements

I would like to thank Samir Khuller and the University of Maryland, College Park. I would not have pursued graduate studies without their intervention. Samir has been a great support to me, as an advisor and sometimes a friend after leaving College Park, and initiated me to the problem which led to my dissertation, I cannot thank him enough.

This thesis would not have been realized without the encouragement, guidance and support of my advisor Rajeev Motwani. He patiently bore all my wildest ideas, and nudged me gently into more fruitful directions, helping me recognize and focus on the more important questions. I would like to thank Baruch Schieber for being my mentor over a few summers and more. I have had a great time visiting IBM T. J. Watson Research Center over these years, and I looked forward to the summers. The graduate fellowships of IBM Research over the last two years, and Stanford University in my initial year helped my research greatly.

I would like to thank Serge Plotkin, for all the informal drop in discussions and encouragement over the years, and for going over this thesis. I have learned immensely from my discussions with Amotz Barnoy, and Seffi Naor in my visits to the east coast, not just related to research. I am grateful to Dan Boneh, Sanjeev Khanna, David Shmoys, Éva Tardos, Vijay Vazirani, and David Williamson for their advice, collaboration and help over different aspects of my research. I would like to thank Guy Even and Nina Mishra for bearing with me over projects that have led to some very fun research. Rajeev Rastogi and Kyuseok Shim hosted me at Bell Laboratories for a summer, it made me familiar with several issues related to practical applications, which has been very helpful even in choosing theoretical problems in my research.

I have been fortunate in having several wonderful colleagues at Stanford. A lot of the ideas in this dissertation were developed with Moses Charikar, Adam Meyerson, Kamesh Munagala, and Liadan O'callahan. The Theory Department at Stanford has been a great source of fun and sustenance, I would miss Chandra Chekuri, Glen Durfee, Ashish Goel, Jeremy Horwitz, Piotr Indyk, and Suresh Venkat. I am grateful to my friends Samir Datta, Anu Ganguly, Namit Jain, Ashish Tiwari, Ashwin Nayak, to name a few, for being supportive and bearing with my random projects over these years. Any such list is hardly exhaustive, I have definitely missed some and I would like to express my gratitude to them, hoping my omission would be overlooked.

I would like to acknowledge the contribution of my parents, for encouraging curiosity, for instruction through fun and games, and my brother for teaching me to formulate an equation, for being the big brother whenever needed. Words would fall well short; I am grateful for their love and patience, making me believe in sunshine.

Contents

Abstract **v**

Acknowledgements **vii**

1 Introduction **1**

1.1 Preliminary Definitions 5

1.2 Overview of Results 6

1.2.1 The k-Median Problem 6

1.2.2 The Facility Location Problem 7

1.2.3 The Single-Sink Edge-Installation Problem 8

1.2.4 Clustering Data Streams 9

1.2.5 Robust Hierarchical Clustering 10

1.3 The Presentation 11

2 A Local Search Approach **14**

2.1 Introduction 14

2.2 Adding a Facility 16

2.2.1 When can we add a facility 17

2.2.2 Which facility to add 18

2.2.3 Improvement from ‘Add’ Heuristic 21

2.3 A Generalized Exchange 22

2.4 An Unified Local Search 27

2.4.1 Local Improvement Step 27

2.4.2 The Local Search Algorithm 31

2.4.3 Analysis of Running time 32
2.5 Bicriteria Bounds and The k-Median Problem: 35
2.5.1 An Approximation for the k-Median Problem 36

3 A Linear Programming Approach **38**
3.1 Introduction 38
3.2 Filtering and Facility Location 41
3.3 Approximating The k-median Problem: An outline 44
3.4 Consolidating Locations 47
3.5 Consolidating Medians 48
3.6 Integrality Gap of a Half-Integral Solution 52
3.6.1 Creating an Initial set of Trees 52
3.6.2 A simple proof of integrality gap 2 52
3.6.3 A gap of factor $\frac{4}{3}$ 53
3.7 A Different View 62

4 Primal Dual Algorithms **64**
4.1 Introduction 64
4.2 Primal Dual Algorithm for Facility Location 66
4.3 Lagrangian Relaxations: Revisiting k Median 70
4.4 Properties of the Dual Solution 74
4.5 Modified Algorithm for k-Median 76
4.5.1 The Approximation Factor Analysis 78
4.5.2 Running Time 86

5 Lower Bounds **89**
5.1 Introduction 89
5.2 The K median Lower Bound 90
5.3 The Facility Location Problem 91
5.3.1 Max SNP-hardness 91
5.3.2 A 1.467 Lower Bound 92
5.4 Gap Examples 95

5.4.1 The k-Median Problem : Primal Linear Program 96
5.4.2 The k-Median Problem: The Dual Constructed 96
5.4.3 Facility Location: The Dual Constructed 97
5.4.4 Facility Location: The Primal Linear Program 98

6 Toolbox – I: Developing Different Approaches 100
6.1 Introduction 100
6.2 Scaling Costs 102
6.2.1 Bicriterion Tradeoff Results 103
6.2.2 Scaling and Capacitated Facility Location 104
6.3 Combining Different Heuristics 105
6.3.1 Local Search And Feasible Fractional Solutions 105
6.3.2 Local Search and Filtering 109
6.3.3 Primal Dual and Local Improvement 111
6.3.4 The Best Known Constant 113
6.4 Half-Integral Analysis: Capacitated k-Median 114
6.4.1 Outline of the algorithm 114
6.4.2 Step 1: Consolidating Demands 115
6.4.3 Step 2: Consolidating Medians 115
6.4.4 Step 3: Obtaining a $\{\frac{1}{2}, 1\}$-integral solution 118
6.4.5 Step 4: Rounding to an integral solution 119
6.4.6 Analyzing the cost 124

7 Toolbox – II: Interchanging Costs 126
7.1 Introduction 126
7.2 The Buy-at-Bulk Facility Location Problem 129
7.3 The Lower Bounded Facility Location Problem 132
7.3.1 The Algorithm 133
7.3.2 Analysis 134
7.4 Demand Dependent Facility Costs 135
7.5 Generalized Clustering 136

8 Fault-Tolerant Facility Location **140**

8.1 Introduction 140

8.2 The Linear Program and a structured solution 141

8.3 An Approximation Algorithm 143

8.3.1 Phase 1: Filtering and Scaling 143

8.3.2 Phase 2: Rounding 144

8.3.3 Facility Location Revisited 147

9 Hierarchical Placement Problems **149**

9.1 Introduction 149

9.1.1 Formal Definitions 152

9.2 The Simple-Placement Problem 153

9.2.1 Small Miss Rate Theorem 153

9.2.2 Solving for small miss rate 155

9.3 The Multi-Level Facility Location Problem 158

9.4 The General-Placement Problem 160

10 Single-Sink Edge-Installation **164**

10.1 Introduction 164

10.1.1 Formal Definitions 165

10.2 The Access Network Design Problem 167

10.2.1 The Algorithm 168

10.2.2 Analysis 169

10.2.3 Dealing with Small Demands 170

10.3 Single-Sink Edge-Installation with Concave Costs 171

10.3.1 Layered Structure of Optimum Solution 171

10.3.2 Removing Similar Pipes 172

10.3.3 Properties of the Near-Optimum Solution 173

10.3.4 The Algorithm 176

10.3.5 Analysis 177

11 Clustering Data Streams **182**

11.1 Introduction 182

11.2 Clustering in Small Space 184

11.2.1 Separability Theorems 184

11.2.2 Composing Two Algorithms: A Black Box Reduction 186

11.2.3 Divide-and-Conquer Strategy 188

11.3 The Data Stream Model 189

11.3.1 $\tilde{O}(nk)$ Time Clustering of Data Streams 191

11.3.2 Clustering in Sub-Quadratic Time 192

11.3.3 Extending to Weighted Input 193

11.3.4 Deterministic Algorithms in $\tilde{O}(nk)$ time 194

11.4 Time Lower Bounds 196

12 Hierarchical Agglomerative Clustering **197**

12.1 Introduction 197

12.2 A Case for Approximation 198

12.2.1 Approximating the Clusters 200

12.3 Related Implementations 202

12.4 Clustering using Representatives: CURE 203

12.4.1 Intuition and Overview 204

12.4.2 Clustering procedure: 205

12.4.3 Enhancements for Large Data Sets 207

12.4.4 Random Sampling 208

12.4.5 Partitioning for Speedup 209

12.4.6 Labeling Data on Disk 211

12.4.7 Handling Outliers 211

12.5 Experimental Results 213

12.5.1 Algorithms 214

12.5.2 Data sets 215

12.5.3 Quality of Clustering 216

12.5.4 Sensitivity to Parameters 219

12.5.5 Comparison of Execution time to BIRCH 223
12.5.6 Scale-up Experiments 224

13 Conclusion **226**

Bibliography **230**

Chapter 1

Introduction

The facility location problem is one of the most well-studied problems in optimization literature. Given a set of client locations, the problem seeks to find a set of optimal locations to build facilities (warehouses) minimizing the sum of the cost of building the facilities along with the cost of assigning clients (stores) to facilities. The facility location problem is NP–hard. This problem has been studied over the last four decades (Balinski [Bal66], Kuehn and Hamburger [KH63], Manne [Man64], and Stollsteimer [Sto61, Sto63]). The problem has been studied from perspectives of worst–case performance guarantees, empirical investigation of different heuristics, average–case analysis, and polyhedral characterization. It is a simple problem to pose and captures a wide variety of applications [CNW90].

Recently the facility location problem has received increasing attention. One interesting application is placing name servers on a large network such as the internet. Given a network, finding the set of locations to place servers such that the sum of the cost of placing the servers and the cost of clients (in their access times) is minimized, is a facility location problem. The sizes of such networks make the optimization problem much more formidable. The same tradeoff manifests if we choose data replicators (mirror sites) to cache information.

But perhaps the most interesting application for the facility location problem has been clustering. The wealth of information embedded in huge databases belonging to corporations (e.g., retail, financial, telecom) has spurred a tremendous interest in the areas of *knowledge discovery* and *data mining*. Clustering, in data mining, is a useful technique for discovering interesting data distributions and patterns in the underlying data. The problem of clustering can be defined as follows: given n data points in a d-dimensional metric space, partition the data points into k clusters such that the data points within a cluster are more similar to each other than data points in different clusters.

One of the earliest known models of clustering is the problem of Weber (see [Web09]), known as the 1-median problem, in which the objective is to find a point which minimizes the sum of distances to all points. This measures the average radius if the point set is considered a single cluster. In case of multiple clusters this extends to the k-median problem. In this problem, the objective is to find k points labelled as medians, where every other point is assigned to its closest median, such that the total sum of the assignment distances is minimized. However in situations where the number of required partitions are unknown, the clustering problem can be modeled as a facility location problem. The problem would associate a cost with every new cluster introduced, and minimize the linear combination of the costs of the clusters along with the average radii of the clusters. The k-center problem was introduced to minimize the maximum radius of a cluster, instead of the average radius. The three problems of uncapacitated facility location problem, the k-median problem and the k-center problem constitute a very large fraction of problems in discrete location theory.

Although clustering has been widely studied, across several disciplines, only a few of the techniques developed scale to support clustering applications of very large datasets. One useful e-commerce application of such large datasets is the clustering of click streams to discover segments in the customer population. Other applications include clustering web pages, multimedia data, retail chain transactions, and telecommunication phone records to discover fraud. Furthermore, such data is typically stored in secondary storage devices and access is costly. These cost considerations necessitate the development of scalable clustering algorithms that run in linear time, revisit the data as little as possible, and consume small space. Thus designing more efficient heuristics, in terms of space and running time, has also become necessary. We extend the clustering algorithms to the data stream model, which only allows a single pass over the data.

The facility location problem models clustering as well as network design problems. We will use the ideas from clustering, and use them to design algorithms for efficient bandwidth allocation problems. The capacitated variants of the problem, where each facility has an associated capacity, will allow us to model edge installation problems in network design involving economies of scale. A high level idea would be to agglomerate sufficient traffic over links to exploit the economy of scale; the agglomeration being a clustering problem.

We will be concerned with developing efficient worst-case heuristics for the facility location problem and its variants. Formally a ρ-approximation algorithm is defined as an algorithm that produces a solution within factor ρ of the optimum, on all instances of the problem. In the next section we will describe the results presented here. However due to the recent interest, this problem has been approached from different directions, including different techniques from approximation theory.

A broader perspective: The facility location problem showcases some of the most interesting techniques in approximation algorithms. These techniques differ mostly in deriving lower bounds on the objective function, which is used in comparison with the heuristic solution to prove the approximation guarantee.

Local search is a common heuristic in optimization problems: considering local neighborhoods, and finding a locally optimum solution. The definitions of neighborhood and the methods of searching give rise to countless variants. Typically such algorithms are simple and provide insight into the structure of the optimal solution. The simplicity ensures easy implementation. If the local operations are inexpensive, and the domain space is small, these algorithms are usually efficient. In case of NP-hard problems, these operations cannot guarantee improvement of any suboptimal solution, but can guarantee improvement if the solution is very far from the optimum. The simplest local changes to a solution in context of the facility location problem is adding or deleting a facility. We will start our discussion from local search since it is the easiest algorithm in our context.

However, the local search approach has a drawback that it does not produce a lower bound, independent of the proof of the approximation factor. Assuming we have instance-specific lower bounds, that are not derived from proof of approximation guarantee of the final solution, we can provide instance-based guarantees. Of course, we would be only be able to prove worst-case bounds, but if we were faced with a real instance, an independent lower bound is of great value. It is conceivable that the final solution constructed will be better than predicted by the exact worst-case of the analysis.

Linear Programming has been one of the most important tools in optimization from the above viewpoint. Posing a problem as an integer linear program and solving its fractional relaxation to get a lower bound has been a first step. This is useful even in the context of exact optimization. In the context of approximation algorithms, the fractional solution of a linear program provides a lower bound if we ensure that the optimum solution satisfies the constraints. The objective in this case is to construct an integral solution which is a (hopefully) small factor of the lower bound using the fractional solution. This step is referred to as the "rounding" step. This approach bas been used for over two decades to bound performances of approximation algorithms [Cha79, Lov75, CFN77, Wol80].

The linear programming formulation has its drawbacks as well. If the number of constraints or variables are large, solving a linear program is non-trivial. In a few cases we can reach a middle ground, where we have a combinatorial algorithm, as well as a lower bound somewhat independent of the solution constructed. For minimization problems, the facility location problem for example, the dual of the linear program is a maximization problem, and by duality theory, any feasible dual solution is a lower bound to the primal objective function. Since the dual problem is a maximization problem, we can conceive of constructing a feasible solution and use this solution as a lower bound. If the structure of the dual linear program is simple, we can create a feasible solution combinatorially. The idea is to create a feasible dual solution, and a particular integral solution of the primal problem which is bounded in terms of the dual solution. This was first explicitly demonstrated in [AKR95], and subsequently the method has been used in a variety of problems; see [GW97] for more details.

The facility location problem provides an example to study the differences between these different approaches. It is also interesting that the best known facility location approximation algorithm is derived from the combination of all these three approaches. The combination of the local search and linear programming heuristics result in approximation algorithms that have better approximation guarantee than each, and is the best result for a fault tolerant generalization of the facility location problem.

1.1 Preliminary Definitions

In this section we will formally define the facility location problem and the k-median problem. The definitions of the generalizations, and network design problems which we will consider later, are relegated to later sections.

The facility location problem is defined as follows: given a network, such that building a facility at location i costs f_i, and assigning location j to facility i costs $d_j c_{ij}$, where d_j is the demand of the location j, and c_{ij} is the metric distance between i and j; identify the best possible locations to build facilities, such that each demand point is assigned, and the total cost of building facilities and assigning demands is minimized. This is also referred to as the metric uncapacitated facility location problem. We will drop the qualifiers and use restrictions wherever appropriate. There are several capacitated variants of the problem, where each facility i can only serve a total demand u_i. The variants depend on the fact that multiple copies of a facility can be bought at a location, or that a demand point can be assigned fractionally to several facilities. If the demand can be split across facilities, it is termed as *splittable*, and *unsplittable* otherwise.

The k-median problem is defined as follows: given a collection of n points, find k points (medians) such that each of the n points is assigned to its closest median, identify the k points such that the total sum of the assignment distances is minimized. We can conceive a variant where each point has a demand (weight) d_j and its contribution to the sum is $d_j c_{ij}$ if its nearest median is at point i.

In fact the k-median problem is the facility location problem if we ignore the cost of the facilities, while enforcing a strict restriction that at most k facilities can be built. The capacitated versions of the k-median problem are defined analogous to capacitated facility location problems.

The k-center problem is identical to the k-median problem, except the objective function minimizes the maximum assignment distance instead of the sum.

1.2 Overview of Results

We will present approximation results for the facility location problem and the k-median problem. The techniques will be subsequently used to obtain approximation algorithms for related problems. We will also present an application of approximation algorithms to design robust clustering algorithms. The results would be presented based on the technique used, and in increasing order of complexity, starting from local search technique. They can be summarized as below:

1.2.1 The k-Median Problem

The k-median problem was shown to be optimally solvable on trees in polynomial time by Kariv and Hakimi [KH79]. Hochbaum [Hoc82], gave an $O(\log n)$ approximation using $O(k \log n)$ medians. Lin and Vitter [LV92b] gave an $O(1)$-approximation using $O(k \log n)$ medians. They also showed that this result is best possible if the distance function is asymmetric or does not follow triangle inequality. In [LV92a] they showed $O(1)$-approximation using $O(k)$ medians in case of general metric spaces.

The first result using exactly k medians was proved by Bartal in [Bar96], which gave an $O(\log^2 n)$-approximation algorithm, through probabilistic approximation of metric spaces. This was subsequently improved in [Bar98] to $O(\log n \log \log n)$. This was improved and derandomized to $O(\log k \log \log k)$ in [CCGG98, CCG+98].

We show the first constant factor approximation for the k-median problem. We show that the fractional linear program for the k-median problem can be rounded within factor $6\frac{2}{3}$. We also how $O(1)$ approximation for the capacitated k-median problem with a small blowup in the capacity. This is joint work with Charikar, Tardos, and Shmoys [CGTS99].

Jain and Vazirani in [JV99] gave a primal-dual 6-approximation algorithm for the k-median problem. We show that the dual construction procedure can be exploited along with augmentation to produce a more structured solution with an approximation guarantee of 4. This also is the upper bound of the integrality gap of the k-Median linear program. This is joint work with Charikar [CG99].

We also show that the k-median problem cannot be approximated within factor $1 + \frac{1}{e}$ unless $NP \not\subseteq DTIME^{\log\log n}$[1].

1.2.2 The Facility Location Problem

Shmoys, Tardos, and Aardal [STA97] gave the first constant factor approximation algorithm for the facility location problem using the linear programming formulation of Balinski [Bal66].

We show that the facility location problem is Max SNP-hard, thus unlikely to have a PTAS. Morover under the assumption $NP \not\subseteq DTIME^{\log\log n}$ the approximation factor cannot be better than 1.47. This is the best known lower bound. We also show how to combine an approximation algorithm with greedy augmentation to improve the algorithm. This is used later to design the best possible approximation factor for the facility location problem. This is joint work with Khuller [GK98].

Korupolu, Plaxton, and Rajaraman [KPR98] analyzed the local search heuristic of Kuehn and Hamburger [KH63], and gave a 5-approximation. Jain and Vazirani [JV99] provided a primal-dual 3-approximation to the facility location problem in time $O(n^2 \log n)$.

We give a 2.41-approximation algorithm based on local search in time $O(n^2 \log n)$. We show that the greedy augmentation and the primal-dual algorithm, give a 1.85 combinatorial approximation algorithm for the facility location problem in time $O(n^3)$. Chudak and Shmoys [Chu98, CS98] gave a linear programming rounding algorithm with approximation factor 1.74. We use their algorithm in some cases of the problem

[1] $DTIME[f(n)]$ is the class of decision problems that can be solved in (deterministic) time $O(f(n))$.

and the resulting algorithm using greedy augmentation, primal-dual, and linear programming rounding improves the constant to 1.73, demonstrating that the analysis is not tight. This is joint work with Charikar [CG99]. Lin and Vitter [LV92b] show a bicriterion tradeoff result for the facility location problem. We also show a better tradeoff result using this technique.

Jain and Vazirani [JV00] gave an $O(\log n)$-approximation algorithm for the fault tolerant facility location problem, in which every client requests a specified number of facilities to be assigned to it. The cost of assignment is the sum of all these assignments. We give an $O(1)$-approximation using linear programming and rounding. We also show that certain aspects of the local search algorithms carry over to this problem. This is joint work with Meyerson and Munagala in [GMM00c].

We consider a version of the facility location in which the facilities must serve a minimum amount of demand to be built. This allows us to consider facility location problems where the cost of the facility depends as a (nonzero polynomial) function of the demand. This is joint work with Meyerson and Munagala in [GMM00b, GMM00a]. Bartal, Charikar, and Raz[BCR00] gave a bicriterion approximation for minsum clustering, where the objective function is the sum of distances of pairs of points in the same partition. We extend their result to suitable normalization functions applied to the sum of distances of pairs in the same cluster, using this variant of the facility location problem.

1.2.3 The Single-Sink Edge-Installation Problem

Consider a service provider or utility company trying to design a network to provide service. The agency can use economies of scale in designing the network, i. e., the cost per unit traffic decreases when routing larger amounts. This is modeled by "buy-at-bulk" network design problems. We are given demands at nodes in a network which have to be routed to their respective destinations using pipes of certain capacities and costs per unit length. The costs obey economies of scale, in the sense that it is cheaper to buy a pipe of larger capacity than many pipes (which sum to the same capacity) of a smaller capacity. The goal is to optimize the total cost of the pipes we buy to route the demands. If all the demand points want to be connected to the same sink node, this is the single-sink edge-installation problem. The cost function is represented as a concave function of the traffic.

The problem of *buy-at-bulk* edge-installation was first defined in [SCRS97]. Awerbuch and Azar [AA97] obtain an $O(\log^2 n)$-approximation to this problem even when all demands have different sinks. Their work is based on techniques to approximate any metric by tree metrics [Bar96]. The approximation factor can be improved to $O(\log n \log\log n)$ using Bartal's result in [Bar98] and derandomized using the results of [CCGG98, CCG+98]. For single-sink edge-installation, Salman et al [SCRS97] show a constant approximation when there is only a single pipe type. Their method is based on the technique of balancing Steiner and shortest path trees [ABP90, KRY94]. Andrews and Zhang [AZ98] define the Access Network Design problem, which is a special case of single sink where the pipe costs and capacities obey certain common constraints, and provide an $O(K^2)$-approximation, where K is the number of pipe types. Meyerson, Munagala and Plotkin [MMP00] provide an $O(\log n)$-approximation to single sink network design where the costs on the edges are arbitrary non-decreasing concave functions of demand.

We show the first constant factor approximation for single-sink edge-installation problem with concave costs. This is joint work with Meyerson and Munagala in [GMM00b, GMM00d].

1.2.4 Clustering Data Streams

Under the Data Stream Model, computation takes place within bounded space, and the data can only be seen via linear scans from the first record onwards (i.e., a data point can be seen only once in a scan, and points must be viewed in order).

The first result in this model was given by Munro and Paterson [MP80] where the space requirement of selection and sorting as a function of the number of passes over the data was studied. The model was formalized by Henzinger, Raghavan and Rajagopalan [HRR98], who gave several algorithms and complexity results related to graph-theoretic problems and their applications. Other papers with results in the data stream model include: a small space approximation of the number of distinct elements in a data stream by Flajolet and Martin [FM83]; approximate selection in a single pass over the data by Manku, Rajagopalan and Lindsay [MRL98, MRL99] and space efficient L^1 difference between two data streams by Feigenbaum, Kannan, Strauss and Vishwanathan [FKSV99].

Charikar, Chekuri, Feder, and Motwani [CCFM97] gave a constant factor algorithm for the incremental k-center problem, which is also a single-pass algorithm requiring $O(nk \log k)$ time and $O(k)$ space. There is a large difference, however, between the k-center and the k-median problem, in construction of good lower bounds. A set of $k+1$ suitably spaced points provide a lower bound for the k-center problem. These points can be thought of as a proof of goodness of the clustering. In the k-median problem, allowing points to have weights, no such succinct proofs exist, and the optimization problem has much more global character. Also the k-median objective function is more robust to the presence of outliers and is desirable in most clustering situations.

We first show separability theorems for the k-median problem, that enable a divide-and-conquer algorithm. Using this paradigm, we show how to achieve a constant-factor algorithm using $\tilde{O}(nk)$ time, $O(n^\epsilon)$ space, and a single pass. We show that no constant factor approximation is possible in $o(nk)$ time. This is joint work with Mishra, Motwani and O'Callahan [GMMO00].

1.2.5 Robust Hierarchical Clustering

A hierarchical clustering is a sequence of partitions in which each partition is nested into the next partition in the sequence. An *agglomerative* algorithm for hierarchical clustering starts with a disjoint set of clusters which places each input data point in an individual cluster. Pairs of items or clusters are then successively merged until the number of clusters reduces to k. At each step, the pair of clusters merged are the ones between which the distance is the minimum. This is one of the most popular algorithms used in information retrieval, biology, medicine, and image processing. The biggest advantage of hierarchical clustering is the resulting hierarchy trees of clusters (dendograms). This allows for exploration of particular clusters of interest wchich is especially useful in refining queries and search. Refer to [Ras92, vR79, Sal89, SM83, Wil88, WMB94] for applications of hierarchical agglomerative clustering in information retrieval and image processing.

However the existing methods of computing the hierarchy are either insensitive to sizes of cluster, or are sensitive to presence of outliers. We use the k-center approximation algorithm to approximate each cluster and use the result to infer the hierarchy. This method of using representative points gives us a robust hierarchical clustering algorithm. We also present sampling and partitioning as an alternate to preclustering. This empirically justifies the divide and conquer approach in the context of large datasets. This is joint work with Rastogi and Shim [GRS98].

1.3 The Presentation

The presentation will have four parts. The presentation will be in terms of the techniques and ideas used rather than the problem they were applied to. The next four chapters (Chapters 2-5) concern the basic facility location and k-median problem. The Chapters 2-4 will be about the different approaches to solving the two problems. Chapter 5 will provide lower bounds for these problems.

The next set of chapters (Chapters 6-7) will be concerned with manipulating the basic heuristics and combining them with other ideas to develop better algorithms. The ideas in these two chapter will be used in various applications for the facility location problem.

The next set of three chapters (Chapters 8-10) concern the application of facility location problem to network design problems.

The last two chapters (Chapters 11-12) consider the question of clustering large datasets, theoretical bounds and empirical results about using approximation algorithms.

In Chapter 2 we consider the local search algorithm for the facility location problems. We will start by investigating the 'Add' operation. This step will be used in several different contexts, in combination with other algorithms, to derive algorithms that are better than both the constituent algorithms. We formulate a generalized exchange step which deletes vertices. This leads to a simple 3-approximation algorithm. Next we show how to unify the two steps into a single one. The results from the unified operation supersedes the result of our generalized exchange step. It has smaller running time and an improved approximation guarantee. However, the most tangible benefit is a bicriterion approximation algorithm, for the k-median problem.

In Chapter 3 we consider linear programming approaches to facility location problems and its variants. We use the strength of linear programming to give the first known constant factor approximation for the k-median problem. We review techniques of agglomerating fractional facilities known as "filtering", and previous constant factor approximations for the facility location problem. In the process we show that the integrality gap of a half-integral solution is $\frac{4}{3}$, which is tight.

In Chapter 4 we consider the primal dual approach to facility location problem. Using Lagrangian relaxation, this leads to an approximation the k-Median problem. We review the primal-dual algorithm for the facility location problem, and show how to use the structural properties of the dual solution to improve the approximation factor of the k-median problem to 4. This also proves that the integrality gap of the k-median linear program is 4.

In Chapter 5 we consider the lower bound results for the facility location and the k-median problem. We show that the facility location problem is unlikely to be approximated below factor 1.46 and the K-median problem below factor $1 + \frac{1}{e}$. We also present constructions of integrality gaps of the linear programs for these problems, as well as the lower bounds on gaps of the dual solution constructed by the primal dual algorithm.

In Chapter 6 we show how to use scaling to achieve better approximation algorithms using the three different approaches. This gives us near-optimal bicriteria tradeoff results for the facility location problem. We also show how to combine these heuristics together to get better approximation guarantees. We apply the half integral analysis in context of the capacitated k-median problem. The use of half-integral solution allows an amortized analysis of the costs.

In Chapter 7 we consider the facility location problem with lower bounds. This problem involves having lower bounds on capacities of the open facilities. This allows us to consider facility location problems where the facility cost depends on the demand served by the facility. It also allows us to show results about the generalization of minsum clustering. In general this variant of the facility location problem allows us to deal with situations where serving a lot of demand from a facility is beneficial. The solution to this problem is used as a subroutine in approximating single-sink edge-installation problems. The chapter demonstrates how to solve capacitated facility location problems by adjusting the metric or the facility costs, and a reduction to uncapacitated facility location problem.

In Chapter 8 we consider the fault-tolerant facility location problem and show the first constant factor approximation for the problem, improving an $O(\log n)$ primal-dual approximation algorithm.

In Chapter 10 we consider hierarchical placement problems. In particular we consider the multilevel facility location problem, and caching on a network. We show how to reduce these problems to the facility location problem and achieve an $O(1)$-approximation for them.

In Chapter 10 we consider the single source edge installation problem with concave costs and its variant, the access network design problem, and show the first constant factor approximation for these problems. We improve previous results of $O(\log n)$-approximation for these problems.

In Chapter 11 we consider the problem of clustering large data sets. We show separability theorems that allow divide-and-conquer approximation algorithm, using small space. We also show how to extend the algorithm to data streams.

In Chapter 12 we consider the problem of hierarchical agglomerative clustering. We suggest a method to make this algorithm robust, while allowing it to recognize clusters of arbitrary sizes and shapes. We present empirical results on its performance and comparison with other known algorithms for clustering large databases.

Chapter 2

A Local Search Approach

2.1 Introduction

Local search and iterative improvement algorithms are one of the most used heuristics in optimization. These algorithms are usually simple to present and will be our starting point. These algorithms repeatedly consider a neighborhood of the solution and search for a locally optimum solution. The definitions of neighborhood and the methods of searching give rise to countless variants. Typically such algorithms are simple and provide insight to the structure of the optimal solution. The simplicity ensures easy implementation. If the local operations are inexpensive, and the domain space is small, these algorithms are usually efficient.

The simplest neighborhood of a solution is considering a solution with one extra facility. We need to determine if opening a facility at a new location would reduce the cost of the solution. It appears plausible, if the assignment cost of our solution is high, the reduction of assignment cost would pay for the cost of the new facility. The next question would be immediate, if we can we delete facilities when the facility cost is high. If we can perform both, we would have a combinatorial approximation algorithm. The idea would be to use the 'Add' strategy to reduce the assignment cost by increasing the facility cost; and subsequently trying to reduce the facility cost by paying increased assignment cost. The latter would be a 'Delete' step. Observe that we would not be always able to delete facilities (for example, if it is the only open facility), but can always attempt an exchange. The proof of the quality of the final solution would hinge on the fact that both the final facility and assignment costs of our solution are small, otherwise we would improve the solution. For the discussion in this chapter we will ensure that the total cost always decrease, however in some cases operations that do not reduce the total cost may allow us to avoid local minima. In fact later in this chapter we will consider local based heuristics for the k median problem, where we will increase the objective function in some steps, since we will try to optimize a different function.

The first combinatorial approximation algorithm for the facility location problem was given by [KPR98]. They analyzed a heuristic named 'Add and Drop', first proposed by Kuehn and Hamburger in [KH63]. The proof hinges on the following outline: if a facility had other facilities close by, the first facility can be closed and the assignments shifted to the nearby open facility; and if it cannot be done it would imply all facilities are far from each other. In this case the optimum must also have its facilities spread out. Consequently we can try exchanging facilities, adding one and deleting one, to improve the solution. This algorithm gives an approximation guarantee of 5, in time $\tilde{O}(n^2k^2)$, on n locations and k possible facilities.

In this chapter, we will provide a simpler combinatorial algorithm. We will start by investigating the 'Add' operation. This step will be used in several different contexts, in combination with other algorithms, see Section 6.3 to derive algorithms that are better than both the constituent algorithms.

Next we will formulate a generalized exchange step, which would delete vertices. This would lead to a simple factor three approximation algorithm. Starting from these two steps, we will show how to unify the two steps into a single one in Section 2.4. The results from the unified operation would supersede the result of our generalized exchange step. However the delete step would serve two purposes: first as a primer for the unified step, separating the arguments required to analyze deletion of facilities as opposed to addition. Second, the delete step demonstrates that a solution with lower cost of facilities can be used. In the context of the k-median problem where the number of facilities are fixed, this argument shows how to construct a feasible assignment with the required number of facilities (k medians) in case we have a solution with many more facilities (medians). This forms the basis of a divide and conquer strategy for problems like k-median, and facility location with uniform facility costs, and ultimately leads to an algorithm for the data stream Model (Chapter 11).

The unified local improvement step would be concerned with the question of the best possible improvement in the solution on the addition of one facility. The algorithm will be 'local' in this sense, we would have at most one new facility at every iteration.

The benefit of the unified step will be several; simplicity, smaller running time and an improved approximation factor. This will also provide a bicriterion approximation algorithm for the k-median problem (Section 2.5). This bicriterion approximation, in combination with the technique of Scaling, would improve several extant approximation algorithms (Section 6.2). Together these two ideas provide a powerful tool in approximating other optimization problems that can be reduced to the facility location problem (Section 9).

The results in this chapter appear in [GK98, CG99, Cha00].

2.2 Adding a Facility

We start by considering the question, under what conditions can we improve an existing solution to the facility location problem by adding a facility. The following observation is central to this discussion:

Fact 2.2.1 *In uncapacitated facility location problem the demand locations are assigned to the nearest built facility. The set of built (or open) facilities specify the solution completely.*

With the above fact, we can answer our question. We compute the best possible assignment in our solution (without location i) and the best assignment with the new set of facilities (including i). The difference is the saving, in assignment cost, we generate by adding the facility i. If the saving is less than the cost of the facility, f_i, adding i as a facility does not reduce the cost of the solution. otherwise, if the saving is more than the cost of facility f_i, we should build a facility at location i.

The question arises when can we improve the solution by adding a facility and which facility to add if there is a choice. The first is an easier question to answer.

2.2.1 When can we add a facility

We prove the following lemma,

Lemma 2.2.1 *If there exists a solution,* SOL, *with facility cost F_{SOL} and assignment cost C_{SOL}, and we have a solution with assignment cost $C > C_{\text{SOL}} + F_{\text{SOL}}$, then there exists a facility whose addition improves the solution.*

Proof: Define $Gain(i)$ to be the decrease in total cost on adding the facility. Thus $Gain(i)$ is the reduction in assignment cost minus the cost of the facility f_i. We will show is that there exists a node with positive $Gain()$. Let $\sigma(j)$ be the facility to which a demand location j is assigned to in our solution.
Consider the set of facilities $\mathcal{F}$, that are used in SOL. We will only be concerned with these facilities. For $i \in \mathcal{F}$, consider the change in assignment cost, if all the demand locations that were assigned to i in SOL, are assigned to i in our solution. Define $Gain'(i)$ to be this change in assignment cost minus the facility cost f_i.
We will argue that

$$Gain(i) \geq Gain'(i)$$

When we computed $Gain'$, we changed the assignments without considering their benefit. It is possible that a location j, assigned to i in SOL, is assigned to a facility closer to it than i in our solution. Consider the change in assignment cost that we would perform, in adding facility i. That would be $Gain(i)+f_i$. These changes, which we would perform, are the best possible changes: we will definitely not reassign j to i, if j was already assigned to a facility closer than i. On the other hand, if i were indeed closer than the facility j is currently assigned to, we would account this term in computing our change in assignment cost. We will also account for locations j', which are not assigned to i in SOL, nonetheless, are closer to i than the facilities they are assigned to in the current solution.

Therefore if we view the change in assignment, We will account for all the positive terms (and possibly more) and drop some negative terms. Our change in assignment cost will be greater (in the signed sense) than the one defined in computing $Gain'()$. Subtracting f_i from both proves the above equation.
Now we add $Gain'(i)$ over all $i \in \mathcal{F}$,

$$\sum_{i \in \mathcal{F}} Gain'(i) = \sum_{i \in \mathcal{F}} (\text{ Change in assignment}) - \sum_{i \in \mathcal{F}} f_i$$

The last term evaluates to F_{SOL}. The first summand can be computed differently. Instead of summing over i to which a demand location j is assigned to in SOL, we can sum over the demand locations. The contribution of every demand location is $d_j(c_{\sigma(j)j} - c_{ij})$. The next equation computes this contribution over all demand points:

$$\sum_j d_j(c_{\sigma(j)j} - c_{ij}) = C - C_{\text{SOL}}$$

Putting all of the above together, we have

$$\sum_{i \in \mathcal{F}} Gain(i) \geq \sum_{i \in \mathcal{F}} Gain'(i) = C - C_{\text{SOL}} - F_{\text{SOL}} > 0$$

Since the sum is positive, there must exist one i with $Gain(i) > 0$. ∎

The above proof demonstrates the kind of arguments we would use to prove assertions about the benefit of including (and later, in a restricted sense, dropping) facilities. We will proceed by defining moves that are sub-optimal compared to ours, but results in the hypothetical solution. We will not require any knowledge of this solution, but argue that since we made optimal moves, our benefits are sum up to more than those of the sub-optimal ones. The sum of the sub-optimal moves will be the difference between the current and the hypothetical solutions.

2.2.2 Which facility to add

In the last lemma we saw a condition when we are guaranteed to find a facility whose addition improves the total cost. The lemma can be reread as: if the current assignment cost is greater than the total cost of the optimum solution, then there exists a facility which improves the cost on addition. However, we will see later why we posed the lemma in terms of a feasible solution, not necessarily the optimum.

The decision on which node to add appears to be simple: add whichever facility that improves the cost. However *the order in which we add facilities* is important for arguing about the quality of the solution; this is an aspect of any iterative improvement strategy. Different heuristics choose different policies to add facilities. In this section we will consider the following strategy, in the next sections we will see two other strategies.

1. Repeatedly add the facility which has the highest positive **Ratio** of improvement to cost.

2. If no such facility exist, stop and output current solution so far.

To show the benefit from such a strategy, we first prove a small corollary.

Corollary 2.2.1 *If there exists a solution* SOL *with facility cost* F_{SOL} *and assignment cost* C_{SOL}*, and the current assignment cost is* C*, then there exists a facility with ratio* $(C - C_{\text{SOL}} - F_{\text{SOL}})/F_{\text{SOL}}$.

Proof: From the proof of Lemma 2.2.1, we have

$$\sum_{i \in \mathcal{F}} Gain(i) \geq C - \text{SOL} - F_{\text{SOL}} \quad \text{and} \quad \sum_{i \in \mathcal{F}} f_i = F_{\text{SOL}}$$

Thus the average ratio of the set of facilities in SOL is as claimed in the lemma. Therefore there must exist at least one facility with the claimed ratio. ■

Now we are in a position to bound the cost of the solution produced by the above strategy. So far we have seen that if the current assignment cost C is greater than $C_{\text{SOL}} + F_{\text{SOL}}$, there exits a facility with good ratio. We will pose the next lemma in a slightly general form, assuming if the assignment cost C is greater than C_{th} guarantees a facility of good ratio. This will ensure that the lemma is self contained, since we will apply the lemma in several different contexts. We have only considered feasible integral solutions in our discussion; we will extend the results to feasible fractional solutions of relaxations of integer linear programs. We will apply this lemma in generalizations of the facility location problem as well. Returning to the lemma:

Lemma 2.2.2 *Starting from a solution with facility cost F and assignment cost C, if for $C > C_{th}$ there exits a facility with ratio $r(C - C_{th})$; then the cost of the final solution output by the greedy strategy is at most*

$$F + \max\left\{\frac{1}{r}\log(r(C - C_{th}) + 1), 0\right\} + C_{th}$$

Proof: If $C \leq C_{th}$, we have nothing to prove; the starting solution satisfies the bound, and subsequently, the solution only improves.

Since the algorithm is iterative, define C_ℓ to be the assignment cost after iteration ℓ, and F_ℓ likewise. Thus $C_0 = C$, and $F_0 = F$. Let i_ℓ be the facility chosen at the ℓ'th iteration. Let the iteration at which C_ℓ was less less than or equal to C_{th} for the first time be $m > 1$. We will show that the bound holds after the m'th iteration.

At iteration ℓ, the decrease in assignment cost was $C_{\ell-1} - C_\ell$, and the cost of facility i_ℓ was $F_\ell - F_{\ell-1}$. By assumption, for $\ell \leq m$,

$$\frac{(C_{\ell-1} - C_\ell) - (F_\ell - F_{\ell-1})}{(F_\ell - F_{\ell-1})} \geq r(C_{\ell-1} - C_{th})$$

The equation rearranges to

$$(F_\ell - F_{\ell-1}) \leq \frac{(C_{\ell-1} - C_\ell)}{r(C_{\ell-1} - C_{th}) + 1}$$

Summing over all ℓ from $1 \ldots m$ we have,

$$F_m + C_m = \sum_{\ell=1}^{m}(F_\ell - F_{\ell-1}) + F_0 + C_m \leq \sum_{\ell=1}^{m}\frac{(C_{\ell-1} - C_\ell)}{r(C_{\ell-1} - C_{th}) + 1} + C_m + F_0$$

Consider the coefficient of C_m on the right hand side. It is $1 - 1/(1 + r(C_{m-1} - C_{th}))$. Thus keeping every other C_ℓ the same, since $C_m \leq C_{th}$, the right hand side is maximized when $C_m = C_{th}$. We would be interested in an upper bound and therefore use this equality for C_m. We would need to upper bound,

$$\sum_{\ell=1}^{m}\frac{(C_{\ell-1} - C_\ell)}{r(C_{\ell-1} - C_{th}) + 1} = \sum_{\ell=1}^{m}\frac{1}{r}\left\{1 - \frac{r(C_\ell - C_{th}) + 1}{r(C_{\ell-1} - C_{th}) + 1}\right\}$$

Now we use the fact that $1 - x \leq \log \frac{1}{x}$, and the right hand side in the above can be bounded as,

$$\sum_{\ell=1}^{m} \frac{1}{r} \log \frac{r(C_{\ell-1} - C_{th}) + 1}{r(C_\ell - C_{th}) + 1} = \frac{1}{r} \frac{r(C_0 - C_{th}) + 1}{r(C_m - C_{th}) + 1} = \frac{1}{r}\left(r(C_0 - C_{th}) + 1\right)$$

Therefore,

$$F_m + C_m \leq F_0 + \frac{1}{r}\left(r(C_0 - C_{th}) + 1\right) + C_{th}$$

Since the cost can only decrease, the lemma holds subsequently. ■

Admittedly, the presentation of Lemma 2.2.2 appears arcane, the lemma will be used in several different context, and even in generalizations of the facility location problem. The lemma, stripped off the current context, is just a proof of an upper bound on the cost of a process defined by the greedy strategy. Notice that the feasibility of the solutions generated was not used in the proof of the lemma. Typically we would ensure feasibility (which is easy in facility location, and will be more involved in generalizations) while *choosing the facility* in the greedy strategy. The lemma would serve as an accounting tool, and hence we posed the lemma in a general setting.

In the current context, if there exists a solution SOL of facility and assignment cost F_{SOL} and C_{SOL} respectively, Corollary 2.2.1 and Lemma 2.2.2 give the following:

Lemma 2.2.3 *Starting from a solution with facility cost F and assignment cost C, and if there exists a solution* SOL *with facility cost F_{SOL} and assignment cost C_{SOL}, then the cost of the final solution output by the greedy strategy is at most*

$$F + F_{\text{SOL}} \max\left\{\log \frac{C - C_{\text{SOL}}}{F_{\text{SOL}}}, 0\right\} + C_{\text{SOL}} + F_{\text{SOL}}$$

Proof: Set $C_{th} = C_{\text{SOL}} + F_{\text{SOL}}$ and $r = 1/F_{\text{SOL}}$ in Lemma 2.2.2. ■

2.2.3 Improvement from 'Add' Heuristic

It is an interesting question, if the 'Add' heuristic is sufficient in itself to prove an approximation bound. In a special metric in which all the distances are 1 and 2, or

1 and 3, it turns out to be the case. These examples are interesting, since for these metrics, we will later show that the best possible approximation factors are achieved by 'Add' Heuristic. See Section 5.3 for the lower bound results.

Corollary 2.2.2 *For a metric with distances* 1 *and* 3 *we have a* 1.463 *approximation.*

Proof: We may assume that the optimal solution opens at least k facilities, for any constant k (otherwise by enumeration over all subsets of size at most k, we can obtain the optimal solution). If we open a single facility, we get $F \leq \frac{1}{k} F_{\text{SOL}}$, and since all demands are covered at distance 3, we get $C \leq 3C_{\text{SOL}}$. By applying a greedy improvement scheme, and by using Lemma 2.2.3, we obtain a solution with cost at most

$$F_{\text{SOL}} \left(\frac{1}{k} + \ln \frac{3C_{\text{SOL}} - C_{\text{SOL}}}{F_{\text{SOL}}} + 1 \right) + C_{\text{SOL}}$$

If we have a a approximation, the above is at most $a(F_{\text{SOL}} + C_{\text{SOL}})$. Simplifying and setting $\gamma = F_{\text{SOL}}/C_{\text{SOL}}$, gives

$$a \geq 1 + \frac{1}{(1 + \frac{1}{\gamma})k} + \frac{\gamma}{\gamma + 1} \ln \frac{2}{\gamma}.$$

Since we may assume that k is at least some fixed constant, by taking the maximum over all $\gamma > 0$ we get $a \geq (1.463 + \epsilon)$. Thus we can choose $a = 1.463 + \epsilon$ for any fixed $\epsilon > 0$. ■

For a metric with distances 1 and 2, we can show a 1.28 approximation.

2.3 A Generalized Exchange

In the last section, we considered adding a facility to our solution. It is a natural question, what happens if we consider deleting vertices. In fact from Lemma 2.2.1, if we cannot add a facility, we have the following inequality: [1]

$$C \leq F_{\text{SOL}} + C_{\text{SOL}}$$

[1]The right hand side can actually be replaced by $C_{\text{OPT}} + F_{\text{OPT}}$, but the equation will remain true even SOL is fractional. We will see the proof later in Section 6.3, but persist in using the general claim, for the sake of uniformity, and hopefully, clarity.

In this section we will be concerned with developing an improvement step that will allow us to improve the solution if the facility cost is very large. However as observed in the beginning of this chapter, we cannot delete facilities arbitrarily and maintain feasibility, for example consider a solution with one very expensive facility. Thus we will consider a generalized exchange operation. We will introduce a facility and delete a few other facilities. Deletion raises the question of reassigning the demand locations assigned to the deleted facility. The simplest answer would be to use fact 2.2.1 again, and reassign the demand locations to their respective closest facilities. We now present the operation, which deletes facilities to improve the overall solution.

1. Assume the set of existing facilities be $\mathcal{F}$. Let i to be the candidate facility, and initially, $Gain(i) = -f_i$.

2. For each facility $i' \in \mathcal{F}$, consider deleting this facility and reassigning the demand locations assigned to this facility to the existing set of facilities.[2] If the total cost decreases add this to $Gain(i)$, and mark i'.

3. If $Gain(i)$ is positive we can open a facility at i, delete the marked facilities, and change the assignments as mandated by the step above.

Notice, as explained before, in the above step we can introduce at most one new facility, which is i, and delete more than one existing facility. Observe that i could belong to $\mathcal{F}$, in which case the above operation deletes a few facilities and reassigns the demand locations. We can prove the following lemma:

Lemma 2.3.1 *If there exists a solution* SOL *with facility cost* F_{SOL} *and assignment cost* C_{SOL}*, and the current assignment cost is* C *and facility cost is* F*, then*

$$\sum_i gain(i) \geq F - C - C_{\text{SOL}} - F_{\text{SOL}}$$

Proof: As we saw in the proof of Lemma 2.2.1, we will define a suboptimal operation which when applied to all the facilities in SOL, will convert our solution to SOL.

[2] We may have deleted some facilities in $\mathcal{F}$ before getting to i'.

For $i' \in \mathcal{F}$, denote its closest neighbor in $\mathcal{F}_{\text{SOL}}$by $m(i')$. Let the facilities in SOL be denoted by $\mathcal{F}_{\text{SOL}}$. For $i \in \mathcal{F}_{\text{SOL}}$, define $Gain'(i)$ to change in cost due to the following:

- For all facilities $i' \in \mathcal{F}$, such that the facility in $\mathcal{F}_{\text{SOL}}$ closest to them is i, delete i' and assign all demand locations assigned to i', to the facility open at i.

For $i \notin \mathcal{F}_{\text{SOL}}$, define $Gain'(i)$ to be 0. We will claim $Gain'(i) \leq Gain(i)$. This follows the same reasoning as in proof of Lemma 2.2.1. Both the terms would have $-f_i$, due to opening a facility at location i. Consider the facilities which has i as the closest neighbor, that is $m(i') = i$. If the deletion of i' and reassignment of all demand assigned to it to i was beneficial (reduced total cost), then surely i' would be marked while computing $Gain(i)$. In fact the contribution to $Gain(i)$ could be better since the demand locations may have a better facility to be assigned to. Thus if $Gain'(i)$ had a positive term, it definitely would be included in computation of $Gain(i)$. On the other hand, $Gain(i)$ will have no negative terms, which $Gain'(i)$ may have. This proves the claim.

Now we will compute $\sum_{i \in \mathcal{F}_{\text{SOL}}} Gain'(i)$. Consider the following:

$$Gain'(i) = -f_i + \sum_{i', m(i')=i} [f_{i'} - \sum_{j, \sigma(j)=i'} d_j(c_{ij} - c_{i'j})]$$

Now i is closer to i' than $\bar{\sigma}(j)$, and by triangle inequality,

$$c_{ij} - c_{i'j} \leq c_{i'i} \leq c_{i'\bar{\sigma}(j)}$$

Thus we have the following inequality

$$\sum_{i \in \mathcal{F}_{\text{SOL}}} Gain'(i) \geq - \sum_{i \in \mathcal{F}_{\text{SOL}}} f_i - \sum_{i \in \mathcal{F}_{\text{SOL}}} \left\{ \sum_{i' \in \mathcal{F}, m(i')=i} \left[f_{i'} - \sum_{j, \sigma(j)=i'} d_j(c_{i'j} + c_{\bar{\sigma}(j)j}) \right] \right\}$$

Now the multiple summations can be simplified to get,

$$\sum_{i \in \mathcal{F}_{\text{SOL}}} Gain'(i) \geq - \sum_{i \in \mathcal{F}_{\text{SOL}}} f_i + \sum_{i' \in \mathcal{F}} f_{i'} - \sum_j d_j c_{\sigma(j)j} + \sum_j d_j c_{\bar{\sigma}(j)j}$$

The (unsigned) summands on the right hand side evaluates to F_{SOL}, F, C, and C_{SOL} respectively.

Therefore

$$\sum_i Gain(i) \geq \sum_{i \in \mathcal{F}_{\mathrm{SOL}}} Gain'(i) \geq -F_{\mathrm{SOL}} + F - C - C_{\mathrm{SOL}}$$

which proves the lemma. ■

The above proof is quite straightforward. Notice that the proof did not require the explicit knowledge of $\mathcal{F}$. In fact it only assumed that the facilities in $\mathcal{F}$ form a feasible solution. The function $Gain'()$ actually defines a feasible solution, whose assignment cost is bounded. Notice, the particular proof of feasibility of the solution involves assignment of demands to nearest facilities. This does not bode well for capacitated cases. But the proof of feasibility carries through if we were building a fixed number of facilities, say k. We will return to this in Chapter 11.

A factor 3 approximation In the last subsection we saw an operation that removed facilities. We could show that as long as the facility cost of our solution was sufficiently high, greater than the assignment cost plus the optimum solution, we can improve the solution. Section 2.2 showed that by considering a different operation, we can improve the solution as long as the assignment cost is greater than the optimum solution. Consider the algorithm: we try to apply either of the moves, as long as we can improve the solution. When we are forced to stop,

$$F \leq C + F_{\mathrm{SOL}} + C_{\mathrm{SOL}} \quad \text{and} \quad C \leq F_{\mathrm{SOL}} + C_{\mathrm{SOL}}$$

Thus we can bound $F + C$ by thrice $F_{\mathrm{SOL}} + C_{\mathrm{SOL}}$. This gives us a factor 3 approximation. However, in the pure 'Add' heuristic we never reconsidered a facility once we added it. This does not hold any more. We have to guarantee that the improvement in each step is significant, thereby requiring a few steps in all. We can choose the facility and the improvement step ('Add' or 'Delete') which gives the maximum improvement. Since there are at most n facilities, the facility with maximum improvement in the 'Add' heuristic must have improvement $\frac{1}{n}(C - F_{\mathrm{SOL}} - C_{\mathrm{SOL}})$.

Similarly the facility with maximum improvement in the 'Delete' heuristic must have improvement $\frac{1}{n}(F - C - F_{\mathrm{SOL}} - C_{\mathrm{SOL}})$. Since the maximum of two quantities is greater than any convex combination of them, the maximum improvement is at least

$$\frac{1}{3}\frac{1}{n}(F - C - F_{\mathrm{SOL}} - C_{\mathrm{SOL}}) + \frac{2}{3}\frac{1}{n}(C - F_{\mathrm{SOL}} - C_{\mathrm{SOL}}) \le \frac{1}{3n}(F + C - 3(F_{\mathrm{SOL}} + C_{\mathrm{SOL}}))$$

Thus in each step the improvement is a fraction of the cost in excess of thrice the optimum (any feasible solution). We will not present the proof here, but after $3n$ such steps, this cost in excess of thrice the optimum, would have decreased by a constant factor. Thus after $O(n \log n/\epsilon)$ steps, we would have a $3 + \epsilon$ approximation, assuming we started from a solution which was at most a polynomial factor of the optimum. We will defer the discussion of finding such a solution till the next section, where we will present a complete algorithm, which unifies the two strategies.

In each such step, the improvement for a candidate facility can be computed in $O(n)$ time, thus requiring $O(n^2)$ time to find the facility with the maximum improvement. Thus we would have an algorithm with running time $O(n^3 \log n/\epsilon)$.

In the next section we will try to reduce the running time. An interesting consequence will be an improvement in the approximation guarantee. Notice that the bound we proved for F involved the term C. If we replace the term C by C_{SOL}, the approximation factor we can prove will not change. But, such an algorithm can be used to get tradeoff results for the facility location problems, where the guarantees on the facility and assignment costs are separate. We would use this algorithm to get a combinatorial bicriteria approximation for the k-median Problem. These tradeoff results become useful in proving approximation guarantees for problems which can be reduced to facility location, but with different approximation preserving guarantees on facility and assignment costs. Also, in both the operations considered, we were adding a facility first. It is a natural question, if we need to consider two different procedures, and not one.

2.4 An Unified Local Search

In the last two sections we saw two local operations which involved adding a facility first. In this section, we will show how to integrate these two local improvement steps into a single operation, and surprisingly, the algorithm will require fewer number of steps.

Assume that we have chosen a facility i to add. We consider the question, what is the best possible improvement we can achieve from adding this facility. First, some demand locations can be reassigned to this newly added facility at i. Also, some of the other existing facilities can be closed, since the benefit from assigning the demand points to them, is less than the cost of the facility.

Interestingly, the latter will be accentuated by the reassignment of demand points to the newly open facility. Thus we will first perform the reassignment to the candidate facility, and subsequently any deletions.

2.4.1 Local Improvement Step

1. Assume i to be the candidate facility to be added, and the set of existing facilities be $\mathcal{F}$. Initially $Gain(i) = -f_i$.

2. For every demand point j, assigned to the facility denoted by $\sigma(j)$, if i is closer to j than $\sigma(j)$ ($c_{ij} < c_{\sigma(j)j}$) mark j for reassignment to i, and

$$Gain(i) \leftarrow Gain(i) + d_j(c_{\sigma(j)j} - c_{ij})$$

3. After all the demand locations have been considered, for each facility $i' \in \mathcal{F}$, consider deleting i' and reassigning the demand locations assigned to i' (and not marked for reassignment in the last step) to the candidate facility i. If the total cost decreases add this decrease to $Gain(i)$, and mark i' for deletion.

4. If $Gain(i)$ is positive we can open a facility at i, delete the marked facilities, and reassign demand locations as mandated by the two steps above.

Observe, in the above it is possible that $i \in \mathcal{F}$. In that case, the $-f_i$ added initially will cancel out later, since we would delete i (as we iterate through $\mathcal{F}$), and reassign the demands to i again, with a saving of f_i. This seems to be a roundabout way of achieving nothing, but is not. In picking $i \in \mathcal{F}$, we also allow reassignments of demand locations not assigned to i. This alleviates the shortcoming in the third step, were we reassigned the demands from a deleted facility, completely, to the candidate facility. It is feasible that reassigning the demands in the best possible fashion (to their respective nearest open facilities) results in a better solution. However, we noted that such a procedure requires more time. Thus by allowing $i \in \mathcal{F}$, we improve the time required in the local improvement step, which we will see later, as well as remove the shortcoming of not performing the best reassignment.

If we ignore the third step in the local improvement, it becomes a simple 'Add' strategy. The delete steps only increase the gain function, therefore the following is immediate from proof of Lemma 2.2.1:

Lemma 2.4.1 *If we have a solution of assignment cost C, and there exists a solution of assignment cost C_{SOL} and facility cost F_{SOL}, then*

$$\sum_i Gain(i) \geq C - C_{\mathrm{SOL}} - F_{\mathrm{SOL}}$$

The next lemma is a consequence of merging the operations together:

Lemma 2.4.2 *If we have a solution of facility cost F, and there exists a solution of assignment cost C_{SOL} and facility cost F_{SOL}, then*

$$\sum_i Gain(i) \geq F - F_{\mathrm{SOL}} - 2C_{\mathrm{SOL}}$$

Proof: Let us denote the assignment cost of the solution by C. As in the local improvement step, a demand location is assigned to $\sigma(j)$ in our solution. Let $\hat{\sigma}(j)$ denote the facility to which the demand point j is assigned, in the solution SOL. Let $\mathcal{F}_{\mathrm{SOL}}$ denote the set of facilities in SOL, and let $m(i)$ denote the facility in $\mathcal{F}_{\mathrm{SOL}}$ closest to i. As we have seen in the previous proofs, we will define a new step which will change the existing solution by $Gain'()$. The change from these steps will be less than $Gain()$, and performed suitably, will result in the solution SOL.

For $i \in \mathcal{F}_{\mathrm{SOL}}$, define $Gain'(i)$ to the the result of the following operation:-

1' Initially $Gain'(i) = -f_i$.

2' For all demand locations j with $\hat{\sigma}(j) = i$, reassign them to i, and add the change to $Gain'(i)$ thus

$$Gain'(i) = -f_i + \sum_{j, \hat{\sigma}(j)=i} d_j(c_{\sigma(j)j} - c_{ij})$$

3' For all $i' \in \mathcal{F}$ with $m(i') = i$, delete i' and reassign the demand (remaining from previous step) to i.

For $i \notin \mathcal{F}_{\mathrm{SOL}}$, $Gain(i)$ is defined to be 0. Thus $Gain'(i)$ can be written as

$$Gain'(i) = -f_i + \sum_{j, \hat{\sigma}(j)=i} d_j(c_{ij} - c_{\sigma(j)j}) + \sum_{i' \in \mathcal{F}, m(i')=i} \left(f_{i'} + \sum_{j, \sigma(j)=i', \hat{\sigma}(j) \neq i} d_j(c_{i'j} - c_{ij}) \right)$$

If we now sum $Gain'(i)$, over $i \in \mathcal{F}_{\mathrm{SOL}}$, the first term adds up to $-F_{\mathrm{SOL}}$. For the second term, we can rewrite the sum as:

$$\sum_{i \in \mathcal{F}_{\mathrm{SOL}}} \sum_{j, \hat{\sigma}(j)=i} d_j(c_{\sigma(j)j} - c_{ij}) = \sum_j d_j(c_{\sigma(j)j} - c_{\hat{\sigma}(j)j}) = C - C_{\mathrm{SOL}}$$

The last term can be divided into two parts, the first of which is simple:

$$\sum_{i \in \mathcal{F}_{\mathrm{SOL}}} \left(\sum_{i' \in \mathcal{F}, m(i')=i} f_{i'} \right) = \sum_{i' \in \mathcal{F}} f_{i'} = F$$

The other part is

$$\sum_{i \in \mathcal{F}_{\mathrm{SOL}}} \left(\sum_{i' \in \mathcal{F}, m(i')=i} \left[\sum_{j, \sigma(j)=i', \hat{\sigma}(j) \neq i} d_j(c_{i'j} - c_{ij}) \right] \right)$$

Now $c_{ij} \geq c_{ii'} + c_{i'j}$. Since $m(i') = i$, we have $c_{\hat{\sigma}(j)i'} \geq c_{ii'}$. Combining these and by triangle inequality, we have

$$c_{i'j} - c_{ij} \geq -c_{ii'} \geq -c_{\hat{\sigma}(j)i'} \geq -c_{\hat{\sigma}(j)j} - c_{i'j}$$

Thus the part we needed to bound is bounded below by

$$\sum_{i \in \mathcal{F}_{\mathrm{SOL}}} \left(\sum_{i' \in \mathcal{F}, m(i')=i} \left[\sum_{j, \sigma(j)=i', \hat{\sigma}(j) \neq i} d_j(-c_{\hat{\sigma}(j)j} - c_{i'j}) \right] \right)$$

If we are interested in bounding the term from below, we can drop the restriction $\hat{\sigma}(j) \neq i$, since it would only add negative terms. Thus the bound is,

$$\sum_{i \in \mathcal{F}_{\text{SOL}}} \left(\sum_{i' \in \mathcal{F}, m(i')=i} \left[\sum_{j, \sigma(j)=i'} d_j(-c_{\hat{\sigma}(j)j} - c_{i'j}) \right] \right) = -\sum_j d_j(c_{\hat{\sigma}(j)j} + c_{\sigma(j)j})$$

Thus the bound evaluates to $-C_{\text{SOL}} - C$. Putting all these together we have,

$$\sum_{i \in \mathcal{F}_{\text{SOL}}} gain'(i) \geq -F_{\text{SOL}} + C - C_{\text{SOL}} + F - C_{\text{SOL}} - C$$

We are almost done, except we have to argue that $Gain(i) \geq Gain'(i)$. To show this we will try to show that all the positive terms that contributed to $Gain'(i)$ will also contribute to $Gain(i)$. However the proof will not be that simple.

Consider the demand locations $\{j | \hat{\sigma}(j) = i\}$. If for such a location j we did not delete $\sigma(j)$ (in the subsequent step) in computing $Gain'(i)$, we will argue as before. If c_{ij} was less than $c_{\sigma(j)j}$, we would have added the same positive term to $Gain(i)$ and $Gain'(i)$. Otherwise, we added a negative term to $Gain'(i)$, and 0 to $Gain(i)$.

The demand locations j with $\hat{\sigma}(j) = i$, for which we deleted $\sigma(j)$ in the next step, in computing $Gain'(i)$ will be accounted differently. We will consider the contributions from these reassignments, along with the contribution from deleting $\sigma(j)$. Consider the net contribution to $Gain'()$ from such an $i' \in \mathcal{F}$.

$$\sum_{j, \sigma(j)=i', \hat{\sigma}(j)=i} d_j(c_{i'j} - c_{ij}) + f_{i'} + \sum_{j, \hat{\sigma}(j) \neq i, \sigma(j)=i'} d_j(c_{i'j} - c_{ij}) = f_{i'} + \sum_{j, \sigma(j)=i'} d_j(c_{i'j} - c_{ij})$$

Now the right hand side can be rewritten as:

$$\left[f_{i'} + \sum_{j, \sigma(j)=i', c_{i'j} \leq c_{ij}} d_j(c_{i'j} - c_{ij}) \right] + \left[\sum_{j, \sigma(j)=i', c_{i'j} > c_{ij}} d_j(c_{i'j} - c_{ij}) \right]$$

The first term (involving $f_{i'}$ and a sum) is the change in cost in considering deletion of i' while computing $Gain(i)$. The second term (just the sum) is the contribution from reassignment to i in computing $Gain(i)$.

We definitely added the second term to $Gain(i)$. If the first term were positive, we also added it to $Gain(i)$, and thus the total contribution to $Gain(i)$ is the same as the contribution to $Gain'(i)$. On the other hand, if the first term were negative, we would not have considered deleting i' while computing $gain(i)$ and would not have added it to $Gain(i)$.

Thus in all cases, the addition to $Gain'(i)$ was less than to $Gain(i)$. The rest of the terms added to $Gain(i)$ will be positive, thus $Gain(i) \geq Gain'(i)$. Summing over all the facilities, since for $i \notin \mathcal{F}_{\mathrm{SOL}}$, $Gain(i) \geq 0$, the lemma follows. ■

Lemma 2.4.3 *Each improvement step can be performed in time $O(n)$.*

Proof: We consider each demand location, if we maintain the facility to which it is assigned to, in $O(1)$ time we can decide if it should be marked for reassignment; and if so, update $Gain()$. At the same time, we can keep track of the change in cost of deleting the facility $\sigma(j)$, the facility to which j is assigned to.

Initially for all facilities in the current solution, the change of cost in deleting a facility is the cost of the facility. Now as we iterate through the demand points, if the candidate facility, i in the algorithm, is farther than $\sigma(j)$, the increase in cost is subtracted from the change of cost in deleting $\sigma(j)$. (Otherwise the difference is added to $Gain(i)$, since j would be reassigned.) We consider a facility for deletion, if the change in cost is positive. Clearly, all this can be done in time $O(n)$, the number of locations in the graph. ■

2.4.2 The Local Search Algorithm

The algorithm is quite simple. We will start from an initial solution and improve it repeatedly. The details follow:

Initial Solution: Order the facilities in order of nondecreasing facility cost. Let them be $i_1, i_2, \ldots, i_n$. First open only the first facility i_1, and assign all demand locations to it.

Next, consider the facilities $i = i_2, i_3, \ldots$ in order. Compute $Gain(i)$, if it is positive, include the facility i in the initial solution. Repeat with the next facility in the order.

Improvement Step: Repeat this step $O(n \log \frac{n}{\epsilon})$ times: pick a facility i uniformly at random. Do not distinguish if i is already in the solution or otherwise. Compute $Gain(i)$, and perform the requisite changes if $Gain(i)$ is positive.

Lemma 2.4.4 *The initial solution is at most n times any feasible solution.*

Proof: Consider the most expensive facility in SOL. Let this be i_ℓ, in the ordering we used in finding the initial solution. Clearly $\ell \leq n$.

Now assume we opened all the facilities $i_1, i_2, \ldots, i_\ell$. our cost will be the cost of these facilities, plus the assignment cost. The cost of these facilities will be at most ℓ times the facility cost of SOL. The assignment cost of this solution, by Fact 2.2.1, will be less than the assignment cost of SOL; since every demand location will have a facility at least as close to it as it had in SOL. Thus the cost of a solution which opened the facilities $i_1, i_2, \ldots, i_\ell$ will be at most $\ell \leq n$ times the feasible solution.

The cost of our initial solution, will definitely be lower. This is because if we ignore the improvement from deleting facilities, opening facilities $i_1, i_2, \ldots, i_\ell$ correspond to adding facilities. Since we will only add a facility if it decreases the total cost, our cost will be less than opening the first ℓ facilities. ■

2.4.3 Analysis of Running time

We will show that the local search algorithm shown above produces a solution of desired quality after $O(n \log \frac{n}{\epsilon})$ iteration steps, with high probability. We will be analyzing a random process and the next lemma will be useful. In a later section we will be using this lemma in two different random processes, and hence we separate the lemma from the proof of the running time required for the local search algorithm.

Lemma 2.4.5 *Suppose X_r is a sequence of independent random variables such that each X_r has n possible values, each non-negative and equally likely such that the values sum up to at least α.*

$$\Pr\left[b > \sum_{r=1}^{\ell} X_r\right] \leq e^{\frac{b}{a}(\log\frac{a\ell}{nb}+1-\frac{a\ell}{nb})}$$

Proof: Let $X = \sum_{r=1}^{\ell} X_r$. Let z be a parameter specified later.

$$\begin{aligned}\Pr[X<\beta] &= \Pr[e^{-zX} < e^{-z\beta}] \\ &\leq \frac{\mathrm{E}[e^{-zX}]}{e^{-z\beta}} \\ &= \frac{\prod_{r=1}^{\ell}\mathrm{E}[e^{-zX_r}]}{e^{-z\beta}}\end{aligned}$$

Let the n possible values of X_r be $X_r(1), \ldots, X_r(n)$. Since $e^{-za_1} + e^{-za_2} < 1 + e^{-z(a_1+a_2)}$ for positive $z, a_1, b_1 \geq 0$,

$$\mathrm{E}[e^{-zX_r}] = \frac{1}{n}\sum_{q=1}^{n} e^{-zX_i(q)} \leq \frac{1}{n}(n-1+e^{-z\sum_{q=1}^{n}X_i(q)}) \leq 1-\frac{1}{n}+\frac{1}{n}e^{-za}$$

Using this, and for $a, b \geq 0$ since $(1+a)^b \leq e^{ab}$, we have

$$\Pr[X<b] \leq \frac{\prod_{r=1}^{\ell}\mathrm{E}[e^{-zX_r}]}{e^{-zb}} \leq \frac{\prod_{r=1}^{\ell}(1+\frac{1}{n}(e^{-za}-1))}{e^{-zb}} \leq e^{\frac{\ell}{n}(e^{-za}-1)+zb}$$

Setting $z = \frac{1}{a}\log\frac{a\ell}{nb}$, proves the lemma. ∎

Now we are ready to prove the following theorem about the local search algorithm,

Theorem 2.4.1 *The algorithm produces a solution of cost at most $(1+\epsilon)(2F_{\mathrm{SOL}} + 3C_{\mathrm{SOL}})$ in $O(n(\log\frac{n}{\epsilon}))$ improvement steps (running time $O(n^2(\log\frac{n}{\epsilon}))$) with high probability ($1-1/p(n)$ for some polynomial $p(n)$).*

Proof: The proof proceeds roughly by showing that $C+F$ decreases rapidly with every step. Let $Cost_t$ denote the cost of the solution before the t'th iteration. Denote by $Gain_t(i)$ the benefit we computed with i as a candidate facility for the t'th iteration. Notice i was chosen uniformly at random. Let ℓ be the last iteration for which $Cost_\ell$ was greater than $(1+\epsilon)(2F_{\text{SOL}}+3C_{\text{SOL}})$. We would be interested in showing $\mathrm{E}[\ell] \le O(n\log\frac{n}{\epsilon})$.

From Lemmas 2.4.1 and 2.4.2, we have

$$\sum_i Gain_t(i) \;\ge\; \frac{1}{2}(C+F-(2F_{\text{SOL}}+3C_{\text{SOL}})) = \frac{1}{2}(Cost_t-(2F_{\text{SOL}}+3C_{\text{SOL}}))$$

Define $X_t(i)$ as $Gain_t(i)/(Cost_t(i)-(2F_{\text{SOL}}+3C_{\text{SOL}}))$, then $\sum_i X_t(i) \ge \frac{1}{2}$.

Suppose i_t was the facility chosen at the t'th iteration,

$$Cost_t - Cost_{t+1} = Gain_t(i_t) = X_t(i_t)(Cost_t(i)-(2F_{\text{SOL}}+3C_{\text{SOL}}))$$

Since $\log\frac{1}{z} \ge 1-z$, the above equation can be modified as:

$$\log\frac{Cost_t-(2F_{\text{SOL}}+3C_{\text{SOL}})}{Cost_{t+1}-(2F_{\text{SOL}}+3C_{\text{SOL}})} \ge \frac{Cost_t-Cost_{t+1}}{Cost_t-(2F_{\text{SOL}}+3C_{\text{SOL}})} = X_t(i_t)$$

Summing the above equation for $t=1,\ldots,\ell-1$.

$$\log\frac{Cost_1-(2F_{\text{SOL}}+3C_{\text{SOL}})}{Cost_\ell-(2F_{\text{SOL}}+3C_{\text{SOL}})} \ge \sum_{t=1}^{\ell} X_t(i_t)$$

Now $Cost_1 \le n(C_{\text{SOL}}+F_{\text{SOL}})$ and $Cost_\ell \ge (1+\epsilon)(2F_{\text{SOL}}+3C_{\text{SOL}})$. Thus

$$\log\frac{n}{\epsilon} > \sum_{t=1}^{\ell} X_t(i_t)$$

Setting $a=1/2$ and $b=\ln(n/\epsilon)$, and $\ell = 2c'n\log\frac{n}{\epsilon}$ in Lemma 2.4.5,

$$\Pr[\log\frac{n}{\epsilon} > \sum_{t=1}^{\ell} X_t(i_t)] \le e^{2\log\frac{n}{\epsilon}(\log c'+1-c')} \le \left(\frac{\epsilon}{n}\right)^{2(c'-1-\log c')}$$

which proves the theorem. ∎

Thus we have a factor 3 approximation for the facility location problem in time $O(n^2\log\frac{n}{\epsilon})$. We will see in the next section, how to extend this result to the k-median Problem.

2.5 Bicriteria Bounds and The k-Median Problem:

In this section, we will show that the unified local search step actually produces a solution in which *both facility and assignment costs can be bounded individually.* This is much stronger than bounding the sum, and will be very useful in several different context.

The following theorem is true with respect to *any feasible* solution, not just the optimal. Obviously this cannot be true for any single solution output by our algorithm; consider the two extreme solutions corresponding to lowest facility cost and highest assignment cost and the reverse situation. Thus our algorithm will actually output $O(\frac{n}{\epsilon}\log n)$ solutions, and we will choose the most appropriate solution from this collection depending on the application. This dependence on ϵ appears unavoidable since there can be exponentially many feasible solutions, and the parameter ϵ allows us to only consider polynomially many.

Theorem 2.5.1 *If there exists a feasible solution* SOL *with facility cost* F_{SOL} *and assignment cost* C_{SOL}*, then in time* $O(\frac{n^2}{\epsilon}\log n)$ *we can find a solution which satisfies*

$$F \leq (1+\epsilon)(F_{\text{SOL}} + 2C_{\text{SOL}}) \quad \textit{and} \quad C \leq (1+\epsilon)(C_{\text{SOL}} + F_{\text{SOL}})$$

Proof: From Theorem 2.4.1 we know that after time $O(n^2 \log \frac{n}{\epsilon})$ time we would have,

$$F + C \leq (1+\epsilon)(2F_{\text{SOL}} + 3C_{\text{SOL}}) \tag{2.1}$$

We would now keep applying the improvement steps, till for the first time the conditions in the theorem are true.[3] Let this be the ℓ'th iteration, after the condition in Equation 2.1 has been achieved. We will use the same notation as in proof of Theorem 2.4.1. In the following discussion, iteration t would refer to the t'th iteration in the phase after Equation 2.1 has been achieved.

For the iterations $t < \ell$, since either F or C is large, by Lemma 2.4.1 and Lemma 2.4.2 we have,

$$\sum_i Gain_t(i) \geq \max[\ F - (F_{\text{SOL}} + 2C_{\text{SOL}}), C - (F_{\text{SOL}} + F_{\text{SOL}})] \geq \epsilon(F_{\text{SOL}} + C_{\text{SOL}})$$

[3] Beyond this point the solution may change, but as we mentioned, we would output several solutions, and one of them will satisfy the conditions of the theorem.

Define $X_t(i) = Gain_t(i)/(F_{\mathrm{SOL}} + C_{\mathrm{SOL}})$. Thus $\sum_i X_t(i) \geq \frac{1}{2}$.

Since the total cost always decreases, and is at most $(1+\epsilon)(2F_{\mathrm{SOL}} + 3C_{\mathrm{SOL}})$, the sum of the improvements over the ℓ iterations can be at most as much as this. This is at most $\frac{4}{\epsilon}(F_{\mathrm{SOL}} + C_{\mathrm{SOL}})$ assuming $\epsilon < 1$. Now we apply Lemma 2.4.5, with $b = \frac{4}{\epsilon}$, and $a = \frac{1}{2}$, and $\ell = 8c'\frac{n}{\epsilon}\log n$,

$$\Pr[\sum_t X_t(i_t) < \frac{4}{\epsilon}] \leq e^{\frac{8}{\epsilon}(\log c' + 1 + \log\log n - c'\log n)}$$

Thus within $O(\frac{n^2}{\epsilon}\log n)$ time of Equation 2.1 being satisfied which is only $O(n^2 \log \frac{n}{\epsilon})$, with high probability the condition in the theorem will be satisfied. ■

2.5.1 An Approximation for the k-Median Problem

The k-median problem is fairly similar to the facility location problem, except that the facility cost is bounded, and all facilities cost the same. Thus the objective is to minimize the assignment cost. This translates to a the number of facilities being restricted, by the parameter k, which is part of the input.

Suppose we have a k-median instance, and we can guess the cost of the optimum solution, OPT}. We only need to guess this value upto a power of $1+\epsilon'$, and hence the number of guesses will be polynomially bounded. Assume the guessed value is ζ and we set the cost of opening a facility at any location to be $\zeta/(\gamma' k)$ for a parameter γ'.

In the facility location problem we have created, we have a feasible solution corresponding to the optimum solution to the original k-median problem, of facility cost $F_{\mathrm{SOL}} = \zeta/\gamma'$ and assignment cost $C_{\mathrm{SOL}} = |\mathrm{OPT}\}|$, with $\zeta \leq |OPT| \leq \zeta(1+\epsilon')$. By Theorem 2.5.1 there exists a solution with facility cost F and assignment cost C, such that

$$F \leq (1+\epsilon')(F_{\mathrm{SOL}} + 2C_{\mathrm{SOL}}) \quad \text{and} \quad C \leq (1+\epsilon')(F_{\mathrm{SOL}} + C_{\mathrm{SOL}})$$

Thus we have $F \leq (1+\epsilon')^2(1+2\gamma')\zeta/\gamma'$, which denotes at most $(1+\epsilon')^2(1+\gamma')k$ facilities, or medians. The assignment cost is at most

$$C \leq (1+\epsilon')(F_{\mathrm{SOL}} + C_{\mathrm{SOL}}) \leq (1+\epsilon')(1+\frac{1'}{\gamma})|\mathrm{OPT}\}|$$

We can choose $\gamma = (1+\epsilon')^2(1+2\gamma') - 1$, and $\epsilon = O(\epsilon'/\gamma)$ sufficiently small, such that $(1+\epsilon')(1+\frac{1}{\gamma}')$ can be expressed as $(1+\frac{1}{\gamma}+\epsilon)$. Notice in the above, we would have to consider $O(\frac{n}{\epsilon'}\log n)$ solution, and choose the best solution with at most $1+2\gamma$ times k medians. Thus we can claim the following theorem:

Theorem 2.5.2 *In time $O(\frac{n^2}{\gamma\epsilon}\log n)$ we can approximate the k-median problem with $k(1+2\gamma)$ medians upto a factor of $(1+\frac{1}{\gamma}+\epsilon)$.*

This is interesting since this is the first bicriterion approximation that achieves a factor arbitrary close to 1, of course with many more medians. This was not possible in any of the previous tradeoff results for this problem, see [LV92a] and [KPR98]. This allows for a local search based clustering algorithm. In fact in several ways the above algorithm is similar to the KMEANS algorithm, used extensively in clustering literature, see [Har79, JD88, DH73].

However the local search algorithm we just saw is more general than the KMEANS algorithm that only considers exchanges. By setting costs to the medians, we also allow to move into solution spaces with more than k medians. This is an interesting generalization, typical of Lagrangian Relaxation methods to solve a problem, in which one or more constraints which are difficult to satisfy are added to the objective function with suitable multipliers. The resulting problem is usually solved easily, but the constraints which were added to the objective function may not be satisfied. The method readjusts the multipliers, penalizing the violated constraints more, and searches till a solution is found. In Section 4.3 we will see the concept of Lagrangian Relaxation being used more formally to derive a constant factor combinatorial approximation algorithm for the k-median problem.

However the bicriterion algorithm allows a simple heuristic for the k-median problem. One advantage of this local search algorithm is its space overhead; primal dual algorithms or linear programming based algorithms typically require more space. We will return to this issue in Chapter 11.

Chapter 3

A Linear Programming Approach

3.1 Introduction

In the previous chapter, we saw a simple, efficient heuristic for the facility location problem using iterative improvement. However, the approach has a drawback. We did not produce a lower bound, independent of the proof of the approximation factor. We would not be aware of the actual quality of our solution, but only can assert that the optimum solution can be one third of our solution.

One of the most important aspects of optimization is a lower bound. Assuming we have instance specific lower bounds, that are not derived from proof of approximation guarantee of the final solution, we can provide instance based guarantees. Of course, we would be only be able to prove worst case bounds, but if we were faced with a real instance, an independent lower bound is of great value. It is conceivable that the final solution constructed will not conform to the exact worst case of the analysis, and has development of heuristics have often demonstrated, will only be a few percent away from the optimum solution. In fact the lower bound is the proof of optimality, if we have an optimum solution, even by accident.

Linear Programming has been one of the most important tools in optimization from the above viewpoint. Posing a problem as an integer linear program and solving its fractional relaxation to get a lower bound has been a first step, in exact optimization. In context of approximation algorithms, the fractional solution of a linear program provides a lower bound if we ensure that the optimum solution satisfies the constraints. The objective from this point is to construct an integral solution which is a (hopefully) small factor of the lower bound using the fractional solution. This step is referred to as the 'rounding' step. This approach bas been used for over two decades to bound performances of approximation algorithms, see [Cha79, Lov75, CFN77, Wol80].

However in certain cases, the specific linear program formulated may not be sufficiently strong. The fractional solution may be significantly smaller than the optimum solution. This is referred to as the 'integrality gap' of the formulation, and is the best approximation factor we can hope to prove. Examples of integrality gap are found for particular problem and formulation pairs. The minimum spanning tree problem for instance, has an integrality gap in the simplest formulation based on cuts. However, a small integrality gap justifies the use of linear programming to design a heuristic, since the lower bound obtained for the instance will be reasonable. The approximation guarantee would in fact prove an upper bound on the integrality gap, if the final solution to the problem is compared to the fractional solution for the same from the linear program.

In this chapter we will consider the linear programming approaches to facility location problems and variants. We will use the strength of the linear programming to give the first known constant factor approximation for the k-median problem. However we will start from linear programming and rounding schemes for the facility location problem. In particular we would review techniques of agglomerating fractional facilities known as 'filtering' in Section 3.2.

The technique of 'filtering' was first introduced in [LV92b]. In [STA97], this technique was used to give the first constant factor (3.16) approximation for the facility location problem using the linear programming formulation of [Bal66]. Filtering augmented with local search, in [GK98], technique gave a 2.47-approximation. The 1.736-approximation algorithm in [Chu98], which is used as a component in designing the best known approximation algorithm, also draws in spirit from the filtering analysis. In fault tolerant facility location, the filtering technique (augmented with local search) provides the best known result, see Chapter 8. This idea of agglomeration, and associated arguments about distance bounds would be a part of the approximation algorithm for the k-median problem.

The k-median problem on trees was shown to be solvable in polynomial time in [KH79, Tam96]. The dynamic programming algorithm in [Tam96], easily extends to a collection of trees. The first non-trivial approximation of $O(\log^2 n)$ for arbitrary metrics, using exactly k medians, follow from result of [Bar96]. This was achieved by embedding the graph metric in a tree probabilistically, such that distances are preserved up to small factors in expectation. This was improved in [Bar98] to $O(\log n \log\log n)$ and subsequently derandomized in [CCGG98, CCG+98]. However an embedding preserves all distances in the metric, and the question arises if preserving all distances is necessary for the approximation. A location is only interested in the distance to the nearest median. All the known results were based on trees, a natural question arises if the fractional assignments can be in form of a tree. [dVPV95] showed that a linear program on a metric induced by a forest has an integrality gap of 2. The question reduces to: if the fractional assignment on an arbitrary metric could be made to resemble the fractional assignment on a metric induced by a forest. The answer is yes, if every location had at least $\frac{1}{2}$ median fractionally. Since in that case, a location will be assigned to itself and its nearest neighbor, fractionally. If we ignore the self-assignments, the assignment graph resembles a tree, since the outdegree of every vertex in the assignment graph is one. There may be at most one loop per tree, defined by mutual nearest neighbors. Subsequently this fractional solution on the 'tree resembling' graph can be rounded. However the condition that every location has at least half a median can be used to improve the result.

In the Section 3.3 we will present the linear programming and rounding technique for the k-median problem. We would consider $\frac{1}{2}$-integral solutions as a stepping stone for the result. In fact the analysis of the algorithm would depend on the integrality gap of a $\frac{1}{2}$-integral solution, for which we will provide tight upper and lower bounds in Section 3.6. This technique of using $\frac{1}{2}$ integral solution will also be useful in approximating facility location restricted to fixed number of facilities, and the capacitated versions. See Section 6.4 for the capacitated k-median problem.

The results in chapter are also in [STA97, CGTS99, Cha00].

3.2 Filtering and Facility Location

As mentioned above, we will formulate the facility location problem as an integer linear program (ILP), solve its fractional relaxation and modify (or round) the fractional solution to an integral solution. The ILP for the facility location problem is given below:

$$\begin{aligned}
\min \sum_i y_i f_i + \sum_{ij} x_{ij} c_{ij} d_j & & \\
x_{ij} & \leq y_i & \text{for each } i,j \\
\sum_i x_{ij} & \geq q & \text{for each j} \\
x_{ij}, y_i & \in \{0,1\} & \text{for each } i,j
\end{aligned}$$

Here y_i indicates whether facility i is chosen in the solution and x_{ij} indicates whether the demand location j is assigned to facility i. The cost of facility i is f_i, and the demand at location j is d_j. c_{ij} is the distance from i to j.

We will deviate from the presentation of [LV92b, STA97], and use the presentation of [GK98], which will allow us a derandomized algorithm. The basic essence is the same, the change is in definition of certain parameters.

Assume we have a solution to the fractional relaxation with $x_{ij}, y_i \geq 0$. We will modify the solution slightly. For every j order the facilities i in non-decreasing order of distance. Let the list be $i_1, \ldots, i_p$. We will set $\bar{x}_{i_1 j} = y_{i_1}$ and so on, for the facilities in this list, till $\sum_{l=1}^{r} y_{i_l} \geq 1$ for the first time. We assign $\bar{x}_{i_r j} = 1 - \sum_{l=1}^{r-1} y_{i_l}$.

What we achieved in the above step is: a location is assigned to its closest facility, with $x_{ij} = y_i$ whenever possible. From Fact 2.2.1, we can easily argue that the assignment cost of the solution $(\bar{x}, y)$ is the best possible. Thus the cost of the new solution is at most the cost of th solution returned by the linear program.

Let F_{LP} denote the fractional facility cost and C_{LP} the fractional assignment cost. Clearly $F_{\text{LP}} = \sum_i y_i f_i$. The assignment distance of location j is denoted by

$$\bar{C}_j = \sum_i \bar{x}_{ij} c_{ij} \quad \text{and} \quad C_{\text{LP}} = \sum_j \bar{C}_j d_j$$

The filtering step consisting of defining a neighborhood $B(\zeta, j)$ for each demand location, where ζ is a parameter in the interval $[0,1]$, as follows:

For every j, consider the facilities with $\bar{x}_{ij} > 0$, let them be $i_1, \ldots, i_r$. Define $B(\zeta, j)$ to be $\{i_1, \ldots, i_p\}$ such that $\sum_{l=1}^{p} y_i \geq \zeta$.

For $\zeta \leq 1$, observe that $\sum i \in B(\zeta, j)\bar{x}_{ij} \geq \zeta$, otherwise we can show $\sum_i \bar{x}_{ij} c_{ij} d_j > \bar{C}_j$ which is not true, by definition.

The rounding step orders the demand locations j, in increasing order of $\bar{C}_j(\zeta)$. Initially all the demand locations are unmarked. It picks the first demand location j' in the list, which is unmarked, and builds a facility at $i' \in B(\zeta, j')$ which has the lowest f_i value for all $i \in B(\zeta, j')$. The demand location j' is assigned to i' and mark j'. For any other demand location j, such that $B(\zeta, j)$ intersects $B(\zeta, j')$ and has not been considered already ($\bar{C}_{j'}(\zeta) \leq \bar{C}_j(\zeta)$) assign j to i', and mark j.

Lemma 3.2.1 $f_{i'} \leq \dfrac{1}{\zeta} \displaystyle\sum_{i \in B(\zeta, j')} y_i f_i$

Proof: For the above discussion, since $f_{i'} \leq f_i$ for $i \in B(\zeta, j')$,

$$\sum_{i \in B(\zeta, j')} y_i f_i \geq \left(\sum_{i \in B(\zeta, j')} y_i \right) f_{i'} \geq \left(\sum_{i \in B(\zeta, j')} \bar{x}_{ij'} \right) f_{i'} \geq (\zeta) f_{i'}$$

which proves the lemma. ■

Define $\bar{C}_j(\zeta)$ to be $\max\{c_{ij} | i \in B(\zeta, j)\}$.

Lemma 3.2.2 *$\bar{C}_j(\zeta) \leq \frac{1}{1-\zeta}\bar{C}_j$, moreover, $\int_0^1 \bar{C}_j(\zeta) d\zeta = \bar{C}_j$.*

Proof: The first part follows from the fact that if $\bar{C}_j(\zeta) > \frac{1}{1-\zeta}\bar{C}_j$, then the demand location j is assigned to the farthest facility in $B(\zeta, j)$ at a distance $\frac{1}{1-\zeta}\bar{C}_j$. By the definition of $B(\zeta, j)$, this means that j is assigned to $1 - \zeta$ fractional facilities at distance $\frac{1}{1-\zeta}\bar{C}_j$. Thus the assignment distance of j cannot be $\bar{C}_j$.

The second part also follows from the definition of $B(\zeta, j)$. Since a particular facility i defines $\bar{C}_j(\zeta)$, over an interval of $\bar{x}_{ij}$. Thus the integral evaluates to $\sum_i \bar{x}_{ij} c_{ij}$, which is $\bar{C}_j$. ■

Now all the $B(\zeta, j')$ we consider, are disjoint. Thus the net facility cost we paid is at most $\frac{1}{1-\zeta}$ times the cost of the fractional facilities in these disjoint $B(\zeta, j')$. Thus the facility cost we paid is $\frac{1}{\zeta}$ times F_{LP}.

Each demand point j is either assigned to a facility in $B(\zeta, j)$, or to a facility in $B(\zeta, j')$ with $\bar{C}_j(\zeta) \geq \bar{C}_{j'}(\zeta)$. In the first case the assignment distance is at most $\frac{1}{1-\zeta}\bar{C}_j$. In the latter case, let i be the facility which is in both $B(\zeta, j)$ and $B(\zeta, j')$. Then

$$c_{i'j} \leq c_{i'j'} + c_{j'i} + c_{ij} \leq \bar{C}_{j'}(\zeta) + \bar{C}_{j'}(\zeta) + \bar{C}_j(\zeta) \leq 3\bar{C}_j(\zeta) \leq \frac{3}{1-\zeta}\bar{C}_j$$

Thus for all cases the assignment distance is at most $\frac{3}{1-\zeta}\bar{C}_j$. Actually Summing over the demand locations, the assignment cost $C(\zeta)$ is at most

$$\sum_j \frac{3}{1-\zeta}\bar{C}_j d_j = \frac{3}{1-\zeta}C_{\mathrm{LP}}$$

In fact we can also observe that

$$\int_0^1 C(\zeta)d\zeta = \int_0^1 \sum_j d_j 3\bar{C}_j(\zeta)d_\zeta = \sum_j d_j 3\int_0^1 \bar{C}_j(\zeta)d_\zeta = 3\sum_j d_j\bar{C}_j = 3C_{\mathrm{LP}}$$

Lemma 3.2.3 *[STA97] The facility cost of the above is at most $\frac{1}{\zeta}F_{\mathrm{LP}}$ and the assignment cost at most $C(\zeta) \leq 3\frac{1}{1-\zeta}C_{\mathrm{LP}}$, moreover $\int_0^1 C(\zeta)d\zeta = 3C_{\mathrm{LP}}$.*

Now we can choose the best ζ for the above result. We will bound the cost of choosing $\zeta \in [a, 1]$ in random, thus bounding the average choice of ζ in this interval. The best choice will clearly be better, and thus the cost of the best solution will be at most

$$\frac{1}{1-a}\left(\int_a^1 \frac{1}{\zeta}F_{\mathrm{LP}}d\zeta + \int_a^1 \sum_j 3\bar{C}_j(\zeta)d\zeta\right) \leq \frac{1}{1-a}\left(\log\frac{1}{a}F_{\mathrm{LP}} + 3C_{\mathrm{LP}}\right)$$

We relaxed the bounds on the integral in the second term, from 0 to 1. Setting $a = e^{-3}$,

Theorem 3.2.1 *[STA97] The facility location problem can be approximated upto a factor of* 3.16.

The filtering step converts the minsum problem to a minmax problem. Notice that the disjoint neighborhoods are in essence choosing an independent set in a graph. In the k-center problem, and similar bottleneck problems with minmax objective, this is a very useful technique. The central idea behind filtering is to accumulate enough fractional facility in a small neighborhood whose radius is bounded. However as observed in the above proof, the fractional facility in the neighborhood cannot be made 1, since the guarantee on the distance breaks down. Thus the filtering step has the feature of increasing the facility cost by some constant fraction.

3.3 Approximating The k-median Problem: An outline

The k-median problem can be stated as the following integer program, where the 0-1 variable y_i, $i \in N$, indicates if the location i is selected as a center, and the 0-1 variable x_{ij}, $i, j \in N$, indicates if location j is assigned to the center at i:

$$\text{minimize} \sum_{i,j \in N} d_j c_{ij} x_{ij} \tag{3.1}$$

subject to

$$\begin{aligned}
\sum_{i \in N} x_{ij} &= 1, && \text{for each } j \in N, && (3.2)\\
x_{ij} &\leq y_i, && \text{for each } i, j \in N, && (3.3)\\
\sum_{i \in N} y_i &\leq k, && && (3.4)\\
x_{ij} &\in \{0,1\}, && \text{for each } i, j \in N, && (3.5)\\
y_i &\in \{0,1\}, && \text{for each } i \in N. && (3.6)
\end{aligned}$$

The constraints (3.2) ensure that each location $j \in N$ is assigned to some center $i \in N$, the constraints (3.3) ensure that whenever a location j is assigned to a center i, then a center must have been opened at i, and (3.4) ensures that at most k centers are open. Our goal would be to argue that these assignments can be rearranged to resemble fractional assignments being induced on a tree metric. However this is only a conceptual tool useful in analysis, the actual algorithm will be the construction of appropriate trees from the solution of the linear program.

Consider the linear programming relaxation to the integer program (3.1)–(3.6), where the 0-1 constraints (3.5) and (3.6) are replaced, respectively, with

$$x_{ij} \geq 0, \qquad \text{for each } i, j \in N, \tag{3.7}$$

$$y_i \geq 0, \qquad \text{for each } i \in N. \tag{3.8}$$

Let $(\bar{x}, \bar{y})$ denote a feasible solution to the LP relaxation and let $\bar{C}_{LP}$ denote its objective function value. For each location $j \in N$, let $\bar{C}_j$ denote the cost incurred by this fractional solution for assigning (one client at) location j, i.e.,

$$\bar{C}_j = \sum_{i \in N} c_{ij} \bar{x}_{ij} \qquad \text{for each } j \in N. \tag{3.9}$$

Note that $\bar{C}_{LP} = \sum_{j \in N} d_j \bar{C}_j$. We will round the fractional solution $(\bar{x}, \bar{y})$ to a feasible solution to the integer program of cost at most $6\frac{2}{3}\bar{C}_{LP}$.

The outline of our approximation algorithm is as follows.

Step 1: Consolidating Locations First we will simplify the problem by reducing the number of locations with positive demand. We will not change the linear programming solution $(\bar{x}, \bar{y})$ but modify only the demands.

Let d' denote the new set of demands. This modification will have the following properties:

- The modification will not increase the cost of the fractional solution $(\bar{x}, \bar{y})$:

$$\bar{C}'_{LP} = \sum_{i,j \in N} d'_j c_{ij} \bar{x}_{ij} \leq \sum_{i,j \in N} d_j c_{ij} \bar{x}_{ij} = \bar{C}_{LP}.$$

- In the resulting instance, all locations with positive demand will be far from each other:

$$c_{ij} > 4 \max(\bar{C}_i, \bar{C}_j) \tag{3.10}$$

for each $i, j \in N$ such that $d'_i > 0$ and $d'_j > 0$.

- Each feasible integer solution for the modified instance with demands d'_j, $j \in N$, can be converted to a feasible integer solution for the original instance, at an added cost of at most $4\bar{C}_{LP}$.

Let $N' = \{j : d'_j > 0\}$ denote the set of locations with positive demand. We will show that this modified instance is simpler, e.g., $|N'| \leq 2k$.

Step 2: Consolidating Medians Next, we simplify the structure of the solution by consolidating nearby fractional centers. First, we modify the solution $(\bar{x}, \bar{y})$ to obtain a new solution (x', y') such that

$$\begin{aligned} y'_j &= 0, \quad \text{for each } j \in N \text{ such that } d'_j = 0, & (3.11) \\ y'_i &\geq \frac{1}{2}, \quad \text{for each } i \in N \text{ such that } d'_i > 0. & (3.12) \end{aligned}$$

We will refer to such a solution as a $\frac{1}{2}$-*restricted solution.* Further, the cost of the $\frac{1}{2}$-restricted solution produced is at most $2\bar{C}_{LP}$.

We further modify the $\frac{1}{2}$-restricted solution, without increasing its cost, to obtain an $\{\frac{1}{2}, 1\}$-solution $(\hat{x}, \hat{y})$: that is, $\hat{y}_i = 0$ for each $i \notin N'$ and $\hat{y}_i$ is either $\frac{1}{2}$ or 1 for each $i \in N'$.

Step 3: Rounding the solution Finally, we show how to convert a feasible $\{\frac{1}{2}, 1\}$-integral solution to the linear programming relaxation to a feasible integral solution of cost at most $\frac{4}{3}$ times the cost of the $\{\frac{1}{2}, 1\}$-integral solution.

Using these steps, we obtain the following theorem.

Theorem 3.3.1 *For the metric k-median problem, the outlined method yields a $6\frac{2}{3}$-approximation algorithm.*

Proof: Suppose that the feasible solution $(\bar{x}, \bar{y})$ to be rounded is an optimal solution to the linear program (3.1)–(3.4) and (3.7)–(3.8). Clearly, $\bar{C}_{LP}$ is a lower bound on the optimal value of the k-median problem. Steps 1–3 above create a feasible integer solution to the modified input of cost at most $2\frac{2}{3}\bar{C}_{LP}$. This can then be converted to an integer solution to the original instance while adding at most $4\bar{C}_{LP}$ to the cost.

This results in a feasible integer solution of cost at most $6\frac{2}{3}\bar{C}_{LP}$, i.e., at most $6\frac{2}{3}$ times the minimum possible. ■

3.4 Consolidating Locations

Recall that $\bar{C}_j = \sum_{i \in N} c_{ij}\bar{x}_{ij}$ is the cost that the linear program pays for assigning one unit of demand at location j. The goal of the first step of the method is to consolidate demands at nearby locations. More formally, we want to guarantee that for all pairs of locations i, j both with positive demand, $c_{ij} > 4\max(\bar{C}_i, \bar{C}_j)$. The modified instance of the location problem is obtained without changing the LP solution $(\bar{x}, \bar{y})$. For some locations j, we will simply "move" the demand d_j to a nearby location $j' \in N$, which is no more than $4\bar{C}_j$ away from location j.

Assume for notational simplicity that $N = \{1, \ldots, n\}$, and the locations are indexed in increasing order of $\bar{C}_j$, i.e., $\bar{C}_1 \leq \ldots \leq \bar{C}_n$. We create the modified instance as follows. Start with $d'_j \leftarrow d_j$, for each locations $j \in N$.

- Consider the locations $j = 1, \ldots, n$ in this order. When considering location j, check if there is another location $i < j$ such that $d'_i > 0$ and $c_{ij} \leq 4\bar{C}_j$. If there is such a location, then add the demand of location j to one such location i: select some i for which $c_{ij} \leq 4\bar{C}_j$, and set

$$d'_i \leftarrow d'_i + d'_j; \tag{3.13}$$

$$d'_j \leftarrow 0. \tag{3.14}$$

Let d' denote the resulting demands, whereas d denotes the original demands. Let N' denote the set of locations with $d'_j > 0$, i.e., $N' = \{j \in N : d'_j > 0\}$.

The feasible solution to the LP relaxation with the original demands, $(\bar{x}, \bar{y})$, is also a feasible solution for the modified input. The next lemma follows directly from the algorithm.

Lemma 3.4.1 *Locations $i, j \in N'$ satisfy (3.10).*

Lemma 3.4.2 *The cost of the fractional solution $(\bar{x}, \bar{y})$ for the input with modified demands is at most its cost for the original input, that is,*

$$\sum_{i,j \in N} d'_j c_{ij} \bar{x}_{ij} \leq \sum_{i,j \in N} d_j c_{ij} \bar{x}_{ij}. \tag{3.15}$$

Proof: The LP objective function value (3.1) of the solution $(\bar{x}, \bar{y})$, for any demands $\bar{d}$, can be written as $\sum_{j \in N} \bar{d}_j \bar{C}_j$. The lemma now follows directly, since d' is obtained by moving demand from a location j to a location i with $\bar{C}_i \leq \bar{C}_j$. ■

Lemma 3.4.3 *For any feasible integer solution (x', y') for the modified input with demands d', there is a feasible integer solution for the original input of cost at most $4\bar{C}_{LP}$ more than the cost of (x', y') with demands d'.*

Proof: Consider a location $j \in N$ that has its demand moved to j' in the modified input. Each location j was moved by at most $4\bar{C}_j$, and so we have that $c_{j'j} \leq 4\bar{C}_j$. If j is assigned to the center to which j' is assigned by x', then, by the triangle inequality, restoring j to its original position increases its unit assignment cost by at most $4\bar{C}_j$. Summing over all j, we obtain the desired bound.

■

3.5 Consolidating Medians

In this section, we consider the problem with demands d'_j for $j \in N'$ as defined in the previous section, i.e., we assume that the modified demands satisfy (3.10).

We simplify the structure of the solution by consolidating nearby fractional centers. First, we modify the solution $(\bar{x}, \bar{y})$ to obtain a new solution (x', y') of cost at most $2\bar{C}_{LP}$ such that $y'_i = 0$ for each $i \notin N'$ and $y'_i \geq \frac{1}{2}$ for each $i \in N'$.

We further modify the $\frac{1}{2}$-restricted solution to obtain, without increasing its cost, a $\{\frac{1}{2}, 1\}$-solution $(\hat{x}, \hat{y})$: that is, $\hat{y}_i = 0$ for each $i \notin N'$ and $\hat{y}_i$ is either $\frac{1}{2}$ or 1 for each $i \in N'$.

The first step will be accomplished using the filtering technique in [LV92b]. The main observation behind this technique is the following lemma, which shows that each location j has at least half of its demand assigned to relatively "nearby" partially open centers: for each $j \in N$, there is a total of at least $1/2$ of an open center within a radius of $2\bar{C}_j$.

Lemma 3.5.1 *For any feasible fractional solution $(\bar{x}, \bar{y})$, $\sum_{i:c_{ij} \leq 2\bar{C}_j} \bar{y}_i \geq \frac{1}{2}$ for each $j \in N$.*

Proof: Recall that $\bar{C}_j = \sum_{i \in N} c_{ij}\bar{x}_{ij}$. In any weighted average, less than half of the total weight can be given to values more than twice the average; that is,

$$\sum_{i:c_{ij} > 2\bar{C}_j} \bar{x}_{ij} < \frac{1}{2}. \tag{3.16}$$

Since $\sum_{i \in N} \bar{x}_{ij} = 1$ and $\bar{x}_{ij} \leq \bar{y}_i$ for each $i \in N$,

$$\sum_{i:c_{ij} \leq 2\bar{C}_j} \bar{y}_i \geq \sum_{i:c_{ij} \leq 2\bar{C}_j} \bar{x}_{ij} \geq \frac{1}{2}. \tag{3.17}$$

This proves the lemma. ∎

Now we are ready to prove the following result.

Theorem 3.5.1 *There is a $\frac{1}{2}$-restricted solution (x', y') of cost at most $2\bar{C}_{LP}$.*

Proof: We will modify the solution $(\bar{x}, \bar{y})$ by moving each fractional center to the location in N' closest it. We will prove that this produces a $\frac{1}{2}$-restricted solution of cost at most twice the cost of the solution $(\bar{x}, \bar{y})$.

We start by setting $x' \leftarrow \bar{x}$ and $y' \leftarrow \bar{y}$. First consider only the constraints (3.11). For each location $i \in N$ at which there is a partially open center (i.e., $y'_i > 0$) and yet $d'_i = 0$, we completely close the fractional center at location i, and move it to the closest location $j \in N'$. The fractional center at i is moved to j by setting

$$y'_j \leftarrow \min(1, y'_i + y'_j); \tag{3.18}$$

$$y'_i \leftarrow 0. \tag{3.19}$$

When we move the fractional center at i to location j, then we also change the assignments accordingly; that is, for each location $j' \in N$, we set

$$x'_{jj'} \leftarrow x'_{jj'} + x'_{ij'}; \tag{3.20}$$

$$x'_{ij'} \leftarrow 0. \tag{3.21}$$

First observe that these changes result in a feasible linear programming solution that also satisfies the constraints (3.11). We will worry about the constraints (3.12) later.

Next we show that these changes at most double the cost of the solution. Consider some location j' with positive demand $d'_{j'}$. For any fractional center i that is moved, say, to location j, consider the corresponding change in the assignment cost of j'. The fraction $x_{ij'}$ previously assigned to i is now assigned to j instead. By the triangle inequality, we get that $c_{jj'} \leq c_{ij} + c_{ij'}$. By the fact that the fractional center at i is moved to j, and not to j', we know that $c_{ij} \leq c_{ij'}$. Hence, $c_{jj'} \leq 2c_{ij'}$. By considering all demand points j' and all fractional centers i that are moved, we see that this claim implies that the cost of (x', y') is at most twice the cost of $(\bar{x}, \bar{y})$.

Finally, we must show that the new solution satisfies the constraints (3.12). Note that by Step 1, any two locations with positive demand are far apart, i.e., the demands d' satisfy (3.10). This implies that for each location j with positive demand, all partially opened centers within a $2\bar{C}_j$ radius of j must be closer to j than to any other location in N'; hence all of these will be moved to j. Lemma 3.5.1 implies that, the sum of the fractional center values on the locations within this radius of j is at least $\frac{1}{2}$. All of these fractional centers are moved to j, and hence we see that $y'_j \geq 1/2$, as required. ■

Next, we obtain a $\{\frac{1}{2}, 1\}$-solution for the modified instance. We begin by examining the structure of any $\frac{1}{2}$-restricted solution. For any $\frac{1}{2}$-restricted solution with fractional center values y, it is easy to express the corresponding optimal fractional assignment x. Consider a location $j \in N'$. The optimal linear programming solution will have $x_{jj} = y_j$, i.e., each location in N' will use its own partially open center to the maximum extent possible. Since $y_j \geq 1/2$, this leaves less than half of j left to be assigned, and so, for any $k \in N'$, we can set $x_{kj} = 1 - x_{jj} (= 1 - y_j)$, and be sure that the resulting solution is feasible (i.e., $x_{kj} \leq 1/2 \leq y_k$). The best alternative is to let k be the location in N' that is closest to j (other than j itself). For each $j \in N'$, let $s(j)$ be the closest location to j in N' (other than j), where ties are broken by choosing the location with smallest index. We have just proved the following lemma.

Lemma 3.5.2 *The minimum cost of a $\frac{1}{2}$-restricted solution (x', y') is $\sum_{j \in N'} d'_j (1 - y'_j) c_{s(j)j}$.* □

This lemma implies that we can view the cost of a $\frac{1}{2}$-restricted solution (x', y') as a function solely of the values y'. Using this lemma, we are ready to prove the following theorem.

Theorem 3.5.2 *For any $\frac{1}{2}$-restricted solution (x', y'), there exists a $\{\frac{1}{2}, 1\}$-integral solution of no greater cost.*

Proof: By Lemma 3.5.2, the cost of the $\frac{1}{2}$-restricted solution (x', y') can be expressed as

$$\sum_{j \in N'} d'_j c_{s(j)j} - \sum_{j \in N'} d'_j c_{s(j)j} y'_j. \tag{3.22}$$

We will construct a $\frac{1}{2}$-restricted solution whose cost is minimum possible. The first term of (3.22) is a constant independent of y'. The second term is maximized if the values y'_j are as large as possible for locations $j \in N'$ with the largest values $d'_j c_{s(j)j}$. The $\frac{1}{2}$-restricted solution of minimum cost is obtained as follows. Let $n' = |N'|$. Sort the locations $j \in N'$ in decreasing order of their weight $d'_j c_{s(j)j}$. Set $\hat{y}_j = 1$ for the first $2k - n'$ locations, and set $\hat{y}_j = 1/2$ for the remaining $2(n' - k)$ locations. The solution $(\hat{x}, \hat{y})$ is a $\{\frac{1}{2}, 1\}$-integral solution. By the above discussion, the cost of $(\hat{x}, \hat{y})$ is at most the cost of the $\frac{1}{2}$-restricted solution (x', y'). ■

Thus, Step 2 produces a $\{\frac{1}{2}, 1\}$-integral solution $\hat{y}_j$, $j \in N'$, so that each location $j \in N'$ is assigned to itself and at most one other location $s(j)$. Of course, if $\hat{y}_j = 1$, then there is no need to consider $s(j)$ at all; we will adopt the convention that if $\hat{y}_j = 1$, then $s(j) = j$.

3.6 Integrality Gap of a Half-Integral Solution

Finally, we show how to convert the $\{\frac{1}{2}, 1\}$-solution obtained in the previous section to an integral solution. We will first show a simple method that increases the cost incurred by at most a factor of 2. Then we outline how to convert to an integer solution while increasing the cost incurred by at most a factor of $\frac{4}{3}$.

Consider the new instance defined on vertices of N', where the demands are the same as in the modified instance produced by Step 1, and the underlying metric space will be a collection of trees. This change in the metric space keeps the length of the tree edges unchanged, increases the distance of some pairs of locations, but the value of the $\{\frac{1}{2}, 1\}$-solution $(\hat{x}, \hat{y})$ remains unchanged.

3.6.1 Creating an Initial set of Trees

We build a collection of trees as follows. For each node $i \in N'$ with $\hat{y}_i = \frac{1}{2}$, draw a directed edge from i to $s(i)$. By the definition of $s(j)$, each component of this graph contains cycles of length at most 2; in fact, each component has at most 1 cycle, which then corresponds to the closest pair of nodes in it. For each cycle, we choose one of the two vertices as a root and delete the directed edge from the root to the other vertex. This will result in a collection of rooted trees. We say that $s(i)$ is the parent of i, if we have a directed edge from i to $s(i)$.

3.6.2 A simple proof of integrality gap 2

Define the level of any node to be the number of edges on the path from the node to the root.

The locations of the centers are chosen as follows. We build centers at all nodes i such that $\hat{y}_i = 1$. We divide the nodes $\{i | \hat{y}_i = \frac{1}{2}\}$ into two subsets corresponding to the odd and even levels and build centers at nodes in the smaller of the two subsets. This ensures that the number of centers built is at most $\sum \hat{y}_i \leq k$.

It is easy to argue this rounding at most doubles the cost of the solution. Each node i with $\hat{y}_i = 1$ gets a center and its contribution to the cost of the solution is 0. For each node i with $\hat{y}_i = \frac{1}{2}$, either i or $s(i)$ is chosen as a center (since one is at an odd level and the other is at an even level). Hence the contribution of i to the cost of the solution is at most $d'_i c_{s(i)i}$ which is twice its contribution to the cost of the $\{\frac{1}{2}, 1\}$-solution.

Thus, the cost of the integral solution produced is at most twice the cost of the $\{\frac{1}{2}, 1\}$-integral solution. If we substitute this bound into the proof of Theorem 3.3.1, we get an 8-approximation algorithm for the k-median problem.

Next we describe a better procedure for rounding the $\{\frac{1}{2}, 1\}$-solution.

3.6.3 A gap of factor $\frac{4}{3}$

We will show that there exists an integral solution of cost at most $\frac{4}{3}$ times that of the $\{\frac{1}{2}, 1\}$-integral solution. This is tight, as shown by the example of a 4 node tree with 0 demand at the root, and demand 1 at each of the 3 children, and $k = 2$.

The basic idea is to produce a probability distribution on integral solutions of expected cost at most $\frac{4}{3}$ times the cost of the $\{\frac{1}{2}, 1\}$-solution.

We will prune the set of trees in the forest in two phases to construct trees with at most 3 levels. The distribution will be defined on these small level trees.

In the earlier proof that the integrality gap is at most 2, a node i either has an open center, or its nearest neighbor $s(i)$ has one. In this section, we will relax this condition and instead require that if neither i nor $s(i)$ has an open center, then $s(s(i))$ must have one. However, that might create a problem when $i \neq s(i)$ and $s(s(i)) = i$, i.e., for pairs of nodes that are closest to each other. In this case, we require that either i or $s(i)$ has an open center.

For each node in the tree, its parent is the location in N' closest to it. Thus, the distance from $s(i)$ to $s(s(i))$ is at most the distance from i to $s(i)$. It would be desirable that the distance from i to $s(s(i))$ is not twice the distance to $s(i)$, which is possible in the worst-case. We will ensure that for each node i, either $s(s(i))$ will be within $\frac{3}{2}$ times the distance to $s(i)$, or there is a center open at i or $s(i)$. The first phase of the tree pruning is designed to guarantee this property. The idea is that if c_{ij}, the distance from i to the closest child j of i, is more than half of the distance from i to $s(i)$, then we delete the edge between i and $s(i)$ from our tree, and make i the root of a new tree. We must be careful with the order in which we consider the locations for such deletions to avoid creating single node components.

Creating small level trees :

Lemma 3.6.1 *We can break the trees in the forest and create a collection of trees Γ_1 such that the trees of Γ_1 span the nodes of the original trees, and every tree $T_i \in \Gamma_1$ with root i satisfies the following properties*

P1: T_i has at least two vertices.

P2: If $\hat{y}_i = \frac{1}{2}$ then the nearest child of i is within a distance $2c_{s(i)i}$.

P3: For location $j \neq i$ that is included in the tree T_i the distance to its parent is $c_{s(j)j}$.

P4: Along a path from a leaf to the root i, the lengths of the edges decrease by at least a factor of 2.

Proof: Consider a tree T in the forest. Recall the level of a node is the number of edges required to reach the root from the node. The subtree rooted at a vertex v represents the tree restricted to the descendants of the node v. The notation $T_1 - T_2$ denotes the tree T_1 restricted to all the vertices not in T_2. Consider the following algorithm:

- We will visit the nodes of each tree by visiting all descendants of a node before visiting the node, and visiting the nearest child of the root as the last node before visiting the root. Consider one of the original trees T. We start by setting $T' = T$, and we will modify T' as we visit the nodes.

- Assume we are visiting a node i that is neither the root, nor the closest child of the root. Let T_i denote the subtree of T' rooted at i. If node i has a child, and the distance from i to its nearest child j is less than twice the distance to its parent $s(i)$, i.e., $c_{ij} \leq 2c_{s(i)i}$, then prune the tree T' by removing the subtree T_i rooted at node i, and adding T_i to the collection Γ_1.

- If i is the next-to-last node in the order and $\hat{y}_{s(i)} = \frac{1}{2}$. In this case $s(s(i)) = i$. We make $s(i)$ a child of i and add this modified subtree T_i' rooted at i to Γ_1.

- If we are visiting the root of T (all other nodes in the tree have already be visited), then the current tree T' is rooted at i, and we add this tree to Γ_1.

The property $P1$ is straightforward to observe. If the last node in the order is visited, then the second last node is still connected to the last node, which implies that T' has at least 2 vertices. In all other cases where a tree is added to $Gamma_1$, it clearly has two or more vertices.

If the tree T_i was included via a pruning step satisfying conditions a and b then due to condition b the property $P2$ follows because the distance to the parent is $c_{s(i)i}$. Otherwise the distance to the nearest child is exactly $c_{s(i)i}$ because either the parent of this node became its child, or this vertex with y value $\frac{1}{2}$ was part of a 2-cycle and its nearest child is the second last vertex in the order which is still present as its child.

Property $P3$ follows since all other edges in T_i are edges in T.

Consider a leaf node j at level 2 or more in T_i. We claim that the distance of $s(j)$ to its parent $s(s(j))$ is at most $\frac{1}{2}c_{s(j)j}$. This implies property $P4$. Assume for contradiction that $c_{s(s(j))s(j)} \geq \frac{1}{2}c_{s(j)j}$, i.e. $c_{s(j)j} \leq 2c_{s(s(j))s(j)}$. If this were the case, when $s(j)$ was visited, the subtree rooted at $s(j)$ would be cut and $s(j)$ would be at level 0. This is a contradiction.

■

Next we will further prune the trees in the collection Γ_1 so that the resulting trees have at most three levels (including the root). Consider the following steps:

(a) Consider a tree $T_i \in \Gamma_1$. Remove all the edges between the levels $2p + 1$ and

$2p+2$ for all $p \geq 0$. (Edges between levels $2p$ and $2p+1$ are undisturbed.)

(b) The above will result into a decomposition into trees having at most two levels (stars). For the singleton nodes created in this process (nodes that are in level $2p+2$ and have no children), attach them to their parent in T_i. Represent this collection by $\Gamma_2(T_i)$.

After these two steps are performed on the entire forest of trees, let Γ be the collection of trees of at most 3 levels formed in this fashion. $\Gamma = \bigcup_T \Gamma_2(T)$.

Distribution over small trees : We will construct a probability distribution over integral solutions for the trees in Γ such that the expected cost is small compared to the cost of the $\{\frac{1}{2}, 1\}$-integral solution. We will need to introduce one more definition.

A tree $T_i \in \Gamma$ is defined to be an **even** tree if $\sum \hat{y}_j$ over vertices j in this tree is integral; otherwise it is defined to be an **odd** tree.

An even tree T_i with $\sum \hat{y}_v = p$ will be assigned p centers. Clearly the number of odd trees will be even. We will randomly choose half of the odd trees to be 1-trees. The others will be 0-trees. A tree T_i with $\sum \hat{y}_j = p + \frac{1}{2}$, for some integer p, will have $p+1$ centers if it is chosen to be a 1-tree, otherwise have p centers. It is immediate that we open exactly k centers.

There are two cases, depending on whether the root i of a tree has $\hat{y}_i$ equal to 1 or $\frac{1}{2}$. We present the former, and simpler, case first. Throughout this discussion, we will choose a subset of k elements from a given set, where the subset is chosen uniformly at random among all such subsets; we shall say that we then *uniformly choose* k elements. The selections for different trees are made independently.

Case 1: Root i of a tree has $\hat{y}_i = 1$. Consider the following distribution to select centers:

1. If the root has $2r$ descendants uniformly choose r of them to build a center.

2. If the root has $2r+1$ descendants, uniformly choose $r+1$ of them if the tree is a 1-tree, and uniformly choose r of them otherwise.

Lemma 3.6.2 *For Case 1, the expected distance of each node i to its nearest center is at most $\frac{2}{3}c_{s(i)i}$.*

Proof: Consider any node i that is not the root. First, we observe that for each node i, the probability that a center is opened at i is $\frac{1}{2}$. This is immediate if the tree is an even tree. For an odd tree, we get this by averaging the probabilities given that the tree is a $0-$tree or a $1-$tree.

Now consider a node i at level 1. If i does not have a center, then the distance to its nearest center, the root, is at most $c_{is(i)}$; otherwise it has a center at distance 0. Thus, the expected distance is $\frac{1}{2}c_{is(i)}$.

Finally, let i be a node at level 2. As before, node i is selected as a center with probability $\frac{1}{2}$. Assume node i is not selected; then the closest center is at distance at most $c_{is(i)}$ if its parent $s(i)$ is selected, and at distance at most $\frac{3}{2}c_{is(i)}$ if neither i nor $s(i)$ are selected. To finish the proof we compute the conditional expectation that $s(i)$ is selected assuming that i is not selected. We consider two cases, depending on whether the tree is even or odd.

Case (a): The tree is even with $2p$ vertices other than the root. Given that i is not selected, $s(i)$ is selected with probability $\frac{p}{2p-1}$. The expected distance of i to its nearest center is bounded by

$$\frac{1}{2}\left(\frac{p}{2p-1}+\frac{(p-1)}{2p-1}\,\frac{3}{2}\right)c_{s(i)i}=\frac{(5p-3)}{4(2p-1)}c_{s(i)i}\leq\frac{2}{3}c_{s(i)i}.$$

Case (b): The tree is odd with $2p+1$ vertices other than the root. If it is a 1-tree, i is not selected with probability $\frac{p}{2p+1}$. Given that i is not selected, $s(i)$ is selected with probability $\frac{p+1}{2p}$. If it is a 0-tree, i is not selected with probability $\frac{p+1}{2p+1}$. Given that i is not selected, $s(i)$ is selected with probability $\frac{1}{2}$. The expected distance of i to its nearest center is at most

$$\frac{1}{2}\left[\frac{p}{2p+1}\left(\frac{p+1}{2p}c_{s(i)i}+\frac{(p-1)}{2p}\,\frac{3}{2}c_{s(i)i}\right)\right]+\frac{1}{2}\left[\frac{p+1}{2p+1}\left(\frac{1}{2}c_{s(i)i}+\frac{1}{2}\,\frac{3}{2}c_{s(i)i}\right)\right]$$

The above expression equals $\frac{5p+2}{4(2p+1)}c_{s(i)i}$ which is at most $\frac{2}{3}c_{s(i)i}$. ■

Case 2: Root i of a tree has $\hat{y}_i = \frac{1}{2}$. Before describing the distribution, we outline its desired properties, as follows:

D1: If the root is not chosen, then all nodes in level 1 of the tree must be chosen.

D2: The root of any tree in Γ will be chosen with probability at least $\frac{2}{3}$.

D3: Each node in level 1 must be chosen with probability at least $\frac{1}{3}$.

D4: Each node in level 2 must be chosen with probability at least $\frac{1}{2}$.

The next lemma explains why property $D1$ is desired.

Lemma 3.6.3 *If property $D1$ is true, if a root node i in tree $T_i \in \Gamma$ is not chosen then it will have a center within a distance at most $2c_{is(i)}$.*

Proof: By property $D1$, if the root i of a tree T_i is not chosen, then its nearest child is chosen. This implies the lemma if a tree with root i was either an original tree, or was created by the first phase of the pruning process. If the tree was created in the second phase of the pruning process, the parent $s(i)$ of i must be a level 1 node in a different subtree. By property $D1$, either $s(i)$ or its parent is chosen, proving the lemma. ■

Lemma 3.6.4 *If properties $D1$-$D3$ are true, then for any node j that is either the root of a tree or a level 1 node, the expected distance to the nearest center is at most $\frac{2}{3}c_{s(j)j}$.*

Proof: Suppose j is a root node. By property $D2$, j is chosen with probability at least $\frac{2}{3}$. By property $D1$ and Lemma 3.6.3, if j is not chosen, it has a center within distance at most $c_{s(j)j}$ from it. This the expected distance to the nearest center is at most $\frac{2}{3}c_{s(j)j}$.

Suppose j is a level 1 node. By property $D3$, j is chosen with probability at least $\frac{1}{3}$. If j is not chosen, by property $D1$, its parent $s(j)$ (the root) must be chosen. Thus the expected distance to the nearest center is at most $\frac{2}{3}c_{s(j)j}$. ■

The root i is at level 0. If the number of level-1 nodes is odd or the number of level-2 nodes is even then we consider the following distribution:

1. The distribution will be the product of these two distributions: the nodes in level 0 (root) and level 1 will have a distribution and independently, the level-2 nodes will have a distribution. We will describe the distribution over level-2 nodes first.

2. If there are $2r$ level-2 nodes, uniformly choose r to open as centers. If there are $2r + 1$ nodes in level 2, (and so the number of level-1 nodes is odd, and this is an odd tree), uniformly choose $r + 1$ of them if this is a 1-tree, and uniformly choose r of them if it is 0-tree.

3. If there is a single level-1 node, then open a center at the root with probability $\frac{2}{3}$ and at the child with probability $\frac{1}{3}$.

4. If the number of level-1 nodes is $2t - 1$ ($t > 1$), open a center at the root and uniformly choose $t - 1$ level-1 vertices as centers.

5. If the number of level-1 nodes is 2, (and so the number of level-2 nodes is even and this is an odd tree), then, if the tree is a 1-tree, build a center at both level-1 nodes with probability $\frac{2}{3}$, and build a center at the root and any one of the level-1 nodes with probability $\frac{1}{3}$. If the tree is a 0-tree, build a center at the root.

6. If the number of level-1 nodes is $2t$ ($t > 1$), open a center at the root and uniformly choose t level-1 nodes if the tree is a 1-tree. Otherwise, open a center at the root and uniformly choose $t - 1$ level-1 nodes as centers.

If there are an even number of level-1 nodes and an odd number of level-2 nodes, we consider a different distribution:

1. If there are 2 level-1 nodes and $2r + 1$ level-2 nodes, then with probability $\frac{2}{3}$ open centers at the root and $r + 1$ uniformly chosen level-2 nodes. With probability $\frac{1}{3}$, open centers at the 2 level-1 nodes and r uniformly chosen level-2 nodes.

2. If there are $2t$ level-1 nodes ($t > 1$) and $2r + 1$ level-2 nodes, always open a center at the root. With probability $\frac{1}{2}$, uniformly choose $t - 1$ level-1 nodes and, independently, uniformly choose $r + 1$ level-2 nodes. With probability $\frac{1}{2}$,

uniformly choose t level-1 nodes and, independently, uniformly choose r level-2 nodes.

Lemma 3.6.5 *In the above distribution, the expected distance to the nearest center for each node j is at most $\frac{2}{3}c_{s(j)j}$.*

Proof: Case 1: The number of level-1 nodes is odd or the number of level-2 nodes is even.

We first show that the distribution satisfies properties $D1$-$D4$. The property $D1$ is true by construction. The root is always present unless the number of level-1 nodes is 1 or 2 When there a single level-1 node, the root is picked with probability $\frac{2}{3}$. In the case of 2 level-1 nodes, the root is always chosen if the tree is a 0-tree and chosen with probability $\frac{1}{3}$ when the tree is a 1-tree. Thus the root is picked with probability $\frac{2}{3}$. This establishes property $D2$.

The property $D3$ is true by inspection, when the number of level-1 nodes is 1 or 2. When the number of level-1 nodes is $2t-1$, a level-1 node is chosen with probability at least $\frac{t-1}{2t-1}$ which is at least $\frac{1}{3}$ because $t > 1$. When the number of level-1 nodes is $2t$, a level-1 node is chosen with probability $\frac{1}{2}$ when it is a 1-tree and with probability $\frac{t-1}{2t}$ when it is a 0-tree. Thus the probability that the level-1 node is chosen is $\frac{1}{4} + \frac{t-1}{4t}$ which is at least $\frac{1}{3}$ since $t > 1$. This proves property $D3$.

We now turn to property $D4$. If there are $2r$ level-2 nodes, the probability of choosing any one is $\frac{1}{2}$. In the case where there are $2r+1$ level-2 nodes, the probability of choosing a level-2 node is also $\frac{1}{2}$ by averaging the probabilities given that the tree is a 0-tree or a 1-tree.

Since properties $D1$-$D3$ hold, by Lemma 3.6.3, the expected distance of a node j to its nearest center is at most $\frac{2}{3}c_{s(j)j}$ when j is either a root node or a level-1 node. We now prove this for level-2 nodes. Observe that the choice of the level-2 nodes is made independently of the choice of the level-1 nodes and the root. If a level-2 node j is not selected (with probability $\frac{1}{2}$), its parent $s(j)$ is chosen with probability at least $\frac{1}{3}$ and otherwise the root is chosen. Thus the expected distance to a center is at most $\frac{1}{2}(\frac{1}{3}c_{s(j)j} + \frac{2}{3}\ \frac{3}{2}c_{s(j)j})$ which is $\frac{2}{3}c_{s(j)j}$.

Case 2: The number of level-1 nodes is even and the number of level-2 nodes is odd.

Suppose we have $2t$ level 1 nodes and $2r+1$ level 2 nodes. Properties $D1$ and $D2$ are true by construction. If $t > 1$, the probability that a level-1 node is chosen is $\frac{1}{2}\frac{(t-1)}{2t} + \frac{1}{2}\frac{t}{2t}$ which is at least $\frac{1}{3}$. If $t = 1$, a level-1 node is chosen with probability $\frac{1}{3}$. This proves $D3$. By Lemma 3.6.3, the expected distance of a node j to its nearest center is at most $\frac{2}{3}c_{s(j)j}$ when j is either a root node or a level-1 node. We now show that this holds for level-2 nodes. However, unlike the previous case, the distribution on the level-2 nodes is not independent of the distribution on the root and the level-1 nodes. We consider the case $t = 1$ and $t > 1$ separately.

Case (a): Suppose $t = 1$. With probability $\frac{2}{3}$, the first configuration is chosen where j does not have a center with probability $\frac{r}{2r+1}$. In this case j is serviced by the root at a distance of at most $\frac{3}{2}c_{s(j)j}$. With probability $\frac{1}{3}$, the second configuration is chosen where v does not have a center with probability $\frac{r+1}{2r+1}$, in which case it is serviced by its parent $s(j)$. The expected distance of j to its nearest center is at most

$$\frac{2}{3}\frac{r}{2r+1}\frac{3}{2}c_{s(j)j} + \frac{1}{3}\frac{r+1}{2r+1}c_{s(j)j} = \frac{(4r+1)}{3(2r+1)}c_{s(j)j} \leq \frac{2}{3}c_{s(j)j}.$$

Case (b): Suppose $t > 1$. With probability $\frac{1}{2}$ the first configuration is chosen where j does not have a center with probability $\frac{r}{2r+1}$. In this case $s(j)$ is chosen with probability $\frac{t-1}{2t}$. Thus the expected distance in this case is at most

$$\frac{r}{2r+1}\left(\frac{t-1}{2t}c_{s(j)j} + \frac{(t+1)}{2t}\ \frac{3}{2}c_{s(j)j}\right) = \left(\frac{5}{4} + \frac{1}{4t}\right)\frac{r}{2r+1}c_{s(j)j}.$$

In the second configuration j is not chosen with probability $\frac{r+1}{2r+1}$, in which case $s(j)$ is chosen with probability $\frac{t}{2t} = \frac{1}{2}$. The expected distance to the nearest center is at most

$$\frac{r+1}{2r+1}\left(\frac{1}{2}c_{s(j)j} + \frac{1}{2}\cdot\frac{3}{2}c_{s(j)j}\right) = \frac{5}{4}\frac{(r+1)}{(2r+1)}c_{s(j)j}.$$

Thus the expected distance over the two configurations is at most

$$\left(\frac{5}{8} + \frac{1}{8t}\ \frac{r}{2r+1}\right)c_{s(j)j} \leq \left(\frac{5}{8} + \frac{1}{16t}\right)c_{s(j)j} \leq \frac{2}{3}c_{s(j)j}.$$

This proves the claim.

■

Since the cost of the nodes in the $\{\frac{1}{2}, 1\}$-solution is $\sum_i \frac{1}{2} d'_i c_{s(i)i}$ and the expected cost of the above distribution is at most $\sum_i \frac{2}{3} d'_i c_{s(i)i}$, we can conclude the following,

Theorem 3.6.1 *The $\{\frac{1}{2}, 1\}$-solution has an integrality gap of at most $\frac{4}{3}$.*

3.7 A Different View

The algorithm presented in the last two sections solved the k-median problem by solving a fractional relaxation of the problem posed as an integer linear program. However we can view the algorithm differently, consider the following:

1. Solve the Fractional relaxation, and identify the set of nodes N', as in Step 1 of the previous algorithm. Instead of 'moving' the demand j to location i, by the condition $i < j$, and $c_{ij} \leq 4\bar{C}_j$, add j to a star whose central vertex is i.

2. Consider the set these central vertices N'. For $i \in N'$ join it by an edge to its nearest location in N'. In case of a ties, we choose the edge which does not create a simple cycle. Thus we have a collection of trees.

3. Ignore all y_i and x_{ij} values and solve the k-median problem optimally on this collection of trees.

What we saw in the last two section was a constructive proof of an integrality gap of $6\frac{2}{3}$ for the above algorithm. If we move the demand locations from the leaf nodes of the initial stars to the central vertex, we have a fractional feasible solution of the same cost as the fractional solution. The step of consolidating medians, shows that there exists a fractional feasible solution of at most twice the original fractional solution. Moreover the assignments are only along the edges of the set of trees we constructed in the above algorithm, since each $i \in N'$ is assigned to itself or its closest location in N'. The integrality gap being $4/3$, we have a solution of cost $8/3$ times the original fractional solution for the modified problem. But this solution only uses edges of the trees we constructed. Adding the cost of moving the demands, we are guaranteed a solution of cost $6\frac{2}{3}$ times the original fractional solution, using only the tree edges we constructed.

Since for a collection of trees, we can solve the k-median problem optimally, our final solution will be no more than $6\frac{2}{3}$ times the initial fractional solution we started from. Thus the previous two sections provided a constructive proof of integrality gap $6\frac{2}{3}$.

However the worst integrality gap example we can construct is factor 2 (Chapter 5). In the next chapter we will see that this gap can be reduced to 4, but not by a constructive rounding procedure.

Chapter 4

Primal Dual Algorithms

4.1 Introduction

In the last two chapters we saw two different approaches to solving the facility location problem. In the local search technique, we was an algorithm which was simple and efficient. However it did not provide us with a good lower bound. On the other hand, the linear programming relaxation gave us a good lower bound, but solving a linear program with $O(n^2)$ variables and constraints is nontrivial. In this chapter we will reach a middle ground, we will have a combinatorial algorithm, as well as a lower bound.

The linear program for the facility location problem we saw, (henceforth referred to as the primal formulation) was a minimization problem. The dual of this linear program is a maximization problem, and by duality theory, any feasible dual solution is a lower bound to the primal objective function. Since the dual problem is a maximization problem, we can conceive of constructing a feasible solution and use this solution as a lower bound. If the structure of the dual linear program is simple, we can create a feasible solution combinatorially. The idea is to create a feasible dual solution, and a particular integral solution of the primal problem which is bounded in terms of the dual solution. This was first explicitly demonstrated in [AKR95], and subsequently the method has been used in a variety of problems, see [GW97] for more details.

The combinatorial construction of a feasible dual provides a lower bound, and since the construction is straightforward, has a small running time. In case of the facility location problem this was first shown in [JV99], who gave a factor 3 approximation.

The facility location problem is the Lagrangian relaxation of the k-median problem, with the Lagrangian multiplier denoting the facility cost. [JV99] showed how to use this fact to get a primal dual approximation algorithm for the k-median problem. For every setting of the Lagrangian multiplier, we can solve a facility location problem. If the multiplier is too large (sum of all the $\binom{n}{2}$ distances), we are expected to open a single facility. At the other extreme, if the multiplier has value 0, we are expected to open every facility. Suppose we had an algorithm that behaved as mentioned, we can perform a binary search over the value of the multiplier and get two solutions which have small difference in the values of the Lagrangian multipliers, but one has less than k facilities and the other more has than k facilities. We can take the linear combination of these two solutions such that the linear combination opens exactly k facilities. This would be a fractional solution. But the linear combination of the two dual solutions will be a feasible solution, an this dual solution will provide a lower bound to the k-median problem. But this would require the approximation guarantee of a special form, we will see this in more detail in Section 4.3. Now the fractional solution which we created will be related to the linear combination of the dual solutions, which in turn would be less than the optimum k-median solution. Thus the fractional solution constructed would be small compared to the integral optimum solution, and can be rounded losing a factor of 2.

This method of using a linear combination of two solutions was used in [Gar96] to give a factor 3 approximation algorithm for the k-MST problem, which asked for a minimum cost tour spanning exactly k vertices. The algorithm used the primal dual Steiner tree algorithm of [GW95], and used a linear combination of the two solutions and the approximation guarantee of the special form to get this approximation. [Gar96] first provided a 5 approximation and proceeded to improve the approximation factor to 3. The improvement came from the observation that the two solutions were generated by the same algorithm for different but close values of the multiplier, and thus similar in structure. This similarity allows for arguing across the two solutions, that a subtree chosen corresponding to one value of the Lagrangian multiplier would be a bounded in cost in case of the other solution since the multiplier values are close.

We will show such an improvement for the k-median problem in Section 4.5. However there would be one difference, that in the k-MST problem, we are interested in the overall tour, and in the case of the k-median problem we have to guarantee that both the solutions appear similar to every demand location. This is because the analysis method tries to bound the cost of the final solution on a location by location basis, by relating the assignment distance of a demand point to its contribution to the dual objective function. This is a shortcoming of the analysis and it is not plausible that the approximation factor can be improved using techniques that amortize the cost. The worst example of integrality gap of the constructed dual solution is 3 (See Chapter 5) and the upper bound is 4.

To ensure that the two solutions are comparable with respect to every demand point, we have to prove continuity of the dual variables as a function of the Lagrangian multiplier. The logarithm of the derivative would indicate the number of binary search steps required to determine Lagrangian multiplier. We would later see that the running time will increase by a factor of n due to this. Even choosing such close values of the multiplier would not be sufficient for our proof technique, and we will try to add facilities from one solution to the other while maintaining feasibility of the dual solutions. This will ensure that if a facility cannot be added to one solution, there exists another facility which is close to it. A similar statement can be inferred from the proof of factor 6, however our bound will be stronger, and hence the improvement ensues.

4.2 Primal Dual Algorithm for Facility Location

The facility location problem can be expressed as an IP whose linear relaxation is as follows:

$$\begin{aligned} \min \sum_i y_i f_i + \sum_{ij} x_{ij} c_{ij} d_j & \\ x_{ij} \quad \leq y_i & \qquad \text{for each } i,j \end{aligned}$$

$$\begin{aligned} \sum_i x_{ij} &\geq 1 && \text{for each } j \\ x_{ij}, y_i &\geq 0 && \text{for each } i, j \end{aligned}$$

Here y_i indicates whether facility i is chosen in the solution and x_{ij} indicates whether node j is serviced by facility i. Throughout this paper the demand nodes will be indexed by j and facilities by i.

The dual of the facility location LP is

$$\begin{aligned} \max \sum_j \alpha'_j & \\ \alpha'_j \leq & \quad \beta'_{ij} + c_{ij} d_j && \text{for each } i, j \\ \sum_j \beta'_{ij} \leq & \quad f_i && \text{for each } i \\ \alpha_j, \beta'_{ij} \geq & \quad 0 && \text{for each } i, j \end{aligned}$$

Transforming the variables, $\alpha'_j = d_j \alpha_j$ and $\beta'_{ij} = d_j \beta_{ij}$, for the sake of simplicity, the dual can be written as.

$$\begin{aligned} \max \sum_j d_j \alpha_j & \\ \alpha_j \leq & \quad \beta_{ij} + c_{ij} && \text{for each } i, j \\ \sum_j d_j \beta_{ij} \leq & \quad f_i && \text{for each } i \\ \alpha_j, \beta_{ij} \geq & \quad 0 && \text{for each } i, j \end{aligned}$$

We describe the primal-dual algorithm very briefly. The graph can be assumed to be a complete bipartite graph between the facilities and the demand nodes. The algorithm maintains dual values for each demand node (α_j) and each edge (β_{ij}). Initially all the variables are set to zero. The algorithm grows the dual variables corresponding to the demand nodes uniformly.

Assume that the parameter τ indicates time, for all locations j which are growing, $\alpha_j = t$. At various times some of the dual variables α_j stop growing, to maintain feasibility of the dual solution.

Three types of events occur during the execution of the algorithm. The first type of event occurs when the dual variable $\alpha_j$corresponding to the demand node j is equal to the edge c_{ij} for a facility i. This happens at $\tau = c_{ij}$. Beyond this point the dual variable β_{ij} corresponding to the edge is raised. The algorithm maintains the invariant that at time t the dual variables α_j that are still being raised are all equal to t and that if $\alpha_j > c_{ij}$ then $\beta_{ij} = \alpha_j - c_{ij}$. An edge between facility i and location j is defined to be *tight* if $\beta_{ij} > 0$.

The second type of event is that at some time $\tau = t_i$ for a facility i the event that $\sum_j d_j \beta_{ij} = f_i$ where f_i is the facility cost of this facility i. The facility i is declared *paid for* and the demand nodes j with positive β_{ij} values stop growing. The node i is defined the *Connecting witness* of j.

The third type of event occurs when for a demand node j the variable $\alpha_j = c_{ij}$ for a paid for (resp. temporarily open) facility i, in which case the variable α_j stops growing. Again node i is the *Connecting witness* of j.

When several events happens at the same time, they are dealt in an arbitrary order. After all the dual variables α_j have stopped growing, the algorithm inspects the subgraph defined by the tight edges.

It recursively picks an arbitrary paid for facility and builds a facility at this location. It then deletes all paid for nodes that can be reached by two tight edges from this facility (thus picking a maximal independent subset of the paid for facilities in the square of the subgraph induced by the tight edges). Denote the set of picked facilities by $\mathcal{F}$.

Define a demand location to be *directly assigned* to facility $i \in \mathcal{F}$ if $\beta_{ij} > 0$. This is an assignment since by construction there cannot be $i, i' \in \mathcal{F}$, with both β_{ij} and $\beta_{i'j}$ positive. All other locations are defined to be *indirectly assigned*. Denote by $\sigma_d(j)$ the facility to which j is directly assigned to. Notice that this 'direct assignment' need not be assignment in the solution to the facility location problem, where demand points will be assigned to their closest facilities.

Lemma 4.2.1 *The primal-dual algorithm returns a solution with facility cost F and a assignment cost C such that, $3F + C \leq 3\text{DSOL}$ where* DSOL *denotes the cost of the dual solution constructed.*

Proof: The following is restatement of the type two events, summed over $i \in \mathcal{F}$.

$$\sum_{i \in \mathcal{F}} \sum_{j, \sigma_d(j) = i} d_j \beta_{ij} = F$$

We can re express this as

$$F = \sum_{\text{directly assigned } j} d_j \beta_{\sigma_d(j)j}$$

Consider a solution in which every demand location is assigned to their nearest facility. Consider a demand location j. If j were directly assigned, then it has a facility within distance $c_{\sigma_d(j)j}$, since the facility $\sigma_d(j)$ was chosen as a facility in the solution. If j were indirectly assigned, consider its connecting witness i. If $i \in \mathcal{F}$, then the assignment distance of j in the final solution is at most $c_{ij} \leq \alpha_j$. If $i \notin \mathcal{F}$, there must be a demand location j' and a paid for facility $i' \in \mathcal{F}$ such that $\beta_{i'j'}$ and β_{ij} were both positive.

Since both $\beta_{i'j'}$ and β_{ij} were positive, $\alpha_{j'}$ is more than $c_{i'j'}$ and $c_{ij'}$. Since $\beta_{ij'} > 0$, we must have $\alpha_{j'} \leq t_i$, since after t_i was paid for, $\alpha_{j'}$ cannot grow. Now $\alpha_j = t_i$. Thus,

$$c_{i'j} \leq c_{i'j'} + c_{ij'} + c_{ij} \leq \alpha_{j'} + \alpha_{j'} + \alpha_j \leq 3\alpha_j$$

Thus the assignment cost is at most

$$C \leq \sum_{\text{directly assigned} j} d_j c_{\sigma_d(j)j} + \sum_{\text{Indirectly assigned} j} 3 d_j \alpha_j$$

Therefore,

$$3F + C \leq \sum_{\text{directly assigned } j} d_j (3\beta_{\sigma_d(j)j} + c_{\sigma_d(j)j}) + \sum_{\text{indirectly assigned } j} 3 d_j \alpha_j$$

Now since $\beta_{\sigma_d(j)j} + c_{\sigma_d(j)j} \leq \alpha_j$ the above can be rewritten as

$$3F + C \leq \sum_j 3 d_j \alpha_j \leq 3\text{DSOL}$$

■

Running Time The algorithm runs in time $O(n^2 \log n)$. The edges can be sorted on distance. There are at most n events corresponding to facilities being paid for. For each of these events, we can update $O(n)$ events, and sort the times at which the next facility is paid for in time $O(n \log n)$.

4.3 Lagrangian Relaxations: Revisiting k Median

Based on the above we can design a primal dual approximation algorithm for the k-median problem that achieves an approximation factor of 6. Consider the Linear Program for the k-median problem,

$$\begin{aligned}
\min \sum_{ij} x_{ij} c_{ij} d_j & & \\
x_{ij} &\leq y_i && \text{for each } i,j \\
\sum_i x_{ij} &\geq 1 && \text{for each } j \\
\sum_i y_i &\leq k && \\
x_{ij}, y_i &\geq 0 && \text{for each } i,j
\end{aligned}$$

Observe that in the above linear program, if the constraint $\sum_i y_i \leq k$, was not present, the linear program has a trivial solution. We can perceive this constraint as a constraint which is hard to satisfy. One of the most interesting techniques of solving linear programs with a few constraints which are difficult to satisfy is through the method of Lagrangian relaxations. We add these constraints to the objective function of the linear program, with a suitable multiplier. The idea is that if the rest of the original program (excluding the constraints added to the objective function) is easier to solve. This is a relaxation, since feasible solutions to the previous program remain feasible. Thus the solution to relaxation provides a lower bound. Since we are eliminating constraints, the resulting program may not satisfy the original constraints, but we can set a multiplier that penalize the violation. The objective then reduce to find the best possible multiplier for the best possible bound. For example the Lagrangian relaxation of the above linear program for the k-median problem is:

$$\begin{aligned} \min \sum_{ij} x_{ij} c_{ij} d_j + z \left(\sum_i y_i - k \right) & \\ x_{ij} &\leq y_i && \text{for each } i,j \\ \sum_i x_{ij} &\geq 1 && \text{for each } j \\ x_{ij}, y_i &\geq 0 && \text{for each } i,j \end{aligned}$$

where the parameter z is the Lagrangian multiplier. Notice that this is a linear program for the Facility Location problem where all the facility costs are z, except the constant term in the objective function. Thus the solution to the facility location problem can be used a lower bound for the k-median problem, after subtracting the kz term. However this lower bound will not be simple to handle, and we would consider the dual programs, to find a dual lower bound to the facility location problem, and subtract kz from it to get a lower bound of the Lagrangian relaxation, and consequently the k-median problem. We can also view the dual programs directly, the dual of the linear program for the k-median problem is:

$$\begin{aligned} \max \sum_j \alpha'_j - zk & \\ \alpha'_j \leq & \quad \beta'_{ij} + c_{ij} d_j && \text{for each } i,j \\ \sum_j \beta'_{ij} \leq & \quad z && \text{for each } i \\ \alpha_j, \beta'_{ij} \geq & \quad 0 && \text{for each } i,j \end{aligned}$$

As in the previous section, transforming the variables, $\alpha'_j = d_j \alpha_j$ and $\beta'_{ij} = d_j \beta_{ij}$, for the sake of simplicity, the dual can be written as.

$$\begin{aligned} \max \sum_j d_j \alpha_j - zk & \\ \alpha_j \leq & \quad \beta_{ij} + c_{ij} && \text{for each } i,j \end{aligned}$$

$$\begin{array}{rcll} \sum_j d_j \beta_{ij} & \leq & z & \text{for each } i \\ \alpha_j, \beta_{ij} & \geq & 0 & \text{for each } i, j \end{array}$$

Now compare the above with the dual of the facility location problem we saw in the previous section, with all the facility costs set to z,

$$\begin{array}{rcll} \max \sum_j d_j \alpha_j & & & \\ \alpha_j & \leq & \beta_{ij} + c_{ij} & \text{for each } i, j \\ \sum_j d_j \beta_{ij} & \leq & z & \text{for each } i \\ \alpha_j, \beta_{ij} & \geq & 0 & \text{for each } i, j \end{array}$$

Thus a feasible solution of the dual program for the facility location problem with cost of all facilities z would give a feasible solution to the dual program for the k-median problem. If we solve the facility location problem for a fixed value of z, the quantity $\sum_j \alpha_j - zk$ will be a lower bound for the dual program for the k-median problem, and thus the primal. Thus if we have a combinatorial algorithm, that approximates the dual of the facility location problem, we can get a combinatorial algorithm for the k-median problem. Hence using a primal dual approximation algorithm for the facility location problem we can hope to achieve a primal dual algorithm for the k-median problem.

The idea is essentially the following: Suppose we had two solutions with $k_1 < k$ and $k_2 > k$ centers respectively, then we can combine the two solutions to get exactly k centers. Clearly when $z = 0$ we open all the facilities. In the same vein when z is larger than n times the diameter of the graph, we open only one facility. Thus we can perform a binary search on the value of z such that we have two values of z which are close and for the smaller value we open $k_2 > k$ facilities and for the larger one we open $k_1 < k$ facilities. Let the dual solutions be $\hat{\alpha_1}$ and $\hat{\alpha_2}$.

At this point if we assume $a + b = 1$ and $ak_1 + bk_2 = k$, we take the linear combination of the dual $a\hat{\alpha_1} + b\hat{\alpha_2}$. We also take the linear combination of the two solutions as a fractional solution of the k-median problem. Now the cost of the fractional solution (which entails the service cost part of the facility location problems only) to the k-median problem can be bound by $3(a(\text{DSOL}_1) + b(\text{DSOL}_2) - az_1k_1 - bz_2k_2)$, where $\text{DSOL}_1, \text{DSOL}_2$ are the value of the dual solutions to the facility location problems. Under the assumption that $z_1 \approx z_2$ (such that their difference is polynomially smaller than the optimal solution to the problem) this reduces to $3(a(\text{DSOL}_1) + b(\text{DSOL}_2) - kz)$. Once again, the linear combination of the two dual solutions, is also a feasible dual solution (assuming $z_1 = z_2$, which introduces a small error) for this value of z. Thus we have a fractional primal solution which can be decomposed into two solutions with integral number of centers, and the cost of the solution is at most thrice the optimal.

The approximation factor follows from the fact that a solution which is a linear combination of two integral solutions, can be rounded to an integral solution opening k centers losing a factor 2 in the approximation factor. We will not prove this fact, but we can easily observe that the above discussion will give a factor 6 approximation algorithm for the k-median problem. We will actually see a better algorithm that will achieve factor 4. The basis of our analysis will be the next theorem.

Let A_j be the assignment cost of j, i.e. the distance to the nearest facility in our solution. A_j will be a random variable, since we will construct a probability distribution over solutions. As we will see later, our algorithm can be derandomized using method of condition probabilities. We will assign a facility opening cost q_j to j, where $\sum_j q_j = zk$. The proof of the following theorem is straightforward.

Theorem 4.3.1 *If for some z, $\sum_j q_j = zk$ and for all j, $\mathrm{E}[A_j] + c \cdot q_j \leq c \cdot \alpha_j(z)$ then the expected cost of the solution is a factor c times the optimal.*

Proof: Follows from the fact that $\sum_j \alpha_j(z) - zk$ is a lower bound, thus summing over all points j, $\sum_j \mathrm{E}[A_j]$ can be c times optimum. The theorem follows from the linearity of expectation, $\mathrm{E}[\sum_j A_j] = \sum_j \mathrm{E}[A_j]$. ■

The above proof uses a guarantee which is true over for all demand nodes j. Since the final solution will be composed of two different solutions, it would require us to show that the two solutions are close – in the sense $|\alpha_j(z_1) - \alpha_j(z_2)|$ is small. For the algorithm in [JV99] this was not required, since the analysis was a two step process.

In our case, it would be sufficient to show $\sum_j |\alpha_j(z_1) - \alpha_j(z_2)|$ is small compared to $\sum_j \alpha_j(z_1)$ or $(\sum_j \alpha_j(z_2))$. The current analysis requires the number of search steps over the value of z, to be n, and thus increases the running time. The number of search steps required is the logarithm of the derivative of the function $\alpha_j(z)$, which in the current analysis is n, and thus the running time increases by a factor n.

4.4 Properties of the Dual Solution

In this subsection we will prove certain continuity properties of the solution produced by the primal-dual facility location algorithm (referred to simply as 'the algorithm' for brevity). These continuity properties will be crucial to the analysis of the modified algorithm later. The dual variables α_j and β_{ij} will represent the dual variables corresponding to vertices and edges as in the previous sections. Let $\alpha_j(z)$ be the final value of α_j produced by the algorithm for z. Similarly, define $\beta_{ij}(z)$. We define $t_i(z)$ be the last time at which $\sum_j \beta_{ij}$ increases in the algorithm for z. Note that $\sum_j \beta_{ij}$ is the total contribution to the facility building cost of i. When the algorithm is run for a particular value of z, $\sum_j \beta_{ij} \leq z$. If the final value of $\sum_j \beta_{ij}$ is indeed z, then we say that facility i is *paid for* and t_i is the time at which facility i is paid for.

The ordering at z refers to the pair $(\mathcal{F}, \mathcal{S})$ where $\mathcal{F}$ is the set of facilities that are paid for in the algorithm for z and $\mathcal{S}$ is the ordered (non-decreasing) sequence of $\alpha_j(z)$ values, $t_i(z)$ values and c_{ij} values, such that consecutive elements of this ordering are related either by equality or strict inequality. We define a *transition point* to be a value of z where infinitesimal changes in z result in a change in the ordering. A transition point can be defined in terms of $t_i(z)$ values as well, our definition in terms of $\alpha_j(z)$ values is purely for convenience. We can prove the following lemma about the minimum separation between two transition points:

Theorem 4.4.1 *Two critical values of z are separated by at least $c = 2^{-(poly(n)+L)}$, where L is the number of bits needed to represent the longest edge.*

Proof: A transition point is the infimum or supremum of a set which is the feasible region of a polynomial sized linear program. ■

We will show that all the functions $\alpha_j(z)$ are continuous. We will also be interested in the times at which nodes are declared open. In fact we will we will extend the above notion of time when a facility is declared open to all nodes and a notion when a node is last contributed to towards building a center. Denoting these quantities as a function of z, say $t_u(z)$ (The formal definition is provided later), we will be interested in the continuity of these functions. Though we will not be able to show that these functions are continuous, we will be able to show piece-wise continuity and hence the existence of left and right limit points of these functions at a critical value.

First let us discuss the continuity of the functions $\alpha_j(z)$.

Continuity of $\alpha_j(z)$ We will prove that $\alpha_j(z)$ is continuous by considering two specific values of z namely z' and $z' + \delta z$ and δz is infinitesimally small. We will show that the values of $\alpha_j(z')$ and $\alpha_j(z' + \delta z)$ are at most a finite multiple of $|\delta z|$ apart. This will imply that $\alpha_j(z)$ is continuous.

Denote by $\alpha_j(z, t)$, the value of dual variable α_j at time t in the algorithm for z. It is easy to observe that $\alpha_j(z, t) = \min(t, \alpha_j(z))$. Define $p_u(z, t) = \sum_j \max(0, \alpha_j(z, t) - c_{uj})$ as the total contribution to the facility building cost of u at time t in the algorithm corresponding to z. A node is declared temporarily open at time t, if and only if $p_u(z, t) = z$.

Theorem 4.4.2 *The dual variables $\alpha_j(z)$ is continuous in z, moreover for a small change δz in value of z, the value of $\alpha_j(z)$ changes by at most $2^n|\delta z|$.*

Proof: Suppose we execute separately and in parallel, the algorithm for z' and $z'' = z' + \delta z$. An *event* is said to occur if a node j gets a connecting witness in one of the two executions. Thus we have $2n$ events in all, two corresponding to each node j. The events for node j occur at times $\alpha_j(z')$ and $\alpha_j(z'')$.

Consider the set of values of $\alpha_j(z')$ and $\alpha_j(z'')$. Sort them in non-decreasing order and break ties arbitrarily. In all there are $2n$ events in this order. Let the i'th event correspond to node j. Look at the set of nodes corresponding to the first $i - 1$ events.

We will prove that for the i'th event, the $\alpha_j(z)$ values of the concerned node differ by at most $f(i)\delta z$. Where the function $f(i) = \sum_r^{i-1} f(r)$. We will proceed by induction. The hypothesis is clearly true for $i = 1$, because no node has been considered yet. We will show that the hypothesis holds for the i'th event, for the node j in question as well. Without loss of generality assume that it was the $\alpha_j(z')$ value. If the value $\alpha_j(z'')$ has been considered already then there is nothing to prove. Therefore we assume that the value $\alpha_j(z'')$ occurs after this event. The node j's dual value got fixed at this value because it got a connecting witness u at the time $t = \alpha_j(z')$. Therefore in the solution corresponding to z', either at this point u was declared open, or $t = \alpha_j(z') = c_{uj}$ and u was declared open already. In both cases $t \geq c_{ij}$. Also since the $\alpha_j(z)$ values are increased uniformly for other nodes, not involved in the first $i-1$ events, the current dual value in the order corresponding to both the orders, at time t is exactly t.

We have $p_u(z', t) = \sum_j \max(0, \alpha_j(z', t) - c_{uj}) = z'$ at this point. For all the nodes involved in the first $i-1$ events, their $\alpha_{j'}(z)$ values are perturbed. For all other nodes the current value of the dual solution in both the orders is the same (t). Therefore, $p_u(z'')$ is at least $z' - \sum_r^{i-1} f(r)\delta z$.

If the node u is not declared temporarily open, then clearly in time at most $t + \sum_r^{i-1} f(r)\delta z + \delta z$ the value of $p_u(z'')$ will be z'' unless $\alpha_j(z'')$ stops growing. So in the ordering corresponding to z'', certainly j will get a connecting witness at or before this time. Some other node with a tight edge to j may be declared temporarily open, but in all cases $\alpha_j(z'')$ is at most $\alpha_j(z) + \sum_r^{i-1} f(r)\delta z + \delta z$. Setting $f(i) = \sum_r^{i-1} f(r) + 1$, we prove the induction hypothesis for all cases.

Now since $f(0) = 0$, we can upper bound $f(i)$ by 2^i. In any case since $f(i)$ is finite, in the limit $\delta z \to 0$, we have $\alpha_j(z') = \alpha_j(z'')$. Hence we can conclude that $\alpha_j(z)$ is continuous. ■

4.5 Modified Algorithm for k-Median

We present a primal dual algorithm with approximation factor 4. In fact this will show an integrality gap of 4 for the natural LP formulation for the k-median problem. We will use a primal dual algorithm for facility location to obtain a solution to the

k-median problem. We set the cost of each facility to be z and run the primal dual algorithm for the resulting facility location problem for different values of z.

The order in which events occur, together with the order in which ties are resolved determines the primal solution produced at the end of of the algorithm. The dual solution produced is independent of the order in which ties are resolved. A binary search on the value of z yields two values z_1, z_2 which are very close such that for $z = z_1$, the algorithm produces $\hat{k}_1 < k$ centers (the *small* solution) and for $z = z_2$, the algorithm produces $\hat{k}_2 > k$ centers (the *large* solution). We will carry on the binary search till the values of z are sufficiently "close". We will discuss this in section 4.5.2 later.

The Augment Step: We incorporate an *augmentation* step which adds facilities if possible in the small and large solutions. Note that this step is performed only after the small and large solutions have been found for two very close values of z; *this is not performed during the binary search.*

We describe how the small solution (corresponding to z_1) is augmented. The augmentation of the large solution (corresponding to z_2) is done in a symmetric fashion other than the stopping condition.

1. Consider all nodes which are centers in the large solution and are not centers in the small solution. Let i be the currently considered node.

2. If the small solution has k centers, stop.

3. If there exists a node i' which is chosen as a center in the current small solution (which resulted from a previous augmentation) and demand node j such that $\beta_{ij}(z_1) > 0, \beta_{ij}(z_2) > 0, \beta_{i'j}(z_1) > 0, \beta_{i',j}(z_2) > 0$, i cannot be added. Otherwise add node i to the small solution.

4. Repeat the above steps until the solution has k nodes or all nodes are considered.

Denote the augmented small solution by $\mathsf{soln}^{(1)}$ and the augmented large solution by $\mathsf{soln}^{(2)}$. Let them open k_1 and k_2 centers respectively. By construction, $k_1 \leq k$ and $k_2 \geq k$. The final solution to the k-median problem will be either $\mathsf{soln}^{(1)}$ or $\mathsf{soln}^{(2)}$

or a subset of centers in either of the solutions, under a suitable distribution. Before we prove the approximation factor for the above algorithm we discuss some of the properties of the dual solution.

4.5.1 The Approximation Factor Analysis

The analysis of the performance of the above algorithm will be divided into two parts. In the first part we will demonstrate certain key properties in the two solutions found after the augment step is complete. We will use the lemmata proved in the first part to prove the approximation factor.

Lemma 4.5.1 *[JV99] For any solution constructed by the primal-dual algorithm (say at value z) and any demand node j there exists node u at distances at most $3\alpha_j(z)$ from j. Moreover $t_u(z) \leq \alpha_j(z)$.*

Proof: Suppose u' is the connecting witness of node j in the orders corresponding to z, then $t_{u'}(z) \leq \alpha_j(z)$ and $c_{u'j} \leq \alpha_j(z)$.

If u' was chosen as a center in the solution (while choosing an independent set, before the augmentation step) then $u = u'$ and the lemma is true.

If u' was not chosen as a center, then there exists a center u in the solution such that $t_u(z) \leq t_{u'}(z)$, and there exists a node j' such that $c_{uj'}$ and $c_{u'j'}$ are both less than $\alpha_{j'}(z)$. Also for this node j' we have

$$\alpha_{j'} \leq t_u(z) \leq t_{u'}(z) \leq \alpha_j(z)$$

Now by triangle inequality $c_{uu'} \leq 2\alpha_j(z)$ and by another application $c_{uj} \leq c_{uu'} + c_{u'j} \leq 3\alpha_j(z)$. Therefore the lemma follows. ■

The next lemma is a consequence of the augment step, and the central piece of the improvement.

Lemma 4.5.2 *If $k_1 < k$, for every node u in $\mathsf{soln}^{(2)}$ (resp. $\mathsf{soln}^{(1)}$) there exists a center in $\mathsf{soln}^{(1)}$ (resp $\mathsf{soln}^{(2)}$) which is at most $2\min(t_u(z_1), t_u(z_2))$ away from u.*

Proof: If u were included in $\mathsf{soln}^{(1)}$, then the lemma is true.

Since u was not included in $\mathsf{soln}^{(1)}$ and that $k_1 < k$, u was checked for augmentation in $\mathsf{soln}^{(1)}$. Thus there exists a vertex i and a vertex v such that,

- v is a center in $\mathsf{soln}^{(1)}$,
- $\beta_{vi}(z_1)$, $\beta_{vi}(z_2)$, $\beta_{ui}(z_1)$, $\beta_{ui}(z_2)$ are positive.

Since $\beta_{vi}(z_1)$ and $\beta_{vi}(z_2)$ are positive, we have $c_{vi} < \alpha_i(z_1)$ and $c_{vi} < \alpha_i(z_2)$. Similarly we have c_{ui} less than both $\alpha_i(z_1)$ and $\alpha_i(z_2)$. Again since $\beta_{ui}(z_1)$ is positive, we have $\alpha_i(z_1) \le t_u(z_1)$ and similarly $\alpha_i(z_2) \le t_u(z_2)$. Thus we can bound c_{vi} and c_{ui} by $\min(\alpha_i(z_1), \alpha_i(z_2))$.

By triangle inequality,

$$c_{uv} \le c_{iv} + c_{iu} < 2\min(\alpha_i(z_1), \alpha_i(z_2)) \le 2\min(t_u(z_1), t_u(z_2))$$

For the case of $u \in \mathsf{soln}^{(1)}$ the argument is exactly the same, except the condition $k_1 < k$ is not required since all the nodes in $\mathsf{soln}^{(1)}$ are candidates for augmentation in $\mathsf{soln}^{(2)}$. ■

Lemma 4.5.3 *For any node u in $\mathsf{soln}^{(1)}$ (or $\mathsf{soln}^{(2)}$), either $\sum_j \beta_{uj}(z_1) = z_1$ or $\sum_j \beta_{uj}(z_2) = z_2$. If there is a single transition point z^* between z_1 and z_2, we have $\sum_j \beta_{uj}(z^*) = z^*$.*

Proof: The second part of the claim follows from the fact that if u is temporarily open in z_1 (resp. z_2) then it is temporarily open for all values of z between z^* and z_1 (resp. z_2). ■

Lemma 4.5.4 *For every demand node j there exists at most one node u in $\mathsf{soln}^{(1)}$ (resp. $\mathsf{soln}^{(2)}$) such that $\min(\beta_{uj}(z_1), \beta_{uj}(z_2)) > 0$. In particular, if there exists a single transition point z^* between z_1 and z_2, then for every demand node j there exists at most one node u in $\mathsf{soln}^{(1)}$ (resp. $\mathsf{soln}^{(2)}$) such that $\beta_{uj}(z^*) > 0$.*

Proof: We prove the lemma for $\mathsf{soln}^{(1)}$, for $\mathsf{soln}^{(2)}$ the arguments are exactly the same.

Suppose not, then there exists two such nodes u and v. We have $\beta_{uj}(z_1)$, $\beta_{uj}(z_2)$, $\beta_{vj}(z_1)$, $\beta_{vj}(z_2)$ all positive.

Consider the node (amongst u and v) which it was included later. It cannot be included in phase 2 corresponding to z_1, since the edges (u,j) and (v,j) both were tight at z_1 (since the $\beta_{uj}(z_1)$ and $\beta_{vj}(z_1)$ are positive). It also cannot be included by the augment step since all $\beta_{uj}(z_1)$, $\beta_{uj}(z_2)$, $\beta_{vj}(z_1)$, and $\beta_{vj}(z_2)$ are positive. Therefore for all cases, u and v could not together be in $\mathsf{soln}^{(1)}$.

Assuming there is exactly one transition point between z_1 and z_2 (including them), if $\beta_{uj}(z^*)$ is positive, then by continuity, in a small open interval around z^*, we have $\beta_{uj}(z) > 0$. Now the fact that $\beta_{uj}(z_1) > 0$ means that $\alpha_j(z_1)$ is strictly greater than c_{uj}. Therefore in the semi-open interval $(z^*, z_1]$ if for one point z we have $\beta_{uj}(z) > 0$, then $\beta_{uj}(z)$ positive throughout. Which is a consequence of the ordering of $\alpha_j(z)$ and the c_{uj} remaining invariant. Likewise for $z \in [z_2, z^*)$. Therefore, $\beta_{uj}(z_1)$ and $\beta_{uj}(z_2)$ are both positive, and the lemma follows by the above argument.

Likewise we can argue that for $\mathsf{soln}^{(2)}$. ■

We will give a solution to the k-median problem based on $\mathsf{soln}^{(1)}$ and $\mathsf{soln}^{(2)}$. In fact, we will give a probability distribution on solutions, but this can be derandomized easily by the method of conditional probabilities. In this subsection we will assume that there is *exactly one* transition point between z_1 and z_2. Denote the transition point by z^*. We will argue for that value of $z = z^*$ in the dual solution. In the later part of this section, we will show how to remove that restriction.

We need to distinguish the cases when the augment step considered all nodes or not. This corresponds to $k_1 < k$ and $k_1 = k$. We consider a definition first.

Definition 1 *If for a node j there exists a node u in* $\mathsf{soln}^{(1)}$ *such that* $\min(\beta_{uj}(z_1), \beta_{uj}(z_2))$ *is positive, define the node to be* directly assigned *to u in* $\mathsf{soln}^{(1)}$*. Otherwise define j to be* indirectly assigned *in* $\mathsf{soln}^{(1)}$*. Likewise define directly or indirectly assigned in* $\mathsf{soln}^{(2)}$.

Case $k_1 = k$ or $k_2 = k$. We can claim that we actually have a factor 3 approximation for the k-median problem. Assume $K_1 = k$. If a node j is directly assigned to u in $\mathsf{soln}^{(1)}$, then we set $q_j = \beta_{uj}(z^*)$ otherwise $q_j = 0$. The claim follows for $k_2 = k$ analogously. See 4.5.5.

Case $k_1 < k$ and $k_2 > k$ Consider the following final solution, with $a = \frac{k_2 - k}{k_2 - k_1}$ and $b = \frac{k - k_1}{k_2 - k_1}$.

1. For every node in $\mathsf{soln}^{(1)}$, find the nearest center in the $\mathsf{soln}^{(2)}$.
2. Choose the pair with the smallest distance.
3. Remove both nodes (for now) and repeat the above until every node in $\mathsf{soln}^{(1)}$ is paired up.
4. For each of the k_1 pairs, independently build a center at the node corresponding to $\mathsf{soln}^{(1)}$ with probability a otherwise build a center at the node corresponding to $\mathsf{soln}^{(2)}$ (with probability b).
5. Of the remaining $k_2 - k_1$ nodes in $\mathsf{soln}^{(2)}$ which are not paired up, choose a subset of $k - k_1$ nodes independently at random and build a center there.
6. Assign every node to its nearest center.

Lemma 4.5.5 *If $k = k_1$ then $\mathsf{soln}^{(1)}$ gives a 3 approximation for the k-median problem.*

Proof: By lemma 4.5.4 we have for any node j, $\beta_{uj}(z^*)$ is positive for at most one node u. Therefore,

$$\beta_{uj}(z^*) = \sum_{u \in \mathsf{soln}^{(1)}} \beta_{uj}(z^*)$$

Summing over all j, and using the fact that by lemma 4.5.3 we have $\sum_j \beta_{uj}(z^*) = z^*$, we have

$$\sum_j q_j = \sum_j \beta_{uj}(z^*) = \sum_j \left(\sum_{u \in \mathsf{soln}^{(1)}} \beta_{uj}(z^*) \right) = \sum_{u \in \mathsf{soln}^{(1)}} \left(\sum_j \beta_{uj}(z^*) \right)$$

Thus

$$\sum_j q_j = \sum_{u \in \mathsf{soln}^{(1)}} z^* = k_1 z^*$$

Suppose j was directly assigned to u in $\mathsf{soln}^{(1)}$. In this case $A_j = c_{uj}$. Therefore $A_j + q_j = c_{uj} + \beta_{uj}(z^*) = \alpha_j(z^*)$, moreover since $q_j \leq \alpha_j(z^*)$, we have

$$A_j + 3q_j \leq 3\alpha_j(z^*)$$

If j was indirectly assigned in $\mathsf{soln}^{(1)}$. By lemma 4.5.1 we have a center u in $\mathsf{soln}^{(1)}$ before augmentation (hence after it as well) such that $c_{uj} \leq 3\alpha_j(z_1)$. Since $q_j = 0$ in this case,

$$A_j + 3q_j \leq 3\alpha_j(z_1)$$

Using continuity arguments as $z \to z^*$, for both cases we have

$$A_j + 3q_j \leq 3\alpha_j(z^*)$$

Now by theorem 4.3.1 we have a 3 approximation for the k-median problem. ■

Each center in $\mathsf{soln}^{(2)}$ is present with probability at least b. Also every node in $\mathsf{soln}^{(1)}$ is present with probability a. In the rest of this subsection we will be use case by case analysis for every proof. We will consider the type of a node j defined as follows. II: Indirectly assigned in solutions $\mathsf{soln}^{(1)}$ and $\mathsf{soln}^{(2)}$; DI: Directly assigned to u in $\mathsf{soln}^{(1)}$ and indirectly in $\mathsf{soln}^{(2)}$; and likewise ID and DD. We will set $q_j = q_j(1) + q_j(2)$. If j is assigned directly to u in $\mathsf{soln}^{(1)}$, set $q_j(1) = a\beta_{uj}(z^*)$ else $q_j(1) = 0$. Similarly $q_j(2) = b\beta_{uj}(z^*)$ if j is directly assigned to u in $\mathsf{soln}^{(2)}$ and 0 otherwise.

Lemma 4.5.6 *For the solution described above, for every node j the following is satisfied;*

$$E[A_j] + c \cdot q_j \leq c\alpha_j(z^*) \qquad \sum_j q_j = kz^*$$

where $c = \max(2(1+a), 2(1+b), 3 + 2ab + 2a^2b^2)$.

Proof: The charge q_j has essentially two parts corresponding to direct assignment or otherwise in the solutions. Denote them by $q_j(1)$ and $q_j(2)$ Using a very similar proof as in the first part of lemma 4.5.5, we can show that over all the direct assignment in $\mathsf{soln}^{(1)}$, we have $\sum_j q_j(1) = ak_1 z^*$, and similarly $\sum_j q_j(2) = bk_2 z^*$. The proofs are exactly the same as in the lemma 4.5.5 with factors a and b present appropriately. Since $ak_1 + bk_2 = k$, it follows that $\sum_j q_j = kz^*$.

We will prove the rest of the lemma case by case on the type of a demand node j.

Case DD: Assume the node j was of type DD. Let j be assigned directly to u and v in solutions $\mathsf{soln}^{(1)}$ and $\mathsf{soln}^{(2)}$ respectively. If u and v are paired up then the expected service cost of j is at most $ac_{uj} + bc_{vj}$. If they are not paired, then assuming u got paired up first, if node u is not present then it definitely has a center within distance $c_{uj} + c_{vj}$ from it. For this case node u is present with probability a conditioned on the fact that u is not present, v is present with probability $(1-a)b$ (Since it is independent of the conditioning.) Also there exists a center within distance $2c_{uj} + c_{vj}$ from j if none are present.

So we first try to assign the node to u, if u is not present then we try to assign the node to v, and if neither is then the center to which u is paired to. The expected service cost for this case is

$$E[A_j] \leq ac_{uj} + b^2 c_{vj} + ab(2c_{uj} + c_{vj})$$

Analogously had v been paired up first, the expected cost is at most

$$E[A_j] \leq bc_{vj} + a^2 c_{uj} + ab(2c_{vj} + c_{uj})$$

The RHS of both the equations can be bound by

$$(a + 2ab)c_{uj} + (b + 2ab)c_{vj}$$

In this case $cq_j = ca\beta_{uj}(z^*) + cb\beta_{vj}(z^*)$. Therefore for this case we can bound $E[A_j] + cq_j$ by

$$(a+2ab)c_{uj} + (b+2ab)c_{vj} + ca\beta_{uj}(z^*) + cb\beta_{vj}(z^*) \leq ca(c_{uj} + \beta_{uj}(z^*)) + cb(c_{vj} + \beta_{vj}(z^*))$$

The second part of the inequality follows since both $1 + 2b$ and $1 + 2a$ are at most 3. The RHS sums to $c(a + b)\alpha_j(z^*)$, which proves the lemma for this case.

Case DI. Consider the node u to which j is assigned to directly in $\mathsf{soln}^{(1)}$. We will have two cases to consider depending on t_u and α_j.

Sub-case $\alpha_j(z_1) \geq t_u(z_1)$: By lemma 4.5.2 there exists a node v in $\mathsf{soln}^{(2)}$ such that,

$$c_{uv} \leq 2\min(t_u(z_1), t_u(z_2)) \leq 2\alpha_j(z_1)$$

So if the node u is paired with v or u is paired off before v then there exists a center within distance $2\alpha_j(z_1)$ of u if u is not present. The expected service cost for these two cases is

$$E[A_j] \leq ac_{uj} + b(c_{uj} + 2\alpha_j(z_1))$$

If v was paired off first, then u is present with probability a, and conditioned on u not being present, v is present with probability $(1-b)b$ at a distance $c_{uj} + 2\alpha_j(z_1)$. If both of them are not present, (with probability ab) then there is a center at a distance $2\alpha_j(z_1)$ from v, which is at most $4\alpha_j + c_{uj}$ from j. We first try to assign j to u, if u is not present then to v and if neither then to the center v is paired to. Thus the expected distance can be bounded by

$$ac_{uj} + b^2(2\alpha_j(z_1) + c_{uj}) + ab(4\alpha_j(z_1) + c_{uj}) = c_{uj} + 2b(1+a)\alpha_j(z_1)$$

In this case $q_j = a\beta_{uj}(z^*)$, therefore we have,

$$E[A_j] + cq_j \leq (\max(1, ca))\alpha_j(z^*) + 2b(1+a)\alpha_j(z_1)$$

We argue that the term $\alpha_j(z_1)$ can be made $\alpha_j(z^*)$, which is a consequence of taking limits $z \to z^*$ over the semi-open interval $(z_1, z^*]$ and the continuity of $\alpha_j(z)$.

It is easy to verify that $\max(1, ca) + 2b(1+a)$ is at most c for $a, b > 0$ with $a + b = 1$.

Sub-case $\alpha_j(z_1) < t_u(z_1)$: By applying lemma 4.5.1 for both the solutions constructed at $z = z_1$ and $z = z_2$, we have that there exists nodes u_1 in $\mathsf{soln}^{(1)}$ and u_2 in $\mathsf{soln}^{(2)}$ at distances at most $3\alpha_j(z_1)$ and $3\alpha_j(z_2)$ from j respectively. Moreover $t_{u_1}(z_1) \leq \alpha_j(z_1)$ and $t_{u_2}(z_2) \leq \alpha_j(z_2)$. From this it is clear that $u \neq u_1$.

If $u_1 = u_2$ or that u_1 is paired to u_2 it is easy to show the required bound of $E[A_j]$. Similarly if $u_1 \in \mathsf{soln}^{(2)}$ and $u_2 \in \mathsf{soln}^{(1)}$. Assume that $u_1 \neq u_2$ and $u_1 \notin \mathsf{soln}^{(2)}$ and $u_2 \notin \mathsf{soln}^{(1)}$.

Applying lemma 4.5.2 on u_1, there exists u_3 in $\mathsf{soln}^{(2)}$is at most a distance $2t_{u_1}(z_1)$ by lemma which is at most $2\alpha_j(z_1)$. If $u_2 = u_3$ or $u_3 \in \mathsf{soln}^{(1)}$ then for node j, it first tries node u, then any of u_1 or u_2, and if none of them (u_1 or u_2) is present, then the pair of u_1 or u_2 whichever got paired first.

$$E[A_j] \leq ac_{uj} + b[\max(3\alpha_j(z_1), 3\alpha_j(z_2)) + 2ab\alpha_j(z_1)]$$

For the case $u_2 \neq u_3$, assuming that u_1 was paired off before u_3 (u_3 need not be paired off) the pair of u_1 is at most $2\alpha_j(z_1)$ from u_1 and the above equation can be used to bound $E[A_j]$. If u_3 were paired off first, the pair of u_3 can be at most $2\alpha_j(z_1)$ from u_3 since at the time u_3 was paired off, u_1 was a possible candidate for the pair.

Now applying lemma 4.5.2 on u_2 there exists u_4 in $\mathsf{soln}^{(1)}$which is distance $2\alpha_j(z_2)$ from u_2. If $u_4 = u$ or u_1 or $u_4 \in \mathsf{soln}^{(2)}$ then again the above equation can be derived (changing some $\alpha_j(z_1)$ terms to $\alpha_j(z_2)$ which we will see is not crucial).

Thus the case we are left to consider is $\{u, u_1, u_2, u_3, u_4\}$ are all distinct and u_3 paired off before u_1. In this case we assign j to u, if u is not present to u_1 or u_2. If both u_1 and u_2 are not present, we try u_3 or u_4. If both u_3 and u_4 are absent, we try the pair of u_3. Thus $E[A_j]$ can be bounded by

$$ac_{uj} + b[\max(3\alpha_j(z_1), 3\alpha_j(z_2)) + ab[(1-ab)\max(2\alpha_j(z_1), 2\alpha_j(z_2)) + ab4\alpha_j(z_1)]]$$

Since $\alpha_j(z_1)$ and $\alpha_j(z_2)$ are $\alpha_j(z^*)$ at the critical point, for this sub-case we have

$$E[A_j] + cq_j \leq ca(c_{uj} + \beta_{uj}(z^*)) + b(3 + 2ab + 2a^2b^2)\alpha_j(z^*) \leq ca\alpha_j(z^*) + cb\alpha_j(z^*)$$

Remark: There is a small dependency in the probability of u_2 and u_4, however since we are interested in a lower bound of the probability of one being present when the other is not, the dependency actually benefits the inequality .

Case II: Assuming j was of type II we make the same argument as in the latter sub-case of type DI. Except that in this case the node u is not present and $q_j = 0$.

$$E[A_j] + cq_j \leq (3 + 2ab + 2a^2b^2)\alpha_j(z^*) + 0$$

Case ID: Assuming j was of type ID, we can proceed exactly along the lines of the case DI interchanging the role of $\mathsf{soln}^{(1)}$ and $\mathsf{soln}^{(2)}$. We will again have two sub-cases, for the first one we will have

$$E[A_j] + cq_j \leq (\max(1, cb))(c_{uj} + \beta_{uj}(z^*)) + 2a(1+b)\alpha_j(z^*)$$

And for the latter sub case the expression derived will be

$$E[A_j] + cq_j \leq cb\alpha_j(z^*) + a(3 + 2ab + 2b^2a^2)\alpha_j(z^*)$$

Thus the lemma is true for all cases. ■

4.5.2 Running Time

The previous subsection essentially showed that the modified algorithm provides a factor 4 approximation if the binary search is carried on till one single transition point is left. In this subsection we will argue in a similar fashion as in [JV99] that the binary search can be terminated when the difference $\Delta(z) = |z_1 - z_2|$ is very small. Define $\Delta(\alpha) = 2^n\Delta(z)$.

However for the purpose of the proof, we used the fact that the $\beta_{ij}(z)$ values are close (for $z = z_1, z_2, z^*$). In fact we would require that the difference in their values be small, that is $1/poly(n)$ of the optimal solution, for a suitably large $poly(n)$. The caveat is that to ensure that the $\beta_{ij}(z)$ values are polynomially close, the z values have to be exponentially close. In fact from the proof of continuity of the $\alpha_j(z)$ values we know that $\Delta(z)$ must have precision $1/(2^{n+L}poly(n))$ for some polynomial $poly(n)$. This essentially says that the binary search needs to be carried on at least $O((n+L)\log n)$ steps. Thus we have a $O(n^2(n+L)\log n)$ algorithm ($n^2 = m$).

Theorem 4.5.1 *If for some z and $\Delta(\alpha) = c_{min}/(16n^2)$ (c_{min} being the smallest non-zero edge)*

$$\sum_j q_j = zk \quad \textit{and} \quad \mathrm{E}[A_j] + c\cdot q_j \leq c\cdot\alpha_j(z) + 4\Delta(\alpha)$$

then the expected cost of the solution is a factor $c + 2/n$ times the optimal.

Proof: Summing the first equation over all nodes j, we get,

$$\begin{aligned}\sum_j E[A_j] + ckz &\leq \sum_j (E[A_j] + c \cdot q_j) \leq c \sum_j \alpha_j(z) + 4n\Delta(\alpha) \\ \sum_j E[A_j] &\leq c\left(\sum_j \alpha_j(z) - zk\right) + 4n\Delta(\alpha)\end{aligned}$$

If the optimal solution is non-zero, it is clearly at most c_{min}. The case where the optimal solution is cost 0 is trivial. Therefore we can say that $E[A_j] \geq c_{min} \geq 16n^2\Delta(\alpha)$. (This implies $\Delta(z) = c_{min}2^{-n-4}/n^2$.) Thus we have

$$\left(1 - \frac{1}{4n}\right) E[A_j] \leq c\left(\alpha_j(z) - zk\right)$$

Since $\sum_j \alpha_j(z) - zk$ is the value of the dual solution to the k-median problem, this implies that the expected cost of the solution is at most a factor $c + 2/n$ times the optimal, using $c < 4$. ■

The idea of the rest of the proof is simple. We will show the above conditions of the theorem are true at $z = z^*$. Notice that we do not need to know the values q_j. The q_j values will be defined as in the previous section. (Actually the definition will depend on $k_1 = k$ or otherwise.) Thus part of the conditions of the theorem are already met.

Lemma 4.5.7

$$\mathrm{E}[A_j] + c \cdot q_j \leq c \cdot \alpha_j(z^*) + 4\Delta(\alpha)$$

Proof: Consider the proofs of lemma 4.5.5 and lemma 4.5.6. By inspection we can conclude that

$$\mathrm{E}[A_j] + c \cdot q_j \leq c \cdot \max\{\alpha_j(z_1), \alpha_j(z_2), \alpha_j(z^*)\}$$

Now all these $\alpha_j(z)$ values differ from $\alpha_j(z^*)$ by $2^n\Delta(z)$, which is $\Delta(\alpha)$ and c is at most 4. ■

The only other observation we require is that c is at most $4 - 2/k$, since either a or b has to be at least $1/k$.

Therefore we can prove the following theorem,

Theorem 4.5.2 *There is a primal-dual* 4*-approximation for the* k*-median problem in time* $O(n^2(L+n)\log n)$

Chapter 5

Lower Bounds

5.1 Introduction

In this chapter we will consider the lower bounds of approximation known for the facility location problems. We will also present some of the constructions of the integrality gaps of the linear programming approaches. The results on facility location problem was first proved in [GK98]. We can prove that the facility location problem is unlikely to be approximated below a factor of 1.47. The complexity assumption was $NP \not\subseteq DTIME^{\log\log n}$[1]. Consequent to these results, [Fei99], and [Svi99] has improved the complexity assumption on which the hardness result is based, to $P \neq NP$.

The hardness factor of 1.46 will carry over to variants like the Fault tolerant facility location, (See Chapter 8), the Hierarchical Placement Problems, (See Chapter 9) and the Edge Installation Problem, (See Chapter 10). We first present a lower bound for the k-median problem. The results in this chapter appear in [GK98].

[1] $DTIME[f(n)]$ is the class of decision problems that can be solved in (deterministic) time $O(f(n))$.

5.2 The K median Lower Bound

We will show a reduction to a problem very similar to dominating set. The dominating set problem is defined as follows: Given a graph $G = (V, E)$, does there exists a set of k vertices such that every vertex is either in this set or adjacent to any vertex in this set.

The following theorem is regarding the hardness of approximating the dominating set.

Theorem 5.2.1 *[Fei96] A polynomial time approximation algorithm for Dominating Set that has an approximation ratio of $c \ln |X|$ for some constant $c < 1$, would imply $NP \subseteq DTIME[n^{O(\log \log n)}]$.*

Using the above we can show the following:

Theorem 5.2.2 *Unless $P = NP$, we cannot have an approximation algorithm of guarantee $1 + \frac{1}{e} - \epsilon$ for the k-median problem.*

Proof: Consider the reduction of the dominating set problem to the k-median problem. Let $G = (V, E)$ be the dominating set instance. Set all edges in the graph G to have distance 1, and the edges not present to have distance 2. Let $|V| = n$.

If there exists a dominating set with k vertices, then there exists a feasible solution of the k-median problem of cost $n - k$. Now suppose the k-median solution returned was at most $(1 + \frac{1}{e} - \epsilon)(n - k)$, that is we had an approximation algorithm with factor $1 + \frac{1}{e} - \epsilon$.

In the returned solution, if r vertices were at distance 1 in the solution, we would have $(n - r - k)$ vertices at distance 2.

$$(2n - r - 2k) \leq (1 + \frac{1}{e} - \epsilon)(n - k)$$

This simplifies to

$$n - r - k \leq (\frac{1}{e} - \epsilon)(n - k) \leq (\frac{1}{e} + \epsilon)n$$

The number of un-dominated vertices in the instance is at most $n - r - k$. Thus in every iteration, the number of vertices remaining would be $(\frac{1}{e} - \epsilon)$. Thus if we require

ℓ iterations,

$$\left(\frac{1-e\epsilon}{e}\right)^{\ell} n \geq 1$$

Since $-\ln(1-\zeta) \geq \zeta$ for $0 \leq \zeta < 1$, if $\epsilon < \frac{1}{e}$, we have

$$\ln n \geq \ell \ln \frac{e}{1-e\epsilon} \geq \ell(1-\ln(1-e\epsilon)) \geq \ell(1+e\epsilon)$$

Now the medians we have chosen over these iterations form a dominating set, since no vertex is at distance 2. Thus the number of dominating vertices is $1/(1+e\epsilon)$ times $\ln n$, which contradicts Theorem 5.2.1. Thus the k-median problem cannot be approximated better than factor $1+\frac{1}{e}-\epsilon$. ■

5.3 The Facility Location Problem

We will first show that the facility location problem is Max SNP-hard by reducing it to bounded vertex cover, shown Max SNP-hard in [PY91]. Since the best known lower bound for the vertex cover problem is $\frac{7}{6}$, [Hås97], this does not give very strong hardness bounds. We present a scheme to produce much stronger lower bounds, based on the recent results on the hardness of approximating Set Cover [Fei96].

5.3.1 Max SNP-hardness

The following problem was shown to be Max SNP-hard in [PY91].

B-Vertex Cover: Given a graph $G = (V, E)$, with the degree of each node bounded by a constant B, find the minimum cardinality vertex cover. A vertex cover is a subset of vertices such that each edge in G has at least one end point in the cover.

We prove the following theorem via reduction from B-vertex cover.

Theorem 5.3.1 *The Uncapacitated Facility Location problem is Max SNP-hard.*

Proof: The B-vertex cover problem has been shown to be Max SNP-hard in [PY91]. We show that for any $\epsilon > 0$, there exists an $\epsilon' > 0$ such that an $(1+\epsilon')$ approximation algorithm for UFL implies a $(1+\epsilon)$ approximation for B-vertex cover. Given an instance of B-vertex cover, we construct an instance of the UFL problem where the vertices of G correspond to facility locations, and the edges correspond to demand locations. The distance (service cost) between facility i (representing vertex i) and demand location j (representing edge j) is 1 if edge j is incident to vertex i, and 2 otherwise. Let the optimal vertex cover have size k (we may assume that we know this, as $1 \leq k \leq |V|$, and we can try each choice for k). The cost of opening a facility is $\frac{|E|}{Bk}$. All demands are 1.

Suppose that the UFL problem can be approximated within a factor of $1+\epsilon'$, ($\epsilon' = \frac{\epsilon}{1+B}$). Corresponding to the optimal vertex cover solution, there exists an assignment of cost at most $(|E| + \frac{|E|}{B})$. We are guaranteed a solution of cost at most $(|E| + \frac{|E|}{B})(1 + \frac{\epsilon}{1+B}) = |E| + (1+\epsilon)\frac{|E|}{B}$.

Suppose the algorithm chooses βk facilities and covers $t|E|$ edges at distance exactly 2. The total cost is $\beta\frac{|E|}{B} + |E| + t|E|$. This cost can be bounded by $|E| + (1+\epsilon)\frac{|E|}{B}$, thus $\beta + tB \leq (1+\epsilon)$.

By choosing the vertices corresponding to the open facilities and one vertex for each of $t|E|$ edges, we obtain a vertex cover. Since $Bk \geq |E|$, the size of the vertex cover is $\beta k + t|E| \leq (\beta + tB)k \leq (1+\epsilon)k$. ■

5.3.2 A 1.467 Lower Bound

We reduce the Set Cover problem to the uncapacitated facility location problem. The basic idea is to use the approximation algorithm for solving UFL to find partial solutions for the set cover instance, by repeatedly running the UFL algorithm. If the algorithm repeatedly finds a small partial cover, covering a large fraction of the elements, then we obtain a $c \ln |X|$ approximation for Set Cover with $c < 1$, which is unlikely.

Consider an instance of Set Cover defined as follows: let the set of elements be $X = \{x_1, \ldots, x_n\}$. Let the collection of sets be defined as $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ where each $S_i \subseteq X$. Moreover, we may assume that k is the size of the minimum set cover and is known to the approximation algorithm. (This is not a restricting assumption, since $1 \leq k \leq m$, and we can run a Set Cover algorithm for each possible value of k.)

Theorem 5.3.2 *[Fei96] A polynomial time approximation algorithm for Set Cover that has an approximation ratio of $c \ln |X|$ for some constant $c < 1$, would imply $NP \subseteq DTIME[n^{O(\log\log n)}]$.*

Based on the above, we can show the following:

Theorem 5.3.3 *If there is a polynomial time algorithm with an approximation factor smaller than 1.463 for the metric uncapacitated facility location problem then $NP \subseteq DTIME[n^{O(\log\log n)}]$.*

Proof: We transform an instance of set cover $(X, \mathcal{S})$ to an instance of the UFL problem. We set $D = X$ and $F = \mathcal{S}$. Each f_i value (facility construction cost) is identical. If $x_j \in S_i$ then $c_{ij} = 1$ (in other words, distances are 1 to the sets (facilities) that this element (demand point) belongs to). We extend c_{ij} to be a metric using shortest paths. We set $d_j = 1$ for each demand point j.

Let γ be a constant whose value will be specified later. We assume that Approx-Algorithm (F, D) is a polynomial time δ approximation algorithm for UFL. The facility location algorithm return a set of facilities (sets). Consider the following algorithm Set Cover $(X, \mathcal{S})$ to solve an instance of Set Cover.

> Create a UFL instance (F, D) corresponding to $(X, \mathcal{S})$
>
> Initially $Cover = \Phi$
>
> While $D \neq \Phi$ Do
>
> > Let $f = \gamma \frac{|D|}{k}$ (cost of a facility)
> >
> > $F' =$ Approx-Algorithm (F, D)
> >
> > Let D' be the elements covered at distance 1
> >
> > Let $Cover \leftarrow Cover \cup F'$, $F \leftarrow F - F'$ and $D \leftarrow D - D'$

Suppose that in iteration ℓ, $n_\ell = |D|$ and $f^{(\ell)}$ is the cost of a facility (uniform costs for all facilities). Since there is a solution of size k to the set cover problem that covers all the elements, there is a solution to the facility location problem of cost $f^{(\ell)}k + n_\ell$ (we build facilities at the k locations corresponding to the set cover, all the service costs are 1, recall $|D| = n_\ell$).

If there is an α approximation algorithm for UFL, then we are guaranteed to obtain a solution of cost at most $\delta(f^{(\ell)}k + n_j)$. We set the value of $f^{(\ell)} = \gamma\frac{n_\ell}{k}$. Our total cost is at most $\delta n_\ell(1+\gamma)$.

We now consider a solution obtained by the approximation algorithm. Suppose we open $b_\ell k$ facilities at cost $f^{(\ell)}$ each. Suppose the number of clients that are within distance 1 of an open facility is $c_\ell n_\ell$. (Both b_ℓ and c_ℓ are ratios, and not constants.) All other clients (uncovered elements) are at distance at least 3. The cost of this solution is at least $f^{(\ell)}b_\ell k + c_\ell n_\ell + 3(n_\ell - c_\ell n_\ell)$. This cost is at most $\delta n_\ell(1+\gamma)$. Thus we obtain (after dividing by n_ℓ)

$$\delta(1+\gamma) \geq \gamma b_\ell + 3 - 2c_\ell \Rightarrow \delta \geq \frac{\gamma b_\ell + 3 - 2c_\ell}{(1+\gamma)}.$$

Define for some fixed constant $0 < c < 1$,

$$c_{b_\ell} = (1 - \frac{1}{e^{\frac{b_\ell}{c}}})$$

Now we end up in two cases,

Case a: For some ℓ, $c_\ell \leq c_{b_\ell}$; then

$$\delta \geq \frac{\gamma b_\ell + 3 - 2c_{b_\ell}}{(1+\gamma)}.$$

If we fix γ and c, and compute the minimum value as a function of b_ℓ we can compute the smallest value for the RHS. This will certainly be a lower bound for δ, our approximation ratio. Taking the derivative with respect to b_ℓ, tells us that the minimum is achieved at $b_\ell = c\ln\frac{2}{c\gamma}$. Substituting this value of b_ℓ gives us

$$\delta \geq \frac{1+c\gamma}{1+\gamma} + \frac{c\gamma}{1+\gamma}\ln\frac{2}{c\gamma}.$$

Choosing $\gamma = 0.463$ gives $\alpha > 1.463$ when c is a constant close to 1.

Case b: For each ℓ, $c_\ell > c_{b_\ell}$. Then we have an algorithm that picks $b_\ell k$ sets and covers $c_\ell n_\ell$ elements each time we run the algorithm. Suppose in iteration ℓ, the number of uncovered elements at the start of the iteration is n_ℓ. Clearly, $n_{\ell+1} = n_\ell(1 - c_\ell)$ and $n_1 = |X|$. Suppose after t iterations $n_{t+1} = 1$. The total number of sets that are picked is $\sum_{\ell=1}^{t} b_\ell k$, with an approximation factor of $\sum_{\ell=1}^{t} b_\ell$.

$$n_{t+1} = 1 = |X| \prod_{\ell=1}^{t} (1 - c_\ell).$$

$$\ln |X| = \sum_{\ell=1}^{t} \ln \frac{1}{1 - c_\ell}.$$

Using the fact that $c_\ell > (1 - \frac{1}{e^{\frac{b_\ell}{c}}})$, we get for each ℓ,

$$b_\ell < c \cdot \ln \frac{1}{1 - c_\ell}.$$

$$\sum_{\ell=1}^{t} b_\ell < c \sum_{\ell=1}^{t} \ln \frac{1}{1 - c_\ell} = c \ln |X| \text{ for } c < 1.$$

Thus Set Cover $(X, \mathcal{S})$ is a $c \ln |X|$ approximation algorithm with $c < 1$. ■

The above result is interesting when contrasted with Corollary 2.2.2, since for metrics with distances one or three, we have a factor 1.47 approximation algorithm.

5.4 Gap Examples

Integrality gaps are tied to the linear programming formulations used, in this section we will demonstrate the best known gaps for the different linear programming formulations we have used in the approximation algorithms. There are some gaps in the upper bounds of these integrality gaps, and the lower bounds we will show; the interesting question being their resolution.

In case of primal dual algorithms, the situation has yet another parameter. Since we are comparing the result to th dual lower bound, the integrality gap depends on the construction of the dual. We will show some results about these algorithms. In [JV99], it was shown that the primal dual algorithm presented can have an approximation

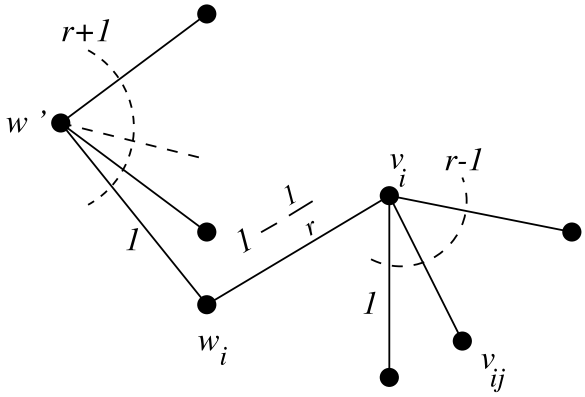


Figure 5.1: Lower bound example for primal dual k-median algorithm

factor 3, we will show that the dual constructed in the process will be a third of the integral solution. These lower bounds turn out to be more interesting, since using combination of local search and primal dual algorithms, we can have much better approximation factors.

5.4.1 The k-Median Problem : Primal Linear Program

Consider the Linear Program considered in Section 3.3. And consider the example of the star graph with $k+1$ leaf vertices. Each of the leaf nodes have demand 1, and $y_i = 1 - \frac{1}{k}$. The vertex at the center has $y_i = \frac{1}{k}$ and demand 0.

Thus overall we have $\sum_i y_i = k$. The cost of the solution is $(k+1)\frac{1}{k} = 1 + \frac{1}{k}$. It is easy to observe that the best integral solution will have a cost of 2. Thus the gap is at least $2 - \frac{2}{1+k}$.

5.4.2 The k-Median Problem: The Dual Constructed

Consider the example shown in Figure 5.1. The instance is defined on a tree rooted at w', and r is a parameter. w' has $r+1$ children $w_1, \ldots, w_{r+1}$. Each node w_i has a single child v_i. Further, each node v_i has $r-1$ children $v_{i1}, \ldots, v_{i(r-1)}$. The distance between w' and w_i is 1, that between w_i and v_i is $1 - 1/r$, and the distance between v_i and v_{ij} is 1. All other distances are the shortest path distances along this tree. The nodes w_i and v_{ij} have unit demands; all other nodes have zero demand. For this

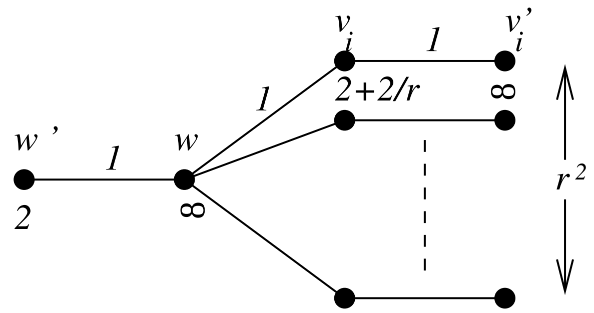


Figure 5.2: Lower bound example for primal dual facility location algorithm

instance $k = 2$. The transition point occurs at $z = \theta = 1 + 1/r$. For $z = \theta^-$, all the v_i nodes are chosen as facilities; this solution has $r + 1$ centers. For $z = \theta^+$, only w' is chosen as a center; this solution has 1 center. At the transition point all the α values are $1 + 1/r$. The value of the dual solution is $r^2 + O(r)$. However, the value of the optimal solution is $3r^2 - O(r)$. Thus the ratio between the two exceeds $3 - \epsilon$ for arbitrarily small ϵ by choosing r sufficiently large.

5.4.3 Facility Location: The Dual Constructed

Consider the example shown in Figure 5.2. The instance is defined on a tree rooted at w', and r is a parameter. w' has a single child w at unit distance from it. w has r^2 children $v_1, \ldots, v_{r^2}$, each at unit distance from w. Further, each v_i has a single child v_i' at unit distance from it. All other distances are shortest path distances along the tree. The node w' has a facility cost of 2 and the nodes v_i have facility costs of $2 + 2/r$; all other nodes have facility costs of ∞. The node w has demand $2r$. Each v_i' has unit demand and the rest of the nodes have zero demand. In the dual solution constructed by the algorithm, the $\alpha(w)$ is $2r(1 + 1/r)$ and the $\alpha(v_i')$ value for each of the v_i' is $1 + 2/r$. The value of the dual solution is $r^2 + 4r + 2$. However, the value of the optimal solution is $3r^2 + 2r + 1$ (corresponding to choosing only w' as a facility). The ratio of the optimal solution value to dual solution value exceeds $3 - \epsilon$ for r suitably large.

5.4.4 Facility Location: The Primal Linear Program

We give a family of examples for which any rounding strategy applied to the natural LP formulation in [Bal66]. will be forced to increase the cost of the integral solution by a factor of about 1.463.

The set F is a set of k vertices. Each $f_i = 1$. The set D is a set of demand points, with each $d_j = 1$. We create $\binom{k}{\ell}$ demand points, one corresponding to *each* subset $S \subset F$ of size ℓ. This demand point is at a service distance c from the facilities in subset S. One should note that the distance to a facility $j \in F - S$ is exactly $3c$. (There must be a demand point that is adjacent to j and a facility in S.)

The cost of the minimum fractional solution to the linear programming formulation, [Bal66, STA97], is at most $\frac{k}{\ell} + c \cdot \binom{k}{\ell}$. (This solution is obtained by assigning a fractional value y_i at each facility of $\frac{1}{\ell}$.) Since each demand point is adjacent to a set S of facilities, it can be fractionally satisfied with cost c.

The optimal integral solution has cost

$$\min_{k'}(k' + c\binom{k}{\ell} + 2c\binom{k-k'}{\ell}).$$

This cost is obtained by choosing k' facilities from the set F, and $\binom{k-k'}{\ell}$ is the number of demand points at distance $3c$ from the set of open facilities. Our ratio r is at least

$$\frac{\min_{k'}(k' + c(\binom{k}{\ell} + 2\binom{k-k'}{\ell}))}{\frac{k}{\ell} + c \cdot \binom{k}{\ell}}.$$

Let $\rho c\binom{k}{\ell} = \frac{k}{\ell}$, where $\rho = 0.463$. Let $\zeta = 1 + \frac{1}{\sqrt{\ell}}$.

There are two possible cases.

Case (a) $k' \geq 2\frac{\zeta-1}{\zeta^2}(k - \ell)$.

In this case we can say that

$$r \geq \frac{k'}{\frac{k}{\ell}(1 + \frac{1}{\rho})} \geq \frac{2\ell\rho(\zeta - 1)}{\zeta^2}\left(\frac{k-\ell}{k}\right)\frac{1}{(1+\rho)}.$$

Since we can choose ℓ such that $1 - \frac{\ell}{k} > \frac{1}{2}$, and $\zeta^2 \leq 2$,

$$r \geq \frac{\rho\sqrt{\ell}}{2(1+\rho)} \geq 1.5$$

The other case is:

Case (b) $k' < 2\frac{\zeta-1}{\zeta^2}(k-\ell)$.

This can be rewritten as $\frac{1}{2}\left(\frac{k'\zeta}{k-\ell}\right)^2 < (\zeta-1)\left(\frac{k'}{k-\ell}\right)$. Using the fact that $e^{-z} \leq 1 - z + \frac{1}{2}z^2$, we obtain

$$e^{-\frac{\zeta k'}{k-\ell}} \leq 1 - \frac{\zeta k'}{k-\ell} + \frac{1}{2}\left(\frac{\zeta k'}{k-\ell}\right)^2.$$

Using the bound $\frac{1}{2}\left(\frac{k'\zeta}{k-\ell}\right)^2 < (\zeta-1)\left(\frac{k'}{k-\ell}\right)$ we obtain

$$e^{-\frac{\zeta k'}{k-\ell}} \leq 1 - \frac{\zeta k'}{k-\ell} + \frac{1}{2}\left(\frac{\zeta k'}{k-\ell}\right)^2 < 1 - \frac{k'}{k-\ell}.$$

Let

$$\nu = \frac{\binom{k-k'}{\ell}}{\binom{k}{\ell}} = \Pi_{i=0}^{i=\ell-1}(1 - \frac{k'}{k-i}) > \Pi_{i=0}^{i=\ell-1}(1 - \frac{k'}{k-\ell}) > \left(e^{-\frac{\zeta k'}{k-\ell}}\right)^{\ell}.$$

Thus

$$r \geq \frac{\min_{k'}(k' + c(\binom{k}{\ell} + 2\nu\binom{k}{\ell}))}{\frac{k}{\ell} + c\cdot\binom{k}{\ell}} > \frac{\min_{k'}(k' + c\binom{k}{\ell}(1 + 2e^{-\frac{\zeta k'\ell}{k-\ell}})}{\frac{k}{\ell} + c\cdot\binom{k}{\ell}}.$$

Let $\chi = \frac{\ell}{k}k'$. We obtain

$$r \geq \min_{\chi} \frac{\chi\rho c\binom{k}{\ell} + c\binom{k}{\ell}(1 + 2e^{-\chi\zeta\frac{k}{k-\ell}})}{c\binom{k}{\ell}(1+\rho)}$$

$$= \min_{\chi} \frac{\chi\rho + 1 + 2e^{-\chi\zeta\frac{k}{k-\ell}}}{1+\rho}.$$

This is minimized for $\chi = \frac{k-\ell}{k\zeta}\ln\frac{2\zeta k}{\rho(k-\ell)}$. This yields

$$r \geq \frac{1 + \frac{(k-\ell)\rho}{k\zeta} + \frac{(k-\ell)\rho}{k\zeta}\ln\frac{2\zeta k}{\rho(k-\ell)}}{1+\rho}.$$

In the limit when $\frac{k-\ell}{k\zeta} \to 1$, the supremum is $1 + \frac{\rho\ln\frac{2}{\rho}}{1+\rho}$ which is maximized for the given choice of $\rho = 0.463$ to be 1.463. The maximum value is obtained for $\rho = 0.463$ when we get $r \geq 1.463$.

Chapter 6

Toolbox – I: Developing Different Approaches

6.1 Introduction

In this chapter we will explore the different ways of using the approaches to the facility location problem. These techniques of combining different heuristics, provide the best known constant approximation for the facility location problem, as well as provide methods to analyze different generalizations. None of the techniques discussed in this section will improve any result significantly, but in combination will improve several approximation results in connection to facility location.

The first technique we will consider is scaling facility costs. In the last three chapters we have seen that the approximation guarantees of the various algorithms are non-uniform in terms of the coefficients in front of the facility and assignment costs. We will try to exploit this asymmetry and design algorithms with better approximation guarantees. The central theme would be to pretend to the algorithm that the facility costs are larger or smaller and force the algorithm to choose a better solution. The first application of the scaling technique will be in tradeoff results. Scaling will also improve approximation guarantees for the capacitated facility location case, and in combination with other heuristics yield one of the components of the best possible facility location approximation.

Combining different heuristics will be the second part of our discussion in this chapter. The combination of linear programming and rounding with greedy 'Add' heuristic yields the best constant factor for the Fault Tolerant Facility Location Problem, see Chapter 8. The combination of scaling, primal dual algorithm with the greedy 'Add' heuristic gives the best possible result, if we want an approximation algorithm that bounds the total cost with unequal coefficients in front of the facility or the assignment costs of a feasible solution. This will be used in Chapter 9 in designing hierarchical networks.

In the last section we will consider the capacitated k Median problem. This will demonstrate the use of half integral solutions in the analysis of the problems where at most a fixed number of facilities can be built. The analysis uses the half integral solution as a way to amortize the cost of rounding the solution to different demand locations. In order to achieve this, we will have to perform a series of rounding steps that will require modification of the assignments. A similar set of path based modification of assignment will be used in context of Lower Bounded Facility Location, in Chapter 7, which will be used significantly in designing access networks, in Chapter 10.

Formally the tradeoff represents a bicriterion approximation. The known results on the tradeoff problem use a (p, q) notation where the first parameter p denotes the approximation factor of the facility cost and q the approximation factor for the assignment cost. The first such result appeared in [LV92a] for the k-median problem, which was $(1+\gamma, 2+2/\gamma)$. In [STA97] a tradeoff of $(1+\gamma, 3+3/\gamma)$ was presented based on filtering techniques, for the facility location problem. In [KPR98] a tradeoff of $(1+\gamma, 3+5/\gamma)$ was presented; using a combinatorial local search algorithm, first proposed by in [KH63]. The techniques of [MRS+98] for bicriteria approximations also yield tradeoffs for facility cost versus assignment cost. Their results are stated in terms of tradeoffs for similar objectives, e.g. (cost,cost) or (diameter,diameter) under two different cost functions. However, their parametric search algorithm will also yield tradeoffs for different objective functions provided there exists a ρ-approximation algorithm to minimize the sum of the two objectives. By scaling the cost functions, their algorithm produces a tradeoff of $(\rho(1+\gamma), \rho(1+1/\gamma))$. For tradeoffs of facility cost versus assignment cost, ρ is just the approximation ratio for the facility location problem.

We will present a $(1+\gamma, 1+2/\gamma)$ tradeoff. This is the first tradeoff result in which we can achieve the coefficient for the assignment cost to be arbitrary close to one. The result presented here is also interesting since the tradeoff of $(1+\gamma, 1+1/\gamma)$ is the best tradeoff possible. This is illustrated by a very simple example.

6.2 Scaling Costs

In this section we will show how cost scaling can be used to show a better approximation guarantee. The idea of scaling exploits the asymmetry in the guarantees of the service cost and facility cost.

We will scale the facility costs uniformly by some factor (say λ). We will then solve the modified problem using local search (or in later sections by a suitable algorithm). While arguing about the cost of the final solution, we will have to scale back to determine the cost in the original instance.

As a simple application of the technique, we can prove the following:

Theorem 6.2.1 *The uncapacitated facility location problem can be approximated to factor $1+\sqrt{2}+\epsilon$ in randomized $O(\frac{n^2}{\epsilon}\log n)$ time.*

Proof: Assume that the facility cost and the service costs of the optimal solution are denoted by F_{OPT} and C_{OPT}. Then after scaling there exists a feasible solution to the modified problem of modified facility cost λF_{OPT} and service cost C_{OPT}. By Theorem 2.5.1, for some small ϵ' we have a solution with service cost C and facility cost F such that

$$F \leq (1+\epsilon')(\lambda F_{\text{OPT}} + 2C_{\text{OPT}}) \qquad\qquad C \leq (1+\epsilon')(\lambda F_{\text{OPT}} + C_{\text{OPT}})$$

Scaling back, we will have a solution of the same service cost and facility cost F/λ. Thus the total cost of this solution will be

$$C + F/\lambda \leq (1+\epsilon')\left[(1+\lambda)F_{\text{OPT}} + (1+\frac{2}{\lambda})C_{\text{OPT}}\right]$$

Setting $\lambda = \sqrt{2}$ gives a $1 + \sqrt{2} + \epsilon$ approximation where $\epsilon = (1 + \sqrt{2}\epsilon')$.

Observe that Theorem 2.5.1 provides us with $O(\frac{n}{\epsilon'} \log n)$ solutions to the scaled instance. If we consider the actual (scaled back) cost of these $O(\frac{n}{\epsilon} \log n)$ solutions, the above proof shows that the best solution must satisfy the theorem. This also illustrates the typical scenarios in which we will use Theorem 2.5.1. ■

6.2.1 Bicriterion Tradeoff Results

Facility location problems model tradeoff scenarios, and how closely we can achieve this tradeoff is an interesting question. We can prove the following bicriterion result:

Theorem 6.2.2 *Let* SOL *be any solution to the facility location problem with facility cost* F_{SOL} *and service cost* C_{SOL}*. For any* $\gamma > 0$*, the local search heuristic proposed (together with scaling) gives a solution with facility cost at most* $(1 + 2/\gamma)F_{\text{SOL}}$ *and service cost at most* $(1+\gamma)C_{\text{SOL}}$*. The approximation is upto multiplicative factors of* $(1 + \epsilon)$ *for arbitrarily small epsilon.*

Proof: Reconsider the proof of Theorem 6.2.1. The guarantee provided by the local search procedure after scaling back yields facility cost C and service cost C such that,

$$F \leq (1 + \epsilon)(F_{\text{SOL}} + 2C_{\text{SOL}}/\lambda) \qquad\qquad C \leq (1 + \epsilon)(\lambda F_{\text{SOL}} + C_{\text{SOL}})$$

Setting $\lambda = 2C_{\text{SOL}}/(\gamma F_{\text{SOL}})$ we get that the facility cost is at most $(1+\gamma)F_{\text{SOL}}$ and the service cost is $(1 + 2/\gamma)F_{\text{SOL}}$ upto factors of $(1 + \epsilon)$ for arbitrary small ϵ. Moreover the guarantee only requires the value of the parameter λ, and not the explicit knowledge of the respective costs. In fact the costs can be guessed upto factors in polynomial in n, and we can try all possible values of the parameter λ upto factors of some $1 + \epsilon'$ and we will be guaranteed that some value of λ exists (upto $1 + \epsilon'$ factor) such that it will be correct. This factor of $(1 + \epsilon')$ can be absorbed by the $1 + \epsilon$ term associated with the tradeoff. Thus in some sense we can achieve the tradeoff in an oblivious fashion.[1] ■

[1] This results holds for arbitrary fractional feasible solutions as well.

Lower bound for tradeoff We present an example to prove that the $(1+\gamma, 1+2/\gamma)$ tradeoff between facility and service costs is almost the best possible when comparing with a fractional solution of the facility location LP. Consider the following instance. The instance $\mathcal{I}$ consists of two nodes u and v, $c_{uv} = 1$. The facility costs are given by $f_u = 1$, $f_v = 0$. The demands of the nodes are $d_u = 1$, $d_v = 0$.

Theorem 6.2.3 *For any $\gamma > 0$, there exists a fractional solution to $\mathcal{I}$ with facility cost F_{SOL} and service cost C_{SOL} such that there is no integral solution with facility cost less than $(1+\gamma)F_{\mathrm{SOL}}$ and service cost less than $(1+1/\gamma)C_{\mathrm{SOL}}$.*

Proof: Observe that there are essentially two integral solutions to $\mathcal{I}$. The first, SOL_1, chooses u as a facility, $F_{\mathrm{SOL}_1} = 1$, $C_{\mathrm{SOL}_1} = 0$. The second, SOL_2, chooses v as a facility, $F_{\mathrm{SOL}_2} = 0$, $C_{\mathrm{SOL}_2} = 1$. For $\gamma > 0$, we will construct a fractional solution for $\mathcal{I}$ such that $F_{\mathrm{SOL}} = 1/(1+\gamma)$, $C_{\mathrm{SOL}} = \gamma/(1+\gamma)$. The fractional solution is obtained by simply taking the linear combination $(1/(1+\gamma))\mathrm{SOL}_1 + (\gamma/(1+\gamma))\mathrm{SOL}_2$. It is easy to verify that this satisfies the conditions of the lemma. ■

Theorem 6.2.3 proves that a tradeoff of $(1+\gamma, 1+1/\gamma)$ for facility cost versus service cost is best possible.

6.2.2 Scaling and Capacitated Facility Location

The scaling idea can also be used to improve the approximation ratio for the capacitated facility location problem. [CW99] prove that a local search algorithm produces a solution with the following properties (modified slightly from their exposition):

$$\begin{aligned} C &\leq (1+\epsilon')(F_{\mathrm{OPT}} + C_{\mathrm{OPT}}) \\ F &\leq (1+\epsilon')(5F_{\mathrm{OPT}} + 4C_{\mathrm{OPT}}) \end{aligned}$$

This gives a $6+\epsilon$ approximation for the problem. However, we can exploit the asymmetric guarantee by scaling. Scaling the facility costs by a factor λ, we get a solution for the scaled instance such that:

$$\begin{aligned} C &\leq (1+\epsilon')(\lambda F_{\mathrm{OPT}} + C_{\mathrm{OPT}}) \\ F &\leq (1+\epsilon')(5\lambda F_{\mathrm{OPT}} + 4C_{\mathrm{OPT}}) \end{aligned}$$

Scaling back, we get a solution of cost $C + F/\lambda$.

$$C + F/\lambda \leq (1+\epsilon')\left[(5+\lambda)F_{\mathrm{OPT}} + (1+\frac{4}{\lambda})C_{\mathrm{OPT}}\right]$$

Setting $\lambda = 2\sqrt{2} - 2$, we get a $3 + 2\sqrt{2} + \epsilon$ approximation.

6.3 Combining Different Heuristics

In this section we will show how to combine various different heuristics for the facility location to get the best approximation possible. Along the way, we will also investigate ways of combining heuristics that do not give the best result for the problem, but gives the best result for some generalizations.

In Chapter 2 we showed that the improvement achieved in an iterative step depended on how the facility and assignment costs of our solution were compared to the facility and assignment costs of a feasible integral solution. In this section we will first show that those results hold with respect to a feasible fractional solution as well. The first benefit of such a result would be a tradeoff result proved in the last section, about fractional solutions. This is interesting because the first tradeoff result of [LV92b] was with respect to fractional solutions. In the next subsection we will consider the 'Add' heuristic in combination with filtering. The next two subsections will present results on combining primal dual algorithm with 'Add' and subsequently developing the best constant factor for approximating facility location.

6.3.1 Local Search And Feasible Fractional Solutions

We first prove the two lemmas concerning the progress of local search step in comparison with a fractional solution. The facility location LP is as follows:

$$\begin{aligned} \min \sum_i y_i f_i + \sum_{ij} x_{ij} c_{ij} & \\ \forall ij \quad x_{ij} \leq y_i & \\ \forall j \quad \sum_i x_{ij} \geq 1 & \\ x_{ij}, y_i \geq 0 & \end{aligned}$$

Here y_i indicates whether facility i is chosen in the solution and x_{ij} indicates whether node j is serviced by facility i.

Lemma 6.3.1 *$\sum Gain(i) \geq C - F_{\mathrm{LP}} - C_{\mathrm{LP}}$, where F_{LP} and C_{LP} are the facility and service costs for an arbitrary fractional solution* SOL *to the facility location LP.*

Proof: Consider the fractional solution SOL to the facility location LP. $F_{\mathrm{LP}} = \sum_i y_i f_i$ and $C_{\mathrm{LP}} = \sum_{ij} x_{ij} c_{ij}$. We will prove that $\sum_i y_i \cdot Gain(i) \geq C - F_{\mathrm{LP}} - C_{\mathrm{LP}}$. Assume without loss of generality that $y_i \leq 1$ for all i and $\sum_i x_{ij} = 1$ for all j.

We first modify the solution (and make multiple copies of facilities) so as to guarantee that for all ij, either $x_{ij} = 0$ or $x_{ij} = y_j$. The number of ij for which this condition is violated are said to be violating ij. The modification is done as follows: Suppose there is violating ij, i.e. there is a facility i and demand node j in our current solution such that $0 < x_{ij} < y_j$. Let $x = \min_j \{x_{ij} | x_{ij} > 0\}$. We create a copy i' of node i (i.e. $c_{i'i} = 0$, $c_{i'j} = c_{ij}$ for all j and $f_{i'} = f_i$.) We set $y_{i'} = x$ and decrease $y_{i'}$ by x. Further, for all j such that $x_{ij} > 0$, we decrease x_{ij} by x and set $x_{i'j} = x$. For all the remaining j, we set $x_{i'j} = 0$. Note that the new solution we obtain is a fractional solution with the same cost as the original solution. Also, all $i'j$ satisfy the desired conditions. Further, the number of violating ij. In at most n^2 such operations, we obtain a solution that satisfies the desired conditions. Suppose $i_1, \ldots, i_r$ are the copies of node i created in this process, The final value of $\sum_{l=1}^{r} y_{i_l}$ is the same as the initial value of y_i.

For a demand node j, let $\sigma(j)$ be the facility assigned to j in the current solution. We now proceed with the proof. Define $Gain'()$ as in proofs of Lemma 2.2.1 and Lemma 2.4.1, that we consider changing the assignments of j to i_l, iff $x_{i_l j} > 0$.

$$Gain'(i_l) = -f_{i_l} + \sum_{j, x_{i_l j} > 0} d_j (c_{\sigma(j),j} - c_{i_l j})$$

Multiplying by y_{i_l}, and using the fact $x_{i_l j} > 0$ iff $x_{i_l j} = y_{i_l}$,

$$y_{i_l} Gain'(i_l) = -f_i y_{i_l} \sum_{j, x_{i_l j} > 0} d_j x_{i_l j} (c_{\sigma(j),j} - c_{i_l j})$$

Summing this over all copies of i, and since $Gain(i) \geq Gain'(i_l)$ for all copies,

$$\sum_i \sum_l -f_i y_{i_l} \sum_{j, x_{i_l j} > 0} d_j x_{i_l j}(c_{\sigma(j),j} - c_{i_l j}) \leq \sum_i \sum_l y_{i_l} Gain(i)$$

Rewriting the equation, using the fact $\sum_l y_{i_l} = y_i$, and $\sum_l x_{i_l j} = x_{ij}$,

$$\sum_i Gain(i) \geq \sum_i y_i Gain(i) \geq -\sum_i y_i f_i + \sum_j d_j c_{\sigma(j)j} \left(\sum_i x_{ij}\right) - \sum_j \sum_i x_{ij} c_{ij} d_j$$

Using $\sum_i x_{ij} = 1$, the lemma follows. ■

The last steps of the above proof can be modified along the lines of the proof of Corollary 2.2.1 to prove the following:

Corollary 6.3.1 *If F_{LP} and C_{LP} are the facility and service costs for an arbitrary fractional solution* SOL *to the facility location LP, then there exists a node with ratio of gain at least $(C - F_{\text{LP}} - C_{\text{LP}})/F_{\text{LP}}$.*

Using Lemma 2.2.2, we get the following lemma similar to Lemma 2.2.3.

Lemma 6.3.2 *Starting from a solution with facility cost F and assignment cost C, and if there exists a solution* SOL*with facility cost $F_{\text{SOL}}$and assignment cost C_{SOL}, then the cost of the final solution output by the greedy strategy is at most*

$$F + F_{\text{LP}} \max\left\{\log \frac{C - C_{\text{LP}}}{F_{\text{LP}}}, 0\right\} + C_{\text{LP}} + F_{\text{LP}}$$

We will now prove a fractional version of Lemma 2.4.2.

Lemma 6.3.3 *$\sum Gain(i) \geq F - F_{\text{LP}} - 2C_{\text{LP}}$, where F_{LP} and C_{LP} are the facility and service costs for an arbitrary fractional solution* SOL *to the facility location LP.*

Proof: Consider the fractional solution SOL to the facility location LP. Let y_i denote the variable that indicates whether facility i is chosen in the solution and x_{ij} be the variable that indicates whether node j is serviced by facility i. $F_{\text{LP}} = \sum_i y_i f_i$ and $C_{\text{LP}} = \sum_{ij} x_{ij} c_{ij} d_j$. We will prove that $\sum_i y_i \cdot Gain(i) \geq F - F_{\text{LP}} - 2C_{\text{LP}}$. As before, assume without loss of generality that $y_i \leq 1$ for all i and $\sum_i x_{ij} = 1$ for all i, j.

As in the proof of Lemma 6.3.1, we modify the fractional solution by making multiple copies of facilities such that for every ij, either $x_{ij} = 0$ or $x_{ij} = y_i$.

Let $\mathcal{F}$ be the set of facilities in the current solution. Mimicking the proof of Lemma 2.4.2, we will "assign" (for deletion) each node $i' \in \mathcal{F}$ to its "nearest" node in the fractional solution. However, since we have a fractional solution, this is a *fractional assignment*, given by variables $m_{i'i} \geq 0$ where the value of $m_{i'i}$ indicates the extent to which i' is assigned to i. Earlier we used $m(i') = i$ when i was closest to i'. Here we will find the $i_1 \in \mathcal{F}_{\mathrm{LP}}$ which is closest to i' and set $m_{i'i_1} = y_{i_1}$, and find the next closest i_2 and so forth. We will again introduce copies of facilities to ensure $m_{i'i} = y_i$ or $m_{i'i} = 0$. Notice that this will still maintain $x_{ij} = y_i$ or $x_{ij} = 0$ which we achieved before.

Observe that the way we have assigned $m_{i'i}$,

$$\sum_i m_{i'i} c_{i'i} \leq c_{i'j} + \sum_i x_{ij} c_{ij}$$

For a demand node j, let $\sigma(j)$ be the facility assigned to j in the current solution.

With every facility i, we will associate a modified solution as follows.

Let $R(i)$ be the set of facilities $i' \in \mathcal{F}$ such that $m_{ii'} > 0$. Note that $x_{ij} = y_i$ for all $j \in D_{\mathrm{SOL}}(i)$ and $m_{ii'} = y_i$ for all $i' \in R(i)$. Consider the solution obtained by including facility i in the current solution and reassigning all nodes with $x_{ij} > 0$ to i. Further, for all facilities $i' \in R(i)$, the facility i' is removed from the solution and all nodes in $D(i') \setminus D_{\mathrm{SOL}}(i)$ are reassigned to i. Let $Gain'(i)$ be the decrease in cost of the solution as a result of this modification, i.e.

$$Gain'(i) = -f_i + \sum_{j, x_{ij} > 0} (c_{\sigma(j)j} - c_{ij}) + \sum_{i' \in R(i)} \left(f_{i'} + \sum_{j, \sigma(j) = i', x_{ij} = 0} (c_{\sigma(j)j} - c_{ij}) \right)$$

Clearly, $Gain(i) \geq Gain'(i)$. We are interested in $\sum_i y_i Gain(i)$ which is at least $\sum_i y_i Gain'(i)$, Now,

$$y_i \cdot Gain'(i) = -y_i f_i + \sum_{j, x_{ij} > 0} x_{ij} (c_{\sigma(j)j} - c_{ij}) + \sum_{i'} m_{ii'} \left(f_{i'} + \sum_{j, \sigma(j) = i', x_{ij} = 0} (c_{\sigma(j)j} - c_{ij}) \right)$$

Observe that we used the fact that $y_i = x_{ij}$ and $y_i = m_{ii'}$ in the relevant cases. Thus $\sum_i y_i Gain(i)$ is at least,

$$-\sum_i y_i f_i + \sum_i \sum_{j, x_{ij}>0} x_{ij} d_j (c_{\sigma(j)j} - c_{ij}) + \sum_i \sum_{i'} m_{ii'} \left(f_{i'} + \sum_{j, \sigma(j)=i', x_{ij}=0} (c_{\sigma(j)j} - c_{ij}) d_j \right)$$

The first term on the right hand side simplifies to $-F_{\mathrm{LP}}$. The second term is rewritten as

$$\sum_i \sum_{j, x_{ij}>0} x_{ij} d_j (c_{\sigma(j)j} - c_{ij}) = \sum_j d_j c_{\sigma(j)j} \sum_i x_{ij} - \sum_j \sum_i x_{ij} c_{ij} d_j = C - C_{\mathrm{LP}}$$

As in proof of Lemma 2.4.2, the last term can be broken into two parts, the first of which is

$$\sum_i \sum_{i'} m_{i'i} f_{i'} = \sum_{i'} f_{i'} \sum_i m_{i'i} = \sum_{i'} f_{i'} = F$$

The other part can be bounded below, since $c_{\sigma(j)j} - c_{ij} \geq -c_{\sigma(j)i}$, as

$$\sum_i \sum_{i'} m_{i'i} \sum_{j, \sigma(j)=i', x_{ij}=0} -c_{\sigma(j)i} d_j \geq \sum_j \sum_i -c_{\sigma(j)i} m_{\sigma(j)i} d_j$$

Notice that we dropped the restriction $x_{ij} = 0$, since we are adding more negative terms by dropping it. Now the above can be bounded as

$$\sum_j d_j \sum_i -c_{\sigma(j)i} m_{\sigma(j)i} \geq \sum_j d_j (-c_{\sigma(j)j} - \sum_i x_{ij} c_{ij}) = -C - C_{\mathrm{LP}}$$

Substituting these expressions, we get

$$\sum_i Gain(i) \geq \sum_i y_i \cdot Gain(i) \geq -F_{\mathrm{LP}} + C - C_{\mathrm{LP}} + F - C - C_{\mathrm{LP}}$$

which proves the lemma. ■

6.3.2 Local Search and Filtering

In this subsection, we will present a very short analysis of combining the filtering and local search approach. Assume that we have a solution of cost, $F(\zeta)$ and $C(\zeta)$, where ζ was the parameter in filtering step (See Section:3.2) where

$$F(\zeta) \leq \frac{1}{\zeta} F_{\mathrm{LP}} \quad C(\zeta) \leq \frac{3}{1-\zeta} C_{\mathrm{LP}} \quad \text{and} \quad \int_0^1 C(\zeta) d\zeta = 3 C_{\mathrm{LP}}$$

Consider the following algorithm, that we first run filtering for all values of ζ, and then apply the 'Add' heuristic in Section 2.2. We pick the node to add with the best ratio of improvement to cost.

Assume c is a constant, to be determined later. Set $\xi = F_{\text{LP}}/C_{\text{LP}}$. Assume that we do not have a c approximation. Thus from the filtering step, over the interval $[\frac{\xi}{c(1+\xi)}, 1]$,

$$C(\zeta) \geq c(F_{\text{LP}} + C_{\text{LP}}) - \frac{F_{\text{LP}}}{\zeta}$$

Now over the interval $[\frac{\xi}{(c-1)(1+\xi)}, 1]$ if $C(\zeta)$ is less than $F_{\text{LP}} + C_{\text{LP}}$, for any value of ζ,

$$F(\zeta) + C(\zeta) \leq \frac{(c-1)(C_{\text{LP}} + F_{\text{LP}})}{F_{\text{LP}}} F_{\text{LP}} + C_{\text{LP}} + F_{\text{LP}} \leq c(C_{\text{LP}} + F_{\text{LP}})$$

In this case we would have a c approximation algorithm by the filtering process itself, and subsequently the cost can only decrease.

Thus over the interval $[\frac{\xi}{(c-1)(1+\xi)}, 1]$ the value of $C(\zeta)$ is greater than $F_{\text{LP}} + C_{\text{LP}}$. By Lemma 6.3.1, and Corollary 6.3.1, there exists a facility with ratio $(C - C_{\text{LP}} - F_{\text{LP}})/F_{\text{LP}}$. Applying this to Lemma 2.2.2, with $C_{th} = F_{\text{LP}} + C_{\text{LP}}$ and $r = 1/F_{\text{LP}}$, we get the final cost of the solution to be at most,

$$F(\zeta) + F_{\text{LP}} \log \frac{C(\zeta) - C_{\text{LP}}}{F_{\text{LP}}} + F_{\text{LP}} + C_{\text{LP}}$$

Using the relationship on $F(\zeta)$, and supposing the above did not give us a c approximation to $F_{\text{LP}} + C_{\text{LP}}$, we get

$$C(\zeta) \geq C_{\text{LP}} + F_{\text{LP}} e^{(c-1)(1+1/\xi) - \frac{1}{\zeta}}$$

Thus from the above two bounds on $C(\zeta)$,

$$\begin{aligned} 3C_{\text{LP}} &\geq \int_0^1 C(\zeta) d\zeta \\ &\geq \int_{\frac{\xi}{c(1+\xi)}}^{\frac{\xi}{(c-1)(1+\xi)}} \left(c(F_{\text{LP}} + C_{\text{LP}}) - \frac{F_{\text{LP}}}{\zeta} \right) d\zeta \\ &\qquad + \int_{\frac{\xi}{(c-1)(1+\xi)}}^1 \left(C_{\text{LP}} + F_{\text{LP}} e^{(c-1)(1+1/\xi) - \frac{1}{\zeta}} \right) d\zeta \end{aligned}$$

the above reduces to

$$3 \geq 1 + \frac{\xi^2}{(c-1)(1+\xi)} - \xi \log \frac{c}{c-1} + \xi e^{(c-1)(1+1/\xi)} \int_{\frac{1}{(c-1)(1+1/\xi)}}^{1} e^{-\frac{1}{\zeta}} d\zeta$$

For $c = 2.47$, the above inequality is violated for all ξ, which shows that the algorithm has an approximation factor of 2.47.

Lemma 6.3.4 *If Lemma 6.3.1 and Lemma 3.2.3 are true then the filtered rounding followed by greedy augmentation has an approximation factor of* 2.47*.*

Although this result will be superseded by several other algorithms in case of the facility location problem, this style of analysis gives the best result for the fault tolerant facility location problem, in Chapter 8.

6.3.3 Primal Dual and Local Improvement

In this section we will show that the ideas of a primal-dual algorithm can be combined with augmentation and scaling to give a better result than obtained by the pure greedy strategy, and the primal dual algorithm itself.

We will not be concerned with the details of the primal dual algorithm here, since we would be using the algorithm as a 'black box'. See Chapter 4 for the primal dual algorithm.

By Lemma 4.2.1, the primal-dual algorithm returns a solution with facility cost F and a service cost S such that, $3F + S \leq 3OPT$ where OPT denotes the cost of the optimal dual solution. The algorithm runs in time $O(n^2 \log n)$.

In itself the primal-dual algorithm does not yield a better result than factor 3. In fact a simple example shows that the dual constructed which is used as a lower bound can be factor 3 away from the true optimum. Once again let SOLbe an arbitrary solution with facility cost $F_{\text{SOL}}$and assignment cost C_{SOL}.

Consider a simpler algorithm before presenting the best known combinatorial algorithm. If in the primal-dual algorithm we were to scale facility costs by a factor of $\lambda = 1/3$ and use primal-dual algorithm on this modified instance we would have a feasible primal solution of cost $F_{\text{SOL}}/3 + C_{\text{SOL}}$. The primal-dual algorithm giving a solution with modified facility cost F and service cost C will guarantee that $3F + S$ is at most 3 times the feasible dual constructed. The dual solution is less than the feasible primal solution of $F_{\text{SOL}}/3 + C_{\text{SOL}}$. After scaling back the solution, the cost of the final solution will be $3F + S$ due to choice of λ, which is at most $F_{\text{SOL}} + 3C_{\text{SOL}}$.

Along with the above run the local search algorithm setting $\lambda = 2$ for which the cost of the final is $3F_{\text{SOL}} + 2C_{\text{SOL}}$ from the proof of theorem 2.5.1. The smaller of the two solutions can be at most $2\frac{1}{3}$ times $F_{\text{SOL}} + C_{\text{SOL}}$.

Corollary 6.3.2 *Using augmentation and scaling along with the primal-dual algorithm the facility location problem can be approximated within factor* $2\frac{1}{3} + \epsilon$ *in time* $O(n^2/\epsilon + n^2 \log n)$.

Suppose we scale by a factor λ, to be specified later, then there exists a feasible fractional solution of facility cost λF_{LP} and fractional assignment cost $C_{\text{LP}}$to the modified problem. By duality theory, this fractional solution is more than any dual solution constructed, and thus if we get facility cost F' and assignment cost C' for the modified problem, the guarantee due to the primal dual algorithm is

$$3F' + C' \leq 3(\lambda F_{\text{LP}} + C_{\text{LP}})$$

. Our actual facility cost is $F = F'/\lambda$ and actual assignment cost $C = C'$.

If at this point we have $C \leq F_{\text{LP}} + C_{\text{LP}}$, then

$$F + C \leq \frac{3\lambda F + C}{3\lambda} + (1 - \frac{1}{3\lambda})C \leq (2 - \frac{1}{3\lambda})F_{\text{LP}} + (1 + \frac{2}{3\lambda})C_{\text{LP}}$$

Recall Lemma 6.3.2. Consider the case that $C > F_{\text{LP}} + C_{\text{LP}}$, we also have $C \leq 3\lambda F_{\text{LP}} - 3\lambda F + 3C_{\text{LP}}$. Since there is a fractional solution of assignment cost C_{LP} and facility cost F_{LP}, by Lemma 6.3.2 the cost is at most,

$$F + F_{\text{LP}} \ \ln\left(\frac{3\lambda F_{\text{LP}} - 3\lambda F + 2C_{\text{LP}}}{F_{\text{LP}}}\right) + F_{\text{LP}} + C_{\text{LP}}$$

Over the allowed interval of F the above expression is maximized at at $F = 2C_{\text{LP}}/(3\lambda)$, with cost

$$(1 + \ln(3\lambda))F_{\text{LP}} + (1 + \frac{2}{3\lambda})C_{\text{LP}}$$

Now for $\lambda > 1/3$ the term $1 + \ln(3\lambda)$ is larger than $2 - 1/(3\lambda)$. Thus the case $C > F_{\mathrm{LP}} + C_{\mathrm{LP}}$ dominates.

Consider the value of λ such that $\ln(3\lambda) = 2/(3\lambda)$, which is $\lambda \approx 0.7192$ and the approximation factor is ≈ 1.8526. The greedy augmentation takes maximum $O(n^3)$ time and the primal-dual algorithm takes $O(n^2)$ time. Thus, we have the following theorem:

Theorem 6.3.3 *The facility location problem can be approximated within factor* ≈ 1.8526 *in time* $O(n^3)$. *In particular the solution can be bounded by* $(1+\ln(3\lambda))F_{\mathrm{LP}} + (1+\frac{2}{3\lambda})C_{\mathrm{LP}}$, *where* F_{LP} *and* C_{LP} *are facility and assignment costs of any feasible, possibly fractional solution.*

6.3.4 The Best Known Constant

It is interesting however that this result which is obtained from the above greedy algorithm can be combined with the algorithm of [Chu98, CS98] to obtain a very marginal improvement in the approximation ratio for the facility location problem.

Theorem 6.3.4 *[Chu98, CS98] If* ρ *denotes the fraction of the facility cost in the optimal solution returned by the Linear Programming formulation of the facility location problem, then the fractional solution can be rounded within a factor of* $1 + \rho \ln \frac{2}{\rho}$ *when* $\rho \leq 2/e$.

The above primal-dual plus greedy algorithm when run with $\lambda = 2/(3(e-1))$ gives an algorithm with approximation ratio $e - \rho(e - 1 - \log\frac{2}{e-1})$. This uses Theorem 6.3.3. It is easy to verify that the smaller of the two solutions has an approximation of 1.728. This is a very marginal (≈ 0.008) improvement over the best known algorithm. However it demonstrates that the facility location problem can be approximated better.

Theorem 6.3.5 *Combining the linear programming approach and the scaled, primal-dual plus greedy algorithms, the facility location problem can be approximated to a a factor* 1.728.

6.4 Half-Integral Analysis: Capacitated k-Median

We will consider the *splittable* version of the problem, in which a node may split its demand fractionally over several centers. We will then use our results for the *splittable* problem to devise algorithms for the *unsplittable* case, where the demand of a node has to be routed from one center only.

6.4.1 Outline of the algorithm

We will use the linear program (3.1)–(3.4) and (3.7)–(3.8) for the k-median problem We add the following constraint to ensure that the total demand assigned to each center is at most M.

$$\sum_j d_j x_{ij} \leq M y_i \qquad \text{for each } i, j \in N. \tag{6.1}$$

Let $(\bar{x}, \bar{y})$ denote a feasible solution to the LP relaxation and let $\bar{C}_{LP}$ denote its objective function value. As before, for each location $j \in N$, let $\bar{C}_j$ denote the cost incurred by this fractional solution for assigning (one client at) location j, i.e.,

$$\bar{C}_j = \sum_{i \in N} c_{ij} \bar{x}_{ij} \qquad \text{for each } j \in N. \tag{6.2}$$

Note that $\bar{C}_{LP} = \sum_{j \in N} d_j \bar{C}_j$. We will round the fractional solution $(\bar{x}, \bar{y})$ to a solution to the integer program of cost at most $16\bar{C}_{LP}$ with capacity at most $3M$. The outline of our approximation algorithm is as follows.

Step 1: We first identify a set of locations N' as in Step 1 of the algorithm for the k-median problem. However, demands are not modified as before.

Step 2: Next, we simplify the structure of the solution by consolidating nearby fractional centers. Every fractional center is assigned to the closest location in N'. In the uncapacitated problem, centers were built only at locations in N'. We cannot do this in the capacitated case, as this would not satisfy the capacity constraints. Instead, for every location i we consider the set of fractional centers assigned to i and redistribute the total fractional center value to the location i and the locations closest to it so that every positive fractional center has value at least a $\frac{1}{2}$. The solution created in this manner could violate the capacity constraints. In this step, we also modify the demands of the locations. Let d' denote the new set of demands. The new demand d'_j of a location j is positive only if the fractional center value $y'_j > 0$.

Step 3: We modify the solution further to produce a $\{\frac{1}{2}, 1\}$-integral solution to the linear programming relaxation. Again, this solution could violate the capacity constraints.

Step 4: Finally, we convert the $\{\frac{1}{2}, 1\}$-integral solution to an integral solution with a factor 3 blowup of capacities.

6.4.2 Step 1: Consolidating Demands

In this step, we identify a set of nodes N' such that

- For all $j \in N - N'$, there exists $i \in N'$ such that $c_{ij} \leq 4\bar{C}_j$.
- For all $i, j \in N'$, $c_{ij} > 4\max(\bar{C}_i, \bar{C}_j)$.

The algorithm to construct the set N' is similar to Step 1 of the algorithm for the k-median problem. In this case, demands are not modified as before.

Assume for notational simplicity that $N = \{1, \ldots, n\}$, and the locations are indexed in increasing order of $\bar{C}_j$, i.e., $\bar{C}_1 \leq \ldots \leq \bar{C}_n$. We begin with $N' = \emptyset$.

- Consider the locations $j = 1, \ldots, n$ in this order. When considering location j, check if there is another location $i < j$, $i \in N'$ such that and $c_{ij} \leq 4\bar{C}_j$. If there is no such location, add j to N', i.e. set $N' \leftarrow N' \cup \{j\}$.

It is immediate from the algorithm that the set N' constructed satisfies the two properties stated above.

6.4.3 Step 2: Consolidating Medians

For location $j \in N$, let $l(j)$ denote the closest location to j in N' (breaking ties arbitrarily). For $i \in N'$, define $M_i = \{j : l(j) = i\}$. Let $z_i = \sum_{j \in M_i} y_j$ denote the total fractional center value in M_i. Rephrasing property 1 of the last Step;

Lemma 6.4.1 $c_{l(j),j} \leq 4\hat{C}_j$.

Recall that the set N' is chosen so that for any locations $i, j \in N'$, $c_{ij} > 4\bar{C}_i$. Hence for each location $j \in N'$, all locations within a $2\bar{C}_i$ radius of i must be closer to i than to any other location in N'. All such locations must belong to M_i. By Lemma 3.5.1, the total fractional center value within a radius $2\bar{C}_i$ of i is at least 1/2. Hence z_i, the total fractional center value within the set M_i is at least 1/2.

For each location $i \in N'$, we redistribute the total fractional center value in the set M_i as follows. Initially set $y'_j = \bar{y}_j$ for all $j \in M_i$ and $x'_{jj'} = \bar{x}_{jj'}$ for all $j \in M_i, j' \in N$.

We move the fractional centers from locations in M_i far away from i to locations closer to i. The following operation transfers an amount p of the fractional center from location j_s to location j_t, $s < t$. When we move the fractional center, we also change the assignments accordingly.

For all $j' \in N$, set

$$\begin{aligned} x'_{j_t j'} &\leftarrow x'_{j_t j'} + \frac{p}{y'_{j_s}} x'_{j_s j'} \\ x'_{j_s j'} &\leftarrow x'_{j_s j'} - \frac{p}{y'_{j_s}} x'_{j_s j'} \end{aligned}$$

Set $y'_{j_t} \leftarrow y'_{j_t} + p$, $y'_{j_s} \leftarrow y'_{j_s} - p$.

We redistribute the fractional centers in M_i by performing a series of such transfer operations.

1. Let j_t be the first location in the sequence $j_1, \ldots, j_m$ such that $y'_{j_t} < 1$. Let j_s be the first location in the sequence $j_{t+1}, \ldots, j_m$ such that $y'_{j_s} > 0$. If either j_s or j_t does not exist, terminate.

2. $p \leftarrow \min(y'_{j_s}, 1 - y'_{j_t})$.

3. Transfer an amount p of the fractional center from j_s to j_t.

4. Go back to step 1.

If at the end of this process, there is a location j_t, $1 < t \leq m$ such that $0 < y'_{j_t} < 1$ (there can be at most one such location), then we assign the demand served by j_t to

j_{t-1}. For all locations $j' \in N$, we set

$$\begin{aligned} x'_{j_{t-1}j'} &\leftarrow x'_{j_{t-1}j'} + x'_{j_t j'} \\ x'_{j_t j'} &\leftarrow 0 \end{aligned}$$

Additionally, we set $y'_{j_t} \leftarrow 0$. Note that as a result of this operation, the load on j_{t-1} can be as much as $2M$.

We will be interested in tracing particular assignments $\bar{x}_{i'j'} > 0$. Notice in step 2, due to the fraction p, an assignment $x'_{j_s j'}$ (which is $\bar{x}_{j_s j'}$, since s is yet to be considered) can be split into several different assignments. We will view these as *different* assignments, to different copies of the node j_s. This appears convoluted, but will allow us to be rid of the term p in the analysis.

Definition 1 *Define an assignment $\bar{x}_{i'j} > 0$ to be terminally modified in this Step iff $z_{l(i')} \geq 1$. If the assignment is modified to i_1; under the simplification using copies of a node,*

$$\begin{aligned} x'_{i_1 j} &\leftarrow x'_{i_1 j} + \bar{x}_{i'j} \\ \bar{x}_{i'j} &\leftarrow 0 \end{aligned}$$

Otherwise define the assignment to be non-terminally modified.

Observe, that in the definition, the terminal modification would also imply $y'_{i_1} = 1$, and $i_1 \in M_{l(i')}$. We will be interested in the cost associated with these modifications, and will use the two following lemmas,

Lemma 6.4.2 *For an assignment $\bar{x}_{ij} > 0$ terminally modified to $i' \in M_{l(i)}$ in step 2, we have $c_{i'j} \leq 3c_{ij} + 8\bar{C}_j$.*

Proof: By construction, i' is before i in the ordering of $M_{l(i)}$, and thus $c_{i'l(i)} \leq c_{l(i)i}$. Now by triangle inequality,

$$\begin{aligned} c_{i'j} &\leq c_{i'l(i)} + c_{l(i)i} + c_{ij} \\ &\leq 2c_{l(i)i} + c_{ij} \\ &\leq 2c_{l(j)i} + c_{ij} \qquad l(i) \text{ is closer than } l(j) \\ &\leq 2(c_{l(j)j} + c_{ij}) + c_{ij} \end{aligned}$$

The lemma follows since $c_{l(j)j} \leq 4\bar{C}_j$ from step 1, Lemma 6.4.1. ∎

Lemma 6.4.3 *For a non-terminally modified assignment $\bar{x}_{ij} > 0$, it must be modified to $l(i)$ and $z_{l(i)} < 1$. Moreover $c_{l(i)j} \leq 2c_{ij} + 4\bar{C}_j$.*

Proof: The first part follows by construction, and $c_{l(i)j} \leq c_{ij} + c_{l(i)i}$. The latter is at most $c_{l(j)i} \leq c_{l(j)j} + c_{ij}$. ∎

The following properties of the transformation will be used later:

Lemma 6.4.4 *1. $y'_i > 0 \Rightarrow y'_i \geq \frac{1}{2}$.*

2. $y'_i < 1 \Rightarrow \sum_{j \in N} d_j x'_{ij} \leq M y'_i$

3. $y'_i = 1 \Rightarrow \sum_{j \in N} d_j x'_{ij} \leq 2M y'_i$

We define new demands d' as follows. For all locations $i \in N'$, set

$$d'_i \leftarrow \sum_{j \in N} d_j x'_{ij}.$$

For all locations $i \in N - N'$, set $d'_i \leftarrow 0$. At this point we would have some locations i with $y'_i = 1$. We will not change this in the subsequent steps (hence terminal modification for assignments that get assigned to i). However, we would have to round the solution associated with the non-terminal assignments.

6.4.4 Step 3: Obtaining a $\{\frac{1}{2}, 1\}$-integral solution

Let $N_1 = \{i \in N : y'_i > 0\}$ be the set of all locations with non-zero fractional centers. Let $N_2 = \{i \in N : 0 < y'_i < 1\}$ be the set of all locations with non-zero fractional centers strictly less than 1. By the transformation in the previous step, N_2 is a subset of N'. Let $|N_1| = \ell$ and $|N_2| = m$.

For all locations $j \in N_1 - N_2$, set $\hat{y}_j \leftarrow 1$. The fractional center values for locations in N_2 are determined as follows. For $j \in N_2$, let $s(j)$ be the nearest location in $N_1 - \{j\}$, breaking ties arbitrarily. Order the vertices in N_2 in increasing order of

$d'_j c_{s(j)j}$, say $j_1, \ldots, j_m$. For all $r > 2(\ell - k)$, set $\hat{y}_{j_r} \leftarrow 1$. For all $r \leq 2(\ell - k)$, set $\hat{y}_{j_r} \leftarrow \frac{1}{2}$.

The fractional assignments x'_{ij} obtained in the previous step are kept unchanged. Note that the solution $(\hat{y}, x')$ at this stage could be infeasible as the fractional center at a location could be reduced to $\frac{1}{2}$ even while another location may be assigned to it to the extent of more than $\frac{1}{2}$. Also, for every fractional center of value $\frac{1}{2}$, the total demand served is at most M. This is expressed in the following lemma:

Lemma 6.4.5 *For any location i,*

$$\hat{y}_i = \frac{1}{2} \quad \Rightarrow \quad \sum_{j \in N} d_j x'_{ij} \leq M$$
$$\hat{y}_i = 1 \quad \Rightarrow \quad \sum_{j \in N} d_j x'_{ij} \leq 2M$$

However we now have a half integral solution and along the lines of Lemma 3.5.2; and this half-integral solution minimizes the $\sum_{i \in N_2}(1 - y_i)d'_i c_{s(i)i}$ under the restriction $y_i \geq \frac{1}{2}$. In particular we are interested in the following:

Lemma 6.4.6 $\sum_{i \in N_2}(1 - \hat{y}_i)d'_i c_{s(i)i} \leq \sum_{i \in N_2}(1 - y'_i)d'_i c_{s(i)i}$.

6.4.5 Step 4: Rounding to an integral solution

We build a collection of trees as in the uncapacitated K median problem, in Section 3.6. For each location $i \in N_2$ with $\hat{y}_i = \frac{1}{2}$, draw a directed edge from i to $s(i)$. Recall that $s(i)$ is the closest location to i in $N_1 - \{i\}$. Hence, each component of the graph obtained contains at most one cycle and this correspond to the closest pair of nodes in it. For each such cycle, we choose one of the two vertices as a root and delete the directed edge from the root to the other vertex. This gives a collection of rooted trees spanning the nodes $i \in N_2$ with $\hat{y}_i = \frac{1}{2}$. As before, $s(i)$ is a parent of i if we have a directed edge from i to $s(i)$.

We will build centers at every node i with $\hat{y}_i = 1$. Additionally, we will build centers at exactly half the nodes i with $\hat{y}_i = \frac{1}{2}$. If a center is built at a node j then all the demand served by j (according to the assignment x') is served by this center.

For any node j with $\hat{y}_j = \frac{1}{2}$, if a center is not built at j, we will assign j to some other node i where a center is built. In this case, all the demand that was originally served by j (according to the assignment x') will now be served by the center at i. We will first decompose each tree in the collection into a set of stars and then obtain integral solutions on these stars.

Consider any tree T in the collection. Recall that the level of a node is the number of edges on the path from the node to the root. Start from the leaf node i in the highest numbered level and remove the subtree rooted at its parent $s(i)$. Observe that this must be a star rooted at $s(i)$. Repeat this procedure till we are left with an empty tree or a singleton root i If $\hat{y}_i = \frac{1}{2}$, consider $s(i)$; this must be a child of i in the tree. In a previous step, we must have removed a star rooted at $s(i)$. We now add i to this star as a child of $s(i)$.

We repeat this procedure on every tree in the collection obtaining a collection of stars spanning the nodes $i \in N_2$ with $\hat{y}_i = \frac{1}{2}$. Note that in every star in the collection, all nodes except perhaps the root have a fractional center of $\frac{1}{2}$. (The root could have a fractional center of $\frac{1}{2}$ or 1).

Definition 2 *An even star is a star such that the total fractional center value summed over nodes in the star is an integer. A star that is not an even star is an odd star.*

Thus, a star rooted at i is an even star if either $\hat{y}_i = 1$ and i has an even number of children, or $\hat{y}_i = \frac{1}{2}$ and i has an odd number of children. On the other hand, a star rooted at i is an odd star if either $\hat{y}_i = 1$ and i has an odd number of children, or $\hat{y}_i = \frac{1}{2}$ and i has an even number of children. Therefore there are four cases to analyze.

We define two kinds of configurations for each star.

1. For an even star rooted at i with $\hat{y}_i = 1$ and children $j_1, \ldots, j_{2r}$ in increasing order of distance from i.

   (a) Build centers at nodes $j_1, j_3, \ldots, j_{2r-1}$. Node j_{2s} is assigned to node j_{2s-1}. The assignment of j_{2s} implies that for all locations $j' \in N$,

$$x'_{j_{2s-1}j'} \leftarrow x'_{j_{2s-1}j'} + x'_{j_{2s}j'}$$

$$x'_{j_{2s}j'} \leftarrow 0$$

For the rest of the discussion we will not present the formulae as above, the term 'assignment' will imply a similar set of equations.

(b) Build centers at nodes $j_2, j_4, \ldots, j_{2r}$. Node j_1 is assigned to the root i and node j_{2s+1} is assigned to j_{2s}.

2. For an even star rooted at i with $\hat{y}_i = \frac{1}{2}$ and children $j_1, \ldots, j_{2r+1}$ in increasing order of distance from i.

   (a) Build centers at the root i and at nodes $j_2, j_4, \ldots, j_{2r}$. Node j_1 is assigned to the root i and node j_{2s+1} is assigned to node j_{2s}.

   (b) Build centers at the root i and at nodes $j_3, j_5, \ldots, j_{2r+1}$. Nodes j_1 and j_2 is assigned to the root i and node j_{2s+2} is assigned to j_{2s+1}.

3. For an odd star rooted at i with $\hat{y}_i = 1$ and children $j_1, \ldots, j_{2r+1}$ in increasing order of distance from the root i.

   (a) Build centers at $j_2, j_4, \ldots, j_{2r}, j_{2r+1}$. Node j_1 is assigned to the root i. Node j_{2r+1} is assigned to itself. All other nodes j_{2s+1} for $s \neq r$ is assigned to j_{2s}. Note this implies $r > 0$, otherwise only j_1 is assigned to the root.

   (b) Build centers at $j_1, j_3, \ldots, j_{2r-1}$. The node j_{2r+1} is assigned to the root i and node j_{2s} is assigned to j_{2s-1}. Again for $r = 0$, we only build j_1 and assign it to itself.

4. For an odd star rooted at i with $\hat{y}_i = \frac{1}{2}$ and children $j_2, j_4, \ldots, j_{2r}, j_{2r}$ in increasing order of distance from the root i.

   (a) Build centers at the root i and at $j_2, j_4, \ldots, j_{2r}$. Nodes j_1 is assigned to the root. Node j_{2s+1} is assigned to the node j_{2s}.

   (b) Build centers at the root i and at $j_3, j_5, \ldots, j_{2r-1}$. The nodes j_1 and j_2 are assigned to the root i and node j_{2s} is assigned to node j_{2s-1}.

At this point we have two configurations each for every star. Let a node i be assigned to $r(i)$ in a configuration for a star S. Define the cost of this configuration of S as $\sum_{i \in S} d'_i c_{r(i)i}$. For the even stars we will choose the configuration of the lower cost. For the odd stars, we will proceed differently. Since there will be even number of such stars, we will denote them by $S_1, S_2, \ldots, S_{2q}$. Let $C_a(\ell)$ denote the cost of star S_ℓ under configuration (a) and likewise $C_b(\ell)$ for configuration (b)., we will assume that the stars are numbered in increasing order of $|C_a(\ell) - C_b(\ell)|$. Our rounding scheme will choose the lower cost configuration for $S_{q+1}, \ldots, S_{2q}$. For the stars $S_1, \ldots, S_q$ we will choose the configuration that yields the smaller number of centers. (this is the configuration (b) by our construction.)

Lemma 6.4.7 *For each star compute the average cost under the two configurations (a) and (b). The total cost is bounded by the sum of average cost over all stars.*

Proof: Since in the even stars we choose the configuration of the smaller cost, we can bound the cost paid by the average over the two configurations.

Thus we need to prove the lemma for the odd stars. Pair up the stars S_ℓ and $S_{2q-\ell+1}$ for $\ell \leq q$. At the worst case we paid the higher of the two costs for S_ℓ. We will use the inequalities,

$$\max(f,g) = \frac{1}{2}|f-g| + \frac{1}{2}|f+g| \qquad \max(f,g) = \frac{1}{2}|f+g| - \frac{1}{2}|f-g|$$

Thus the cost we paid over the pair S_ℓ and $S_{2q-\ell}$ can be bounded by

$$\begin{aligned} \min(C_a(2q-\ell), C_b(2q-\ell)) + \max(C_a(\ell, C_b(\ell)) &\leq \frac{1}{2}|C_a(2q-\ell) + C_b(2q-\ell)| \\ + \frac{1}{2}|C_a(\ell) + C_b(\ell)| + \frac{1}{2}|C_a(\ell) - C_b(\ell)| &- \frac{1}{2}|C_a(2q-\ell) - C_b(2q-\ell)| \end{aligned}$$

Consider the last two terms in the above expression. The way we defined the ordering of S_ℓ, we have

$$|C_a(\ell) - C_b(\ell)| \leq |C_a(2q-\ell) - C_b(2q-\ell)|$$

Which proves the lemma. ■

Lemma 6.4.8 *The cost of the stars (with respective configurations) chosen by the above algorithm can be bounded by* $\sum_{j \in N_2, \hat{y}_j = \frac{1}{2}} d'_j c_{s(j)j}$.

Proof: Consider a node j which was chosen as a root of a star. The cost for this node is 0. So we are left with the case that j is a child node in the star it belongs to.

All the nodes fall into three categories: it has a center in one configuration and is assigned to its sibling, at most $2c_{s(j)j}$ away, in the other; it has a center in one configuration and assigned to the root in the other; it is assigned to the root always. In all the three cases the average cost paid by j is at most $d'_j c_{s(j)j}$, which proves the lemma. ■

Lemma 6.4.9 *The cost of the the solution of the non-terminal assignments increases by the cost of the stars, which is* $2 \sum_{i \in N_2} (1 - y'_i) c_{s(i)i} d'_i$.

Proof: Consider all the non-terminal assignments to i, which gets reassigned to $r(i)$. The new reassignment distance for a location j is at most $c_{r(i)i} + cij$ by the triangle inequality. Summing over all j,

$$\sum_j d_j x'_{ij} (c_{r(i)i} + cij) = c_{r(i)i} \sum_j d_j x'_{ij} + \sum_j d_j x'_{ij} cij$$

The first term on the right is $c_{r(i)i} d'_i$ which when summed over all $i \in N_2$ is the cost of the stars. The last term on the right hand side of the equation is the cost associated with the non-terminally modified assignments before this step.

The cost of the stars is:

$$\sum_{j \in N_2, \hat{y}_j = \frac{1}{2}} d'_j c_{s(j)j} = \sum_{j \in N_2} 2(1 - \hat{y}_j) d'_j c_{s(j)j}$$

Using Lemma 6.4.6 with the above equation, the lemma follows. ■

6.4.6 Analyzing the cost

The cost of terminal assignments can be bound by

$$\sum_j \sum_{i:z_{l(i)}\geq 1} (3c_{ij} + 8\bar{C}_j)\bar{x}_{ij}d_j \tag{6.3}$$

The cost of the non-terminal assignments after Step 2 can be bound. We would have to add the cost of the stars, to get the final cost associated with non-terminal assignments.

The cost of non-terminal assignments after Step 2 is:

$$\sum_j \sum_{i:z_{l(i)}<1} (2c_{ij} + 4\bar{C}_j)\bar{x}_{ij}d_j \tag{6.4}$$

To account for the cost of the stars, each location i in N_2 is charged $2(1-y'_i)d'_j c_{s(i)i}$, and this cost is distributed over the demand locations. Notice that $i \in N_2$ implies $z_i < 1$.

$$\sum_{i:z_i<1} 2(1-y'_i)d'_i c_{s(i)i} \leq \sum_j \sum_{i:z_i<1} 2(1-y'_i)d_j x'_{ij} c_{s(i)i}$$

Now the cost accruing to demand location j is

$$\sum_{i:z_i<1} 2(1-y'_i)d_j x'_{ij} c_{s(i)i}$$

Consider the set $M_i = \{i' | l(i') = i\}$. Notice that $x'_{ij} = \sum_{i':l(i')=i} \bar{x}_{i'j}$.

We have two cases : first, if $i \neq l(j)$. In this case, $s(i)$ is at most as far away as $l(j)$ from i, also i is closer to i' than $l(j)$. Now

$$c_{s(i)i} \leq c_{l(j)i} \leq c_{ii'} + c_{i'l(j)} \leq 2c_{i'l(j)}$$

We use the facts $c_{i'l(j)} \leq c_{i'j} + c_{jl(j)}$, and Lemma 6.4.1. The cost can be bounded as

$$\sum_{i:z_i<1, i\neq l(j)} \left[\sum_{i',l(i')=i} 2(1-y'_i)\bar{x}_{i'j} 2(c_{i'j} + 4\bar{C}_j)d_j \right]$$

Simplifying the double summation, and using $2(1-y'_i) \leq 1$, the above reduces to

$$\sum_{i':z_{l(i')}<1, l(i')\neq l(j)} x_{i'j}(2c_{i'j} + 8\bar{C}_j)d_j \tag{6.5}$$

The case $i = l(j)$ is the more complicated case, we would need to estimate $c_{s(i)i}$. It is sufficient to estimate $(1 - y'_i)c_{s(i)i}$.

Now $y'_i < 1$ implies there must be a location $i' \notin M_i$ and $\bar{x}_{i'i} > 0$; since the location i is assigned at least (fractionally) to one center. The closest such $i' \notin M_i$ can be at most $\bar{C}_i/(1 - y'_i)$ away from j, otherwise $\sum'_i \bar{x}_{i'i} C_{i'i} > \bar{C}_i$.

Since $i' \notin M_i$, it must be $l(i') \neq i$. Now $l(i')$ will be closer to i' than i, we can conclude $c_{s(i)i} \leq c_{ii'} + c_{l(i')i'} \leq 2c_{ii'}$. If $l(j) = i$, we have $\bar{C}_i \leq \bar{C}_j$, since i was ordered before j in Step 1. Thus,

$$(1 - y'_i)c_{s(i)i} \leq 2\bar{C}_i \leq 2\bar{C}_j$$

Therefore;

$$\sum_{i:z(i)<1, i=l(j)} \sum_{i':l(i')=i} d_j 2(1 - y'_i)c_{s(i)i}\bar{x}_{i'j} \leq \sum_{i':z(l(i'))<1, l(i')=l(j)} 4\bar{C}_j d_j \bar{x}_{i'j} \tag{6.6}$$

From the two equations 6.5 and 6.6, the charge to location j from the cost of the stars can be bounded by,

$$\sum_{i':z(l(i))<1} (2c_{i'j} + 8\bar{C}_j)d_j\bar{x}_{i'j}$$

Summing the above Equation for all j and using Equation 6.4, the total cost from non-terminally modified paths is

$$\sum_j \sum_{i':z(l(i'))<1} (4c_{i'j} + 12\bar{C}_j)d_j\bar{x}_{i'j}$$

Combining with Equation 6.3, the total cost is

$$\sum_j \sum_{i'} (4c_{i'j} + 12\bar{C}_j)d_j\bar{x}_{i'j} \leq \sum_j (4\bar{C}_j + 12C_j)$$

Thus we can prove the following theorem,

Theorem 6.4.1 *The capacitated k-median problem with splittable assignments can be approximated to factor 16 increasing the capacity by factor 3.*

We can derive the following using [ST93],

Theorem 6.4.2 *The capacitated k-median problem with unsplittable assignments can be approximated to factor 16 increasing the capacity by factor 4.*

Chapter 7

Toolbox – II: Interchanging Costs

7.1 Introduction

In this chapter we will investigate capacitated facility location problems. Interestingly the technique of solving these problems will be reducing them to the uncapacitated facility location variants. Consider the following problem: that the facilities we are placing can serve an upper bound of demand, but we can place as many facilities at a given location (of course we pay for each facility placed). It is also conceivable that the cost of placing multiple facilities at a location is cheaper than adding their cost up. This typifies the bulk or volume discounts associated with economies of scale. We would refer to this problem as the buy at bulk facility location problem. Later we would consider problems for which there is a lower bound on the demand a facility i may serve. This is the Lower Bounded Facility Location (LBFL) problem. We will review the definitions for both these problems later.

It is an interesting fact that the methods to approach these problems would be through reductions to the uncapacitated facility location problems. In the buy at bulk facility location problem, we would add the facility costs to the metric, to achieve a newer metric instance. The argument would be that the slight excess cost paid in the second metric instance is sufficient to pay for copies of the facility required to handle the demand at a location. In the case of lower bounded facility location we will take the reverse path, by adding the cost of serving a minimum demand to the facility cost. The central idea being that if a facility is open, the cost to route the minimum demand to this location has been paid for already, what remains is the reassignment of demands. These ideas would become clearer as we proceed, but the interesting aspect is the handling of capacity constraints by interchanging facility and metric costs in the reductions.

The capacitated facility location can have several variants. The most common variant (CFLP) is a facility location problem with the added constraint that every facility has an upper bound M on the capacity. For this problem we can achieve a constant factor approximation using filtering (See [STA97]), but with a relaxation of the upper bound of the capacity by a constant factor. One such variant studied in [CS99], allowed multiple copies of facilities to be placed at the same location (all with the same capacity M). This allowed the capacity to increase twofold, of course, with a twofold increase in cost. We will be concerned with this version, also the demands will be unsplittable: assigned completely to one facility.

However in our discussion, we will have capacities that would depend on the location, μ_i at location i, for which [JV99] proposed a constant factor primal dual approximation algorithm. We will be more interested in showing reductions to the uncapacitated facility location problem. In the first section, we will consider such a reduction for the buy at bulk facility location problem.

In the buy at bulk models we will consider, the price of placing t facilities will be strictly be less than $t+1$ facilities at the same place. Consider a distribution network, when facilities can be placed only if there exists a certain *minimum* demand. This is an example of Lower Bounded Facility Location problem. In a further step we can assume a situation in which the cost of placing $t+1$ copies is cheaper than placing t copies, provided the minimum demand for the $t+1$ copies is satisfied. Assume the situation of negotiating a smaller fee to host a conference, if a minimum number of persons are guaranteed to stay in the hotel. This problem will also become very interesting in context of single sink edge installation problem, (See Chapter 10). In the second section we will study this lower bounded facility location problem. Each facility will have a different lower bound L_i, and cost f_i.

We will present a reduction of the above problem to the uncapacitated facility location problem. However we will go one step further and show that we can achieve a constant factor integrality gap with respect to any feasible fractional solution to the integer linear programming formulation. This would be very important in context of edge installation problems, where we would use this approximation under the guarantee that a fractional feasible solution exists, but we would not have an explicit fractional solution. Unlike facility location [STA97, CG99, JV99], the lower bound makes it hard to round the linear relaxation directly. This arises from the fact that the filtering steps of Lin and Vitter in [LV92b] do not work. Thus fractional solutions cannot be rounded by previous approaches.

We will extend the results of the first two sections in considering a facility location problem where the facility cost is a (slowly varying) function of the demand. This will act as a primer to consider some application of the facility location problem in clustering, in the next section. As discussed before, the facility location problem also models clusterings, with a tradeoff in the number of clusters versus the average radius. In the variants we will consider here, the *Minsum Clustering* partitions the dataset into k parts, and the objective function to minimize is the sum of distances of pairs of points in the same cluster. Intuitively this results in defining tighter clusters than by the k-median problem, in which the objective is to minimize the sum of the distances to the cluster center.

We can define the class of clustering based on pairwise distances. In *generalized k-clustering*, we are given n points in a metric space with distance function c. The goal is to partition the points into k clusters, $\chi_1, \chi_2, \ldots, \chi_k$ so that the following objective function is minimized:

$$\sum_{l=1}^{k} \frac{1}{|\chi_l|^\gamma} \sum_{i,j \in \chi_l} c_{ij}$$

It is easy to see that an equivalent formulation to a factor of 2 is the following. We wish to partition the points into k clusters $\chi_1, \chi_2, \ldots, \chi_k$ with centers $q_1, q_2, \ldots, q_k$ so that the following objective function is minimized:

$$\sum_{l=1}^{k} \frac{1}{|\chi_l|^{\gamma-1}} \sum_{i \in \chi_l} c_{iq_l}$$

Note that for $\gamma = 0$, this is Minsum clustering, while for $\gamma = 1$, this is the k-median problem upto constant factors.

Definition 3 *A bicriteria approximation algorithm for generalized k-clustering is a (a, b) approximation if the cost of the solution is within a factor of a of the optimal cost, and the number of clusters produced is bk.*

The technique in [BCR00] gives a constant bicriteria approximation for the case when γ is a constant less than 1, in particular the minsum clustering as well. We show how to extend the result to constant $\gamma \geq 1$, thereby generalizing it to all constant γ. Our approximation factor will however be exponential in γ.

The material in this chapter also appears in [CG99, GMM00b, GMM00a].

7.2 The Buy-at-Bulk Facility Location Problem

consider the following capacitated variant of facility location: A facility at location i is associated with a capacity u_i. Multiple facilities can be built at location i; each can serve a demand of at most μ_i.

Given an instance I of capacitated facility location, we construct an instance I' of uncapacitated facility location with a modified distance function

$$c'_{ij} = c_{ij} + \frac{f_i}{\mu_i}.$$

The facility costs in the new instance are the same as the facility costs f_i in the capacitated instance. This construction is similar to the construction in [JV99]. Note that the distance function is asymmetric. In analyzing most facility location algorithms, we usually assume a symmetric distance function. However, for the facility location algorithms that we will consider, the analysis goes through even for distance functions of the form $c_{ij} + g(i)$, where c_{ij} is a metric and $g(i)$ is an arbitrary non-negative function. Since the asymmetric distance function we use is of this form, we can use the approximation ratios guaranteed by such algorithms for our case. Henceforth, when we refer to an approximation algorithm for uncapacitated facility location, we will mean an algorithm whose analysis applies to the case of asymmetric distance functions of this form.

The following lemma relates the assignment and facility costs for the original capacitated instance I to those for the modified uncapacitated instance I'

Lemma 7.2.1 *Given a solution SOL to I of assignment cost C_{SOL} and facility cost F_{SOL} there exists a solution to I' of assignment cost at most $C_{\text{SOL}} + F_{\text{SOL}}$ and facility cost at most F_{SOL}.*

Proof: Let y_i be the number of facilities built at i in SOL. Let x_{ij} be an indicator variable that is 1 iff j is assigned to i in SOL. $C_{\text{SOL}} = \sum_{ij} x_{ij} c_{ij}$ and $F_{\text{SOL}} = \sum_i y_i f_i$. The capacity constraint implies that $\sum_j x_{ij} \leq u_i \cdot y_i$. Using SOL, we construct a feasible solution SOL′ to I' as follows: The assignments of nodes to facilities is the same as in SOL. A facility is built at i iff at least one facility is built at i in SOL.

Clearly the facility cost of SOL′ is at most F_{SOL}. The assignment cost of SOL′ is

$$\begin{aligned}
\sum_{ij} x_{ij} c'_{ij} &= \sum_{ij} x_{ij} \left(c_{ij} + \frac{f_i}{u_i} \right) \\
&= \sum_{ij} x_{ij} c_{ij} + \sum_i f_i \frac{\sum_j x_{ij}}{u_i} \\
&\leq \sum_{ij} x_{ij} c_{ij} + \sum_i f_i y_i \\
&= C_{SOL} + F_{SOL}.
\end{aligned}$$

■

The following lemma states that given a feasible solution to I', we can obtain a solution to I of no greater cost.

Lemma 7.2.2 *Given a feasible solution* SOL′ *of I', there exists a feasible solution* SOL *of I such that the cost of* SOL *is at most the cost of* SOL′.

Proof: Let Y_i be an indicator variable that is 1 iff a facility is built at i in SOL′. Let x_{ij} be an indicator variable that is 1 iff j is assigned to i in SOL′. The solution SOL for instance I is obtained as follows: The assignments of nodes to facilities is exactly the same as in SOL$'$. The number of facilities built at i is

$$y_i = \left\lceil \frac{\sum_j x_{ij}}{u_i} \right\rceil .$$

Thus $y_i \leq \frac{\sum_j x_{ij}}{u_i} + Y_i$. The cost of SOL$'$ is

$$\begin{aligned}
\sum_{ij} x_{ij} c'_{ij} + \sum Y_i f_i &= \sum_{ij} x_{ij} \left(c_{ij} + \frac{f_i}{u_i} \right) + \sum_i Y_i f_i \\
&= \sum_{ij} x_{ij} c_{ij} + \sum_i f_i \left(\frac{\sum_j x_{ij}}{u_i} + Y_i \right) \\
&\geq \sum_{ij} x_{ij} c_{ij} + \sum_i f_i y_i
\end{aligned}$$

Note that the last expression denotes the cost of SOL. Thus the cost of SOL is at most the cost of SOL$'$. ■

We now explain why the analysis of certain facility location algorithms applies to the case of asymmetric distance functions of the form $c'_{ij} = c_{ij} + g(i)$ where c_{ij} is a metric and $g(i)$ is an arbitrary non-negative function. We need to argue that in all cases where the triangle inequality is invoked, the analysis still goes through for the more general distance function.

We now show that the analysis of the LP rounding algorithm in [Chu98] applies when the distance function is of the special form considered. The only invocation of the triangle inequality is in the proof of Lemma 8, in bounding the conditional expected backup service cost of demand node k. The proof considers two cases. In either case, the service cost c_{ik} is bounded by $c_{ik} \leq c_{ij_0} + c_{lj_0} + c_{lk}$. Observe that this inequality is of the same form as the inequality above considered in the proof of the primal-dual algorithm. (Here i' is replaced by i, j by k, j' by j_0 and i by l). Hence this inequality applies to the asymmetric distance function we construct. Thus the analysis of the LP rounding algorithm applies for our case. The approximation ratio in Theorem 6.3.5 is obtained by combining the combinatorial algorithm with the LP rounding algorithm of [Chu98, CS98]. Hence, the bounds reported in Theorem 6.3.5 apply to our case as well.

Suppose we have instance I of capacitated facility location. We first construct the modified instance I' of uncapacitated facility location as described above. We will use an algorithm for uncapacitated facility location to obtain a solution to I' and then translate it to a solution to I. Let $OPT(I)$ (resp. $OPT(I')$) denote the cost of the optimal solution for instance I (resp. I'). First, we show that a ρ approximation for uncapacitated facility location gives a 2ρ approximation for the capacitated version.

Lemma 7.2.3 *Given a ρ approximation for uncapacitated facility location, we can get a 2ρ approximation for the capacitated version.*

Proof: Lemma 7.2.1 implies that $OPT(I') \leq 2OPT(I)$. Since we have a ρ approximation algorithm for uncapacitated facility location, we can obtain a solution to I' of cost at most $\rho OPT(I') \leq 2\rho OPT(I)$. Now, Lemma 7.2.2 guarantees that we can obtain a solution to I of cost at most $2\rho OPT(I)$, yielding a 2ρ approximation for the capacitated version. ■

The above lemma implies an approximation ratio of ≈ 3.456 in polynomial time using the guarantee of Theorem 6.3.5, combining the LP rounding and combinatorial algorithms. This can be improved slightly using scaling, however it will be detailed and not in context of the reductions we wish to present in this chapter. We will now present the problem of facility location with lower bounds on the capacities. The reduction will be different, but underlying concepts of devising a metric to charge the facility cost and changing facility costs to charge the metric distance, are very interesting.

7.3 The Lower Bounded Facility Location Problem

This problem is a variant of the classical facility location problem. We are given a network $G(V, E)$ with a distance function c on the edges and a set of demand points. The cost of opening a facility at location i is f_i. In addition, there is a lower bound of

L_i on the demand a facility opened at i must satisfy. The goal is to open facilities and allocate the demands to the open facilities so that an open facility at i has at least L_i demand routed to it. The cost of our solution is the sum of the average distance traveled by the demands and the cost of the open facilities. We wish to minimize this cost.

Definition 4 *An approximation algorithm for load balanced facility location is a (a, b) approximation if the cost of the solution is within a times the optimal cost and facility i, if opened, serves at least $\frac{L_i}{b}$ demand.*

Let us denote by r the best known approximation ratio for classical facility location. We present a $(2r, 3)$ approximation to this problem. This generalizes to a $(\frac{1+a}{1-a}r, \frac{1}{a})$ approximation for $a < 1$.

7.3.1 The Algorithm

The algorithm proceeds in two basic steps. Our transformations work for the fractional solution obtained from the linear relaxation of the integer program discussed above, so our final approximation guarantee is against the fractional solution.

Facility Location: For facility i, we add the cheapest way to route at least L_i units of demand to i to the facility cost f_i. We next solve regular facility location with these facility costs. We take demand points in increasing order of distance to i till we have collected L_i amount of demand, this is the cheapest way to route the above said demand.

Rounding to Remove Facilities: Consider any open facility i that serves less than $\frac{L_i}{3}$ amount of demand. We close the facility and route the demands it serves to their closest open facilities. This transformation does not increase the cost of our solution.

We consider the open facilities in arbitrary order. Suppose facility i serves less than $\frac{L_i}{3}$ amount of demand, we close the facility and route the demands it serves to their closest open facilities. At the end of this process, we are guaranteed that each open facility i serves at least $\frac{L_i}{3}$ amount of demand, and each demand goes to the closest open facility.

7.3.2 Analysis

We will present the analysis with respect to an arbitrary fractional solution, SOL. The linear program can be expressed as follows:

$$\text{Minimize} \sum_i \sum_j d_j c_{ij} x_{ij} + \sum_i f_i y_i$$

$$\begin{array}{rcll} \sum_i x_{ij} & \geq & 1 & \text{for each} j \\ x_{ij} & \leq & y_i & \text{for each} i, j \\ \sum_j d_j x_{ij} & \geq & L_i y_i & \text{for each} i \\ x_{ij}, y_i & \in & \{0,1\} & \text{for each} i, j \end{array}$$

Lemma 7.3.1 *Consider any feasible fractional solution to the load balanced facility location problem with facility cost F_{SOL} and assignment cost C_{SOL}. We can construct a feasible fractional solution to the modified instance of the regular facility location problem of $2C_{\text{SOL}} + F_{\text{SOL}}$.*

Proof: Look at any fractional facility i. Since the feasible solution is routing at least L_i amount of demand to any open facility, the facility cost we assign in the new problem is at most the routing cost of the demand connected to that facility. Thus the total additional facility cost is at most C_{SOL}. ■

We have to show that removing a facility does not increase the total facility plus routing cost of the solution. For this, we show a feasible way to route the demands it serves so that the cost does not increase.

Lemma 7.3.2 *Removing a facility i serving less than $\frac{L_i}{3}$ amount of demand cannot increase the cost of our solution.*

Proof: Suppose we are closing facility i. Consider the closest demand point j which does not send demand to this facility. Suppose $c_{ij} = D$. If j is being served by i', $c_{i'j} < D$, as each demand point goes to the closest open facility.

Also, the facility cost $f'_i \geq \frac{2L_i}{3}D$. This follows because the facility serves only $\frac{L_i}{3}$ amount of demand, while the facility cost f'_i is f_i plus the cost of serving at least L_i units of demand.

When we close the facility, we can afford to use f'_i towards re-routing the demand it serves. We send the demand to i', the facility serving j. The extra cost for doing this is at most the cost of taking the demand from i to j and from there to i'. This distance is at most $2D$, and the demand is at most $\frac{L_i}{3}$, and so the total re-routing cost is at most $\frac{2L_i}{3}D$. ■

We now solve the facility location instance mentioned above. The cost of the solution we obtain is within a factor of $r = 1.728$ to the optimal solution for that instance. Therefore the total cost in the solution we compute is bounded in terms of the routing cost of the original fractional solution to within a factor of $2r$. The above can be summarized in the following theorem,

Theorem 7.3.1 *The load balanced facility location problem has a* $(2r, 3)$ *approximation where each demand is served by its closest open facility.*

We can scale the facility costs to improve the lower bounds. We will state the following tradeoff theorem.

Theorem 7.3.2 *The load balanced facility location problem has a* $(\frac{1+a}{1-a}r, \frac{1}{a})$ *approximation for* $a < 1$ *where each demand is served by its closest open facility.*

Proof: We start off by adding λ times the cheapest way to serve L_i units of demand to facility i to its cost. It is immediate that the approximation is $((\lambda+1)r, \frac{1}{a})$, where $\lambda = \frac{2a}{1-a}$. ■

7.4 Demand Dependent Facility Costs

We will now consider a variant of facility location where the facility cost is a *non-increasing* positive function of the amount of demand the facility serves. This model is appropriate in some situations where there is a startup operating cost for a facility (for example, a supermarket), but this cost decreases because of profits made with increasing number of customers.

Let us denote the cost function for facility i as f_i so that the cost of serving demand d is $f_i(d)$. Fix constants α and b larger than 1.

We assume the function f_i satisfies the following "nice" property. Let a and b be some constants larger than 1. For all demands ζ, $\frac{f_i(\zeta)}{f_i(a\zeta)} \leq b$. We also assume that all demands are larger than 1.

We can obtain a constant approximation for this problem by converting it to an instance of load balanced facility location. For each facility i, we create many copies $i_0, i_1, \dots$ of this facility. Copy i_j has lower bound a^j and cost $f_i(a^j)$. The metric and the demand points remain the same.

We now solve the modified instance as a load balanced facility location instance. In our solution, we will lose a factor of b because of scaling of the demands. Also, if we insist the lower bound is at least $\frac{1}{a}$ times the original bounds, our approximation ratio for the algorithm is $\frac{a+1}{a-1}r$. Since the actual cost is the cost of routing $\frac{1}{a}$ fraction less demand to the facility, our facility costs go up by another factor of b. Therefore the final approximation ratio is $\frac{b^2(a+1)}{a-1}r$, which is a constant.

7.5 Generalized Clustering

In this section, we define a generalization of the regular clustering problem, and show how to solve it using load balanced facility location. Recall the equivalent formulation to the *generalized k-clustering*, with a loss of factor 2 in the objective function. We are given n points in a metric space with distance function c. The goal is to partition the points into k clusters, $C_1, C_2, \dots, C_k$ with centers $q_1, q_2, \dots, q_k$ so that the following objective function is minimized:

$$\sum_{l=1}^{k} \frac{1}{|C_l|^{\gamma-1}} \sum_{i \in C_l} c_{iq_l}$$

The following was known for the case $\gamma = 0$, it can be extended to all $\gamma \leq 1$.

Theorem 7.5.1 *[BCR00] For $\gamma = 0$, the above has a $O(1)$ approximation with a constant factor blowup in the number of medians.*

The methods of [BCR00] does not carry over to the case $\gamma \geq 1$. We will consider this case in this section and demonstrate that the facility location with lower bounds allows us to prove a theorem similar to above.

For every point i, we create n copies of the point and denote them $i^{(1)}, i^{(2)}, \ldots, i^{(n)}$. We will use the copy $i^{(r)}$ if the point i is the center of a cluster of size r. We can now formulate generalized k-clustering for $\gamma \geq 1$ as an integer program. Let $y_i^{(r)}$ denote whether i is the center of a cluster of size r. Let $x_{ij}^{(r)}$ be set to 1 if point j is in a cluster of size r with center at i.

$$\text{Minimize} \sum_i \sum_j \sum_r x_{ij}^{(r)} d(i,j) \frac{1}{r^{\gamma-1}}$$

$$\begin{array}{rcll} x_{ij}^{(r)} & \leq & y_i^{(r)} & \text{for each } i,j,r \\ \sum_i \sum_r x_{ij}^{(r)} & \geq & 1 & \text{for each } j \\ \sum_j x_{ij}^{(r)} & \geq & r y_i^{(r)} & \text{for each } i,r \\ \sum_i \sum_r y_i^{(r)} & \leq & k & \end{array}$$

We can now formulate this as a facility location problem with lower bounds. Every point j has demand 1. Facility location $i^{(r)}$ has cost 1 and lower bound r. The new integer program formulation is therefore:

$$\text{Minimize} \sum_i \sum_j \sum_r x_{ij}^{(r)} d(i,j) \frac{1}{r^{\gamma-1}} + \sum_i y_i^{(r)}$$

$$\begin{array}{rcll} x_{ij}^{(r)} & \leq & y_i^{(r)} & \text{for each } i,j,r \\ \sum_i \sum_r x_{ij}^{(r)} & \geq & 1 & \text{for each } j \\ \sum_j x_{ij}^{(r)} & \geq & r y_i^{(r)} & \text{for each } i,r \end{array}$$

Let F^* and S^* denote the optimum facility cost and service cost respectively. Note that $F^* = k$ for our problem.

For facility $i^{(r)}$, we add b times the cheapest way to ship r amount of demand to that facility. The optimum solution pays $F^* + bS^*$ for the facility cost, and S^* for the service cost.

Note that the distance from j to $i^{(r)}$ is $\frac{d(i,j)}{r^{\gamma-1}}$, and this is not a metric. But, note that the distances are multiplied by a number that depends only on the facility and not on the demand point. It is shown in [BCR00] that the primal dual algorithm in [JV99] can be modified so that it works even if the metric is multiplied by a number that depends just on the facility.

We can therefore perform the first step of the load balanced facility location algorithm just as before. For the second step where we close facilities, we consider facilities which do not satisfy the lower bound in *increasing* order of r.

Lemma 7.5.1 *If we close facilities in Step 2 of the load balanced facility location algorithm in increasing order of r, then, the cost of our solution does not go up.*

Proof: Suppose we close a facility $i^{(r)}$ because it did not have enough demand coming to it. Let D be the distance in the modified metric to the closest point j which was not connected to $i^{(r)}$. Then, j went to some facility $l^{(r')}$ which was less than D distance in the modified metric, and $r' \geq r$. We therefore have: $d(i,j) = Dr^{\gamma-1}$ and $d(l,j) \leq Dr'^{\gamma-1}$.

Let the cost of routing the demands to i be C, and that to l be C'. If x is the amount of demand routed to i, we have:

$$C' \leq C\left(\frac{r}{r'}\right)^{\gamma-1} + xD + xD\left(\frac{r}{r'}\right)^{\gamma-1} \leq C + 2Dx$$

Therefore, when we add b times the cheapest service cost for satisfying the lower bound, we can obtain a constant approximation so that every open facility $i^{(r)}$ serves at least $r\frac{b}{2+b}$ demand. ∎

Now, the algorithm in [JV99] gives the following approximation:

$$3F + S \leq 3(F^* + S^*)$$

We have modified the facility costs so that F^* becomes $F^* + bS^*$. We therefore have:

$$3F + S \leq 3(F^* + S^*(1+b))$$

Also, every open facility $i^{(r)}$ serves at least $r\frac{b}{2+b}$ demand. Now, we apply Lagrangian relaxation. We scale the facility costs by a factor of λ and then solve the modified problem. In the final solution, we scale back the facility costs. We will therefore have:

$$3\lambda F + S \leq 3(\lambda F^* + S^*(1+b))$$

We now guess the value of λ as $\frac{S^*}{aF^*}$. We therefore have:

$$\begin{aligned} F &\leq (1+(1+b)a)F^* \\ S &\leq 3(1+b+\tfrac{1}{a})S^* \end{aligned}$$

Now, we are paying a routing cost which is scaled for a cluster of size r, while the actual cluster size could be $r\frac{b}{2+b}$. This causes our service cost to go up by a factor of $\left(1+\frac{2}{b}\right)^{\gamma-1}$. We therefore have the following theorem:

Theorem 7.5.2 *For any $\gamma \geq 1$, generalized k-clustering has a bicriteria approximation of factor $3(1+b+\frac{1}{a})(1+\frac{2}{b})^{\gamma-1}$ using $1+(1+b)a$ clusters.*

Chapter 8

Fault-Tolerant Facility Location

8.1 Introduction

The facility location problem models the tradeoff of developing resources (facilities) and the utility (reduction in assignment cost) accruing from such. In several applications, caching on a network, for example, fault tolerance is also a facet. The placement of caches should be resistant to failures of nodes and links. The facility location problem does not provide any guarantee about the second closest facility to any node. In a fault-tolerant situation, the cost of a location that requires a "backup" would be a linear combination of the costs of assigning a demand location to the two facilities.

In this chapter we consider the problem of fault-tolerant facility location in which every location j specifies to be assigned to a number r_j of facilities. The cost of assignment of this location is a weighted combination of these r_j assignments. [JV00] provided a primal dual approximation which was logarithmic in the largest requirement. Even for all requirements one, the primal dual algorithm has an integrality gap of at least 3. However, the fault tolerant variant of the k-center problem has constant factor approximation algorithms [CGR98, KPS97, Kru95, Ple87]. Here resolve this issue by providing a constant factor approximation for the fault-tolerant facility location problem.

The fault-tolerant facility location is complicated, compared to the fault-tolerant

k-center problem. Ideas like considering a threshold graph to reduce to the unweighted problem, or searching for a suitable power of a graph to generate a lower bound, do not work. We instead use linear program rounding to provide a constant-approximation - it demonstrates that the natural LP formulation for this problem has smaller integrality gap than any dual formulation.

We use filtering in a fashion similar to [LV92b, STA97], however we combine it with scaling and uncrossing steps. These steps allows us to ensure that while we are considering a filtered neighborhood, if a demand point is assigned to several distinct facilities within this neighborhood we will round in such a way that the entire assignment can be rearranged to maintain feasibility. They cannot be swapped simply: if the first facility has a higher weight in calculating the assignment cost than the backups, for example.

We subsequently use greedy local improvement as in Section 6.3.4 to construct a solution of integrality gap 2.47. The core of the similarity is that once a set of facilities are fixed, the assignment costs are also determined. However, the similarity does not seem to extend to allow us to delete facilities as in the combinatorial facility location algorithms in [KPR98, CG99]. The clustered rounding techniques of [Chu98, CS98] also appears to have this problem with feasibility. An approximation algorithm for the facility location problem based only on add and exchange operations would be the stepping stone for a combinatorial approximation for this problem.

The material in this chapter appears in [GMM00c].

8.2 The Linear Program and a structured solution

This problem can be formulated as an integer program. Here, y_i denotes whether facility i is open, and $x_{ij}^{(r)}$ denotes that demand j is assigned to facility i and facility i is the r^{th} closest open facility to j.

$$\text{Minimize} \sum_i \sum_j \sum_r c_{ij} w_j^{(r)} x_{ij}^{(r)} + \sum_i f_i y_i$$

$$\begin{array}{rcll} \sum_i x_{ij}^{(r)} & \geq & 1 & \text{for each } j, r \\ \sum_r x_{ij}^{(r)} & \leq & y_i & \text{for each } i, j \\ y_i & \leq & 1 & \text{for each } i \\ x_{ij}^{(r)}, y_i & \in & \{0,1\} & \text{for each } i, j, r \end{array}$$

The linear relaxation of the above-mentioned integer program gives us a fractional solution. We will convert the solution (x, y) to a solution $(\bar{x}, y)$ such that the cost of the new solution does not increase, and the new solution satisfies certain useful properties. This is not required for the algorithm; this structured solution is constructed so that we can compare our solution to this structured solution, to argue about the approximation factor.

We will treat a demand point j as having r_j copies under the constraint: no two copies of any demand point are assigned to the same facility. In the fractional setting this reduces to the condition $\sum_r x_{ij}^{(r)} \leq y_i \leq 1$.

For every demand point j, we reassign it to facilities, fractionally, as follows. Order the facilities in non-decreasing distance from j, breaking ties arbitrarily. The ordering is fixed for the facility for the algorithm. The first demand copy $j^{(1)}$, is assigned to the initial set of facilities that sum up to 1 fractionally. The last facility i in this set can be incompletely assigned, i.e. $\bar{x}_{ij}^{(1)} < y_i$. For the second copy, we start from this facility i, and ensure $\bar{x}_{ij}^{(1)} + \bar{x}_{ij}^{(2)} = y_i$. After that we again pick up one unit of facility fractionally, $\sum_i \bar{x}_{ij}^{(2)} = 1$, and repeat this for all the copies of the demand point. The following lemma is true by construction:

Definition 5 *Define $\mathcal{C}_j^{(r)} = \sum_i \bar{x}_{ij}^{(r)} c_{ij}$. Define $\mathcal{C}_j^{(r)}(\zeta)$ to be the distance at which the r'th copy of the demand point j picks up at least ζ fractionally; thus, $\int_0^1 \mathcal{C}_j^{(r)}(\zeta) d\beta = \mathcal{C}_j^{(r)}$.*

Lemma 8.2.1 *The cost of the solution does not increase; $\sum_{j,r} w_j^{(r)} \mathcal{C}_j^{(r)} = C^*$.*

Lemma 8.2.2 *For any facility i and demand j, there exist at most two values of r such that $\bar{x}_{ij}^{(r)} > 0$. Further, if two such values exist they must be consecutive.*

Once the (fractional) facilities are fixed, it is simple to see that the above reassignment is (one of) the best possible. Intuitively, the copies of the demand with larger weight w (and thus smaller r) go to the closer open facilities.

8.3 An Approximation Algorithm

The algorithm will try to round the fractional solution. It proceeds in two phases. The idea is to use the filtering technique combined with reassignment of the fractional demands, such that each copy of the demand goes to a different facility. Recall, we treat the different copies of a demand as separate, and denote the r^{th} copy of demand j by $j^{(r)}$. Fix $\zeta \in (0,1)$, to be determined later.

8.3.1 Phase 1: Filtering and Scaling

In this section we will modify the above solution to create a new solution $(\hat{x}, \hat{y})$, which we will round it in the next phase. This phase uses filtering. For every demand $j^{(r)}$, we consider the fractionally assigned facilities in increasing order of distance. Let i be the first facility in the ordering of $j^{(r)}$ such that $\sum_{i'}^{i} \hat{x}_{i'j}^{(r)} \geq 1 - \zeta$. In other words, i is the first facility for which j picks up $1 - \zeta$ facility fractionally.

For all i' appearing before i in our ordering, we set $\hat{x}_{i'j}^{(r)} = \bar{x}_{i'j}^{(r)}$. We set $\hat{x}_{ij}^{(r)}$ so that the total assignment of $j^{(r)}$ is *exactly* $1 - \zeta$. For all i' appearing after i in the ordering, we set $\hat{x}_{i'j}^{(r)} = 0$.

We scale the $\hat{x}_{ij}^{(r)}$ by $\frac{1}{1-\zeta}$ so that $\sum_i \hat{x}_{ij}^{(r)} = 1$ for all $j^{(r)}$. We next set $\hat{y}_i = \min\left\{\frac{y_i}{1-\zeta}, 1\right\}$.

Lemma 8.3.1 *If $\hat{x}_{ij}^{(r)} > 0$, then $c_{ij} \leq \frac{1}{\zeta}\mathcal{C}_j^{(r)}$.*

We first show that $(\hat{x}, \hat{y})$ is feasible. For this, it is enough to show the following lemma:

Lemma 8.3.2 *For all i, j, we have $\sum_r \hat{x}_{ij}^{(r)} \leq \hat{y}_i$.*

Proof: Before filtering, by Lemma 8.2.2, we knew that at most two copies of a demand went to any one facility. Suppose we are considering facility i and demand j. If exactly one copy, say r is assigned to i, the inequality trivially holds, as $\hat{x}_{ij}^{(r)} \leq \hat{y}_i$.

We therefore assume that two copies of j are assigned to i. Let $j^{(r)}$ and $j^{(r+1)}$ be assigned to i. Note that by the construction in Section 8.2, i is the furthest assigned facility to $j^{(r)}$ and the closest to $j^{(r+1)}$.

The interesting case is $y_i \geq 1-\zeta$, otherwise $\sum_r \hat{x}_{ij}^{(r)} \leq y_i$ was true, and the lemma follows as we scale both the left and right hand sides by the same amount.

Let us look at the $\hat{x}_{ij}^{(r)}$ values before scaling (but after filtering). Therefore we need to show $\sum_r \hat{x}_{ij}^{(r)} \leq 1-\zeta$, then, scaling could not have increased this value beyond 1. When we were considering $j^{(r)}$ for filtering, we must have set $\hat{x}_{ij}^{(r)} = \max(0, \bar{x}_{ij}^{(r)} - \zeta)$, as i is the furthest assigned facility to $j^{(r)}$. We now consider two cases:

Case 1: $\hat{x}_{ij}^{(r)} = 0$. Then, $\hat{x}_{ij}^{(r+1)} \leq 1-\zeta$ because of filtering on $j^{(r+1)}$.

Case 2: $\hat{x}_{ij}^{(r)} = \bar{x}_{ij}^{(r)} - \zeta$. This implies $\hat{x}_{ij}^{(r)} + \hat{x}_{ij}^{(r+1)} = \bar{x}_{ij}^{(r)} + \bar{x}_{ij}^{(r+1)} - \zeta \leq 1-\zeta$, as $\bar{x}_{ij}^{(r)} + \bar{x}_{ij}^{(r+1)} \leq y_i \leq 1$.

This completes the proof. ■

Lemma 8.3.3 *Let $r_1 < r_2$. For any demand j, all the edges used by $j^{(r_1)}$ are at most as long as the edges used by $j^{(r_2)}$ in the filtered and scaled solution.*

Proof: The rearrangement from Section 8.2 guarantees this on the un-filtered solution; filtering does not change the ordering of the edges. ■

8.3.2 Phase 2: Rounding

At the beginning of this phase we have a solution $(\hat{x}, \hat{y})$. In this phase we will round the fractional solution. We will preserve $\sum_r \hat{x}_{ij}^{(r)} \leq \hat{y}_i$ as an invariant.

Step A: Arrange all the copies (irrespective of demand points) in increasing order of the distance to the farthest fractional facility serving it. We will pick the copies in this order, and repeatedly apply Step B.

Step B: Assume we have picked the r'th copy of the demand point j. Let the set of facilities serving it be $P_j^{(r)}$. Note that copies of j will be picked in increasing order.

We will build a facility at the cheapest facility i in $P_j^{(r)}$. We show how to modify the solution:

1. We select facilities i' with $\hat{x}^{(r)}_{i'j} > 0$ starting with i until the total fraction by which selected facilities are open is exceeds 1. Let $Y = \sum_{i'} \hat{y}_{i'} \geq 1$ be the total fraction by which these facilities are open.

2. The last picked facility i'' may be picked partially, if $Y > 1$. Since all $\hat{y}_i \leq 1$, $i'' \neq i$. Make two copies, i_1 and i_2, of facility i''. Set $\hat{y}_{i_2} = Y - 1$, and $y_{i_1} = \hat{y}_{i''} - \hat{y}_{i_2}$. For any j' distribute $\hat{x}^{(r)}_{i''j'}$ arbitrarily between the two copies; maintaining $\sum_r \hat{x}^{(r)}_{i'j} \leq \hat{y}_{i'}$ for both $i' = i_1$ and $i' = i_2$. The copy i_1 is selected and i_2 is not. Denote the set of picked facilities by $\hat{P}$.[1]

3. For any j' (inclusive of j), consider its copies $r_1, r_2, \ldots, r_k$ served by $\hat{P}$.

   Compute $X^{(1)}_{j'} = \sum_{i' \in \hat{P}} \hat{x}^{(r_1)}_{i'j'}$, and $X^{(2)}_{j'}, \ldots, X^{(k)}_{j'}$ likewise. These quantities denote the fractions to which the copies of j' are assigned to the facilities in $\hat{P}$ respectively. Compute $Y^{(1)}_{j'} = \sum_{i' \notin \hat{P}} \hat{x}^{(r_1)}_{i'j'}$, and similarly $Y^{(2)}_{j'}, \ldots, Y^{(k)}_{j'}$. These quantities denote the fractions by which the copies of j' are assigned to facilities outside the set $\hat{P}$, respectively.

4. We open a facility completely at i. For every j' fractionally assigned to the set $\hat{P}$, we assign the smallest numbered copy (r_1) of j' to be completely served by i.

   Uncrossing Step: We now reassign the remaining copies of j' completely to facilities outside the set $\hat{P}$ by performing an uncrossing step. We consider the fraction $Y^{(1)}_{j'}$ by which the copy $j'^{(r_1)}$ was assigned to facilities not in $\hat{P}$, and re-assign this to the other copies of j' which were originally assigned to the set $\hat{P}$ as follows: Consider the fraction by which $j'^{(r_1)}$ was assigned to the closest facility not in $\hat{P}$. We assign this fraction to $j'^{(r_2)}$ until either $j'^{(r_2)}$ is completely satisfied, or we have assigned the fraction completely. In the former case, we move to $j'^{(r_3)}$; in the latter case, we consider the next closest facility not in $\hat{P}$ that was previously connected to $j'^{(r_1)}$, and repeat.

   During uncrossing, we maintain the invariants $\sum_{i'} \hat{x}^{(r_t)}_{i'j'} = 1$ and $\sum_t \hat{x}^{(r_t)}_{i'j'} \leq \hat{y}_{i'}$.

[1] $i_1 \in \hat{P}$ and $i_2 \notin \hat{P}$. The logic is that since a facility is being built at i, we can build facility later at i''; but we need to account for $\hat{x}$. Note $\sum_{i' \in \hat{P}} \hat{y}_{i'} = 1$.

Lemma 8.3.4 *For any j', we can always re-assign the copies $r_2, r_3, \ldots, r_k$ completely outside set $\hat{P}$ by the uncrossing step.*

Proof: The only thing we need to check is that we maintain Now,

$$X_{j'}^{(t)} + Y_{j'}^{(t)} = 1 \quad \text{for all } \ 1 \leq t$$

Consider the total fraction by which demand j' is assigned outside the set $\hat{P}$. This was originally $\sum_t Y_{j'}^{(t)}$. We remove fraction $Y_{j'}^{(1)}$ (because we assign $j'^{(r_1)}$ completely in set $\hat{P}$) and add $\sum_t X_{j'}^{(t)} - X_{j'}^{(1)}$ because of the uncrossing. Therefore, the final fraction is:

$$\sum_t Y_{j'}^{(t)} + \sum_t X_{j'}^{(t)} - (Y_{j'}^{(1)} + X_{j'}^{(1)})$$

which evaluates to exactly $k-1$. This means that the re-assignment is always possible. ∎

At the end of one iteration of Step B, we have opened facility i completely. For every demand fractionally assigned to the set $\hat{P}$, the smallest assigned copy is completely assigned to i. Every other copy is fractionally re-assigned completely outside set $\hat{P}$. We drop the set $\hat{P}$ and the copies $j'^{(r_1)}$ from further consideration.
Using arguments similar to Section 3.2, it follows:

Lemma 8.3.5 *The r'th copy of a demand point j is assigned within a distance $\frac{3}{\zeta}\mathcal{C}_j^{(r)}$, thus the service cost is at most $\frac{3}{\zeta}C^*$. The facility cost of the above solution is at most $\frac{1}{1-\zeta}F^*$.*

Proof: Since $\sum_{i \in \hat{P}} y_i = 1$, and we charge this to the cheapest facility, the facility cost cannot go up in this step. But since we scaled the $\hat{y}_i$ in Phase 1, our cost could go up by $\frac{1}{1-\zeta}$. Since the distances form a metric, and we are using the demand with the smallest $\hat{\mathcal{C}}_j^{(r)}$, the distance cannot have more than tripled. Note that since we use the assignments of r_1 (and r_1 was the smallest numbered copy), by Lemma 8.3.3, the new assignments introduced in uncrossing can only decrease. This, combined with the distance bound from Lemma 8.3.1 completes the proof. ■

Setting $\zeta = \frac{3}{4}$ we have:

Theorem 8.3.1 *Fault-Tolerant Facility Location has a factor* 4 *approximation in polytime.*

Choosing the best ζ gives us a factor 3.16 approximation.

8.3.3 Facility Location Revisited

We will show how to improve the approximation factor, and more importantly the similarities between uncapacitated facility location and the fault-tolerant version. Consider the 'Add' heuristic, in Section 2.2.

Recall that, the $Gain(i)$ of a facility i is the decrease in *total cost* (decrease in assignment cost minus the facility cost of i) of the solution on addition of facility i to the solution. The facility with the best gain ratio is the facility i with best ratio of $Gain(i)/f_i$. If the $Gain(i)$ is positive, the heuristic adds this node and repeats; and stops otherwise.

The computation of the assignment cost is easy following the observation that once the set of facilities are fixed, every demand point chooses the facilities serving it, in increasing order of distance. The improvement of the solution depends on the quality of gain we can prove at every step; the following lemma can be proved:

Lemma 8.3.6 *If the current costs of facility and assignment be F and C, respectively, and there exists a fractional solution with costs F^* and C^* with $C \geq F^* + C^*$, then there exists a node with ratio $(C - F^* - C^*)/F^*$.*

Proof: Consider the fractional solution $(\bar{x}, \bar{y})$, and copy nodes fractionally and sufficiently to ensure that $\bar{x}_{ij}^{(r)} = \bar{y}_i$ or 0 for all i, j, and r. Now consider placing the facilities of the fractional solution (or their copies) with the fraction $\bar{y}_i$; and changing the assignment as mandated by the solution. Define $Gain'(i)$ to be the change (possibly negative) of cost by this operation. Notice at this step we may be making suboptimal assignments to this fractional facility thus $Gain'(i) \leq Gain(i)$. Now

$$\sum_i y_i Gain'(i) = C - C^* - F^* \quad \text{and} \quad \sum_i y_i f_i = F^*$$

In the first summation, we have to sum over the different copies of a facility since they have separate $Gain'()$. This shows the average $Gain'()$ to be as claimed in the lemma, and $Gain()$ being only better, proves the lemma. ■

The above argument is exactly the same for the original facility location problem, as proved in Lemma 2.2.3. This is because the fraction solution ensures fault-tolerance by its feasibility; in our solution, we assign the smallest numbered copy (r) of a demand point to this new facility, if closer than the assignment distance for this copy in the current solution. This has a domino effect, since the next smallest copy $(r+1)$ now gets assigned to where the previous (r'th) copy was assigned to and so forth. Other than the computation of $Gain()$, the arguments are exact.

By the arguments in Lemma 6.3.4 the next theorem follows.

Theorem 8.3.2 *Fault-Tolerant Facility Location has a* 2.47 *approximation algorithm in polytime.*

Chapter 9

Hierarchical Placement Problems

9.1 Introduction

In this chapter we consider applications requiring layered placement of resources. Examples of such problems include hierarchical caching on the internet, multi-level facility location problems, and edge installation problems. In this chapter we develop techniques to approximate these problems within constant factors. We will be concerned with hierarchical caching and multilevel placement problems in this chapter. We will use this chapter also as a stepping stone to develop a constant factor approximation for the single-sink edge-installation problem, in the next chapter (Chapter 10).

Considerable research has been devoted to optimizing internet access time through the use of various caching strategies. The central idea is to replicate data to reduce the access costs of the users. Caching schemes model tradeoffs between the storage and management of the copies and the average time to access a certain object in a network. The facility location problem has been used to model such adaptive data replication and placement problems [MW88]. However in most caching scenarios the caching is performed hierarchically. Small caches are placed close to the users while larger ones reside in the backbone of the network [Wes97, CLM99].

We consider the problem of hierarchical placement of caches, using the web as our example. We model the caches at each layer of the hierarchy as having a fixed miss rate. Thus we consider a hierarchical caching scheme as follows: demand locations communicate with caches of type 1, which in turn communicate with caches of type 2 and so on, until we reach the servers located at the top-most level. We assume that each cache in level i of the hierarchy has miss rate σ_i. We refer to caches in level i of the hierarchy as type-i caches.

The miss rate of each layer is modeled to be an arbitrary fixed constant. Each demand location in the network could have many users, so it seems natural to model an average user profile instead of considering each user to be distinct. There is also some evidence that the distribution of the fraction of times different pages are accessed is Zipfian [BCF+99, ZIRO99]. A natural caching scheme is to cache the most popular pages [TRS97, BCF+99, ZIRO99]. Under this scheme, the miss rate is independent of the incoming request rate and depends on the Zipf distribution over pages of the incoming request stream and the size of the cache [BCF+99]. Capacitated models are frequently introduced to prevent cache overload. Constraints in capacities immediately pose questions of "economies of scale" or "buy at bulk" constraints [SCRS97]. We address this issue as well.

The quality of a solution is measured on two axes, the cost of placing the caches and the average time to serve the requests. A simple formulation is to optimize the sum of the placement cost and the service cost, as in facility location problems. For most such problems heuristics that try optimizing both the criterion follow without much difficulty from those which attempt minimizing the sum. The access time will be modeled by a metric distance function on the possible locations in the network. We will define a formal model in Section 9.1.1. This problem subsumes facility location problem and is Max SNP-hard. We will obtain a constant factor approximation, with running time depending polynomially on the number of demand locations and the number of layers.

This problem arises in a variety of other contexts as well. Consider the problem of caching multimedia clips [DS96] in a hierarchical fashion. It can be shown analytically that the interval caching scheme illustrated in [DS96, AM00] is optimal for this application. Assuming a statistical distribution for the interval between successive requests we can show that the reduction in rate is a function of the length of the clip, size of the cache and the distribution, but is independent of the actual request rate itself. This implies that if we place the caches in a hierarchy, so that the caches in the same layer have the same size, the reduction in request rate due to a cache at level i is a fraction σ_i that depends just on the layer. In such a scenario, we can define the problem of Hierarchical Placement just as we did for caching web pages above.

A constant factor approximation for the buy at bulk facility location problem follows from [JV99]. Although not explicitly stated, the only observation one requires is that the algorithm of [JV99] carries through for capacities which are dependent on location only. Simply making copies of a facility (as many copies as supported by the buy at bulk constraints at this location) solves this problem. Although this is a trivial extension of the result in [JV99], it will be central in handling capacities and buy at bulk constraints in the context of Hierarchical Placement.

The Multi-Level facility location problem was previously studied in [TL84, KvEH77, ALLQ96, VRE82]. In a typical application, we have to route material produced at factories to consumers through a hierarchy of warehouses. We place large warehouses close to the factories, and smaller ones close to the consumers. The cost of placing a warehouse depends on the level the warehouse is to function in, its size, and its location. This problem closely resembles the web caching problems discussed above, except that there is no reduction in the demand from one layer to the next. We provide a constant factor combinatorial approximation for this problem with running time polynomial in the size of the input and the number of levels.

This problem has been studied in [TL84, KvEH77, ALLQ96, VRE82, STA97, ACS99]. In [STA97] a factor 4 approximation was given for this problem, with a fixed number of layers. However, the algorithm requires n^k variables and equations for k levels for running time exponential in the number of layers. In [ACS99], an ellipsoid algorithm was used to solve this exponential size linear program, with polynomially many constraints, to achieve a factor 3 approximation in polynomial time. We show how to reduce this problem to repeated solving of the facility location problem, providing an $O(1)$ approximation whose running time depends only polynomially on k. Thus we achieve a simpler, combinatorial approxiamtion algorithm for the multi level facility location problem.

The material in this chapter appears in [GMM00d].

9.1.1 Formal Definitions

We are given an undirected network $G(V, E)$ with a distance function on the edges, a set of demand points, and a set of possible locations and caches of types $1, \ldots, k$ with miss rates $\sigma_1, \ldots, \sigma_k$. Each cache is specified by the tuple $\langle u, y, i\rangle$, where u denotes the location of the cache, y its capacity and i its type (level). The cost of such a cache is denoted by $f^y_{u,i}$.

We have to place the caches satisfying the following constraints:

1. Each of the demand points are served by type 1 caches. Each of the type 1 caches placed behave as demand points that need to be served by type 2 caches. We can consider the demand points to be level 0 caches.

2. The demand of a type 1 cache is σ_1 times the amount of the demand served by the type 1 cache. The caches of type 2 onwards behave similarly as the caches of type 1, with miss rates σ_i for level i.

3. The cost of the solution is the cost of placing all the caches and the cost of servicing all the demands. The latter is the sum of the distance times the demand of the demand points, the type 1 caches, and caches of higher types.[1]

We present three versions of this problem in increasing order of difficulty. There are other intermediate versions, but we omit discussion as the techniques used will be similar. The first problem demonstrates the general geometric property we would consider to solve this problem, the second problem shows how to deal with costs which are dependent on location. The third combines those techniques in their generality, with capacities and buy at bulk thrown in.

[1]It is also conceivable that the cost of an access is accounted by the distance to the type (level) 1 cache, and in case of a miss, σ_1 times the distance back to the original demand node and then to the type (level) 2 cache. In this case the demand node is aware of the miss and finding the next level cache itself. This objective function decomposes into several applications of the facility location problem, one for each level. In this model we are assuming that the demand node is not aware of a cache miss and the redirection is performed by the caches, as in most caching scenarios.

SIMPLE-PLACEMENT: In this version, the cost of a cache is independent of the location and just depends on the type of the cache. The caches have no capacity in terms of the amount of incoming demand.

MULTI-LEVEL: Here, all the σ_i are 1, but the cost of a facility depends on the location as well as the type of the cache. This is the classic multi-level facility location problem.

GENERAL-PLACEMENT: Cache of type i now has have capacity M_i in terms of the amount of incoming demand. We are allowed to place multiple caches at a location. The cost of a cache depends both on the location and the type of the cache.

It is straightforward that these problems extend the facility location problem, and therefore are NP-Hard. In fact these problems cannot be approximated within factor 1.47 unless $P = NP$, see Chapter 5. In the next sections, we will present constant factor approximations for these problems.

9.2 The Simple-Placement Problem

Recall that in this version of the problem, the cost of a cache of type i is f_i and is independent of the location of the cache. Each cache can support infinite user demand, we will address this restriction in the later sections. We first present a small miss rate theorem, which is crucial to our algorithm.

9.2.1 Small Miss Rate Theorem

We reduce the problem to a new instance where the miss rates are small. We will used primed quantities to distinguish the variables of new instance from the original one.

Theorem 9.2.1 *Given an instance of the* SIMPLE-PLACEMENT *problem with parameters $\sigma_1, \ldots, \sigma_k$, the problem can be reduced to a different caching problem with $k' \leq k$*

types of caches and a different set of parameters $\sigma'_1, \ldots, \sigma'_{k'}$, such that each σ'_ℓ is a constant less than a where $0 < a < 1$, and at most a $\frac{1}{a}$ blowup in service cost.

Proof: We show this by providing a new instance in which the σ'_i are at most a, and any solution in the newer instance can be realized in the older instance at the same cost. We will also show that the optimal solution in the older instance can be modified to give a feasible solution in the newer instance of cost at most $\frac{1}{a}$ times.

We first find a j such that $\prod_{i=1}^{j_1} \sigma_j < a$. We will pretend that the caches of types 1 through j form a single cache in the new instance.

We repeat the above for caches of type i for $i > j_1$, that is we find j_2 such that $\prod_{i=j_1+1}^{j_2} \sigma_i < a$. Thus the number k' will be the number of times this step can be performed. The new σ'_1 will be $\prod_{i=1}^{j_1} \sigma_i$ and σ'_2 will be $\prod_{i=j_1+1}^{j_2} \sigma_i$ and so on.

In the new instance the cost of a cache of type 1 will be the sum of the costs of caches of types 1 through j_1 in the original instance. Similarly the cost of a cache of type 2 in the new instance will be the sum of the costs of the caches of types $j_1 + 1$ through j_2.

The demand points remain the same.

It is easy to see that a solution in the new instance gives a solution of the old instance by simply placing caches of types 1 through j_1 in the locations where a type 1 cache is placed in the new instance. And similarly for caches of type 2 and onwards in the new instance are replaced by (possibly) sets of caches of the old instance. It is easy to see that the cost remains unchanged in this process.

Thus we showed that a solution of a certain cost in the new instance can be realized in the older instance at the same cost. We will now show that the optimal solution of the older instance can be modified into a feasible solution of the newer instance by at most a constant factor blowup in the cost.

Consider the optimal solution. Consider the level j_1 at which the product of the parameters $\prod_{i=1}^{j_1} \sigma_i < a$ for the first time. If we were to short-circuit all the paths from the demand points to the type j_1 caches, the cost of service will be at most $\frac{1}{a}$ times original.

This is because the product of the σ_i values of levels 1 through $j_1 - 1$ in the original instance is greater than a. Notice that since every demand point is assigned to an unique type 1 cache and each such to an unique type 2 cache and so on, each demand can be associated with an unique cache of type j.

We can now place caches of type 1 through $j_1 - 1$ at the same location u where a type j_1 cache is placed, and since every cache of type i is assigned to an unique cache of type $i+1$, the optimal assignment had also built at least these many caches for this cache of type j_1. Thus the cost of caches does not go up in this process. Thus we only increased the service cost by a factor of $\frac{1}{a}$. This completes the reduction. ■

This tells us that any instance of SIMPLE-PLACEMENT can be reduced to the case where the miss rate σ_i is less than a.

9.2.2 Solving for small miss rate

The algorithm is natural and simple. We repeatedly solve an uncapacitated facility location problem for each of the levels. The reason why such a solution works is that since the miss rate is sufficiently bounded away from one, the secondary demands introduced at the facilities and their contributions decrease fast and geometrically. Recall that each σ_i for each level is at most a.

We scale all the facility costs by a factor λ, we will set the value of this parameter later.[2] We solve the uncapacitated facility location for the first level. Next, we solve the facility location problem for the second level with demands located at the facilities opened at the earlier level, but scaled down by σ'_1. We continue the process till we have considered all the levels.

We denote the best approximation ratio for the facility location problem by r. For level i, let the total cost of the caches placed by our solution be F_i and that placed by the optimum solution on the new cache types be F_i^*. Also, let the service cost be S_i and S_i^* respectively.

We need to bound the cost of the optimum facility location solution at a given layer. Let A_i denote the cost of serving our algorithm's level $i-1$ facilities by the optimal's level i facilities. We can prove the following:

[2]In the final analysis we can only bound a linear combination of the facility and service costs with different coefficients, this will allow us to make both equal.

Lemma 9.2.1 *The facilities of the level i of the optimal solution provides a feasible solution to our problem at the level i of facility cost F_i^* and service cost at most $A_i \leq S_i^* + \sum_{j=1}^{i-1} a^j(S_{i-j} + S_{i-j}^*)$.*

Proof: It is easy to see the lemma is true for $i = 1$. We will proceed by induction, we would need to show that the cost of the solution of level $i+1$ is $S_{i+1}^* + a(A_i + S_i)$.

Consider the facilities we have built at level i, (although we have bounds via the approximation algorithm, the way the approximation apportions the facility and service costs at level i are not in our control). We can view the effect of solving the facility location problem at level i being that the demands at level $i-1$ were moved to facilities we constructed at level i (with a factor σ_i reduction).

Now consider each of these demand points (at the facilities of level i we chose to build) connected to the optimal's facilities in level $i+1$. We will actually send the demand at a point via several paths, but if we send all the demand along the cheapest cost path the cost will only be lower.[3] We will account for the cost in three parts; the first part is the cost of moving all these demands to their original locations (in level $i-1$), the second is the cost of moving these demands to the level i facilities in optimal's solution, and the third is moving them to the level $i+1$ facilities in optimal's solution. The last part of the cost is S_{i+1}^*, (notice that although this seems obvious – it is true because the demand decreases uniformly for all the demand points, as they are moved from level to level; the proof entails observing the path of every demand point and showing that the multiplier for this is $\prod_{j=1}^{i} \sigma_j$, which is a constant for all demand points), the second part of the cost is $\sigma_i A_i$, (by the hypothesis and the fact that demands are scaled down by σ_i) and similarly the first part is $\sigma_i S_i$. The proof follows from the fact that $\sigma_i \leq a$. ■

Theorem 9.2.2 *The above algorithm is a constant approximation for the* Simple-Placement *Problem.*

[3] The reason behind such a description is that it allows to prove bounds without looking at the actual values of the demands points created at each level, thereby eliminating considerable amount of short lived notations.

Proof: Let C be the total cost of our solution and let C^* be the optimal cost.

Note that $C = \sum_{i=1}^{k} S_i + F_i$, and $C^* = \sum_{i=1}^{k} aS_i^* + F_i^*$. From Lemma 9.2.1 and from an r approximation of the facility location problem we have,

$$\lambda F_i + S_i \leq r(\lambda F_i^* + S_i^* + \sum_{j=1}^{i-1} a^{i-j}(S_j^* + S_j))$$

Summing over all i,

$$\begin{aligned}\sum_i(\lambda F_i + S_i) &\leq r\lambda \sum_i F_i^* + r\sum_i S_i^* + r\sum_i \sum_{j=1}^{i-1} a^{i-j} S_j + r\sum_i \sum_{j=1}^{i-1} a^{i-j} S_j^* \\ &\leq r\lambda \sum_i F_i^* + r\sum_i S_i^* + \frac{ra}{1-a}\sum_i S_i + \frac{ra}{1-a}\sum_i S_i^*\end{aligned}$$

Setting $\lambda = 1 - \frac{ra}{1-a}$ we get

$$\sum_i F_i + \sum_i S_i \leq r\sum_i F_i^* + \frac{r}{1-a-ra}\sum_i S_i^*$$

Therefore we have a $\frac{r}{1-a-ra}$ approximation for the problem where each σ_i is at most a.

We are almost done, since the SIMPLE-PLACEMENT Problem is modified to an instance of bounded σ_i losing a factor $1/a$ in the distance. Therefore we have a constant factor approximation algorithm for SIMPLE-PLACEMENT. ∎

The best known value for $r = 1.728$ (Theorem 6.3.3), and thus the approximation ratio is 12.33 for SIMPLE-PLACEMENT (choosing $a = 0.17$). We can obtain an approximation factor of 8.03 using a different facility location tradeoff as illustrated below.

A Different Facility Location Tradeoff: We can use Theorem 6.2.2 to get a tradeoff result, and once again using λ scaling initially,

$$\lambda F_i + S_i \leq (1 + \ln\frac{2}{\gamma})\lambda F_i^* + (1+\gamma)(S_i^* + \sum_{j=1}^{i-1} a^{i-j}(S_j^* + S_j))$$

Simplifying the sum as in the last theorem, we can prove the following

Theorem 9.2.3 *The* SIMPLE-PLACEMENT *problem with each $\sigma_i \leq a$ has an approximation of* $\max\{1 + \ln \frac{2}{\gamma}, \frac{1+\gamma}{1-2a-a\gamma}\}$.

For the general case the approximation factor is $\max\{(1+\ln\frac{2}{\gamma}), \frac{1+\gamma}{a(1-2a-a\gamma)}\}$, which is minimized for $a = \frac{1}{2(2+\gamma)}$ and subsequently for $\gamma = 0.002$ evaluates to 8.03.

9.3 The Multi-Level Facility Location Problem

In this version, there is no reduction in demand from one layer to the next, *i.e.*, all the σ are 1. Let f_{ji} be the cost of placing a facility of layer i at location j.

We show how to reduce this problem to repeated solving of the facility location problem, providing an $O(1)$ approximation whose running time depends only polynomially on k. Note that the size of the output for this problem also depends polynomially on k. We introduce a technique for handling location dependent costs in placement problems. The next section will further generalize our algorithms to handle capacitated facilities as well. We can show the following

Theorem 9.3.1 *The Multi-Level facility location problem has a $9.2(1+\epsilon)$-approximation running in time polynomial in the size of the input, k, and $\frac{1}{\epsilon}$.*

Proof: We will show that we can construct a new problem instance which is just the capacitated facility location problem with the freedom to place multiple facilities(caches) at a location, with just a constant factor loss in cost.

In the new instance, let $\langle v, y\rangle$ denote a cache of capacity y at location v. Noticed we are not using the third parameter since all $\sigma_i = 1$ except the level k facilities (caches).

Let us consider the cost of cache $\langle u, y\rangle$, we will compute these for all v, all y in powers of $1 + \epsilon$ in the new instance.

1. Construct a labeled directed graph with $K+1$ copies of the original graph. Let the copies be $G(0)$ to $G(K)$. From a node v in $G(i-1)$ (for $1 \leq i \leq j$) draw an arc of cost $y \cdot c_{vw} + f_{wi}$ to w in $G(i)$.

2. For each node v find the shortest path from the copy of v in $G(0)$ to the copy of v in $G(K)$. Denote the cost of this path by R_v^y.

3. The cost of $\langle u, y \rangle$ is set to be R_u^y.

We can convince ourselves that a solution of this new instance can be achieved in the older instance with at most the same cost. Consider a location v with cache in the new instance with capacity x. If in the older instance, we first bring all this demand to location v, and then send it on the round trip denoted by R_v^x. Notice that on this round trip, the demand will be routed through locations and caches of type 1 type 2, and so forth up to type K (corresponding to older instance) before reaching v. Thus we can short circuit using triangle inequality and achieve a routing of cost at most denoted by the cost of the newer instance.

We will now show that the optimal solution of the older instance can be modified into a feasible solution at most a constant factor blowup in cost. Consider all the demand at a cache of type K in the older solution. We can short circuit the demand from points to this cache, say at v, directly. That does not change the service cost. However the cache now costs more, but by not too much. Consider the closest demand point p from this location v, if we send the entire demand at location v to p and back through the path by which demand point p is connected to v, the cost will be at most two times the service cost. Thus if the demand, say y was a power of $1+\epsilon$, then we can choose $x = y$ and R_v^x would have been at most this round trip cost plus the costs of the caches of type 1 through K, (of the original instance) along the path from p to v. However we have to account for the fact that y may not be a power of $1+\epsilon$ and we may end up choosing an x which is at most $(1+\epsilon)y$. Thus the cost of routing the demand may lose another factor of $1+\epsilon$. Thus the cost of the new solution is at most $3(1+\epsilon)$ of the original.

We now use the a constant factor approximation for capacitated facility location in [JV99] to obtain a $12(1+\epsilon)$ approximation.

However we are going to use the following two observations: the transformation shows that the optimum solution of the older instance (with facility cost F^* and service cost S^*) provides a feasible solution with the facility cost at most $(1+\epsilon)(F^*+2S^*)$

and service cost S^*. Now the second step of reducing this capacitated problem following [JV99] introduces the following transformation (see [CG99]); that a solution to the capacitated problem with facility and service costs F' and S' reduce to an uncapacitated problem with facility cost F' and service cost $F' + S'$. Using these inequalities, and the tradeoff result used in SIMPLE-PLACEMENT we can improve the factor to 9.2. ∎

9.4 The General-Placement Problem

We will now combine the ideas in the previous two sections to solve the most general case. Each cache is denoted by $\langle u, y, i\rangle$ where the location is u, capacity restriction y, and type i. Multiple caches can be placed at the same location (of identical or different types) in order to accommodate large demands.

Reduction to Small Miss Rate We show how to reduce the problem with arbitrary σ_i to the case where all σ'_i are below a. The following theorem applies:

Theorem 9.4.1 *Given an instance of the* GENERAL-PLACEMENT *problem with parameters $\sigma_1, \ldots, \sigma_k$, the problem can be reduced to a different caching problem with $k' \leq k$ types of caches and a different set of parameters $\sigma'_1, \ldots, \sigma'_{k'}$, such that each σ'_ℓ is a constant less than a where $0 < a < 1$, in time polynomial in input size and $\frac{1}{\epsilon}$, and at most a $\frac{3(1+\epsilon)}{a}$ blowup in cost.*

Proof: As before, we will identify again types 1 through k' of the caches in the new instance. That is σ'_1 will be $\prod_{i=1}^{j} \sigma_i$ for the first j when $\prod_{i=1}^{j} \sigma_i \leq a$. The parameter σ'_2 will be $\prod_{i=j+1}^{j'} \sigma_i$ for the first j' for which the quantity $\prod_{i=j+1}^{j'} \sigma_i \leq a$.

The caches in the new instance will be denoted by triples, $\langle v, y, 1\rangle'$ denoting a cache at v of type 1 in the new instance having capacity y (allowing copies to be placed simultaneously). The primed triples will indicate caches in the new instance.

Let us consider how to assign the cost of cache $\langle u, y, 1\rangle'$. We will compute these for all v, all y in powers of 2 and all types 1 through k' in the new instance. (The type $1'$ in the new instance is corresponding to types 1 through j_1 in the old instance). We will use ideas from the Multi-Level Facility Location Problem. The idea is that although the demands decrease, we can approximate the sublevels in the original problem (1 through j for example) by a Multi-Level facility location problem, appropriately modified to account for capacities.

1. Construct a labeled directed graph with $j+1$ copies of the original graph. Let the copies be $G(0)$ to $G(j)$. From a node v in $G(i-1)$ (for $1 \le i \le j$) draw an arc of cost $y \cdot c_{vw} + \min_{\langle u,x,i\rangle} \left\{ \lceil \frac{y}{x} \rceil f^x_{wi} \right\}$ to w in $G(i)$. If there are no $\langle u,x,i\rangle$, we will not introduce the arc or introduce an arc of cost inf (very large positive value). The expression reads complicated, but encodes the best way of routing demand y from v to w under restriction that w has a cache of type i, and $\langle w,x,i\rangle$ are the possible caches that can be used.

2. For each node v find the shortest path from the copy of v in $G(0)$ to the copy of v in $G(j)$. Denote the cost of this path by R^y_{vj}.

3. The cost of $\langle u,y,1\rangle'$ is set to be R^y_{uj}.

4. Likewise we set the costs $\langle u,y,2\rangle'$ etc.

As in the proofs of Theorem 9.3.1 and Theorem 9.2.1, we can show that converting the optimal solution of the old problem to a feasible instance of the new problem causes us to lose a small factor. It is not hard to see for a solution of cost $\bar{C}$ corresponds to a solution of same cost in the original solution. This is because the way the costs of the new facilities are constructed, the demand at a node where the new solution places a $\langle u,y,i\rangle'$ cache; there will be only one cache placed with demand D satisfying $y \le D(1+\epsilon) \le y(1+\epsilon)^2$. Otherwise we can choose a facility of smaller cost. The way R^y_{ui} is ascertained, we can clearly rout along the round trip path in the calculation of R^y_{ui}.

However the more interesting part is to show that there exists a feasible solution to the new problem which is bounded in terms of the cost of any feasible solution with facility cost F and service cost S. Consider the routing along the levels 1 through j

in this solution (later levels handled analogously). First we will ignore the decrease in demands, this makes the service cost S/a. (Once again, we only need to ignore decrease from levels 1 through $j-1$ whose product of σ-values is at least a.) The facility cost increases due to the excess demand flowing. However at each of these places there already is one facility built, to handle at most $1/a$ times the demand the facility cost is at most F/a. (Since $\lceil x/a \rceil \leq \lceil x \rceil / a$.)

At this point for this modified solution which is a collection of trees, for each tree we can find out the cost per unit demand along the path. (We divide the cost of the copies of the facilities in proportion to the demands served, notice it helps that the demands remains same over the levels.) Along with this add the distance from the root to the leaf node. Pick the leaf node which minimizes the sum. The cost of sending the entire demand served by the tree, from the root to this leaf node and back over this path the cost is at most the service cost plus the cost of the tree. (Summing over the two parts we added up.)

Therefore up to $(1+\epsilon)$ factors we have a feasible solution of the new problem with facility cost $(F+2S)/a$, and service cost S/a. Thus the cost of the new problem is bounded by $3\bar{C}$. ■

Solving Small Miss rate Case The algorithm is the same as in Section 9.2. We solve capacitated facility location [JV99] at each of the new levels in turn, using $\langle u, y, i\rangle'$ as the facilities. The proof of the following theorem proceeds along the same lines as the proof of Theorem 9.2.2.

Theorem 9.4.2 *Given an instance of* GENERAL-PLACEMENT *where the σ_i are bounded by a, we can approximate the problem within $O(1)$.*

Proof: Once again we multiply all facility costs by λ. Consider once again that we are in level i. Consider the solution of using optimal's level i facilities (which are integral) and a fractional assignment of the demand points. Note that the total demand has decreased by $\prod_{j=1}^{i} \sigma_i$ which is same for all demand points.

For the demands we have (at facilities F_i of our solution) first break them into the original (level 0) demand points. Now assign them to the optimum's integral facilities. *This is a fractional assignment.* But the fractional service cost can be bounded by $S_i^* + \sum_{j=1}^{i-1} a^{i-j}(S_j^* + S_j)$. Now we perform the transformation to reduce capacitated facility location to the uncapacitated case. This adds λF_i^* to the fractional assignment cost. Now the fractional version of theorem 6.2.2 gives,

$$\lambda F_i + S_i \leq \lambda F_i^*(1 + \ln \frac{2}{\gamma}) + (1+\gamma)[\lambda F_i^* + S_i^* + \sum_{j=1}^{i-1} a^{i-j}(S_j^* + S_j)]$$

Summing over all i we can bound $\lambda \sum_i F_i + \sum_i S_i$ by

$$(2 + \ln \frac{2}{\gamma} + \gamma)\lambda \sum_i F_i^* + (1+\gamma)(1 + \frac{a}{1-a}) \sum_i S_i^* + (1+\gamma)\frac{a}{1-a} \sum_i S_i$$

Setting $\lambda = 1 - (1+\gamma)\frac{a}{1-a}$ we can bound $\sum_i (F_i + S_i)$, which is the cost paid by us, to be $(2 + \ln \frac{2}{\gamma} + \gamma)$ times optimal's facility cost plus $\frac{1+\gamma}{1-(2+\gamma)a}$ times the service cost. This proves the theorem. ∎

Combining Theorem 9.4.1 and Theorem 9.4.2 gives an $O(1)$ approximation for GENERAL-PLACEMENT.

Chapter 10

Single-Sink Edge-Installation

10.1 Introduction

Consider a service provider or utility company trying to design a network to provide service. The agency can use economies of scale in designing the network, that the cost per unit traffic decreases when routing larger amounts. This is modeled by the edge-installation problem, also known as *Buy-at-Bulk Network Design* [SCRS97, AA97, AZ98, MMP00]. We are given demands at nodes in a network which have to be routed to their respective destinations using pipes of certain capacities and costs per unit length. The costs obey economies of scale, it is cheaper to buy a pipe of larger capacity than many pipes (which sum to the same capacity) of a smaller capacity. The goal is to optimize the total cost of the pipes bought while routing all the demand. Andrews and Zhang [AZ98] define a special case of this problem called the *Access Network Design* problem, where all demands need to be routed to a central core network and the costs and capacities of the pipes obey certain common constraints. They show applications of this problem in designing telephone networks. The "buy-at-bulk" constraints can also be extended to costs of nodes or vertices. The capacitated facility location in Chapter 7 is an example where the buy-at-bulk constraints are on facilities. In this chapter the constraints will be on edges. The buy-at-bulk constraints are sub-additive and therefore are usually represented by concave cost functions. We will use the term buy-at-bulk and concave costs interchangeably.

The problem of buy-at-bulk edge-installation was first defined in [SCRS97]. Awerbuch and Azar [AA97] obtain an $O(\log^2 n)$ approximation to this problem even when all demands have different sinks. Their work is based on techniques to approximate any metric by tree metrics [Bar96]. The approximation factor can be improved to $O(\log n \log\log n)$ using Bartal's result in [Bar98] and derandomized using the results of [CCGG98, CCG+98]. For *single-sink* edge-installation, Salman et al [SCRS97] show a constant approximation when there is only a single pipe type. Their method is based on the technique of balancing Steiner and shortest path trees [ABP90, KRY94]. Andrews and Zhang [AZ98] define the Access Network Design problem, which is a special case of single-sink edge-installation where the pipe costs and capacities obey certain common constraints, and provide an $O(K^2)$ approximation, where K is the number of pipe types. Guha et al [GMM00d] give a constant approximation to this problem. Meyerson et al [MMP00] provide an $O(\log n)$ approximation to single-sink edge-installation where the costs on the edges are arbitrary non-decreasing concave functions of demand. This was improved to a constant in [GMM00b].

The material in this chapter appears in [GMM00b, GMM00d].

10.1.1 Formal Definitions

We are given a network $G(V, E)$ with a length function on the edges, and a sink node $s \in V$. We have a set $R \subseteq V$ of demand points which we have to route to the sink by buying pipes and laying them along the edges in the network. We are given K types of connections (pipes) each with a fixed cost and a capacity. The cost of placing a pipe of fixed cost σ_k along a path of length L will be $\sigma_k L$. We are given a set S_1 of demand nodes and a single sink s. Each demand node $v \in S_1$ needs to transport some amount of demand d_v to the sink. We are asked to buy a set of pipes as cheaply as possible so as to route all demands to the sink. We are allowed to buy multiple copies of a pipe along the same link.

We will use an alternate formulation of this problem, introduced in [AZ98]. Instead of each pipe having a capacity u_k, the pipes will have incremental cost defined by $\delta_k = \sigma_k / u_k$. This represents the per-unit-flow cost of the pipe. If we transport d units of demand along a path of length L using pipe k, we will pay a total of $L(\sigma_k + \delta_k d)$. It's not hard to see that a solution under this formulation costs at least as much as the same solution under the capacitated model, and at most twice as much as the solution under the capacitated model.

If we number the pipes in increasing order of capacity, we observe the following conditions: $\sigma_1 < \sigma_2 < \cdots < \sigma_K$, and $\delta_1 > \delta_2 > \cdots > \delta_K$. We will define $f_k(D) = \sigma_k + \delta_k D$ to be the per-unit-distance cost of routing demand D along a pipe of type k. The capacity of pipe k is $u_k = \sigma_k/\delta_k$.

The goal is to minimize the cost of buying the pipes along the edges to route all demand to the sink. Since the cost per unit demand goes down with the capacity of the pipes, there exists a near-optimum solution (within a factor of two) which always buys edges along a tree [AA97].

The Access Network Design problem is a special case of single-sink edge-installation with additional restrictions on the costs of the pipe types. The main restriction is that a type k pipe is cheaper only when it routes significant demand. Formally, the restrictions can be stated as follows:

1. For $2 \leq k \leq K$, if $d < \frac{\sigma_k}{\delta_k}$, then $d\delta_{k-1} + \sigma_{k-1} < d\delta_k + \sigma_k$. For this to make any physical sense, we would actually require $d < b\frac{\sigma_k}{\delta_k}$ for some $b < 1$. Since it will not effect our proofs, we will simplify our notation by assuming $b = 1$.

2. The smallest demand looks like the smallest pipe capacity, or more precisely, $d \cdot \delta_1 > \sigma_1$.

3. $\sum_{\kappa<k} \sigma_\kappa = O(\sigma_k)$.

Under these restrictions, [AZ98] showed that there exists a near-optimal (within a constant multiplier on the cost) solution which is a tree satisfying the following properties:

1. Each demand is routed through pipes of consecutive types, i.e. types $1, 2, \ldots, \kappa$. ($\kappa \leq k$).

2. For all pipe types k, any pipe of that type has at least $u_k = \frac{\phi_k}{\delta_k}$ amount of demand flowing through it.

We can therefore compare ourselves against the optimal solution that satisfies the above mentioned properties. We can assume every pipe type is used for routing all demands by placing self loops of increasing types at the sink, provided the self loops of type k we insert at the sink are not required to have u_k amount of demand in them.

10.2 The Access Network Design Problem

We can prove the following:

Theorem 10.2.1 *There exists a solution to the Access Network Design problem in which we only use pipe types satisfying the condition $\sigma_i = \frac{\delta_{i+1}}{\delta_i} \leq \alpha$, and in which any pipe of type i has at least u_i amount of demand flowing through it. The cost of this solution is within $\frac{2}{\alpha}$ of the original optimum which used all pipe types and which had at least u_k demand in any pipe of type k.*

Proof: Note that since we are using pipes of larger types in increasing layers, the incremental cost δ per unit of traffic keeps decreasing. In fact, we can make sure that δ goes down by a constant fraction $a < 1$ with a $\frac{1}{\alpha}$ increase in cost. The way we do this is the following:

Consider pipes of increasing types starting at type 1. Let $\sigma_i = \frac{\delta_{i+1}}{\delta_i}$. Let k' be the largest number such that $\prod_{i=1}^{k'} \sigma_i \geq \alpha$. We remove all pipe types $2, \ldots, k'+1$ and use only pipe of type 1 instead of all these pipes. We next consider pipes starting at type $k'+2$ and repeat this filtering process. This is the same as in hierarchical placement problems.

When the above is completed, we are left with a set of pipe types satisfying the following properties:

- For consecutive pipe types i and $i+1$, $\frac{\delta_{i+1}}{\delta_i} \leq \alpha$.
- If we used a pipe of type i instead of a pipe of type j, then $\delta_j > a\delta_i$ and $\phi_j > \phi_i$.

■

The problem reduces to layered facility location, just as in hierarchical placement, except that we need to meet the lower bound of u_k at each layer.

We combine the constant approximation for load balanced facility location with the hierarchical placement arguments from Section 9.2 to derive a constant factor approximation.

10.2.1 The Algorithm

Let $\sigma_k = \frac{\delta_k}{\delta_{k-1}}$. We can assume with a loss of $\frac{2}{\alpha}$ in the approximation ratio that all $\sigma_k \leq a < 1$. Our algorithm will lay pipes in increasing order of types.

Let T_k be the forest we construct in level k. The demand points in this layer is the roots of the forest T_{k-1}. Actually all these forests will be collections of star graphs. Let $s \in V$ be the sink for all the demand. It is easy to observe that multiple entry points in the core network can be all collapsed into a single node.

First, the fixed cost we incur in constructing T_k is at most the routing cost, provided the demand at each pipe (except pipes originating at s) in layer k is more than $u_k = \frac{\phi_k}{\delta_k}$. Let S_k and F_k denote our fixed and incremental cost at layer k, respectively. Let the corresponding values of these quantities in the optimal solution be S_k^* and F_k^* respectively.

We use the layered facility location technique discussed in Section 9.2 to decide the routing with layer k pipes. Lemma 9.2.1 will apply in our case because the incremental cost of routing drops by at least α from one layer to the next.

Our facility location subroutine is the load balanced facility location algorithm discussed above. The demand points are the demands sitting at the roots of the trees in T_{k-1}, and the lower bound on the facilities except s is u_{k+1}. The facility location s has no lower bound. This is because once some demand is at s, it is connected to the network. We route the demands to the facilities using pipes of type k. We now have the tree T_k. Note that our algorithm ensures that we open a facility at s, and all demand at s is routed to itself.

We can obtain a constant approximation of our routing cost in terms of the fractional optimal routing cost at layer k given the demands at level k. Let us call this fractional feasible solution Z_k. We need to find a way to upper bound this cost Z_k in terms of the actual optimal cost.

10.2.2 Analysis

Let us now consider the level k, and try to compute an upper bound of routing the demand sitting at the roots of the forests in T_{k-1} to level $k+1$ using pipes of type k so that each demand point except s in layer $k+1$ has at least u_{k+1} amount of demand. One way of sending this demand to split it and route it back to the original demand nodes at level 0 and from there along the optimal tree to level $k+1$. This gives us the required bounds as shown below. We assume that r is the approximation for load balanced facility location when the lower bound we guarantee is $\frac{L}{8}$ instead of L. Notice we have to treat the demand point s separately since once some demand is routed to it in our solution, we cannot reassign it since that cost is not accounted for. However in the fractional solution the demand may be used to satisfy lower bounds u_{k+1} for other nodes. We show that the integrality gap is constant even when the demands at special nodes are not rerouted.

Lemma 10.2.1 $Z_k \leq \left(\sum_{j=1}^{k-1}(S_{k-j} + S^*_{k-j})\alpha^j + S^*_k\right)$

Proof: Suppose we have computed the load balanced facility location solution till layer k. We have some demands sitting at the roots of the forests in T_{k-1}.

For each demand point at the roots of T_{k-1}, we split it in proportion to the demands back to layer 1 along the paths in T_{k-1} and route it back to layer $k+1$ along the optimal tree. This gives us a feasible fractional solution. Since we use layer k pipes, and since the incremental cost of pipes decreases by at least α at each layer, the cost of routing decreases by α in each layer. The proof now follows the same spirit as the proof of Lemma 9.2.1. ■

Lemma 10.2.2 $S_k + F_k \leq 9r \cdot Z_k$.

Proof: Note that Z_k is the fractional optimum incremental cost of routing the demands at layer k. We have to pay some fixed and incremental cost for routing this demand using pipes of type k. Note that had the demands at all locations in the roots of T_{k-1} been at least $\frac{u_k}{8}$, we can bound the fixed cost as 8 times the incremental cost. The only demand point that does not meet the lower bound is s, but our facility location algorithm ensures that all demand sitting at s is routed to a facility opened at s, so that we do not pay any cost for routing this demand. Our facility location algorithm routes the demand without splitting to layer $k+1$ so that each facility in that layer except s has demand at least $\frac{u_{k+1}}{8}$.

This gives an r approximation to Z_k, where r is the approximation ratio for load balanced facility location when the lower bound is $\frac{L}{8}$ instead of L. Since we have to pay fixed cost which could be a factor 8 of this incremental cost, the lemma follows. ■

Theorem 10.2.2 *The Access Network Design Problem has a constant approximation.*

Proof: From the above lemmas,

$$S_k + F_k \leq 9r\left(\sum_{j=1}^{k-1}(S_{k-j} + S^*_{k-j})\alpha^j + S^*_k\right)$$

Note that our solution has value $\sum_k(S_k+F_k)$ while the optimum has value $\sum_k(S^*_k+F^*_k)$. Also note that we have lost a factor of $\frac{2}{\alpha}$ in consolidating the pipe types so that the geometric decrease theorems apply.

Using the same arguments as before, it is easy to see that we have an approximation ratio of $\frac{18r}{a(1-a(1+9r))}$. ■

We finally note that we can remove the assumption that the smallest demand looks like the smallest pipe type.

10.2.3 Dealing with Small Demands

Access Network Design required that the smallest demand is at least the size of the smallest pipe, $u_1 = \frac{\phi_1}{\delta_1}$. We can do away with this restriction as follows. We construct a Steiner tree connecting all the demands to the sink s of minimum cost. We can get a 2 approximation to the cost of the minimum cost Steiner tree.

Let the optimum solution of the Access Network Design instance be OPT.

Let us consider the fixed cost of our Steiner tree using pipes of type 1 and compare this with the fixed cost optimum pays in its tree using the K pipe types. Clearly, we are within a factor of 2 of OPT, as we use the pipe type with least fixed cost ϕ.

We now route the demands along this tree towards s. Consider all vertices the demands d in whose upstream edge is at least u_1, but whose downstream edges do not have u_1 amount of demand. We "cut" the tree at these vertices, which we call "consolidated" vertices. Our demand is now routed along the Steiner tree using pipes of type 1 till we hit a consolidated vertex. The demands at the consolidated vertices are at least u_1. The routing cost we paid till now is at most the fixed cost, which is at most $2OPT$. We have therefore already paid at most $4OPT$ in going to the "consolidated" vertices.

Now, let us consider the cost of the optimum solution with the demands at the "consolidated" vertices. One way of routing to s is to go back to the original demand nodes using our Steiner tree, and from there to s along the optimum tree for the original instance. We therefore pay at most $5OPT$ in doing this.

It is now clear that we can obtain a constant approximation even if the demands are arbitrarily small compared to u_1.

10.3 Single-Sink Edge-Installation with Concave Costs

10.3.1 Layered Structure of Optimum Solution

First, observe that as we increase demand along an edge, there are break-points at which it becomes cheaper to use the next larger pipe type. Let γ_k be the demand for which it becomes cheaper to use a pipe of type $k+1$ compared to a pipe of type k. Assume without loss of generality that $0 = \gamma_0 < u_1 < \gamma_1 < u_2 < \gamma_2 < \cdots < u_K < \gamma_K = \infty$.

Observe now that if the demand amount is in the range $[\gamma_{i-1}, u_i]$, we can ignore the incremental cost with a factor 2 loss in cost, and the cost of the edge will just be σ_i times the length of the edge, independent of the demand. If on the other hand, the demand is in the range $[u_i, \gamma_i]$, we can ignore the fixed cost with a factor 2 loss in cost, and the cost of the edge per unit length is δ_i times the demand.

This implies that the optimum solution can be converted with a factor 2 loss in cost to a layered solution. Layer i has a Steiner forest using pipes of type i followed by a forest of shortest path trees using pipes of the same type. Each pipe in the Steiner forest has at least γ_{i-1} demand and each pipe in the shortest path forest has at least u_i amount of demand.

10.3.2 Removing Similar Pipes

Our algorithm will progressively construct partial solutions using each pipe type in turn. In order to bound the total cost, we must guarantee that pipes are very different from one another in terms of fixed and incremental costs. We will eliminate various pipe types in order to guarantee the following conditions for some positive $a < 1/2$.

1. For any $k < K$, we have $\sigma_k < a\sigma_{k+1}$.

2. For any $k < K$, we have $a\delta_k > \delta_{k+1}$.

We need to prove that we can guarantee these conditions without increasing the cost of the optimum solution by too much.

Lemma 10.3.1 *We can eliminate pipes in order to guarantee that among the remaining pipes we have $\sigma_k < a\sigma_{k+1}$ while increasing the fixed cost of the optimum solution by at most $1/\alpha$. The incremental cost of the optimum solution can only decrease.*

Proof: We find the largest pipe k such that $\sigma_k \geq \alpha\sigma_{k+1}$. We eliminate this pipe, replacing it in the optimum solution with pipe $k+1$. We renumber the pipes and repeat. Notice that if at some point some pipe type is replaced by pipes of type k, then we will always keep pipes of type k in the final solution (since every higher pipe type than k is at least α higher fixed cost). When this finishes, we will have the desired property. The original optimum solution with pipe replacements has fixed cost at most $1/\alpha$ larger since any pipe which was replaced was replaced by a pipe with at most $1/\alpha$ bigger fixed cost. The incremental cost can only decrease, since higher fixed cost implies smaller incremental cost. ■

A similar process allows us to guarantee the second condition.

Lemma 10.3.2 *We can eliminate pipes in order to guarantee that among the remaining pipes we have $a\delta_k > \delta_{k+1}$ while increasing the incremental cost of the optimum solution by at most $1/\alpha$. The fixed cost of the optimum solution can only decrease.*

Combining these two lemmas, we can guarantee the two conditions with only a constant increase in the cost of the solution.

Theorem 10.3.1 *There exists a solution which uses only the remaining pipes after elimination, and which has cost at most $1/\alpha$ times the cost of the original optimum solution.*

10.3.3 Properties of the Near-Optimum Solution

We will define b_k to be such that $f_{k+1}(b_k) = 2af_k(b_k)$. In essence, b_k is sufficient demand that it becomes considerably cheaper to use a pipe of type $k+1$ rather than a pipe of type k. We first show that $u_k \leq b_k \leq u_{k+1}$.

Lemma 10.3.3 $b_k \leq u_{k+1}$.

Proof: From the definition of b_k, we can write:

$$\sigma_{k+1} + \delta_{k+1}b_k = 2a(\sigma_k + \delta_k b_k)$$

Solving this equation for b_k yields:

$$b_k = \frac{\sigma_{k+1} - 2a\sigma_k}{2a\delta_k - \delta_{k+1}} \leq \frac{\sigma_{k+1}}{2a\delta_k - \delta_{k+1}} \leq \frac{\sigma_{k+1}}{\delta_{k+1}} = u_{k+1}$$

. ■

Lemma 10.3.4 $b_k \geq u_k$

Proof: When we have b_k flow, it is cheaper to use a pipe of type $k+1$ rather than a pipe of type k. It follows that $\sigma_{k+1} + \delta_{k+1} b_k < \sigma_k + \delta_k b_k$. Solving this for b_k, we can see that

$$b_k > \frac{\sigma_{k+1} - \sigma_k}{\delta_k - \delta_{k+1}}$$

Since $a < 1/2$, it follows that $\sigma_{k+1} > 2\sigma_k$ and we can conclude that $b_k > u_k$. ■

Lemma 10.3.5 *For any demand $D \geq b_k$, $f_{k+1}(D) \leq 2af_k(D)$.*

Proof: Suppose $D = b_k + x$ for some $x \geq 0$. Then,

$$f_{k+1}(D) = \sigma_{k+1} + \delta_{k+1}(b_k + x) = 2\alpha(\sigma_k + \delta_k b_k) + \delta_{k+1} x$$

Noting that $\delta_{k+1} \leq a\delta_k$, it immediately follows that $f_{k+1}(D) \leq 2af_k(D)$. ■

There exists a near-optimum solution which uses pipe type $k+1$ only if at least b_k demand is being routed. This solution also routes all demand which enters a node using pipes of type k out of that node using pipes of types k or $k+1$. This structural observation about a nearly (within constant factor) optimum solution will be important in our proof of the approximation ratio.

Theorem 10.3.2 *There exists a solution which uses pipe type $k+1$ on a link only if at least b_k demand is being routed across that link, and which routes all demand which entered a node using pipe k out of that node using pipes k and $k+1$. This solution pays at most $\frac{2}{a}+1$ times the optimum.*

Proof: We consider the nodes of the optimum solution tree from the bottom up. Suppose a node has flow outgoing on a pipe of type k. We can conclude that all incoming flow was on pipes of type k or less, since otherwise we could improve the optimum solution by changing one of the pipe types. Consider the flow incoming on pipes of type i in increasing order of i. Either the total flow incoming on pipes of type i is at least b_i or it is not. If it is at least b_i, then we add a pipe of length zero from this node to itself; this pipe has type $i+1$ and carries the flow which was incoming on pipes of type i. Adding this pipe does not increase the cost of the solution, since the pipe has length zero. If there is not b_i demand incoming on pipes of type i, then we add a new pipe of type i from the node to its parent which will carry all the flow which was incoming on pipes of type i. The total flow traveling from this node to its parent has not changed. We can see that the new solution constructed will have the desired properties; we must guarantee that it is within a constant of optimum. Consider an edge in the optimum tree. The original optimum placed a pipe of type k here. We may have placed an additional pipe of each type 1 through $k-1$ along this edge. The pipe of type i routes at most b_i flow. The total cost of these additional pipes is therefore at most $\sum_{i=1}^{i=k-1} f_i(b_i)$. Using Lemma 10.3.3 and the definition of b_i, we can guarantee that $f_i(b_i) = \frac{1}{2a} f_{i+1}(b_i) \leq \frac{1}{2a} f_{i+1}(u_{i+1}) = \frac{2\sigma_{i+1}}{2a}$. Substituting this into the equation, the additional pipe cost is at most $\frac{2}{2a}\sum_{i=2}^{i=k} \sigma_i$. Because each fixed cost is at most half the next higher fixed cost, we can bound this sum by $\frac{4}{2a}\sigma_k$ and the total cost of the solution has increased by at most a factor of $\frac{2}{a}+1$.

This breaks down only at the sink itself, where we may have less than b_k flow arriving on pipes of type k yet there is no "parent node" to which we can route. Suppose that this is the case, but that the total flow in the network is at least b_k (if the total flow in the network is less than b_k we can simply discard pipes of type higher than k). Since not all the flow has reached the sink on pipes of type k, it follows that somewhere there exists a pipe of type $k+1$. This means some location gathered at least b_k flow. Suppose we route downward from the sink along the graph to this location using pipes of type k. If any edge now has pipes of type k traveling in both directions, we simply remove the pipe towards the sink. We stop when we reach a node where we have more than b_k flow; this must happen eventually because we're routing towards a node with sufficient flow. We maintain the property that we have introduced at most one additional pipe of type k on top of pipes of type $k+1$ and up, and that these pipes route at most b_k flow. We have already accounted for the cost of this modification. ■

We will define C_k^* to be the total cost which this near-optimum solution pays using pipes of type k. The total cost of the solution is therefore $\sum_{k=1}^{k=K} C_k^* = C^*$.

10.3.4 The Algorithm

We will now present an algorithm for single-sink edge-installation based on the structural observations we made above. The scaling idea from the previous section measures that we can compare the cost of our solution in each layer against the respective costs of the optimum solution, similar to the analysis used in [GMM00d] for Access Network Design.

Let s denote the sink node. Our algorithm constructs forests in layers. We will illustrate the construction for layer i. Let S_i be the set of demand points we have at this stage. S_1 is the original set of demand points. Layer i will use pipe type i exclusively.

We will use the load balanced facility location problem [GMM00d] as a sub-routine below. This problem is a variant of the classical facility location problem, where we have a lower bound on the amount of demand any open facility must serve. We can approximate this to a constant factor of μr provided we relax the lower bound by factor $b = \frac{\mu-1}{\mu+1}$. Here, r is the best known approximation for facility location, and can be taken as 1.728.

Steiner Trees Construct a Steiner tree on S_i. The edge cost per unit length is σ_i. Transport the demands from S_i upwards along the tree. If on any edge, the amount of demand is larger than u_i, we "cut" the tree at that edge. This gives us a forest on S_i where each edge has at most u_i demand through it.

Consolidate Consider any root in this forest. This has at least u_i amount of demand coming to it from nodes in S_i. Let the set of demand points sending demand to some root j be S_{ij}. Pick a node at random from S_{ij} in proportion to its demand and send all the demand at j to this node.

Shortest Path Trees We solve a load balanced facility location instance on S_1 with the facility lower bound b_i on all nodes and the edge cost per unit length δ_i. If there does not exist b_i total demand, then we instead route directly to the sink. We get a forest of shortest path trees. We route our current demands along these trees to their roots.

Consolidate Consider any root in this forest. Some set of nodes from S_1 were assigned to this root, and their (original) total demand is at least bb_i. We choose one such node at random, in proportion to their original demands. We send all the demand from the root to the chosen node We set S_{i+1} to these new demand locations.

Our solution will route the demands through the forests of increasing pipe types. This solution need not be a tree, but can easily be converted to one of no greater cost.

10.3.5 Analysis

We define d_v to be the demand of node v in the original (S_1) demands. We define D_v to be the demand at node v in the current stage of the algorithm.

Let T_i^I be the incremental cost of the Steiner Tree at layer i and T_i^F be its fixed cost. The total cost of the Steiner Tree at layer i is $T_i = T_i^I + T_i^F$.

Let P_i^I be the incremental cost of the shortest path tree at layer i and P_i^F be its fixed cost. The total cost of the shortest path tree at layer i is $P_i = P_i^I + P_i^F$.

Let N_i be the total cost of the consolidation steps for layer i. The total cost of our solution is therefore $\sum_i (T_i + P_i + N_i)$.

At each layer we will construct an overall solution to the problem on the nodes S_i. Let $C_i(j)$ represent the total cost which this solution on the nodes S_i pays using pipes of type j.

Lemma 10.3.6 *At the end of any consolidation step, every node has* $\mathrm{E}[D_v] = d_v$.

Proof: We will prove this by induction on the steps i. Suppose that the statement is true at some step. We will show that it is true at the next step.

Suppose that the demand at node v after the previous consolidation step was x_v. By the induction hypothesis, $\mathrm{E}[x_v] = d_v$. There are two cases to consider; either we performed a Steiner Tree step or a Shortest Path Tree step.

Suppose we have just performed a Steiner Tree step. The current node is routed to some root with total demand D. We then choose a node for consolidation. The probability that we choose node v is x_v/D. If we choose v, demand D will be placed there; otherwise no demand can be placed there. Thus the expected amount of demand placed at v is x_v; by induction we can claim that $\mathrm{E}[D_v] = \mathrm{E}[x_v] = d_v$ as desired.

Suppose we have just performed a Shortest Path Tree step. The current node is routed to some root which would have demand D if the demands were as in the S_1 stage. The probability we consolidate to v is d_v/D; if we do, the total demand at v will be the total of the current demands of all the nodes routed to this root. Let V be the set of nodes routed to v. $\mathrm{E}[D_v]$ is therefore $\mathrm{E}[\frac{d_v}{D}\sum_{w\in V} x_w]$. Observe now that:

$$\mathrm{E}[\sum_{w\in V} x_w] = \sum_{w\in V} \mathrm{E}[x_w] = \sum_{w\in V} d_w = D$$

Thus $\mathrm{E}[D_v] = d_v$. ■

Lemma 10.3.7 $\mathrm{E}[N_i] \leq T_i + P_i$.

Proof: The consolidation step following a tree construction always has expected cost at most the cost of the tree construction. ■

Lemma 10.3.8 $\mathrm{E}[P_i^I] \leq \mu r \sum_{j=1}^{j=i} a^{i-j} C_j^*$.

Proof: Suppose the demands at the sources were those from S_1. Then one possible solution would be the optimum problem solution up until pipes of type $i+1$ were used. We know that the optimum solution must gather the desired b_i flow before using pipes of type $i+1$. It follows that we can find a solution with cost at most μr times the incremental cost of the optimum using pipes of type 1 through i. Since we will always pay the incremental cost δ_i, and the incremental costs scale by a, we can guarantee a total cost of at most $\sum_{j=1}^{j=i} a^{i-j} C_j^*$ for this solution. Our actual demand at each node has expected value equal to the original demand, so the expected value of P_i^I is bounded as above. ■

Lemma 10.3.9 $P_i^F \leq P_i^I$.

Proof: The Steiner Tree stage guarantees at least u_k demand or zero everywhere. If an edge has zero demand flowing on it, we will pay zero for that edge. Otherwise there is at least u_k demand on the edge and we pay an incremental cost which exceeds the fixed cost. ■

Lemma 10.3.10 *Let D_v be the demand at $v \in S_i$. Then,* $\mathrm{E}[D_v] \geq bb_{i-1}$.

Proof: We obtain the nodes S_i by solving a load balanced facility location instance on S_1 with lower bounds b_{i-1}. In this solution, each node in S_i except s has demand at least bb_{i-1}. Consider any node w in S_1, and suppose that the demand our solution so far has there is x_w. Then, $\mathrm{E}[x_w] = d_w$. Let V be the set of nodes which get routed to v. Then, $\mathrm{E}[D_v] = \mathrm{E}[\sum_{w \in V} x_w] = \sum_{w \in V} \mathrm{E}[x_w] = \sum_{w \in V} d_w \geq bb_{i-1}$. ■

Lemma 10.3.11 *At stage i we can construct a solution which uses only pipes i and higher. This solution has cost $C_i(j)$ using pipes of type j, where $\mathrm{E}[C_i(j)] \leq C_j^*$ for $j > i$, $C_i(j) = 0$ for $j < i$, and $\mathrm{E}[C_i(i)] \leq \sum_{j=1}^{j=i} \frac{1}{b}(2a)^{i-j} C_j^*$.*

Proof: For $i = 1$ we use the near-optimum solution itself and the claim follows immediately.

Consider stage i. If we use the pipes as in the near-optimum solution, our expected cost using each pipe type j would be equal to C_j^*. For each pipe of type $j < i$, we remove the pipe if the total demand flowing across it is zero. Otherwise we replace the pipe with a pipe of type i. The cost of this replacement pipe is $f_i(D)$

where D is the demand flowing across it. Given that the demand is nonzero, the node must lie along the path from one of the chosen consolidation nodes from the previous stage. Each of these nodes has expected demand at least bb_{i-1}. It follows that $\mathrm{E}[f_i(D)] \leq \frac{1}{b}(2a)^{i-j} f_j(D)$. We can therefore bound the cost of this modified solution using pipes of type i by an expected $\sum_{j=1}^{j=i} \frac{1}{b}(2a)^{i-j} C_j^*$. ■

Lemma 10.3.12 $\mathrm{E}[T_i^F] \leq 2\sum_{j=i+1}^{j=K} a^{j-i} C_j^* + 2\sum_{j=1}^{j=i} \frac{1}{b}(2a)^{i-j} C_j^*$.

Proof: The solution given in Lemma 10.3.11 is one possible Steiner tree. The fixed cost of this Steiner tree is bounded by the following expected cost:

$$\sum_{j=i+1}^{j=K} a^{j-i} C_j^* + \sum_{j=1}^{j=i} \frac{1}{b}(2a)^{i-j} C_j^*$$

This holds because the cost on a pipe of type $i+k$ will be reduced by a^k since we pay only for a pipe of type i. We can find a Steiner Tree of at most twice this cost, so the claim follows. ■

Lemma 10.3.13 $T_i^I \leq T_i^F$.

Proof: Since we cut the tree at any edge with more than u_k demand along it, we guarantee that the fixed cost paid on any edge we actually use exceeds the incremental cost. ■

Theorem 10.3.3 *There is a constant-approximation for single-sink edge-installation.*

Proof: The total cost of our solution is bounded by $\sum_i 2(2T_i^F + 2P_i^I)$. Using Lemmas 10.3.8 and 10.3.12, we conclude that the expected cost of our solution is bounded by the following:

$$4\sum_i (2\sum_{j=i+1}^{j=K} a^{j-i} C_j^* + 2\sum_{j=1}^{j=i} \frac{1}{b}(2a)^{i-j} C_j^* + \mu r \sum_{j=1}^{j=i} a^{i-j} C_j^*)$$

By reversing orders of summation, we can bound this by:

$$4(\frac{2}{1-a} + \frac{2}{b(1-2a)} + \frac{\mu r}{1-a})C^*$$

This is our approximation against the near-optimum solution. Using Theorem 10.3.2 allows us to bound our overall approximation ratio by:

$$(\frac{4}{a})(1 + \frac{2}{a})(\frac{2}{1-a} + \frac{2}{b(1-2a)} + \frac{\mu r}{1-a})$$

■

Chapter 11

Clustering Data Streams

11.1 Introduction

A data stream is an ordered sequence of points that can be read only once or a small number of times. Formally, the data stream model is defined to be a sequence of points $x_1, \ldots, x_i, \ldots, x_n$ such that the data is read in increasing order of the indices i. Along with usual performance measures, a data stream algorithm is also measured by the number of passes the algorithm must make over the data stream and the amount of memory required. In this chapter we give constant factor approximation algorithms for the k-median problem that naturally fit into this data stream setting.

Clustering has been widely studied recently across several disciplines. However only a few of the techniques developed scale to support clustering applications of very large datasets. One useful e-commerce application of such large datasets is the clustering of click streams to discover segments in the customer population. Other applications include clustering web pages, multimedia data, retail chain transactions, telecommunication phone records to discover fraud, etc. Further, such data is typically stored in secondary storage devices and access is costly. This leads us to consider developing scalable clustering algorithms that run in linear time, revisit the data as little as possible, and consume small space. These concerns can be summarized in the problem of clustering data streams.

One of the first results in data streams was the result of [MP80] where the space requirement of selection and sorting as a function of the number of passes over the data was studied. It showed that $O(n^{\frac{1}{p}})$ memory was both necessary and sufficient (upto logarithmic terms) for an algorithm allowed at most p passes for selection. The model was formalized in [HRR98] and several algorithms and complexity results related to several graph theoretic problems and their applications. It addressed tradeoff questions relating to number of passes, randomness, and approximation. Other papers with results in the data stream model include: a small space approximation of the number of distinct elements in a data stream in [FM83]; approximate selection in a single pass over the data in [MRL98, MRL99] and space efficient L^1 difference between two data streams in [FKSV99].

[CCFM97] gave a constant factor algorithm for incremental k-center problem, which is also a single pass algorithm using $O(nk \log k)$ time and $O(k)$ space. However there is a large difference between the k-center and the k-median problem, in construction of good lower bounds. A set of $k+1$ suitably separated points provide a lower bound for the k-center problem. These points can be thought of as a proof of goodness of the clustering. In the k-median problem no such succinct proofs exist, and the optimization problem has much more global character.

We first show separability theorems in Section 11.2 for the k-median problem that enables a divide-and-conquer algorithm. This itself reduces the number of passes required in the sampling based algorithm to 2. Using this divide-and-conquer paradigm we show in Section 11.3 how to achieve a constant factor algorithm in time $O(n^{1+\epsilon})$ time and $O(n^{\epsilon})$ memory and a single pass. Next in Section 11.3.1 using randomization we show how to reduce the running time to $O(nk)$ while reducing the number of passes to one. In Section 11.4 we discuss why it is not possible to obtain any bounded approximation ratio in deterministic $\Omega(nk)$ time A natural question arises is what approximation can we achieve in deterministic $O(nk)$ algorithm. We show how in Section 11.4 how to achieve a poly-$\log n$ approximation ratio in a single pass in deterministic $O(nk)$ time.

The material in this chapter appears in [GMMO00].

11.2 Clustering in Small Space

One of the first requisites of clustering a data stream is that the computation be carried out in small space. Our goal will be first to show that clustering can be carried out in small (n^ϵ for n data points) space, without being concerned with the number of passes. Subsequently we will see how to implement the algorithm in one pass. We will begin by showing a reduction theorem for the k-median problem. This reduction carries over to other clustering metrics such as the sum of squared distances and so on.

Recall that in the k-median problem we are given a set of n points in a metric space with distance function c_{ij}, and point i has weight d_i. The objective is to identify k medians, such that every point i is assigned to its closest median, such the weighted sum of the assignment distance is minimized.

11.2.1 Separability Theorems

We will start by the following theorem on relating the optimum solution of the k-median to a possibly infeasible solution, where the medians need not belong to the set of points. These separability theorems follow from the proofs of the generalized exchange procedure in Section 2.3. In the generalized exchange we demonstrated that there exists a solution, using the facilities of the unknown feasible solution, of assignment cost bounded by the linear combination of the assignment costs of our solution and the feasible solution. In the following discussion, we will use the observation that the feasible solution uses only k facilities (or medians). We will present this idea in a slightly different way for ease of subsequent use.

Theorem 11.2.1 *Given an instance of the k-median problem and a solution of cost C, where the medians may not belong to the set of input points, there exists a solution of cost $2C$ where all the medians belong to the set of input points.*

Proof: Consider the solution of cost C, and let the points $j_1, \ldots, j_q$ be assigned to median i. Since median i may not be in the input, consider the point j_l which is closest to i as the median (instead of i). Thus the assignment distance of every point

j_r doubles, since $c_{j_r j_l}$ can be bound by $c_{j_l i} + c_{j_r i}$. Thus over all the n points in the original set, the assignment distance can at most double, summing to at most $2C$. ■

Now consider the following theorem on partitioning th data set,

Theorem 11.2.2 *Given a set of points, and an arbitrary partitioning of it in disjoint sets $\chi_1, \ldots, \chi_\ell$, the sum of the optimum solution values for the k-median problem on these ℓ sets of points is at most twice the cost of the optimum k-median problem solution for the entire n points.* [1]

Proof: Consider the medians used for the optimum k-median solution. If each partition uses these medians, the cost of the solution will be exactly the cost of the optimal solution. This follows since the objective function for k-median is the sum of distances to the nearest median for every point. However the set of medians chosen by the optimum solution need not be present in a partition. But in the case where the medians points can be arbitrary points in the space, the above theorem is proved.

In case we have to chose the medians from the given set of points, the k-medians used by the optimum solution will not be available to every partition. In this case use Theorem 11.2.1 to construct a solution which is at most 2 times the cost of the optimum solution. ■

This theorem sets the stage for a divide-and-conquer algorithm for the k-median problem. However we would have to come up with a strategy for the conquer step after we solve the k-median problem on these sets. A natural idea is to construct an instance where all the points which have the median i shift their weight to the point i in the new instance. Since we are changing the instance, we must show that the newly created instance has at least a good feasible solution. Notice that the set of points in the new instance is much smaller and may not even contain the medians of the optimum solution.

---

[1] *The factor 2 is avoided in the Euclidean case, if we allow that medians can be arbitrary points in space, rather than requiring that they be points from the original data set.*

Theorem 11.2.3 *If the cost of the solution we got by partitioning the point set is C, and if C^* is the cost of the optimum solution to the k-median problem for the entire set, then there exists a solution of cost $2(C + C^*)$ to this new instance.* [2]

Proof: As in the proof of the previous theorem, we will consider the k medians in the optimum solution.

Each point i' in the new instance can be thought of a collection of points. Consider a point i whose weight d_i was shifted to point p' in the new instance. In that case, i' was the median to which i was assigned to and we can consider the point i to be located at i' in the new instance. Thus the objective function can be bound by summing over i, the assignment distance of i' weighted by d_i.

The median to which i was assigned to in the optimal solution be $\sigma(i)$. Thus i' has a median at distance $c_{i'\sigma(i)}$. Thus the contribution of i in the new instance is $d_i c_{i'\sigma(i)}$. This is bound by $d_i c_{ii'} + d_i c_{i\sigma(i)}$. The first term, summed over all points i, evaluates to C and the second term evaluates to C^*.

Thus we showed an assignment to the medians of the optimal solution at cost $C + C^*$. Using Theorem 11.2.1, the theorem follows. ■

11.2.2 Composing Two Algorithms: A Black Box Reduction

In this subsection we will present a reduction technique that will solve the k-median problem in the following two steps. We define the algorithm LEVEL(2):

Algorithm LEVEL(2) In the first stage it will use a clustering algorithm that outputs more than k medians, but significantly fewer than n, the original number of data points, such that the sum of distances of the original points to the nearest of these medians will at most be a small factor off the optimal k-median solution of the entire dataset. Essentially, this will be a bicriteria approximation denoted by (a, b) where we use ak medians at most and the cost of the solution is at most b times the

[2] *Again, the factor 2 is avoided if we use the Euclidean distance and allow medians to be arbitrary points.*

optimal k-median solution. Notice that we could also have $a = 1$. Using local search techniques we can make b arbitrarily close to 1; we will discuss this more later.

In the second stage, we will output the solution to a k-median instance defined on the ak points obtained from the first stage. The weight of each such point found will be the sum of the weights of the points in the original dataset assigned to it in the first stage.

The reduction will show that if we have a c-approximation algorithm for the second stage, the final approximation factor can be bounded. We can prove the following,

Theorem 11.2.4 *The algorithm* LEVEL(2) *has an approximation factor of* $2c(1 + b) + b$.

Proof: Let the optimal k-median solution be of cost C^*. Then the cost of the solution C at the end of the first stage is at most bC^*. Thus by Theorem 11.2.3, there exists a solution to the k-median problem on the modified instance of cost $2(C + C^*)$. Since we have a c-approximation, we have a solution of cost $2c(1 + b)C^*$ to the modified instance.

Consider the following thought experiment: if point i was assigned to median i' in the first solution, assume the point i is located i' in a new instance. Thus instead of summing over i', we can account the cost of the solution by summing over i the assignment distance of i' weighted by d_i. If the assignment distance of i' is $c_{i'\sigma(i')}$ in our solution, $\sum_i d_i c_{i'\sigma(i')}$ is $2c(1 + b)C^*$.

If we consider a solution to the original instance where i is assigned to $\sigma(i')$, the contribution of i is at most d_i times $c_{i\sigma(i')}$. The distance $c_{i\sigma(i')}$ is bounded by $c_{ii'} + c_{i'\sigma(i')}$. Thus the total contribution can be bounded by $\sum_i d_i(c_{ii'} + c_{i'\sigma(i')})$. Now $\sum_i d_i c_{ii'}$ is C, and $\sum_i d_i c_{i'\sigma(i')}$ is $2c(1 + b)C^*$, which proves the theorem. ■

It appears that we have undertaken a fruitless endeavor, since we had constant-factor approximation algorithms for k-median to begin with. But if we used an $f(n)$-space algorithm, with a 2-level scheme each phase can be made to cluster $\sqrt{n}$ points, and thus the space required can be made $f(\sqrt{n})$. The black-box nature of the algorithm will now allow us to devise a divide-and-conquer strategy based algorithm.

11.2.3 Divide-and-Conquer Strategy

We will generalize the algorithm LEVEL(2) informally described in "base case" is the algorithm LEVEL(2) described above, which operates on a single partition, clusters its input points down to ak points, and further reduces these to k.

Algorithm LEVEL($\ell + 1$)

Partition the input into disjoint subsets.

Cluster each subset to ak intermediate medians using the bicriterion algorithm.

Collect all intermediate medians.

Apply LEVEL(ℓ) on the intermediate medians.

We can claim the following theorem,

Theorem 11.2.5 *Algorithm LEVEL(ℓ) for constant ℓ, gives a constant factor approximation for the k-median problem.*

Proof: Assume that the approximation factor for the ℓth level is A_ℓ. From Theorem 11.2.2 we know that the cost of the solution of the first level is $2b$ times optimal. From Theorem 11.2.4 we get that the approximation factor A_j would satisfy a simple recurrence,

$$A_\ell = 2A_{\ell-1}(2b+1) + 2b$$

The solution of the recurrence is $c_0 \cdot (2(2b+1))^\ell$ for some constant C_0. This is $O(1)$ given ℓ is a constant. ■

We need to store the intermediate medians using up space, and this seems to limit the number of subsets into which we partition the original data set. If we partition it into more than $\sqrt{n/k}$ subsets, then since each generates at least k points we will require more than $\sqrt{nk}$ space. Thus it seems that we require $\sqrt{nk}$ space at least.

In the next section we will see a way to get around this problem. We will implement the hierarchical scheme more carefully and obtain a clustering algorithm for data streams. We have to do away with the storage of the intermediate medians, and thus design a more interesting recursive algorithm.

11.3 The Data Stream Model

Under the Data Stream Model computation takes place within bounded space M, and a data point can be seen only once in a scan, and points must be viewed in order.

In this section we will modify the multi-level algorithm to operate on data streams. We will present a one-pass, $O(1)$-approximation in this model assuming that the bounded memory M is not too small, more specifically n^ϵ if n denotes the number of records in the stream.

This model and the analysis have similarities to incremental clustering and online models. However our approach will be a bit different. We will maintain a forest of assignments. We will complete this to k trees, and all the nodes in a tree will be assigned to the median denoted by the root of the tree. Thus we allow ourselves to change the clustering completely on seeing a new point. This is different compared to online model, where we do not have restriction on space. If we allow ourselves to change the clustering for an online model, we can output the best approximation for every prefix of the point set seen, storing all of them. Mettu and Plaxton [MP00] considered a different formulation, where we can store all the points, but output a list of medians such that the first p points of this list is an $O(1)$-approximation for the p-median problem, for all p.

First we will show how solve the problem of storing intermediate medians. Next we will inspect the space requirements and running time.

Data Stream Algorithm To achieve this, we will modify our multi-level algorithm slightly. The algorithm will be the following:

1. Input the first m points; use a bicriterion algorithm to reduce these to $O(k)$ (say $2k$) points. As usual, the weight of each intermediate median is the number of points assigned to it in the bicriterion clustering. (Assume m is a multiple of $2k$.)

2. Repeat the above till we have seen $m^2/(2k)$ of the original data points. At this point we have m intermediate medians. We will term these as first-level medians.

3. Cluster these m first-level medians into $2k$ second-level medians and proceed.

4. In general, maintain at most m level-i medians, and, on seeing m level-i medians, generate $2k$ level-$i+1$ medians, with the weight of a new median as the sum of the weights of the level-i medians assigned to it.

5. When we have seen all the original data points (or we want to have a clustering of the points we have seen so far) we cluster all the intermediate medians into k final medians.

This is a fairly simple and one of the most natural ways to cluster the data. It is easy to observe that this algorithm is identical to the multi-level algorithm described before.

The number of levels required by this algorithm is at most $O(\log(n/m)/\log(m/k))$. If we have $k \ll m$ and $m = O(n^\epsilon)$ for some constant $\epsilon < 1$, we have an $O(1)$-approximation. Using linear programming or primal dual algorithms we will have $m = \sqrt{M}$ where M is the memory size (ignoring factors due to maintaining intermediate medians of different levels). We argued that the number of levels would be a constant when $m = n^\epsilon$ (and hence when $M = n^{2\epsilon}$) for some $\epsilon < 1/2$).

Linear Space Clustering The approximation quality which we can prove (and intuitively the actual quality of clustering obtained on an instance) will depend on the number of levels we have. From this perspective it is profitable to use a space-efficient algorithm.

We can use the local search algorithm in Section 2.5 to provide a bicriterion approximation in *space linear in* m, the number of points clustered at a time. The advantage of this algorithm is that it maintains only an assignment and therefore uses linear space. However the complication is that for this algorithm to achieve a bounded bicriterion approximation, we need to set a "cost" to each median used, so that we penalize if many more than k medians are used. The algorithm solves a facility location problem after setting the cost of each median to be used. However this can be done by guessing this cost in powers of $(1+\gamma)$ for some small γ and choosing the best solution with at most $k + 2k/\gamma$ medians. In the last step, to get k medians we would have to use the primal-dual algorithm. This would allow us to cluster with $m = M$ points at a time provided $k \le M \log n / \log M$.

The Running Time The running time of this clustering algorithm is dominated by the contribution from the first level. The local search algorithm is quadratic and the total running time is $O(n^{1+\epsilon})$ where $M = n^{\epsilon}$. We argued before, however, that ϵ will not be very small and hence the approximation quality which we can prove will remain small.

We therefore claim the following theorem

Theorem 11.3.1 *We can solve the k-median problem on a data stream with time $O(n^{1+\epsilon})$ and space $\theta(n^{\epsilon})$ up to a factor $2^{O(\frac{1}{\epsilon})}$.*

We have two avenues to pursue. The running time will be lower-bounded by the space we require, and we improve this bottleneck to get linear space clustering, but first, to achieve scalability, our goal will be to get clustering in time $\tilde{O}(nk)$. This will mean an amortized update of $O(k\ polylog(n))$. In the next section we will motivate how to achieve this, and provide evidence that ours is a hard bound for the running time of a clustering algorithm.

The second issue is to present an algorithm with approximation guarantee which is polynomial in $\frac{1}{\epsilon}$. We will show how to achieve this in Section 11.4.

11.3.1 $\tilde{O}(nk)$ Time Clustering of Data Streams

Let us recall the algorithm we have developed so far. We have $k^2 \ll M$, and we are applying an alternate implementation of a multi-level algorithm.

We are clustering $m = O(M)$ (assuming $M = O(n^{\epsilon})$ for constant $\epsilon > 0$) points and storing $2k$ medians to "compress" the description of these data points. We use the local search-based algorithm (See Section 2.5). We keep repeating this procedure till we see m of these descriptors or intermediate medians and compress them further into $2k$. Finally, when we are required to output a clustering, we compress all the intermediate medians (over all the levels there will be at most $O(M)$ of them) and get $O(k)$ penultimate medians which we cluster into exactly k using the primal dual algorithm, in Chapter 4.

11.3.2 Clustering in Sub-Quadratic Time

We will use the results in [Ind99] on metric space algorithms that are sub-quadratic. The algorithm as defined will consist of two passes and will have constant probability of success. For high probability results, the algorithm will make $O(\log n)$ passes. As stated, the algorithm will only work if the original data points are *unweighted*. Consider the following algorithm:

1. Draw a sample of size $s = \sqrt{nk}$.
2. Find k medians from these s points using the primal dual algorithm.
3. Assign each of the n original points to its closest median.
4. Collect the n/s points with the largest assignment distance.
5. Find k medians from among these n/s points.
6. We have at this point $2k$ medians.

Theorem 11.3.2 *[Ind99] The above algorithm gives an $O(1)$ approximation with $2k$ medians with constant probability.*

The above algorithm [3] provides a constant-factor approximation for the k-median problem (using $2k$ medians) with constant probability. The above experiment is repeated $O(\log n)$ times for high probability. We will not run this algorithm by itself, but as a sub-step in our algorithm. The algorithm requires $\tilde{O}(nk)$ time and space. Using this algorithm with the local search tradeoff results in Section 6.2.1 reduces the space requirement to $O(\sqrt{nk})$.

[3] The algorithm presented here, without the last step, is essentially the same as in [Ind99], however the primal dual algorithm which requires $O(n^2)$ time to solve k-median problem was not known when the result was published. The result proved therein was using $O(n^2k^2)$ local search algorithm in [KPR98] which was a bicriterion approximation.

11.3.3 Extending to Weighted Input

We need this sampling-based algorithm to work on weighted input. It is necessary to draw a random sample based on the weights of the points; otherwise the medians with respect to the sample do not convey much information. The simple idea of sampling points with respect to their weights does not help. The philosophy of the above method is that a random sample will be reasonable for most points, that there will not be many outliers (at most n divided by the sample size, upto constants), and that in the second phase it is sufficient to account for these outliers.

If the points have weights, in the first step we may only eliminate k points. Therefore sampling according to weights does not carry through. Contrast this with the algorithm in [FKV98] where the points were in Euclidean space and the measure was sum of squares of distances. Both these facts were crucial for their algorithm.

We suggest the following modification. The basic idea is scaling. We can round the weights to the nearest power of $(1+\epsilon)$ for $\epsilon > 0$. In each group we can ignore the weight and lose a $(1+\epsilon)$ factor. Since we have an $\tilde{O}(nk)$ algorithm, summing over all groups, the running time is still $\tilde{O}(nk)$. The correct way to implement this is to compute the exponent values of the weights and use only those groups which exist, otherwise the running time will depend on the largest weight.

The Full Algorithm We will use this sampling-based scheme to develop a one-pass and $O(nk)$-time algorithm that requires only $O(n^{\epsilon})$ space.

- Input the first $O(M/k)$ points, and use the randomized algorithm above to cluster this to $2k$ intermediate median points.
- Use a local search algorithm to cluster $O(M)$ intermediate medians of level i to $2k$ medians of level $i+1$.
- Cluster the final $O(k)$ medians to k medians.

Notice that the algorithm remains one pass, since the $O(\log n)$ iterations of the randomized sub-algorithm just add to the running time. Thus, over the first phase, the contribution to the running time is $\tilde{O}(nk)$. Over the next level, we have $\frac{nk}{M}$ points, and if we cluster $O(M)$ of these at a time taking $O(M^2)$ time, the total time for the second phase is $O(nk)$ again. The contribution from the rest of the levels decreases geometrically, so the running time is $\tilde{O}(nk)$. As shown in the previous sections, the number of levels in this algorithm is $O(\log_{\frac{M}{k}} n)$, and so we have a constant factor approximation for $k \ll M = \theta(n^\epsilon)$ for some small ϵ. [4]

Thus we claim the following theorem,

Theorem 11.3.3 *The k-median problem has a constant-factor approximation algorithm running in time $O(nk \log n)$, in one pass over the data set, using n^ϵ memory, for small k.*

11.3.4 Deterministic Algorithms in $\tilde{O}(nk)$ time

One natural question we can ask ourselves is what we can achieve if we do not have randomization. We already have seen how to get an $O(n^{1+\epsilon})$-time clustering algorithm in n^ϵ space which gives a constant factor approximation. However this constant factor grows as $2^{\frac{1}{\epsilon}}$ and if we were to ask for an $\tilde{O}(nk)$-time algorithm we would have an approximation factor polynomial in (n/k). Modifying our approach slightly, we can show the following,

Theorem 11.3.4 *In $\tilde{O}(nk)$ deterministic time, we have a poly-$\log n$ approximation for the k median problem in n^ϵ space and a single pass.*

Proof: First we will have to construct an algorithm that runs in time $\tilde{O}(nk)$. Then we can reduce the space required in the same way as for the previously described randomized algorithm.

Consider the primal-dual algorithm which gives a constant factor (say c) approximation for the k-median problem. This algorithm takes time (and space) an^2 some constant a. (We will have to evaluate the constants in the running time, as we will shortly see that these will blow up.) Now consider the algorithm that chooses p_1 partitions of equal size and applies the algorithm to these partitions repeatedly, and

[4] We could have used the sampling-based algorithm in the intermediate steps as well, however such a recursive, sampling-based algorithm will have greater errors, in theory and very likely in practice.

then to the nk/p_1 points so obtained, to get k points. If we choose $p_1 = (n/k)^{\frac{2}{3}}$, the running time of the algorithm is $2an^{\frac{4}{3}}k^{\frac{2}{3}}$. Call this algorithm A_1. The space required by this algorithm is $2an^{\frac{4}{3}}k^{\frac{2}{3}}$ also. However by Theorem 11.2.4 we have an approximation of $2c^2 + 3c$.

Now define A_2 to partition the dataset in p_2 partitions and apply A_1 on all of them and the resulting intermediate medians (notice we can easily ensure an implementation to get a one-pass algorithm). Solving to minimize the running time will yield $p_2 = (n/k)^{4/5}$. Therefore the running time and space required both become $4an^{\frac{16}{15}}k^{\frac{14}{16}}$.

If we carry this process on that A_i calls A_{i-1} on p_i partitions, without much difficulty we can prove that the running time and the space required by the algorithm will be $a2^in^{\left(1+\frac{1}{2^{2^{i-1}}-1}\right)}k^{\left(1-\frac{1}{2^{2^{i-1}}-1}\right)}$. However the approximation factor c_j grows as $c_j = 2c_{j-1}^2 + c_{j-1}$.

To get the exponent of n in the running time to be 1, it is sufficient to have $i = \theta(\log\log\log n)$. This makes the running time nk (hiding poly $\log\log n$ factors) and gives approximation $O(\log^p n)$ since the approximation factor is 4^{2^i}. Thus we have a poly-$\log n$ approximation in $\tilde{O}(nk)$ space and time. Now we can use this in our previous algorithm to get $O(\log^p n)$ approximation in n^ϵ space and $\tilde{O}(nk)$ time, without using randomization. ■

The above actually shows that we have an $O(n^{1+\epsilon})$-time clustering with approximation guarantee polynomial in $\frac{1}{\epsilon}$. Combining this with Theorem 11.3.1 we get the following,

Theorem 11.3.5 *The k-median problem can be approximated in time $\tilde{O}(n^{1+\epsilon\delta})$ and space $\theta(n^\delta)$ upto a factor of $O(poly(\frac{1}{\epsilon})2^{\frac{1}{\delta}})$.*

.

11.4 Time Lower Bounds

We worked fairly hard to get a clustering algorithm that requires time $O(nk)$, ignoring the log factors. It is easy to see that this is also a lower bound in some sense, to get any approximation factor.

The lower bound result can be motivated by the following example. Assume that $k = n - 1$ (we have used $k \ll n$ as the motivating factor, but let us assume this for the sake of the example) and that all pairs of points, except one, have distance 1 from each other, and that this other pair has distance 0. For many clustering measures like k-center and k-median, to get the optimal solution, for which the objective function has value 0, we will have to discover the endpoints of the edge with 0 distance. Any other assignment will have objective function value 1, so no approximation factor is possible in $o(n^2)$ or $o(nk)$ time in general metrics. Although this works for randomized case as well, k has to be large ($\theta(n)$) for this lower bound to hold.

We will extend this example in case of $k \ll n$. We will partition the graph into k cliques. A pair of points in the same clique has distance 0. Any other edge will have distance 1. Again the assignment function will have to discover all the cliques, to have objective function value 0.

Theorem 11.4.1 *[KLMR98] Any deterministic clustering algorithm where the objective function is a linear combination distances, must make $\Omega(nk)$ probes to the distance function to achieve a bounded approximation.*

Chapter 12

Hierarchical Agglomerative Clustering

12.1 Introduction

Existing clustering algorithms can be broadly classified into *partitional* and *hierarchical* [JD88]. Partitional clustering algorithms attempt to determine k partitions that optimize a certain criterion function. The k–Median problem, the k–Center problem, the minsum clustering problem, the generalized clustering are all examples of such. In this chapter we will discuss the hierarchical approach which is quite different.

A hierarchical clustering is a sequence of partitions in which each partition is nested into the next partition in the sequence. An *agglomerative* algorithm for hierarchical clustering starts with the disjoint set of clusters, which places each input data point in an individual cluster. Pairs of items or clusters are then successively merged until the number of clusters reduces to k. At each step, the pair of clusters merged are the ones between which the distance is the minimum. This is one of the most popular algorithms used in information retrieval, biology, medicine, image processing. The biggest advantage of the hierarchical clustering is the resulting hierarchy trees of clusters or dendograms. This allows for exploration of particular clusters of interest. This is specially useful in refining queries and search. See

[Ras92, vR79, Sal89, SM83, Wil88, WMB94] for applications of hierarchical agglomerative clustering in information retrieval and image processing.

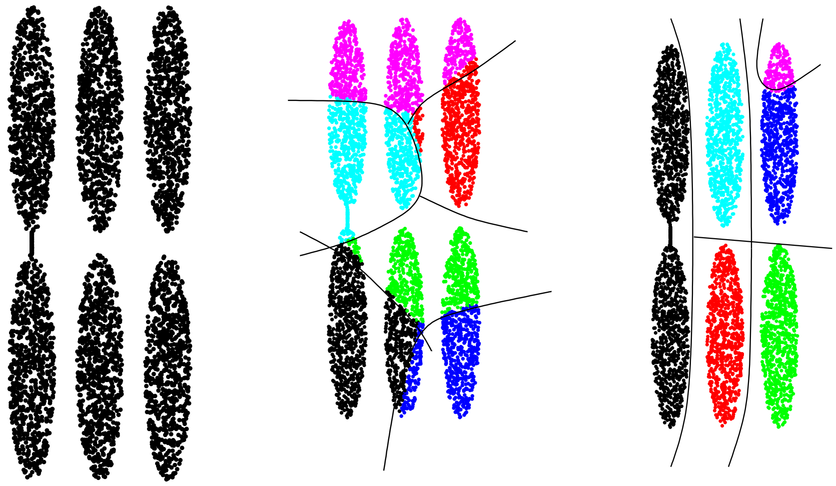
The material in this chapter appears in [GRS98].

12.2 A Case for Approximation

The hierarchical merging of clusters raises the question how the distance between clusters should be measured. The widely used measures for distance between clusters are as follows (m_ℓ is the mean for cluster χ_ℓ and n_ℓ is the number of points in χ_ℓ).

$$\begin{aligned} d_{mean}(\chi_1, \chi_2) &= \|m_1 - m_2\| \\ d_{ave}(\chi_1, \chi_2) &= 1/(n_1 n_2) \sum_{p \in \chi_1} \sum_{p' \in \chi_2} \|p - p'\| \\ d_{max}(\chi_1, \chi_2) &= \max_{p \in \chi_1, p' \in \chi_2} \|p - p'\| \\ d_{min}(\chi_1, \chi_2) &= \min_{p \in \chi_1, p' \in \chi_2} \|p - p'\| \end{aligned}$$

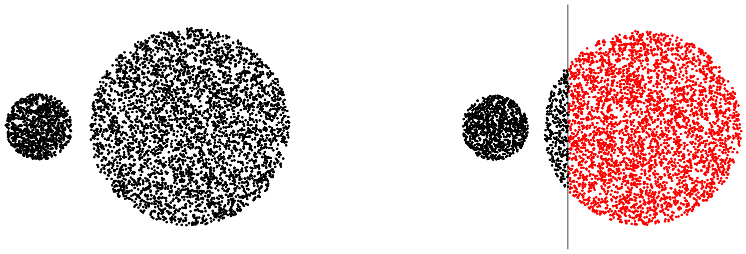
For example, with d_{mean} as the distance measure, at each step, the pair of clusters whose centroids or means are the closest are merged. On the other hand, with d_{min}, the pair of clusters merged are the ones containing the closest pair of points. All of the above distance measures have a minimum variance flavor and they usually yield the same results if the clusters are compact and well-separated. However, if the clusters are close to one another (even by outliers), or if their shapes and sizes are not hyperspherical and uniform, the results of clustering can vary quite dramatically. Consider the example data points in Figure 12.1. The desired elongated clusters are shown in Figure 12.1(a). However, d_{mean} as the distance measure, causes the elongated clusters to be split and portions belonging to neighboring elongated clusters to be merged. The resulting clusters are as shown in Figure 12.1(b). On the other hand, with d_{min} as the distance measure, the resulting clusters are as shown in Figure 12.1(c). The two elongated clusters that are connected by narrow string of points are merged into a single cluster. This "chaining effect" is a drawback of d_{min} – basically, a few points located so as to form a bridge between the two clusters causes points across the clusters to be grouped into a single elongated cluster.



(a) (b) (c)

Figure 12.1: Clusters generated by hierarchical algorithms

From the above discussion, it follows that neither the centroid-based approach (that uses d_{mean}) nor the all-points approach (based on d_{min}) work well for non-spherical or arbitrary shaped clusters. A shortcoming of the centroid-based approach is that it considers only one point as representative of a cluster – the cluster centroid. For a large or arbitrary shaped cluster, the centroids of its sub-clusters can be reasonably far apart, thus causing the cluster to be split. The all-points approach, on the other hand, considers all the points within a cluster as representative of the cluster. This other extreme, has its own drawbacks, since it makes the clustering algorithm extremely sensitive to outliers and to slight changes in the position of data points.



(a) (b)

Figure 12.2: Problem of labeling

When the number N of input data points is large, hierarchical clustering algorithms break down due to their non-linear time complexity (typically, $O(N^2)$) and huge I/O costs. In order to remedy this problem, in [ZRL96], the authors propose a new clustering method named BIRCH, which represents the state of the art for clustering large data sets. BIRCH first performs a *preclustering phase* in which dense regions of points are represented by compact summaries, and then a centroid-based hierarchical algorithm is used to cluster the set of summaries (which is much smaller than the original dataset).

The preclustering algorithm employed by BIRCH to reduce input size is incremental and approximate. During preclustering, the entire database is scanned, and cluster summaries are stored in memory in a data structure called the CF-tree. For each successive data point, the CF-tree is traversed to find the closest cluster to it in the tree, and if the point is within a threshold distance of the closest cluster, it is absorbed into it. Otherwise, it starts its own cluster in the CF-tree.

Once the clusters are generated, a final labeling phase is carried out in which using the centroids of clusters as seeds, each data point is assigned to the cluster with the closest seed. Using only the centroid of a cluster when redistributing the data in the final phase has problems when clusters do not have uniform sizes and shapes as in Figure 12.2(a). In this case, as illustrated in Figure 12.2(b), in the final labeling phase, a number of points in the bigger cluster are labeled as belonging to the smaller cluster since they are closer to the centroid of the smaller cluster.

12.2.1 Approximating the Clusters

The above discussion showed us that while computing intercluster distances, reducing the clusters to a single point or centroid is not a good idea for clusters with unequal sizes (in physical space denoted by the metric space). This appears to encode too little information about the clusters. On the other hand the single link clustering method represents a cluster by all the points belonging to it, and is storing too much information; hence is also lead astray by presence of a few outliers.

The best approach appears to be in a middle ground between the above two. We start by asking the question, can we represent clusters by small amount of information, more than one central point and much less than all the points in a cluster ? Since in an arbitrary metric metric all we have at our disposal are points, the question can be formulated if we can represent clusters by storing a small, k, number of points.

The first premise of storing k points to represent a cluster would be that every point in the existing cluster be close to at least one of the representative points. Of course, this can be modified to at least two, or even, all the points. But we will proceed with the simplest approach, that every point in the cluster be close to at least one representative point. This is a classic example of the *k–Center problem*. The difference with previous approaches is that we are not using the k–Center problem to cluster the data set, but approximate individual clusters, which are clustered using hierarchical clustering. This also alleviates the labeling problem we mentioned in context of centroid based clustering, since each cluster is represented by k points. However we will formulate a better procedure as we proceed.

Several algorithms exist for the k–center problem, we will choose the approach in [Gon85]. In this approach, given $r < k$ points, we choose the next farthest point from this set. Butt it is sufficient if we choose a point which is more than half the distance away from the r points. This allows us to avoid points on the boundaries of the cluster, which are more likely to be outliers. We will discuss this procedure in more detail later.

The quadratic complexity is not very promising in context of clustering very large datasets, and we will use sampling based techniques. The theoretical results in the Chapter 11, indicate that both 'divide and conquer', and 'sampling' are viable approaches, in terms of clustering with worst case bounded error guarantees. However, the sampling approach has a better running time and error guarantee, and we will use this approach. The idea would be to sample the data set, cluster the sample, and use a small fraction of this sample to label the original dataset.

12.3 Related Implementations

In recent years, a number of clustering algorithms for large databases have been proposed [NH94, ZRL96, EKSX96, HKKM97]. In [NH94], the authors propose a partitional clustering method for large databases which is based on randomized search. Each cluster is represented by its *medoid*, the most centrally located point in the cluster, and the objective is to find the k best medoids that optimize the criterion function. The authors reduce this problem to that of graph search by representing each set of k medoids as a node in the graph, two nodes being adjacent if they have $k-1$ medoids in common. Initially, an arbitrary node is set to be the current node and a fixed number of iterations are performed. In each iteration, a random neighbor of the current node is set to be the current node if it results in better clustering. The computation of the criterion function for the random neighbor requires the entire database to be examined. It is experimentally shown that CLARANS outperforms the traditional *k-medoid* algorithms. However, CLARANS may require several passes over the database, the runtime cost of which could be prohibitive for large databases.

In [EKX95], the authors use the R*-tree[SRF87, BKSS90, Sam89] to improve the I/O efficiency of CLARANS on large databases by (1) drawing samples from leaf pages to reduce the number of data points (since data points are packed in leaf nodes based on spatial locality, a sample point in the leaf page can be a good representative point), and (2) focusing on relevant points when evaluating the "goodness" of a neighbor.

Since multiple I/O scans of the data points is a bottleneck for existing clustering algorithms, in [ZRL96], the authors present a clustering method named BIRCH whose I/O complexity is a little more than one scan of the data. BIRCH first pre-clusters the data into the maximum possible and finest possible subclusters that can fit in main-memory. For the pre-clustering phase, BIRCH employs a CF-tree which is a balanced tree structure similar to the B-tree and R-tree family[Sam89]. After pre-clustering, BIRCH treats each of the subcluster summaries as representative points, and runs a well-known approximation algorithm from [Ols93], which is an agglomerative hierarchical clustering algorithm.

BIRCH and CLARANS work well for convex or spherical clusters of uniform size.

However, they are unsuitable when clusters have different sizes or when clusters are non-spherical (see Figure 12.1). For clustering such arbitrary shaped collections of points (e.g., ellipsoid, spiral, cylindrical), a density-based algorithm called DBSCAN was proposed in [EKSX96]. DBSCAN requires the user to specify two parameters that are used to define minimum density for clustering – the radius *Eps* of the neighborhood of a point and the minimum number of points *MinPts* in the neighborhood. Clusters are then found by starting from an arbitrary point and if its neighborhood satisfies the minimum density, including the points in its neighborhood into the cluster. The process is then repeated for the newly added points.

While DBSCAN can find clusters with arbitrary shapes, it suffers from a number of problems. DBSCAN is very sensitive to the parameters *Eps* and *MinPts*, which in turn, are difficult to determine. Furthermore, DBSCAN also suffers from the robustness problems that plague the all-points hierarchical clustering algorithm – in case there is a dense string of points connecting two clusters, DBSCAN could end up merging the two clusters. Also, DBSCAN does not perform any sort of preclustering and executes directly on the entire database. As a result, for large databases, DBSCAN could incur substantial I/O costs. Finally, with density-based algorithms, using random sampling to reduce the input size may not be feasible – the reason for this is that unless sample sizes are large, there could be substantial variations in the density of points within each cluster in the random sample.

12.4 Clustering using Representatives: CURE

In this section, we present CURE's hierarchical clustering algorithm whose salient features are: (1) the clustering algorithm can recognize arbitrarily shaped clusters (e.g., ellipsoidal), (2) the algorithm is robust to the presence of outliers, and (3) the algorithm has linear storage requirements and time complexity of $O(n^2)$ for low-dimensional data. The n data points input to the algorithm are either a sample drawn randomly from the original data points, or a subset of it if partitioning is employed. An analysis of issues related to the size of the random sample and number of partitions is presented in Section 12.4.3.

12.4.1 Intuition and Overview

As we mentioned earlier, traditional clustering algorithms are unsuitable when clusters have arbitrary shapes (e.g., ellipsoidal) or there is a large variance in the sizes of clusters. For instance, in the centroid-based hierarchical clustering algorithm, clusters, large and small, are represented by their centers. However, this approximation of a cluster by its center gets progressively worse as the size of the cluster increases - thus, causing elongated or large clusters to be split into smaller subclusters. On the other hand, even though the all-points approach based on d_{min} is good for identifying clusters with arbitrary shapes, it is fragile in the presence of outliers.

The clustering algorithm starts with each input point as a separate cluster, and at each successive step merges the closest pair of clusters. In order to compute the distance between a pair of clusters, for each cluster, c representative points are stored. These are determined by first choosing *c well scattered* points within the cluster, and then shrinking them toward the mean of the cluster by a fraction α. The distance between two clusters is then the distance between the closest pair of representative points – one belonging to each of the two clusters. Thus, only the representative points of a cluster are used to compute its distance from other clusters.

The c representative points attempt to capture the physical shape and geometry of the cluster. Furthermore, shrinking the scattered points toward the mean by a factor α gets rid of surface abnormalities and mitigates the effects of outliers. The reason for this is that outliers typically will be further away from the cluster center, and as a result, the shrinking would cause outliers to move more toward the center while the remaining representative points would experience minimal shifts. The larger movements in the outliers would thus reduce their ability to cause the wrong clusters to be merged. The parameter α can also be used to control the shapes of clusters. A smaller value of α shrinks the scattered points very little and thus favors elongated clusters. On the other hand, with larger values of α, the scattered points get located closer to the mean, and clusters tend to be more compact.

```
procedure merge(u, v)
begin
w := u ∪ v
w.mean := (|u|u.mean + |v|v.mean) / (|u| + |v|)
tmpSet := ∅
for i := 1 to c do {
   maxDist := 0
   foreach point p in cluster w do {
      if i = 1
         minDist := dist(p, w.mean)
      else
         minDist := min{dist(p, q) : q ∈ tmpSet}
      if (minDist ≥ maxDist){
         maxDist := minDist
         maxPoint := p
      }
   }
   tmpSet := tmpSet ∪ {maxPoint}
}
foreach point p in tmpSet do
   w.rep := w.rep ∪ {p + α*(w.mean-p) }
return w
end
```

Figure 12.3: Procedure for merging clusters

12.4.2 Clustering procedure:

Initially, for each cluster u, the set of representative points u.rep contains only the point in the cluster. Thus, in Step 1, all input data points are inserted into the k-d tree. The procedure build_heap (in Step 2) treats each input point as a separate cluster, computes u.closest for each cluster u and then inserts each cluster into the heap (note that the clusters are arranged in the increasing order of distances between u and $u.closest$).

Once the heap Q and tree T are initialized, in each iteration of the while-loop, until only k clusters remain, the closest pair of clusters is merged. The cluster u at the top of the heap Q is the cluster for which u and $u.closest$ are the closest pair of clusters. Thus, for each step of the while-loop, extract_min (in Step 4) extracts the top element u in Q and also deletes u from Q. The merge procedure (see Figure 12.3) is then used to merge the closest pair of clusters u and v, and to compute new representative points for the new merged cluster w which are subsequently inserted into T (in Step 8). The points in cluster w are simply the union of the points in the two clusters u and v that were merged. The merge procedure, in the for-loop (steps 4-17), first iteratively selects c well-scattered points. In the first iteration, the point farthest from the mean is chosen as the first scattered point. In each subsequent iteration, a point from the cluster w is chosen that is farthest from the previously chosen scattered points. The points are then shrunk toward the mean by a fraction α in Step 19 of the merge procedure.

For the merged cluster w, since the set of representative points for it could have changed (a new set of representative points is computed for it), we need to compute its distance to every other cluster and set w.closest to the cluster closest to it (see steps 11 and 12 of the cluster procedure). Similarly, for a different cluster x in Q, x.closest may change and x may need to be relocated in Q (depending on the distance between x and x.closest). A brute-force method for determining the closest cluster to x is to compute its distance with every other cluster (including w). However, this would require $O(n)$ steps for each cluster in Q, and could be computationally expensive and inefficient. Instead, we observe that the expensive computation of determining the closest cluster is not required for every cluster x. For the few cases that it is required, we use T to determine this efficiently in $O(\log n)$ steps per case. We can classify the clusters in Q into two groups. The first group of clusters are those who had either u or v as the closest cluster before u and v were merged. The remaining clusters in Q constitute the second group. For a cluster x in the first group, if the distance to w is smaller than its distance to the previously closest cluster (say u), then all we have to do is simply set w to be the closest cluster (see Step 17). The reason for this is that we know that the distance between x and every other cluster is greater than the distance between x and u. The problem arises when the distance between x and w is larger than the distance between x and u. In this case, any of the other clusters could become the new closest cluster to x. The procedure closest_cluster (in Step 15) uses the tree T to determine the closest cluster to cluster x. For every point p in x.rep, T is used to determine the nearest neighbor to p that is not in x.rep. From among the nearest neighbors, the one that is closest to one of x's representative points is determined and the cluster containing it is returned as the closest cluster to x. Since we are not interested in clusters whose distance from x is more than $dist(x, w)$, we pass this as a parameter to closest_cluster which uses it to make the search for nearest neighbors more efficient. Processing a cluster x in the second group is much simpler – x.closest already stores the closest cluster to x from among existing clusters (except w). Thus, if the distance between x and w is smaller than x's distance to it's previously closest cluster, x.closest, then w becomes the closest cluster to x (see Step 21); otherwise, nothing needs to be done. In case x.closest for a cluster x is updated, then since the distance between x and its closest cluster may have changed, x may need to be relocated in the heap Q (see steps 18 and 22).

An improved merge procedure: In the merge procedure, the overhead of choosing representative points for the merged cluster can be reduced as follows. The merge procedure, in the outer for-loop (Step 4), chooses c scattered points from among all the points in the merged cluster w. Instead, suppose we selected the c scattered points for w from the $2c$ scattered points for the two clusters u and v being merged (the original scattered points for clusters u and v can be obtained by unshrinking their representative points by α). Then, since at most $2c$ points, instead of $O(n)$ points, need to be examined every time a scattered point is chosen, the complexity of the merge procedure reduces to $O(1)$. Furthermore, since the scattered points for w are chosen from the original scattered points for clusters u and v, they can be expected to be fairly well spread out.

12.4.3 Enhancements for Large Data Sets

Most hierarchical clustering algorithms, including the one presented in the previous subsection, cannot be directly applied to large data sets due to their quadratic time complexity with respect to the input size. In this section, we present enhancements and optimizations that enable CURE to handle large data sets. We also address the issue of outliers and propose schemes to eliminate them.

12.4.4 Random Sampling

In order to handle large data sets, we need an effective mechanism for reducing the size of the input to CURE's clustering algorithm. One approach to achieving this is via *random sampling* – the key idea is to apply CURE's clustering algorithm to a random sample drawn from the data set rather than the entire data set. Typically, the random sample will fit in main-memory and will be much smaller than the original data set. Consequently, significant improvements in execution times for CURE can be realized. Also, random sampling can improve the quality of clustering since it has the desirable effect of filtering outliers.

Efficient algorithms for drawing a sample randomly from data in a file in one pass and using constant space are proposed in [Vit85]. As a result, we do not discuss sampling in any further detail, and assume that we employ one of the well-known algorithms for generating the random sample. Also, our experience has been that generally, the overhead of generating a random sample is very small compared to the time for performing clustering on the sample (the random sampling algorithm typically takes less than a few seconds to sample a few thousand points from a file containing hundred thousand or more points).

Of course, one can argue that the reduction in input size due to sampling has an associated cost. Since we do not consider the entire data set, information about certain clusters may be missing in the input. As a result, our clustering algorithms may miss out certain clusters or incorrectly identify certain clusters. Even though random sampling does have this tradeoff between accuracy and efficiency, our experimental results indicate that for most of the data sets that we considered, with moderate sized random samples, we were able to obtain very good clusters. In addition, we can use chernoff bounds to analytically derive values for sample sizes for which the probability of missing clusters is low.

We are interested in answering the following question: what should the size s

of the random sample be so that the probability of missing clusters is low ? One assumption that we will make is that the probability of missing a cluster u is low if the sample contains at least $f|u|$ points from the sample, where $0 \leq f \leq 1$. This is a reasonable assumption to make since clusters will usually be densely packed and a subset of the points in the cluster is all that is required for clustering. Furthermore, the value of f depends on the cluster density as well as the intercluster separation – the more well-separated and the more dense clusters become, the smaller is the fraction of the points from each cluster that we need for clustering. Chernoff bounds can be used to prove the following theorem.

Theorem 12.4.1 *For a cluster u, if the sample size s satisfies*

$$s \geq fN + \frac{N}{|u|}\log(\frac{1}{\delta}) + \frac{N}{|u|}\sqrt{(\log(\frac{1}{\delta}))^2 + 2f|u|\log(\frac{1}{\delta})} \qquad (12.1)$$

then the probability that the sample contains fewer than $f|u|$ points belonging to cluster u is less than δ, $0 \leq \delta \leq 1$.

Thus, based on the above equation, we conclude that for the sample to contain at least $f|u|$ points belonging to cluster u (with high probability), we need the sample to contain more than a fraction f of the total number of points – which seems intuitive. Also, suppose u_{min} is the smallest cluster that we are interested in, and s_{min} is the result of substituting $|u_{min}|$ for $|u|$ in the right hand side of Equation (12.1). It is easy to observe that Equation (12.1) holds for $s = s_{min}$ and all $|u| \geq |u_{min}|$. Thus, with a sample of size s_{min}, we can guarantee that with a high probability, $1 - \delta$, the sample contains at least $f|u|$ points from an arbitrary cluster u. Also, assuming that there are k clusters, with a sample size of s_{min}, the probability of selecting fewer than $f|u|$ points from any one of the clusters u is bounded above by $k\delta$.

12.4.5 Partitioning for Speedup

As the separation between clusters decreases and as clusters become less densely packed, samples of larger sizes are required to distinguish them. However, as the input size n grows, the computation that needs to be performed by CURE's clustering algorithm could end up being fairly substantial due to the $O(n^2 \log n)$ time complexity. In this subsection, we propose a simple partitioning scheme for speeding up CURE when input sizes become large.

The basic idea is to partition the sample space into p partitions, each of size $\frac{n}{p}$. We then partially cluster each partition until the final number of clusters in each partition reduces to $\frac{n}{pq}$ for some constant $q > 1$. Alternatively, we could stop merging clusters in a partition if the distance between the closest pair of clusters to be merged next increases above a certain threshold. Once we have generated $\frac{n}{pq}$ clusters for each partition, we then run a second clustering pass on the $\frac{n}{q}$ partial clusters for all the partitions (that resulted from the first pass).

The idea of partially clustering each partition achieves a sort of *preclustering* – schemes for which were also proposed in [ZRL96]. The preclustering algorithm in [ZRL96] is incremental and scans the entire data set. Each successive data point becomes part of the closest existing cluster if it is within some threshold distance τ from it – else, it forms a new cluster. Thus, while [ZRL96] applies an incremental and approximate clustering algorithm to all the points, CURE uses it's hierarchical clustering algorithm only on the points in a partition. In some abstract sense, CURE's partitioning scheme behaves like a sieve working in a bottom-up fashion and filtering out individual points in favour of partial clusters that are then processed during the second pass.

An important point to note is that, in the first pass, the closest points in each partition are merged only until the final number of clusters reduces to $\frac{n}{pq}$. By ensuring that $\frac{n}{pq}$ is sufficiently large compared to the number of desired clusters, k, we can ensure that even though each partition contains fewer points from each cluster, the closest points merged in each partition generally belong to the same cluster and do not span clusters. Thus, we can ensure that partitioning does not adversely impact clustering quality. Consequently, the best values for p and q are those that maximize the improvement factor $\frac{q-1}{pq} + \frac{1}{q^2}$ while ensuring that $\frac{n}{pq}$ is at least 2 or 3 times k.

The partitioning scheme can also be employed to ensure that the input set to the clustering algorithm is always in main-memory even though the random sample itself may not fit in memory. If the partition size is chosen to be smaller than the main-memory size, then the input points for clustering during the first pass are always main-memory resident. The problem is with the second pass since the size of the input is the size of the random sample itself. The reason for this is that for every cluster input to the second clustering pass, we store all the points in the cluster. Clearly, this is unnecessary, since our clustering algorithm only relies on the representative points for each cluster. Furthermore, the improved merge procedure in Section 12.4 only uses representative points of the previous clusters when computing the new representative points for the merged cluster. Thus, by storing only the representative points for each cluster input to the second pass, we can reduce the input size for the second clustering pass and ensure that it fits in main-memory.

12.4.6 Labeling Data on Disk

Since the input to CURE's clustering algorithm is a set of randomly sampled points from the original data set, the final k clusters involve only a subset of the entire set of points. In CURE, the algorithm for assigning the appropriate cluster labels to the remaining data points employs a fraction of *randomly* selected representative points for each of the final k clusters. Each data point is assigned to the cluster containing the representative point closest to it.

Note that approximating every cluster with multiple points instead a single centroid as is done in [ZRL96], enables CURE to, in the final phase, correctly distribute the data points when clusters are non-spherical or non-uniform. The final labeling phase of [ZRL96], since it employs only the centroids of the clusters for partitioning the remaining points, has a tendency to split clusters when they have non-spherical shapes or non-uniform sizes (since the space defined by a single centroid is a sphere).

12.4.7 Handling Outliers

Any data set almost always contains *outliers*. These do not belong to any of the clusters and are typically defined to be points of non agglomerative behaviour. That is, the neighborhoods of outliers are generally sparse compared to points in clusters, and the distance of an outlier to the nearest cluster is comparatively higher than the distances among points in bonafide clusters themselves.

Every clustering method needs mechanisms to eliminate outliers. In CURE, outliers are dealt with at multiple steps. First, random sampling filters out a majority of the outliers. Furthermore, the few outliers that actually make it into the random sample are distributed all over the sample space. Thus, random sampling further isolates outliers.

In agglomerative hierarchical clustering, initially each point is a separate cluster. Clustering then proceeds by merging closest points first. What this suggests is that outliers, due to their larger distances from other points, tend to merge with other points less and typically grow at a much slower rate than actual clusters. Thus, the number of points in a collection of outliers is typically much less than the number in a cluster.

This leads us to a scheme of outlier elimination that proceeds in two phases. In the first phase, the clusters which are growing very slowly are identified and eliminated as outliers. This is achieved by proceeding with the clustering for some time until the number of clusters decreases below a certain fraction of the initial number of clusters. At this time, we classify clusters with very few points (e.g., 1 or 2) as outliers. The choice of a value for the fraction of initial clusters at which outlier elimination gets triggered is important. A very high value for the fraction could result in a number of cluster points being incorrectly eliminated – on the other hand, with an extremely low value, outliers may get merged into proper clusters before the elimination can kick in. An appropriate value for the fraction, obviously, is dependent on the data set. For most data sets we considered, a value of around $\frac{1}{3}$ performed well.

The first phase of outlier elimination is rendered ineffective if a number of outliers get sampled in close vicinity – this is possible in a randomized algorithm albeit with low probability. In this case, the outliers merge together preventing their elimination, and we require a second level of pruning. The second phase occurs toward the end. Usually, the last few steps of a clustering are the most important ones, because the granularity of the clusters is very high, and a single mistake could have grave consequences. Consequently, the second phase of outlier elimination is necessary for good clustering. From our earlier discussion, it is easy to observe that outliers form very small clusters. As a result, we can easily identify such small groups and eliminate them when there are very few clusters remaining, typically in the order of k, the actual number of clusters desired. We have found the above two-phase approach to outlier elimination to work very well in practice.

12.5 Experimental Results

In this section, we study the performance of CURE and demonstrate its effectiveness for clustering compared to BIRCH and MST[1] (*minimum spanning tree*). From our experimental results, we establish that

- BIRCH fails to identify clusters with non-spherical shapes (e.g., elongated) or wide variances in size.
- MST is better at clustering arbitrary shapes, but is very sensitive to outliers.
- CURE can discover clusters with interesting shapes and is less sensitive (than MST) to outliers.
- Sampling and partitioning, together, constitute an effective scheme for preclustering – they reduce the input size for large data sets without sacrificing the quality of clustering.
- The execution time of CURE is low in practice.
- Sampling and our outlier removal algorithm do a good job at filtering outliers from data sets.
- Our final labeling phase labels the data residing on disk correctly even when clusters are non-spherical.

In our experiments, in addition to CURE, we consider BIRCH and MST. We first show that, for data sets containing elongated or big and small clusters, both BIRCH

[1]Clustering using MST is the same as the all-points approach described in Section 12.2, that is, hierarchical clustering using d_{min} as the distance measure.

and MST fail to detect the clusters while CURE discovers them with appropriate parameter settings. We then focus on analyzing the sensitivity of CURE to parameters like the shrink factor α and the random sample size s. Finally, we compare the execution times of BIRCH and CURE on a data set from [ZRL96], and present the results of our scale-up experiments with CURE. In all of our experiments, we use Euclidean distance as the distance metric. We performed experiments using a Sun Ultra-2/200 machine with 512 MB of RAM and running Solaris 2.5. The clusters in the figures in this section were generated using our labeling algorithm and visualizer. Our visualizer assigns a unique color to each cluster[2].

12.5.1 Algorithms

BIRCH: We used the implementation of BIRCH provided to us by the authors of [ZRL96]. The implementation performs preclustering and then uses a centroid-based hierarchical clustering algorithm with time and space complexity that is quadratic in the number of points after preclustering. We set parameter values to the default values suggested in [ZRL96]. For example, we set the page size to 1024 bytes and the input size to the hierarchical clustering algorithm after the preclustering phase to 1000. The memory used for preclustering was set to be about 5% of dataset size.

CURE: Our version of CURE is based on the clustering algorithm described in Section 12.4, that uses representative points with shrinking towards the mean. As described at the end of Section 12.4, when two clusters are merged in each step of the algorithm, representative points for the new merged cluster are selected from the ones for the two original clusters rather than all points in the merged cluster. This improvement speeds up execution times for CURE without adversely impacting the quality of clustering. In addition, we also employ random sampling, partitioning, outlier removal and labeling as described in Section 12.4.3. Our implementation makes use of the k-d tree and heap data structures. Thus, our implementation requires linear space and has a quadratic time complexity in the size of the input for lower

[2]The figures in this paper were originally produced in color. These can be found in our online submission.

Symbol	Meaning	Default Value	Range
s	Sample Size	2500	500 - 5000
c	Number of Representatives in Cluster	10	1 - 50
p	Number of Partitions	1	1 - 50
q	Reducing Factor for Each Partition	3	–
α	Shrink Factor	0.3	0 - 1.0

Table 12.1: Parameters

dimensional data.

The partitioning constant q (introduced in Section 12.4.5) was set to 3. That is, we continue clustering in each partition until the number of clusters remaining is 1/3 of the number of points initially in the partition. We found that, for some of the data sets we considered, when q exceeds 3, points belonging to different clusters are merged, thus negatively impacting the clustering quality. Also we handle outliers as follows. First, at the end of partially clustering each partition, clusters containing only one point are considered outliers, and eliminated. Then, as the number of clusters approaches k, we again remove outliers – this time, the ones containing as many as 5 points. Finally, in almost all of our experiments, the random sample size was chosen to be about 2.5% of the initial data set size. Table 12.1 shows the parameters for our algorithm, along with their default values and the range of values for which we conducted experiments.

MST: When $\alpha = 0$ and the number of representative points c is a large number, CURE reduces to the MST method. Thus, instead of implementing the MST algorithm, we simply use CURE with the above parameter settings for α and c. This suffices for our purposes since we are primarily interested in performing qualitative measurements using MST.

12.5.2 Data sets

We experimented with four data sets containing points in two dimensions whose geometric shape is as illustrated in Figure 12.4. The number of points in each data set is also described in Table 12.2. Data set 1 contains one big and two small circles

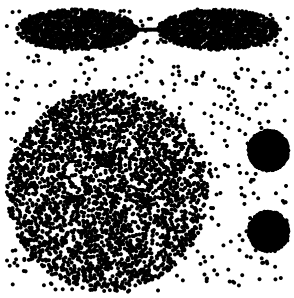
	No. of Points	Shape of Clusters	No. of Clusters
Data Set 1	100000	Big and Small Circles, Ellipsoids	5
Data Set 2	100000	Big and Small Ellipsoids	10
Data Set 3	121560	Big and Small Rings	10
Data Set 4	100000	Circles of Equal Sizes	100

Table 12.2: Data Sets

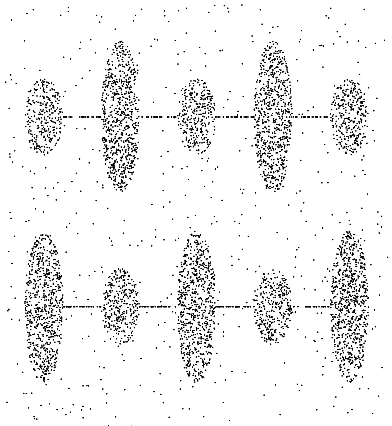
that traditional partitional and hierarchical clustering algorithms, including BIRCH, fail to find. The data set also contains two ellipsoids which are connected by a chain of outliers. In addition, the data set has random outliers scattered in the entire space. The purpose of having outliers in the data set is to compare the sensitivity of CURE and MST to outliers. Data set 2 contains big and small ellipsoids while Data set 3 has a sequence of 2 concentric rings. Again, as for Data set 1, a chain of outliers connect the clusters and outliers are distributed uniformly in the data set. Finally, Data set 4 is identical to one of the data sets used for the experiments in [ZRL96]. It consists of 100 clusters with centers arranged in a grid pattern and data points in each cluster following a normal distribution with mean at the cluster center (a more detailed description of the data set can be found in [ZRL96]). We show that CURE not only correctly clusters this data set, but also that its execution times on the data set are much smaller than BIRCH.

12.5.3 Quality of Clustering

We run the three algorithms on data sets 1–3 to compare them with respect to the quality of clustering. Figure 12.5 shows the clusters found by the three algorithms for Data set 1. As expected, since BIRCH uses a centroid-based hierarchical clustering algorithm for clustering the preclustered points, it cannot distinguish between the big and small clusters. It splits the larger cluster while merging the two smaller clusters adjacent to it. In contrast, the MST algorithm merges the two ellipsoids because it cannot handle the chain of outliers connecting them. CURE successfully discovers the clusters in Data set 1 with the default parameter settings in Table 12.1, that is, $s = 2500$, $c = 10$, $\alpha = 0.3$ and $p = 1$. The moderate shrinking toward the mean by a

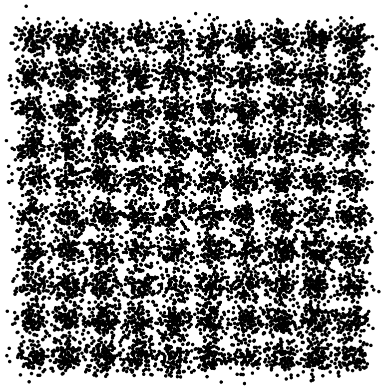


(a) Data set 1



(b) Data set 2

(c) Data set 3



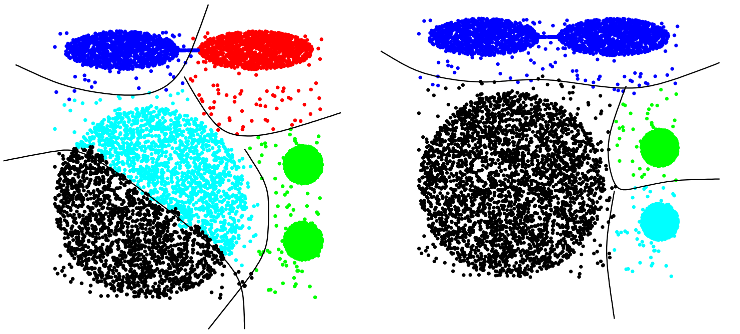
(d) Data set 4

Figure 12.4: Data sets

factor of 0.3 enables CURE to be less sensitive to outliers without splitting the large and elongated clusters.

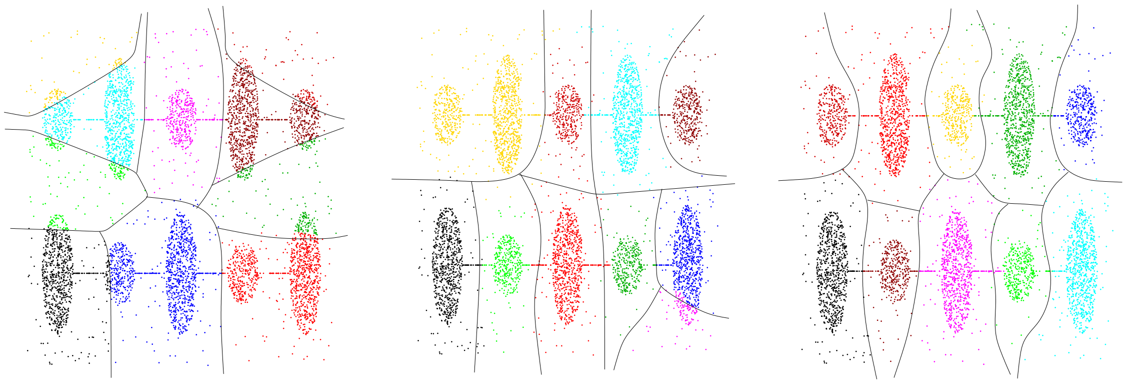
We obtain similar results with data sets 2 and 3. For instance, with Data set 2 (see Figure 12.6), BIRCH separates the extreme portions of the elongated clusters, and then merges these across the clusters. Also, it merges the middle portions of some adjacent pairs of clusters. MST, on the other hand, due to the chain of outliers between clusters, merges the first two elongated clusters in the top row. In contrast, CURE finds the right clusters with the same parameter settings as for Data set 1, that is, the default parameter settings.

Finally, with Data set 3, we obtain the best clustering results compared to BIRCH



(a) BIRCH (b) MST METHOD (c) CURE

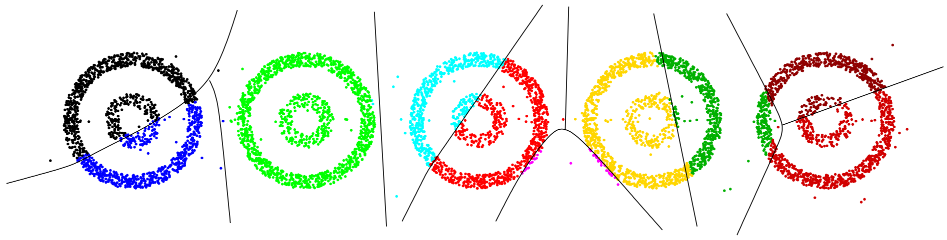
Figure 12.5: Data set 1



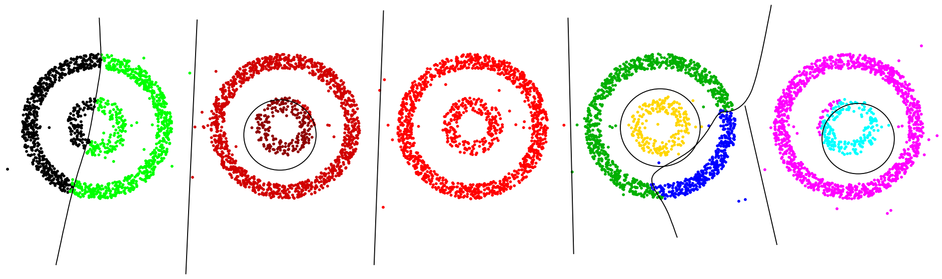
(d) BIRCH (e) MST METHOD (f) CURE

Figure 12.6: Data set 2

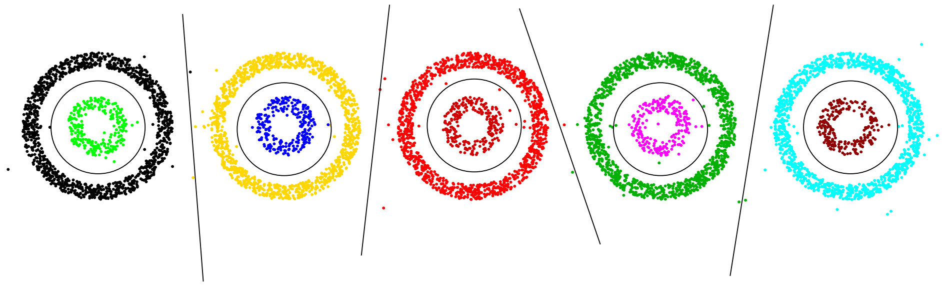
and MST as illustrated in Figure 12.7. BIRCH splits almost every pair of concentric rings into two halves and then merges the halves of the outer and inner rings into a single cluster. MST exhibits a similar behavior on the first pair of rings; on the third, it merges both the outer and inner rings (due to the outliers connecting them) while on the fourth it splits the outer ring into two clusters. Only CURE identifies each ring as a separate cluster with the following values for the parameters: $s = 3000$, $c = 100$, $\alpha = 0.15$ and $p = 1$. Note that we require a larger number of representative points and a smaller α to capture the more complex geometry of the concentric rings.



(a) BIRCH



(b) MST METHOD



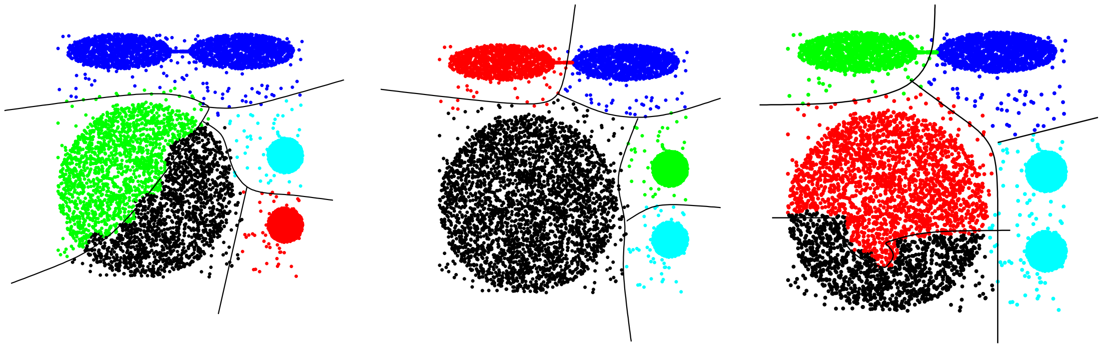
(c) CURE

Figure 12.7: Data set 3

12.5.4 Sensitivity to Parameters

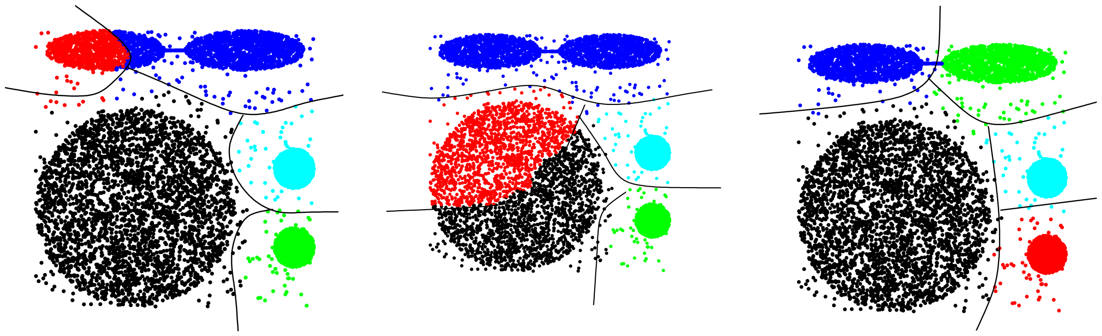
In this section, we perform a sensitivity analysis for CURE with respect to the parameters α, c, s and p. We use Data set 1 for our study. Furthermore, when a single parameter is varied, the default settings in Table 12.2 are used for the remaining parameters.

Shrink Factor α: Figure 12.8 shows the clusters found by CURE when α is varied from 0.1 to 0.9. The results, when $\alpha = 1$ and $\alpha = 0$, are similar to BIRCH and



(a) $\alpha = 0.1$ (b) $\alpha = 0.2 - 0.7$ (c) $\alpha = 0.8 - 0.9$

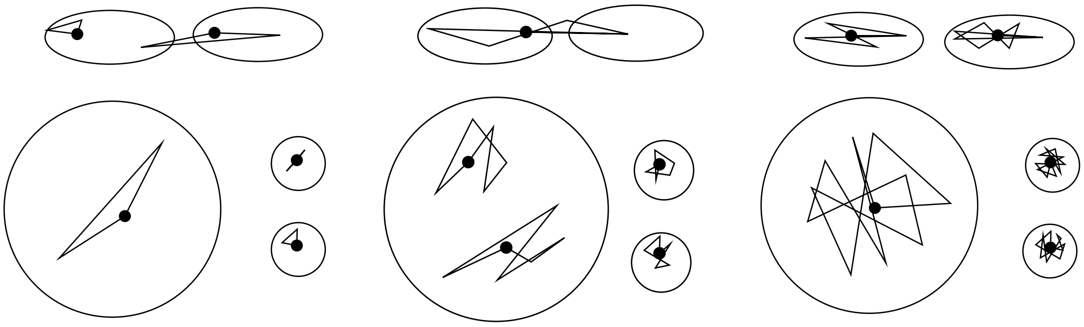
Figure 12.8: Shrink factor toward centroid



(a) $c = 2$ (b) $c = 5$ (c) $c = 10 - 100$

Figure 12.9: Number of representatives

MST, respectively. These were presented in the previous subsection and thus, we do not present the results for these values of α. As the figures illustrate, when α is 0.1, the scattered points are shrunk very little and thus CURE degenerates to the MST algorithm which merges the two ellipsoids. CURE behaves similar to traditional centroid-based hierarchical algorithms for values of α between 0.8 and 1 since the representative points end up close to the center of the cluster. However, for the entire range of α values from 0.2 to 0.7, CURE always finds the right clusters. Thus, we can conclude that 0.2–0.7 is a good range of values for α to identify non-spherical clusters while dampening the effects of outliers.



(a) $c = 2$ (b) $c = 5$ (c) $c = 10$

Figure 12.10: Representatives of clusters

Number of Representative Points c: Figure 12.9 illustrates the results of clustering using CURE when the number of representative points are varied from 1 to 100. For smaller values of c, we found that the quality of clustering suffered. For instance, when $c = 5$, as shown in Figure 12.9(b), the big cluster is split. This is because a small number of representative points do not adequately capture the geometry of clusters. However, for values of c greater than 10, CURE always found right clusters. In order to illustrate the effectiveness of the representative points in capturing the geometry of clusters, we also plot, in Figure 12.10, the representatives for clusters at the end of clustering. The circular dot is the center of clusters and the representative points are connected by lines. As the figure illustrates, the representative points take shapes of clusters. Also, Figure 12.10(b) explains why the big cluster is split when we choose only 5 representative points per cluster. The reason is that the distance between the closest representative points belonging to the two subclusters of the large cluster becomes fairly large when c is 5.

Number of Partitions p: We next varied the number of partitions from 1 to 100. With as many as 50 partitions, CURE always discovered the desired clusters. However, when we divide the sample into as many as 100 partitions, the quality of clustering suffers due to the fact that each partition does not contain enough points for each cluster. Thus, the relative distances between points in a cluster become large compared to the distances between points in different clusters – the result is that points belonging to different clusters are merged.

There is a correlation between the number of partitions p and the extent to which each partition is clustered as determined by q (a partition is clustered until the clusters remaining are $\frac{1}{q}$ of the original partition size). In order to preserve the integrity of clustering, as q increases, partition sizes must increase and thus, p must decrease. This is because with smaller partition sizes, clustering each partition to a larger degree could result in points being merged across clusters (since each partition contains fewer points from each cluster). In general, the number of partitions must be chosen such that $\frac{s}{pq}$ is fairly large compared to k (e.g., at least 2 or 3 times k).

Random Sample Size s**:** We ran CURE on Data set 1 with random sample sizes ranging from 500 upto 5000. For sample sizes up to 2000, the clusters found were of poor quality. However, from 2500 sample points and above (2.5% of the data set size), CURE always correctly identified the clusters.

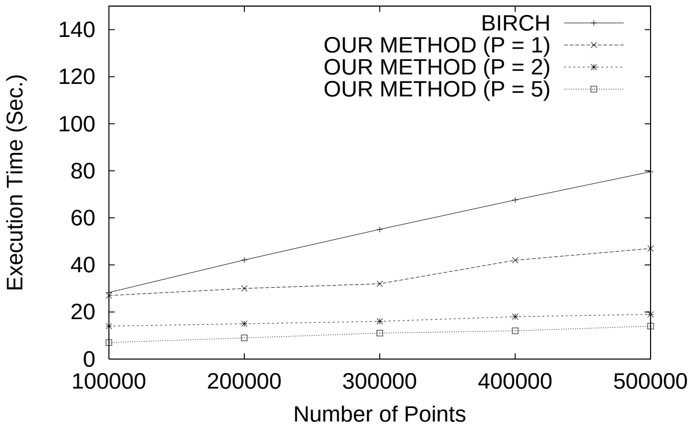


Figure 12.11: Comparison to BIRCH

12.5.5 Comparison of Execution time to BIRCH

The goal of our experiment in this subsection is to demonstrate that using the combination of random sampling and partitioning to reduce the input size as is done by CURE can result in lower execution times than the preclustering scheme employed by BIRCH for the same purpose. We run both BIRCH and CURE on Data set 4 – this is the data set that centroid-based algorithms, in general, and BIRCH, in particular, can cluster correctly since it contains compact clusters with similar sizes. CURE, too, finds the right clusters with a random sample size of 2500, $\alpha = 1$ (which reduces CURE to a centroid-based algorithm), one representative for each cluster (that is, the centroid), and as many as 5 partitions.

Figure 12.11 illustrates the performance of BIRCH and CURE as the number of data points is increased from 100000 to 500000. The number of clusters or their geometry is not altered - thus, each cluster becomes more dense as the number of points increases. For CURE, we consider three values for the number of partitions: 1, 2 and 5, in order to show the effectiveness of partitioning. The execution times do not include the time for the final labeling phase since these are approximately the same for BIRCH and CURE, both of which utilize only the cluster centroid for the purpose of labeling.

As the graphs demonstrate, CURE's execution times are lower than BIRCH's. In addition, partitioning further improves our running times by more than 50%. Finally, as the number of points is increased, execution times for CURE increase very little since the sample size stays at 2500, and the only additional cost incurred by CURE is that of sampling from a larger data set. In contrast, the executions times for BIRCH's preclustering algorithm increases much more rapidly with increasing data set size. This is because BIRCH scans the entire database and uses all the points in the data set for preclustering. Thus, the above results confirm that our proposed random sampling and partitioning algorithm are very efficient compared to the preclustering technique used in BIRCH.

12.5.6 Scale-up Experiments

In order to determine the effects of the random sample size s, number of representatives c and number of partitions p on execution times, we next report the results of scale-up experiments on Data set 1. We obtained similar results with the other data sets. In our running times, we do not include the time for the final labeling phase.

Random Sample Size: In Figure 12.12(a), we plot the execution time for CURE as the sample size is increased from 500 to 5000. The graphs confirm that the computational complexity of CURE is quadratic with respect to the sample size.

Number of Representatives: Figure 12.12 (b) illustrates the execution time for CURE as the number of representative points is varied from 1 to 50. The graph shows a gradual and linear increase in execution time as c is increased.

Number of Partitions: The effect on execution time, as the number of partitions is increased is as shown in Figure 12.12 (c). In Section 12.4.5, we showed that the computational complexity with p partitions is approximately $\frac{1}{p} + \frac{1}{q^2}$ times the complexity with only one partition. Thus, since q is fixed at 3, initial increases in p from 1 onwards, result in larger reductions in execution time. However, once p increases beyond a certain point, the second term containing q dominates and subsequent increases in p result in miniscule improvements in performance. From the graph, it follows that for $q = 3$, once p reaches 5, subsequent partitioning does not achieve significant reductions in running time.

Note that, if we were to increase q, then the threshold for the number of partitions beyond which performance saturates, also increases. The reason for this is that with a larger value for q, the second term of $\frac{1}{p} + \frac{1}{q^2}$ begins to dominate for larger values of p, and thus performance gains can be obtained until the number of partitions reaches a much larger value. However, a word of caution – the choice of values for p and q must be dictated by not only performance considerations, but also considerations related to clustering quality as described earlier in Section 12.5.4.

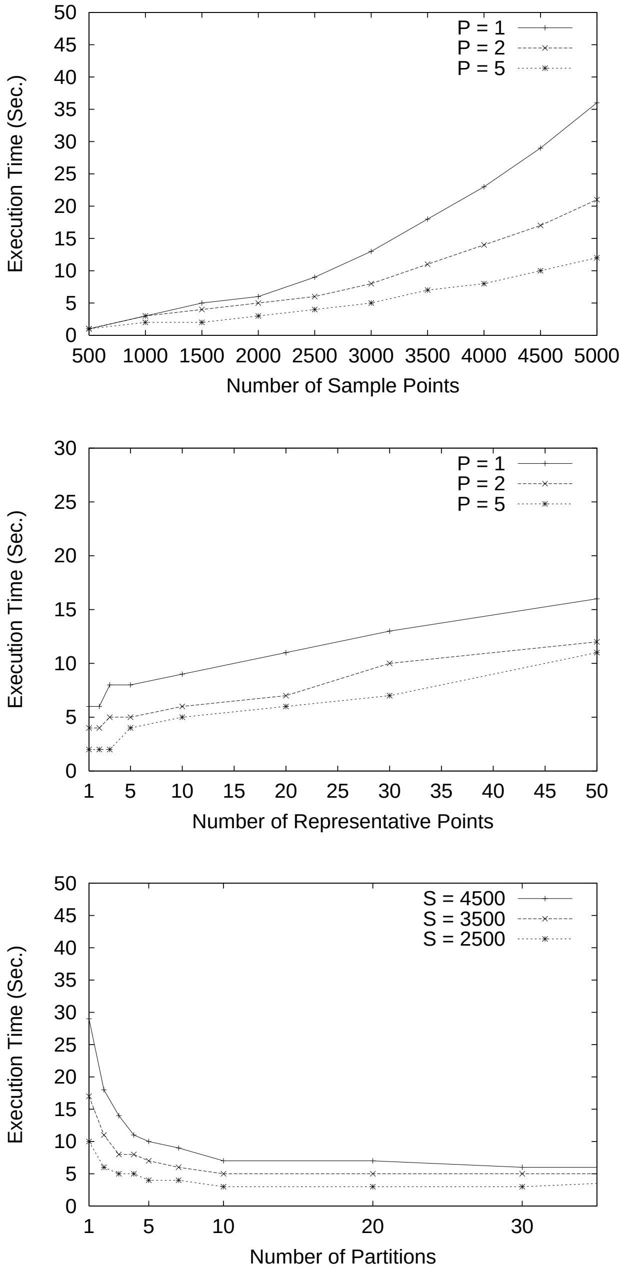


Figure 12.12: Scale-up experiments

Chapter 13

Conclusion

In the first few chapters, (Chapters 2,3, 4, 5), we saw several different techniques of approximating the facility location problem, and the k−Median problem. Next in Chapters 6, and 7, we saw different ideas of combining the algorithms, and reducing the capacitated variants to the uncapacitated variants of the problem. The third section, (Chapters 8, 9, 10), was regarding the application of facility location problem to network design problems. The last two chapters (Chapters 11, 12) considered the question of clustering large datasets. Several interesting questions remain,

- A factor 2 approximation algorithm for the k−Median problem: It is quite likely that the primal solution to the linear program can be modified better than the current best $6\frac{2}{3}$. It is quite vexing that the integrality gap of the primal linear program solution is 4, but that is achieved through primal dual algorithms, without any knowledge of the primal solution. An example contradicting this claim will also be interesting, since it would tell us how to design an algorithm. So far the solutions exploit the tree structure, the primal dual solution also can be viewed as a collection of trees to be rounded. The algorithms use a set of edges (obtained from linear program, or primal dual algorith) which define a tree. An interesting question would be to analyze a solution that uses edges that do not define a tree.

- An optimal approximation for the facility location problem: The lower bound proved has the property that the best approximation ratio from the can be $1 + \rho \log 2(1-\rho)/\rho$, where ρ is the fraction of the facility cost of the optimum dominating set compared to the total cost, and we choose it. On the other hand the algorithm of [Chu98, CS98] show the upper bound to be $1 + \rho \log 2/\rho$ for $\rho \leq 2/e$, where ρ is the fraction of the facility cost in the optimum solution of the linear program. Theorem 6.3.3 shows how to achieve an approximation factor $1 + \rho + \rho \log 2(1-\rho)/\rho$ for $\rho \leq 2/3$ where ρ is the fraction of facility cost of any feasible fractional solution. The cases where ρ is high is easier to handle. The function forms of the upper bounds is very similar to the form of the lower bound, and it is conceivable that accounting properly we may be able to design an optimal approximation algorithm for facility location. The lower bound is proved for special metrics with distances 1 and 3; it would be interesting if this can be increased. Such a step should also provide directions for improving the lower bound for the k–Median problem. Another question would be to develop an unified facility location algorithm and analysis that achieves the best approximation factor, rather than the better of two algorithms approach.

- Space Lower bounds for approximation of the k–Median Problem: Most of the lower bounds in approximation theory concentrate on the quality of approximation under suitable complexity theoretic assumption, typically $P \neq NP$. However in several optimization problems over large data sets, the space requirements can be more restrictive than the time bounds. For example a heuristic with quadratic space requirement may not be feasible since the device may run out of address space. In this context, the question of tradeoff of approximation with respect to the space required by algorithms can be explored. In Chapter 11 we saw that we can cluster n points in space $O(n^{\delta})$, with an approximation factor of $2^{\frac{1}{\delta}}$. This holds in the stream model irrespective of randomization. The randomization allows the running time to be smaller, the deterministic version either runs in time $O(n^{1+\epsilon\delta})$ or suffers a loss of $poly(\frac{1}{\epsilon})$ in the approximation factor. Thus the approximation factor is dominated stronger by restrictions in space than time. Lower bounds on space bounded computation are shown by communication complexity, [FKSV99] show how to extend some of these arguments to data streams. However it would be a step forward if we can develop methods for a broader class of approximation algorithms. In the contex of k–Median, simple experiments of dividing the input across two parties achieve nothing, since the assumption of a metric distance function provides enough information. It is sufficient that both parties cluster their data and exchange information about centers and weights. The space bound of such a protocol is $O(k\ polylog(n))$. This protocol would give a fairly good, $O(1)$, approximation to the entire solution, but the upper bounds we develop in Chapter 11 only allow us to prove an approximation factor of $n^{1/\log\log n}$. Any such progress in resolving these gaps would possibly extend to other algorithms for large point sets in metric spaces.

- Constant approximation for minsum clustering: In Chapter 7, we saw references to generalized clustering, and minsum clustering. IN [BCR00], a bicriterion approximation for the minsum clustering is provided. If the number of clusters are preserved, the best known approximation guarantee is $\log n$. It is an interesting question if we can achieve a constant factor preserving the number of clusters. For most other objective functions, the current approximation factors are constants, with the objective function being Max SNP hard. This minsum objective is one of the few which are yet to be correctly classified in terms of the approximation class it belongs to.

- We saw several approximation factors for facility location problems and its variants, one obvious question would be regarding the tightness of these approximations. Most of the approximations we showed were through reductions of facility location, ad are much worse compared to the lower bound of 1.47. The approximation for single sink ede installation is a few thousand. The approximation factor increases due to the hierarchical nature of the analysis. One such example would be the Multi-Level facility location problem for which solving a primal linear program of size $O(n^k)$, (actually its dual with polynomial variables, using ellipsoid algorithm and a separation oracle) and subsequent rounding gives an approximation factor 3, (See [ACS99]), whereas the reduction to facility location gives a larger approximation factor, (See Chapter 9). Similarly improving approximation factors for fault tolerant facility location, access network design, single sink edge installation, capacitated facility location, or capacitated k–Median would provide insight into developing better heuristics.

Bibliography

[AA97] B. Awerbuch and Y. Azar. Buy-at-bulk network design. *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 542–47, 1997.

[ABP90] B. Awerbuch, A. Baratz, and D. Peleg. Cost-sensitive analysis of communication protocols. *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, pages 177–87, 1990.

[ACS99] K. Aardal, F. Chudak, and D. B. Shmoys. A 3 approximation algorithm for the k-level uncapacitated facility location problem. *Information Processing Letters*, 72:161–167, 1999.

[AKR95] A. Agarwal, P. Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized steiner problem on networks. *SIAM Journal on Computing*, pages 440–456, 1995.

[ALLQ96] K. Aardal, M. Labbé, J. Leung, and M. Queyranne. On the two-level uncapacitated facility location problem. *INFORMS J. Comput.*, 8:289–301, 1996.

[AM00] Matthew Andrews and Kamesh Munagala. Online algorithms for caching multimedia streams. *manuscript*, 2000.

[AZ98] Matthew Andrews and Lisa Zhang. The access network design problem. *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pages 40–49, 1998.

[Bal66] M. L. Balinski. On finding integer solutions to linear programs. *Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems*, pages 225–248, 1966.

[Bar96] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. *37th IEEE symposium on Foundations of Computer Science*, pages 184–193, 1996.

[Bar98] Y. Bartal. On approximating arbitrary metrics by tree metrics. *30th ACM Symposium on Theory of Computing*, 1998.

[BCF+99] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. *Proceedings of INFOCOM*, 1999.

[BCR00] Y. Bartal, M. Charikar, and D. Raz. Approximating p-clustering in metric spaces. *manuscript*, 2000.

[BKSS90] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R^*-tree: an efficient and robust access method for points and rectangles. *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 322–331, 1990.

[CCFM97] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997.

[CCG+98] M. Charikar, C. Chekuri, A. Goel, S. Guha, and S. Plotkin. Approximating a finite metric by a small number of tree metrics. *39th IEEE Symposium on Foundations of Computer Science*, 1998.

[CCGG98] M. Charikar, C. Chekuri, A. Goel, and S. Guha. Rounding via trees: Deterministic approximation algorithms for group steiner trees and k-median. *30th ACM Symposium on Theory of Computing*, 1998.

[CFN77] G. Cornuéjols, M. L. Fisher, and G.L. Nemhauser. Location of bank accounts to optimize float: An analytical study of exact and approximate algorithms. *Management Science*, pages 789–810, 1977.

[CG99] M. Charikar and S. Guha. Improved combinatorial algorithms for facility location and k-median problems. *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, 1999.

[CGR98] S. Chaudhury, N. Garg, and R. Ravi. The p-neighbor k-center problem. *Information Processing Letters*, 65(3):131–134, 1998.

[CGTS99] M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys. A constant factor approximation algorithm for the k-median problem. *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, 1999.

[Cha79] V. Chatal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.

[Cha00] M. Charikar. Approximation algorithms for clustering problems. *PhD Thesis, Stanford University*, 2000.

[Chu98] F. Chudak. Improved approximation algorithms for uncapacitated facility location. *Proceedings of Integer Programming and Combinatorial Optimization*, LNCS 1412:180–194, 1998.

[CLM99] C. Chiang, M. Liu, and M. Muller. Caching neighbourhood protocol: A foundation for building dynamic web caching hierarchies with proxy servers. *Proceedings of International Conference on Parallel Processing*, 1999.

[CNW90] G. Cornuéjols, G.L. Nemhauser, and L.A. Wosley. The uncapacitated facility location problem. In P. Mirchandani and R. Francis, editors, *Discrete Location Theory*, pages 119–171. John Wiley and Sons, Inc., New York, 1990.

[CS98] F. Chudak and D. B. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *Unpublished manuscript*, 1998.

[CS99] F. Chudak and D. B. Shmoys. Improved approximation algorithms for the capacitated facility location problem. *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages S875–S876, 1999.

[CW99] F. Chudak and D. P. Wiliiamson. Improved approximation algorithms for capacitated facility location problems. *Proceedings of Integer Programming and Combinatorial Optimization*, LNCS 1610:99–113, 1999.

[DH73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[DS96] A. Dan and D. Sitaram. A generalized interval caching policy for mixed interactive and long video environments. *Multimedia Computing and Networking*, January 1996.

[dVPV95] S. de Vries, M. Posner, and R. Vohra. On a generalization of the p center problem. *Information Processing Letters*, 56:67–71, 1995.

[EKSX96] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial database with noise. *International Conference on Knowledge Discovery in Databases and Data Mining (KDD-96)*, 1996.

[EKX95] M. Ester, H. Kriegel, and X. Xu. A database interface for clustering in large spatial databases. *International Conference on Knowledge Discovery in Databases and Data Mining (KDD-95)*, 1995.

[Fei96] U. Feige. A threshold of $\ln n$ for approximating set-cover. *Proceedings of 28th ACM Symposium on Theory of Computing*, pages 314–318, 1996.

[Fei99] U. Feige. Personal communiction. 1999.

[FKSV99] J. Feigenbauma, S. Kannan, M. Strauss, and M. Vishwanathan. An approximate l^1–difference algorithm for massive data sets. *Proceedings of 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 501–511, 1999.

[FKV98] A. Frieze, R. Kannan, and S. Vempala. Fast monte carlo algorithms for finding low rank approximation. *Proceedings of 39th Annual IEEE Symposium on Foundations of Computer Science*, 1998.

[FM83] P. Flajolet and G. N. Martin. Probabilistic counting. *Proceedings of 24th Annual IEEE Symposium on Foundations of Computer Science*, pages 76–82, 1983.

[Gar96] N. Garg. A 3 approximation for the minimum tree spanning k vertices. *Proceedings of 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 302–309, 1996.

[GK98] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 649–657, 1998.

[GMM00a] S. Guha, A. Meyerson, and K. Munagala. Facility location with demand dependent costs and generalized clustering. *Stanford University Technical Note, STAN-CS-TN-00-96*, 2000.

[GMM00b] S. Guha, A. Meyerson, and K. Munagala. Hierarchical placement and network design problems. *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science*, 2000.

[GMM00c] S. Guha, A. Meyerson, and K. Munagala. Improved approximation algorithms for fault tolerant facility location. *Stanford University Technical Note, STAN-CS-TN-00-96*, 2000.

[GMM00d] S. Guha, A. Meyerson, and K. Munagala. Improved combinatorial algorithms for single sink edge installation problems. *Stanford University Technical Note, STAN-CS-TN-00-96*, 2000.

[GMMO00] S. Guha, N. Mishra, R. Motwani, and L. O'callahan. Clustering data streams. *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science*, 2000.

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

[GRS98] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 73–84, 1998.

[GW95] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, April 1995.

[GW97] M. Goemans and D. P. Williansom. The primal dual method for approximation algorithms and its application to network design problems. In D. Hochbaum, editor, *Approximation Algorithms for NP-Hard problems*, pages 144–191. International Thomson Publishing, 1997.

[Har79] P. Hartigan. *Clustering Algorithms.* 1979.

[Hås97] J. Håstad. Some optimal inapproximability results. *Proceedings of 29th ACM Symposium on Theory of Computing*, pages 1–10, 1997.

[HKKM97] E. Han, G. Karypis, V. Kumar, and B. Mobasher. Clustering based on association rule hypergraphs. *Technical report, 1997 SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, 1997.

[Hoc82] D. Hochbaum. Heuristic for the fixed cost median problem. *Math Programming*, 22:148–162, 1982.

[HRR98] M. R. Henzinger, P .Raghavan, and S. Rajagopalan. Computing on data streams. *Technical Report 1998-011, Digital Eqipment Corporation, Systems Research Center*, May, 1998.

[Ind99] P. Indyk. Sublinear time algorithms for metric space problems. *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 428–434, 1999.

[JD88] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, 1988.

[JV99] K. Jain and V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. *Proceedings of the Twenty-Ninth Annual IEEE Symposium on Foundations of Computer Science*, 1999.

[JV00] K. Jain and V. Vazirani. An approximation algorithm for the fault tolerant metric facility location problem. *APPROX*, 2000.

[KH63] A. A. Kuehn and M. J. Hamburger. A heuristic program for locating warehouses. *Management Sciences*, 9:643–666, 1963.

[KH79] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems, part ii: p-media ns. *SIAM Journal on Applied Mathematics*, pages 539–560, 1979.

[KLMR98] L. E. Kavraki, J. C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot path planning. *Journal of Computer and System Sciences*, 57:50–60, 1998.

[KPR98] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–10, 1998.

[KPS97] S. Khuller, R. Pless, and Y. Sussmann. Fault tolerant k-center problems. *Third Italian Conference on Algorithms and Complexity (CIAC)*, 1997.

[Kru95] S. O. Krumke. On a generalization of the p center problem. *Information Processing Letters*, 56:67–71, 1995.

[KRY94] S. Khuller, B. Raghavachari, and N. Young. Balancing minimum spanning and shortest path trees. *Algorithmica*, 14(4):305–321, 1994.

[KvEH77] L. Kaufman, M. vanden Eede, and P. Hansen. A plant and warehouse location problem. *Operations Research Quarterly*, 28:547–557, 1977.

[Lov75] L. Lovasz. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, pages 383–390, 1975.

[LV92a] J. H. Lin and J. S. Vitter. Approximation algorithms for geometric median problems. *Information Processing Letters*, 44:245–249, 1992.

[LV92b] J. H. Lin and J. S. Vitter. ϵ-approximations with minimum packing constraint violations. *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, 1992.

[Man64] A. S. Manne. Plant location under economies of scale decentralization and computation. *Math Programming*, 11:213–235, 1964.

[MMP00] Adam Meyerson, Kamesh Munagala, and Serge Plotkin. Cost-Distance: Two metric network design. *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science*, 2000.

[MP80] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, pages 315–323, 1980.

[MP00] R. Mettu and C. G. Plaxton. The onlike median problem. *Proceedings of the 41st IEEE Foundations of Computer Science*, 2000.

[MRL98] G .S .Manku, S. Rajagopalan, and B. Lindsay. Approximate medians and other quantiles in one pass with limited memory. *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 426–435, 1998.

[MRL99] G .S .Manku, S. Rajagopalan, and B. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large databases. *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 251–262, 1999.

[MRS+98] M. V. Marathe, R. Ravi, R. Sundaram, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III. Bicriteria network design problems. *Computing Research Repository: Computational Complexity*, 1998.

[MW88] A. Milo and O. Wolfson. Placement of replicated items in distributed databases. *Proceedings of International Conference on Extending Database Technology*, 1988.

[NH94] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. *Proceedings of the 20'th International Conference on Very Large Data Bases*, 1994.

[Ols93] C. F. Olson. Parallel algorithms for hierarchical clustering. *Technical report, University of California at Berkeley*, 1993.

[Ple87] J. Plesnik. A heuristic for the p-center problem in graphs. *Discrete and Applied Mathematics*, 17:263–268, 1987.

[PY91] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[Ras92] E. Rasmussen. Clustering algorithms. In *Information Retrieval: Data structures and Algorithms*. Prentice–Hall, Englewood Cliffs, NJ, 1992.

[Sal89] G. Salton. *Automatic Text Processing*. Addison–Wesley, 1989.

[Sam89] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1989.

[SCRS97] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian. Buy-at-bulk network design: Approximating the single-sink edge installation problem. *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 619–628, 1997.

[SM83] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* McGraw–Hill Book Company, New York, 1983.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+ tree: a dynamic index for multi-dimensional objects. *Proceedings of the 13th International Conference on Very Large Data Bases*, pages 507–518, 1987.

[ST93] D. B. Shmoys and É. Tardos. Scheduling unrelated machines with costs. *Proceedings of the Fourth Annual SIAM-ACM Symposium on Discrete Algorithms*, pages 448–454, 1993.

[STA97] D. B. Shmoys, É. Tardos, and K. Aardal. Approximation algorithms for facility location problems. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 265–274, 1997.

[Sto61] J. F. Stollsteimer. The effect of technical change and output expansion on the optimum number, size and location of pear marketing facilities in a california pear producing region. *PhD Thesis, University of California, Berkeley*, 1961.

[Sto63] J. F. Stollsteimer. A working model for plant numbers and locations. *Journal of Farm Economics*, 45:631–645, 1963.

[Svi99] M. Sviridenko. Personal communiction. 1999.

[Tam96] A. Tamir. An $o(pn^2)$ algorithm for the p-median problem and related problem on tree graphs. *Operations Research Letters*, 19:59–94, 1996.

[TL84] D. Tcha and B. Lee. A branch-and-bound algorithm for the multi-level uncapacitated location problem. *European J. Oper. Res.*, 18:35–43, 1984.

[TRS97] I. Tatarinov, A. Rousskov, and V. Soloviev. Static caching of web servers. *Proceedings of the Sixth International Conference on Computer Communications and Networks*, 1997.

[Vit85] J. S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.

[vR79] C. J. van Rijsbergen. *Information Retrieval.* Butterworths, London, 1979.

[VRE82] T. Van Roy and D. Erlenkotter. A dual based procedure for dynamic facility location. *Management Sci.*, 28:1091–1105, 1982.

[Web09] A. Weber. A ueber den standort der industrien. *Erster Teil. Reine Theorie der Standorte*, pages 302–309, 1909.

[Wes97] Duanne Wessels. Configuring hierarchical squid caches. *National Laboratory for Advanced Network Research*, 1997.

[Wil88] P. Willett. Recent trends in hierarchical document clustering. *Information Processing and Management*, 24:577–597, 1988.

[WMB94] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabyltes: Compressing and Indexing Documents and Images.* Van Nostrand Reinhold, New York, 1994.

[Wol80] L. A. Wolsey. Heuristic analysis, linear programming and branch and bound. *Mathematical Programming Study*, 13:121–134, 1980.

[ZIRO99] J. Zhang, R. Izmailov, D. Reininger, and M. Ott. Web caching framework: Analytical models and beyond. *IEEE workshop on Internet Applications*, 1999.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 103–114, 1996.