Approximation Algorithms for Min Sum k-Clustering and Balanced k-Median

by

Rohit Sivakumar

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

In this thesis, we consider two closely related clustering problems, Min Sum k-Clustering (MSkC) and Balanced k-Median (BkM). In *Min Sum k-clustering*, one is given a graph and a parameter k, and has to partition the vertices in the graph into k clusters to minimize the sum of pairwise distances between vertices in the same cluster. In the *Balanced k-Median* problem, one is given the same set of input, and has to partition the vertices into k clusters, $C_1, \ldots, C_k$, where each cluster C_i is centered at a node c_i, while minimizing the total assignment costs for the points in the metric; here the cost of assigning a point j to a cluster C_i is equal to $|C_i|$ times the (j, c_j) distance.

The most important contribution of this thesis, described in Chapter 4, is an $O(\log n)$-approximation for both these problems, where n is the number of nodes in the input graph. This improves over the previous best $O(\epsilon^{-1} \log^{1+\epsilon} n)$-approximation by Bartal, Charikar and Raz [1] by a factor of $O(\epsilon^{-1} \log^{\epsilon} n)$. As in the work of Bartal et al., our approximation for general metrics uses probabilistic embeddings into Hierarchically Separated Trees (HSTs). Specifically, we prove that both MSkC and BkM have constant approximation algorithms in HSTs.

Acknowledgements

I would like to thank my supervisor, Mohammad Salavatipour, for his support and guidance over the last two years, and for his patient reviews of various stages of this thesis. I would also like to thank Babak Behsaz and Zachary Friggstad, without whom this work would have not been possible.

I feel deeply indebted to both the staff and the graduate students at the Institute of Mathematical Sciences, Chennai. The hours I spent at their summer school in Theoretical Computer Science has been my first and biggest inspiration to pursue research in algorithms.

I am thankful to my parents for their overwhelming support, and for constantly and unconditionally placing my interests ahead of theirs.

Finally, I would like to thank the Department of Computing Science for financially supporting me during my course of study.

Table of Contents

Chapter 1

Preliminaries

1.1 Motivation

Clustering is a popular research field in the computing world, which aims at partitioning a given set of data into groups such that items in the same group have similar properties. Clustering is also a well-studied problem in the Operations Research community, where it is viewed as an optimization problem with the goal of identifying clusters to minimize a given objective function. In this thesis, we consider two closely related optimization problems, namely *Balanced k-Median* and *Min Sum k-Clustering* and devise approximation algorithms for them.

Given a collection of n points called nodes, the distances between every pair of nodes, and a parameter $k < n$, k-Median refers to the problem of selecting k of the n given nodes as *centers* to minimize the sum of distances from every other node to its closest center. This NP-hard problem has been studied extensively, both by the Operations Research as well as the Theoretical Computer Science communities. The current best known approximation algorithm for this problem is by Byrka et al. [21], who gave a factor $(2.611 + \epsilon)$ approximation earlier this year, for any arbitrary constant $\epsilon > 0$. In a solution to k-Median, a center c is said to serve a node v if c is v's closest open center.

Since the k-Median problem solely aims at minimizing the sum of distances from the nodes to centers, it is possible to have optimal solutions where a large fraction of the n nodes are served by a single center, while other centers serve very few or no other nodes. In real world scenarios like *facility location*, where the centers represent locations where facilities are opened and the remaining nodes represent clients, such solutions can be unpleasant due to the huge imbalance in the load on the open facilities.

The most obvious solution to mitigate these unpleasant scenarios is to modify the k-median objective to place hard capacity constraints on the opened facilities. These constraints typically give an upper bound on the number of nodes each open center can serve. This problem, referred to as the Capacitated k-Median, has been extremely hard to approximate. Even after a long line of research work spanning over fifteen years [16, 17, 18, 19, 20], there does not exist a true approximaton to this problem which meets all the input capacity constraints.

In the main result of this thesis, covered in Chapter 4, we give an approximation algorithm for a problem called *Balanced k-Median*, which penalizes large capacities in the k-Median solution without actually placing hard capacity constraints on the centers. The penalization is done by means of a balancing function on the size of the clusters. We also give auxiliary results for this problem in Chapters 2 and 3. The rest of this chapter provides an introduction to the algorithmic and graph theoretic machinery essential to understanding the chapters that follow.

1.2 Graph Theoretic Fundamentals

A graph G is represented by two sets (V, E) where V is the set of *vertices* (or *nodes*) and E is the set of *edges*. An edge $e \in E$ of a graph is an unordered pair (u, v) (also denoted as uv) where u and v are two distinct vertices in V, also referred to as the *endpoints* of e. Two vertices $u, v \in V$ are *adjacent* if and only if there exists an edge $(u, v) \in E$. An edge $e = (u, v)$ is said to be *directed* if it has an orientation where one of its endpoints is called the *head* of the edge and its other endpoint is called the *tail* of the edge. A graph is said to be *directed* if its edges are directed and is called *undirected* otherwise. In this thesis, we restrict ourselves to problems involving undirected graphs. A *weighted graph* is a graph $G(V, E)$ where each edge $e \in E$ has an associated weight (or cost) given by the function $w : E \to \mathbb{R}$. The weight of an edge can be used to represent features like the amount of resource spent (or earned) while moving from one endpoint of the edge to another.

A $v_1 - v_k$ walk in G is a sequence of vertices $(v_1, v_2, \cdots, v_k)$ such that for any two consecutive vertices v_i and v_{i+1}, the edge $(v_i, v_{i+1}) \in E$ for all $1 \leq i < k$. The cost of the walk is equal to the sum, $\sum_{i=1}^{k-1} w(v_i v_{i+1})$. A $v_1 - v_k$ path in G is a $v_1 - v_k$ walk where each vertex v_i is distinct. The shortest $u - v$ path in G is a path that begins at $u \in V$, ends at $v \in V$, and has the minimum cost. We denote the cost of this path by $c_G(u, v)$. The diameter of the graph G is equal to $\max_{u,v \in V} c_G(u, v)$, the cost of the longest shortest path in G. A cycle in G is a walk $(v_1, v_2, \cdots, v_k)$ where $v_1 = v_k$ and every other vertex $v_i, 2 \leq i < k$, is distinct. A graph is acyclic if it does not contain any cycles. An undirected graph $G(V, E)$ is *connected* if and only if there exists a $u - v$ path between each pair of vertices $u, v \in V$. A tree is a connected acyclic graph. A complete graph $G(V, E)$ is a graph in which every pair of distinct vertices $u, v \in V$ is connected by a unique edge in E. The open neighborhood of a vertex $v \in V$ in the graph G, denoted by $N_o(v)$, is the set of vertices adjacent to V. The closed neighborhood of v, $N_G(v) = N_o(v) \bigcup \{v\}$.

A *hypergraph* H is a pair (V, E) where V is the set of vertices and each $X \in E$ is a non-empty subset of V, called a *hyperedge*. It is a generalization of graphs where an edge can connect any number of vertices. The *size* of every hyperedge X is the cardinality of the set X. A *k-uniform hypergraph* is a hypergraph, all of whose hyperedges have size k. A generic graph we saw earlier, can also be called a *2-uniform hypergraph*.

A graph (or hypergraph) $G(V, E)$, whose vertices V can be partitioned into k disjoint sets $V_1, V_2, \cdots, V_k$ such that no edge $e \in E$ connects vertices from the same partition V_i , $i \in \{1, 2, \cdots, k\}$ is called a *k-partite graph.* When $k = 2$ or 3, these graphs are called *bipartite* and *tripartite*, respectively.

A function $d : V \times V \to \mathbb{R}$ defined over pairs of vertices of a complete graph $G(V, E)$ is called a *metric* if it satisfies the following properties for every triple $u, v, w \in V$.

- $d(u, u) = 0$
- $d(u, v) = d(v, u)$
- $d(u, v) + d(v, w) \geq d(u, w)$

The first property says that a metric cannot have weighted self-loops and the second property implies that the metric is symmetric. The third and most important property of a metric is triangle inequality. According to this property, for any pair (u, w) of vertices, the cost of every $u - w$ path in the graph is at least equal to the weight of the $u - w$ edge in the graph. A graph $G'(V, E')$ is called the *shortest path metric completion* of G if it has the same set of vertices as G and for two vertices $u, v \in V$, the weight of the (u, v) edge in G' is equal to $c_G(u, v)$, the cost of the shortest u, v path in G. It is easy to see that the edge weights in G' satisfy the metric property. The edge weights $d : V \times V \to \mathbb{R}$ of a graph $G(V, E)$ forms a *line metric* if it satisfies the properties $d(u, u) = 0$ and $d(u, v) = d(v, u)$, and its vertices V can be represented by an ordered sequence $v_1, v_2, \cdots, v_n$ such that $d(v_i, v_j) = \sum_{p=i}^{j-1} d(v_i, v_{i+1})$ for each pair of vertices $v_i, v_j \in V$ and $i < j$.

1.3 Approximation Algorithms

Given a set of feasible solutions $\mathcal{F}$ and an objective function $\mathcal{O}$, the goal of an optimization problem is to find a feasible solution $f \in F$ that minimizes (or maximizes) the objective $\mathcal{O}$. Since many natural optimization problems of practical relevance in fields such as *vehicle routing* and *job scheduling* are NP-Hard, these problems cannot be solved optimally and efficiently unless $P = NP$. Under these circumstances, we fall back to the approach of finding near-optimal algorithms to solve these problems. Such near-optimal solutions whose difference from the optimum can be bounded mathematically for every instance of the problem are called *Approximation Algorithms*.

An approximation algorithm $\mathcal{A}$ with a polynomial running time for a problem P has an *approximation ratio* of $\alpha \geq 1$ if, for every instance I of P, the cost $C(I)$ of the solution returned by algorithm $\mathcal{A}$ is within a factor α of the cost $C^*(I)$, of the optimal solution. (i.e) $\max_I(\frac{C(I)}{C^*(I)}, \frac{C^*(I)}{C(I)}) \leq \alpha$.

A randomized polynomial time algorithm $\mathcal{R}$ for a problem P has an *approximation ratio* of $\alpha \geq 1$ if, for every instance I of P, the cost $C(I)$ of the solution returned by algorithm $\mathcal{R}$ is within a factor α of the cost $C^*(I)$ of the optimal solution, in expectation. (i.e) $\mathbb{E}[\max_I(\frac{C(I)}{C^*(I)}, \frac{C^*(I)}{C(I)})] \leq \alpha$. A constant factor approximation algorithm is an approximation algorithm whose approximation ratio is upper bounded by a constant.

A *Polynomial Time Approximation Scheme* (PTAS) is a set of algorithms which takes an instance of an optimization problem and a constant $\epsilon > 0$ and, in polynomial time, produces a solution whose cost is within a factor $(1 + \epsilon)$ of the optimal solution.

An *(α, β) Bi-criterea Approximation* to a problem Π is an approximation algorithm which takes an instance, I of Π, as input and returns a solution $\mathcal{S}$ of cost at most α times the optimum of Π; but this solution $\mathcal{S}$ could exceed a chosen bounded-valued parameter specified by a constraint in Π by a factor of at most β.

An optimization problem is APX-Hard if there is no PTAS for it unless

$P = NP$. In other words, these are problems which cannot have polynomial time approximation algorithms with an approximation ratio less than a constant, $c > 1$. This constant, c, is called the *hardness gap* of the optimization problem.

1.3.1 Linear Programming

Linear Programming is a widely used technique for developing approximation algorithms. An Integer Linear Program (ILP) consists of a linear objective function and a set of linear inequalities called constraints, each containing variables capable of taking values from a discrete integral set. The following set of equations represent a generic ILP formulation of an optimization problem.

$$\begin{aligned}
&\textit{minimize} \sum_{j=1}^{n} c_j x_j && \\
&\textit{subject to} \sum_{j=1}^{n} a_{ij} x_j \geq b_i, && i = \{1, 2, \cdots, m\} \\
&\qquad\qquad x_j \in \{0, 1\} && j = \{1, 2, \cdots, n\}
\end{aligned}$$

The linear programming relaxation of an integer program is obtained by relaxing the constraint that each variable must be 0 or 1 to a weaker constraint that each variable takes real values in the interval $[0, 1]$. Thus, the constraints of the aforestated ILP can be relaxed in the following manner into a linear program (LP).

$$
\begin{aligned}
&\textit{minimize} \sum_{j=1}^{n} c_j x_j \\
&\textit{subject to} \sum_{j=1}^{n} a_{ij} x_j \geq b_i, && i = \{1, 2, \cdots, m\} \\
&x_j \geq 0 && j = \{1, 2, \cdots, n\} \\
&x_j \leq 1 && j = \{1, 2, \cdots, n\}
\end{aligned}
$$

The constraints of a linear program determine the feasible region of the optimization problem. In other words, a given solution $\mathcal{S}$ to a set of linear constraints is feasible if and only if none of the constraints are violated by the assignment $\mathcal{S}$. A well-known property of linear programs is that they can be solved optimally and efficiently. Since the feasible region for an ILP is a subset of the feasible region of its LP relaxation, the optimal value returned by solving the LP is a lower bound on the optimal ILP solution for a minimization problem.

The *integrality gap* between a minimization integer program P and its LP relaxation R_P is defined as $\sup \frac{OPT_P(I)}{OPT_{R_P}(I)}$ where the supremum is taken over all instances I of P and $OPT_P(I)$ and $OPT_{R_P}(I)$ are the optimal solutions to instance I of P and R_P, respectively.

1.3.2 Approximation: An Example

We illustrate some of the concepts seen above with the help of a simple approximation algorithm [24] for the metric k-center problem. The choice of this problem comes from its relatively simple approximation algorithm and its relevance to clustering.

Problem Statement: Given a metric graph $G(V, E)$ with edge weights $d(u, v)$ between vertices u and v and a positive integer k such that $1 \leq k \leq |V|$,

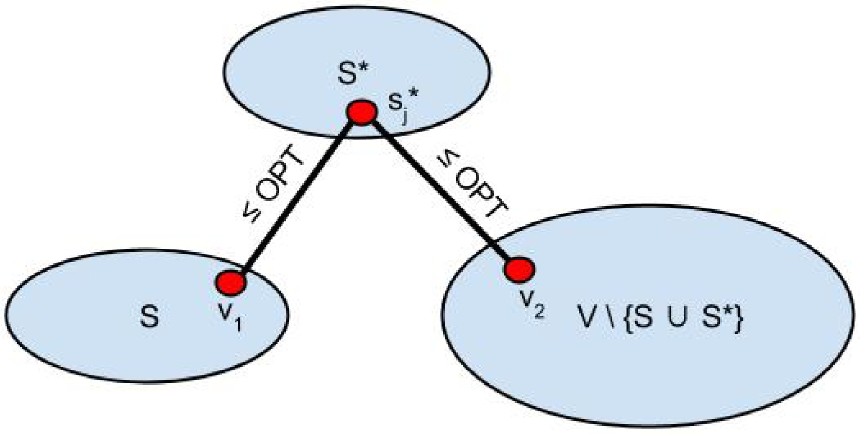


Figure 1.1: A 2-approximate k-center

the objective is to find a subset $S \subseteq V$ such that $|S| = k$ and S minimizes $\max\limits_{v \in V} \min\limits_{s \in S} d(v, s)$.

In the following algorithm by Gonzalez [24], $d(v, S)$ denotes $\min\limits_{s \in S} d(v, s)$ and $arg \max\limits_{v} d(v, S)$ denotes $arg \max\limits_{v} \min\limits_{s \in S} d(v, s)$.

Approximation Algorithm for k-Center

```
1. Start with an arbitrary vertex $s_1 \in V$; $S \leftarrow \{s_1\}$
2. **for** $i \leftarrow 2$ to $k$ **do**
3.         $s_i \leftarrow arg \max\limits_{v} d(v, S)$
4.         $S \leftarrow S \cup \{s_i\}$
5. return $S$
```

Lemma 1 *The above algorithm is a 2-approximation to metric k-center.*

Proof. Let $S^* = \{s_1^*, s_2^*, \cdots, s_k^*\}$ be an optimal set of centers and let $OPT = \max\limits_{u \in V} \min\limits_{s_i^* \in S^*} d(u, s_i^*)$. Consider an arbitrary vertex $v \in V \backslash S$. By pigeon-hole principle, there exist two distinct elements, v_1 and v_2, in the set $S \cup \{v\}$ which are served by the same center, s_j^*, in the solution S^*. Moreover, $|\{v_1, v_2\} \cap S| \geq 1$ by our choice of v_1 and v_2.

If $|\{v_1, v_2\} \cap S| = 2$, then both $d(v_1, s_j^*)$ and $d(v_2, s_j^*)$ are upper bounded by OPT. Therefore, by triangle inequality, $d(v_1, v_2) \leq 2 \cdot OPT$. Without loss of generality, if v_1 was added to S before v_2, then we know by our choice of v_2 that $d(v, S) \leq d(v_2, v_1) \leq 2 \cdot OPT$.

On the other hand, if $|\{v_1, v_2\} \cap S| = 1$, then exactly one of v_1 or v_2 is in our solution S while the other vertex must be v. By metric property, $d(v_1, v_2) \leq d(v_1, s_j^*) + d(v_2, s_j^*) \leq 2 \cdot OPT$ proving that $d(v, S) \leq 2 \cdot OPT$.

From the above arguments, $d(v, S) \leq 2 \cdot OPT$, for every vertex $v \in V \backslash S$. For every other vertex $v' \in S$, it is clear that $d(v', S) = 0$. This validates the claim of our lemma. ■

1.3.3 A Collection of Optimization Problems

The following list contains three NP-hard problems which are referenced in the subsequent chapters of this thesis.

- ***k*-Median:** Given a metric graph, $G(V, E)$ with edge weights $d(u, v)$ and a positive integer $k \geq 1$, the objective of k-median is to find a subset $S \subseteq V$ such that $|S| = k$, to minimize the value of the expression, $\sum\limits_{v \in V} \min\limits_{s \in S} d(u, s)$.

- **Dominating Set:** Given a graph $G(V, E)$, Dominating Set is the problem of finding the smallest cardinality subset $S \subseteq V$ such that $\bigcup\limits_{v \in S} N_G(v) = V$.

- **Three Dimensional-Matching (3DM):** Given a 3-uniform tripartite hypergraph $H(V, E)$, the objective of 3DM is to determine whether or not there exists a subset of hyperedges, $X \subseteq E$ such that $|X| = \frac{|V|}{3}$ and $\bigcup\limits_{e=(u,v,w) \in X} \{u, v, w\} = V$.

Chapter 2

Problem Statements and Initial Results

2.1 Problem Definition

In the Balanced k-Median problem (BkM), we are given a graph $G(V,E)$ with edge weights $d(u,v)$ between vertices $u,v \in V$, and a parameter k. The goal of this problem is to partition the set V into k clusters $C_1, C_2, \cdots, C_k$, with each cluster C_i centered at a distinct vertex $c_i \in C_i$, $i \in \{1, 2, \cdots, k\}$ to minimize the value of $\sum\limits_{i=1}^{k} |C_i| \sum\limits_{j \in C_i} d(c_i, j)$. This objective function penalizes large clusters to give a balanced partition with respect to the cluster sizes.

Much like BkM, in Min Sum k-Clustering (MSkC), we are given a graph $G(V,E)$ and a parameter k. The objective of MSkC is to partition the set V into k clusters, $C_1, C_2, \cdots, C_k$, minimizing the sum of distances between pairs of vertices in the same cluster. i.e: to minimize $\sum\limits_{i=1}^{k} \sum\limits_{u,v \in C_i} d_{u,v}$.

2.2 Previous Work

The Min Sum k-Clustering problem was introduced in 1976 by Sahni and Gonzales [2] who proved the problem to be NP-Hard. This is the dual of the Max k-Cut problem, which involves partitioning the input graph into k clusters with the objective of maximizing the sum of weights of edges between vertices in different clusters. Frieze and Jerrum [10] gave a $\frac{k}{k-1}$ approximation algorithm for the Max k-Cut problem. Kann et al. [11], in 1997 proved that Max k-Cut cannot be approximated better than a factor $1 + \frac{1}{34k}$ unless $P = NP$. In the same paper, the authors prove that MSkC cannot be approximated beyond a factor $O(n^{2-\epsilon})$ for any $k > 2$.

Bartal et al. [1] devised the first approximation algorithm for the Min Sum k-Clustering problem. They gave an $O(\epsilon^{-1} \log^{1+\epsilon} n)$-approximation to both MSkC and BkM in metric spaces for any arbitrary constant $\epsilon > 0$, which was the only approximation algorithm known for the general metric instance of MSkC before our $O(\log n)$-approximation [3] in 2015. In Bartal et. al's paper, the authors also give a $O(1)$ bicriterea approximation for BkM that uses at most $O(k)$ clusters.

In addition to the above results, Min Sum k-Clustering has been further studied in restricted settings. Fernandez de la Vega et al. [9] gave a $(1 + \epsilon)$-approximation algorithm for MSkC with a running time of $O(n^{3k} 2^{\epsilon^{-k^2}})$. BkM can be solved in time $n^{O(k)}$ by guessing the center locations and their capacities, and then finding a minimum-cost assignment from the clients to these centers [12]. The same algorithm can be used to yield a 2-approximation for MSkC when k is regarded as a constant. For the special case that $k = 2$, this approximation ratio was improved to a PTAS by Indyk [13]. For the case where the distances do not form a metric, when $k = 2$, an $O(\sqrt{\log n})$-approximation is known for MSkC as this is just a reformulation of the Minimum Uncut problem [14]. For the case that $k = o(\log n / \log \log n)$, Czumaj and Sohler [6] gave a $(4+\epsilon)$-approximation for MSkC. For MSkC in geometric spaces, Schulman [15] gave an algorithm that runs in time $n^{o(\log \log n)}$, returns a solution of total cost $(1 + \epsilon)$ times optimal, and partitions $(1 - \epsilon)$ factor of all the points optimally.

The Balanced k-Median problem, on the other hand, was first introduced by Bartal et al. in [1] and has only been studied in the context of approximating MSkC in metric spaces [6, 1]. It is known from [1] that the optimal costs of MSkC and BkM on a metric instance are factor 2 from one another. For $k = o(\log n/\log\log n)$, the authors of [6] give a sampling technique for Balanced k-Median, which gives a $(2+\epsilon)$-approximate BkM solution.

A problem closely related to Balanced k-Median is Capacitated k-Median, whose objective is to partition the given graph into k clusters with hard capacities on the cluster sizes, while minimizing the sum of distances from every node in the graph to its cluster center. Much like BkM, this is a well-known clustering problem where clients are not guaranteed to be served by their nearest open center (unlike k-center or k-median). This problem has resisted many approximation techniques. An evidence to the difficulty of the problem is that a natural LP relaxation of the Capacitated k-Median has an unbounded integrality gap even when we are allowed to exceed the number of open facilities by a factor of $(2-\epsilon)$.

The first bi-criterea approximation to this problem was developed by Charikar et al. [16], who gave a 16-approximation algorithm for the case of uniform capacities that violates the capacity constraints by a factor of 4. Chuzhoy and Rabani [17] presented the first approximation algorithm for the case of non-uniform capacity constraints. In their solution, the capacity constraints are voiolated by a factor of at most 50 while the cost of their solution is at most 40 times the optimum. In 2013, Gijswijt and Li [18] developed an algorithm which reduced the number of open facilities to $2k+1$ and their algorithm returns a solution of cost at most $(7+\epsilon)$ times the optimum.

Li [19] reduced the margin of infeasibility for this problem by proposing an algorithm which opens at most $k(1+\epsilon)$ facilities and gives an $e^{O(\frac{1}{\epsilon^2})}$ approximate solution to Capacitated Uniform k-Facility location, a generalization of Capacitated k-Median. Recently, he extended this result to the case where capacities are non-uniform [20].

At the heart of both the Balanced k-Median and the Capacitated k-Median objectives is the problem of k-Median, where we partition the given set of points into k sets $C_1, \cdots, C_k$, each having a center c_i to minimize the total sum of distances of the points to their respective centers. There is a long line of research on this problem within the approximation algorithms community. Some of the recent results are [22, 23, 21], which bring down the approximation ratio to $2.611 + \epsilon$. However, we believe that BkM is much harder to approximate than k-Median because the assignment of clients to centers are not obvious in BkM, even when one is given the location of the open centers.

2.3 Hardness Results

The Min Sum k-Clustering problem was proved to be NP-hard by Sahni and Gonzales [2], when they first introduced it in 1976. The Balanced k-Median problem on the other hand, has only been looked at in [1] and [6], and in both cases, the focus of the authors has been to use BkM as a tool for approximating MSkC. In this section, we prove hardness results for BkM through Theorem 1. We also define a new problem called, the Generalized Balanced k-Median and prove that it is APX-Hard on metric spaces. In the rest of this thesis, we say that a center (or facility) at a location v_f serves (or covers) a client v_c if the vertex v_c belongs to the cluster centered at v_f. Moreover, the total number of clients served by the open facility v_f is called the *capacity* or *multiplier* of the facility.

Theorem 1 *It is NP-hard to find an α-approximate Balanced k-Median for any $\alpha \geq 1$, if the input instance is not a metric.*

Proof. We prove this theorem by a reduction from the *Dominating set* problem. Given an instance of dominating set on an unweighted graph $G(V, E)$, we create a new weighted complete graph $G'(V, E')$ with the same vertex set V. The weights of the edges in E' are assigned as follows:

$$w(e) = \begin{cases} 1 & e \in E \\ (\alpha + 1)n^3 & e \notin E \end{cases}$$

We claim that G has a dominating set of size at most k if and only if the optimal BkM solution in G' has a cost of at most n^3. We also claim that the optimal BkM in G' has a cost of at least $(\alpha+1)n^3$ if the size of the minimum dominating set in G is greater than k. A reduction of this type is called a *Gap Introducing Reduction* and the proof of our claims are as follows:

Suppose that G has a dominating set of size k. Then, there exist k vertices $d_1, d_2, \cdots, d_k \in V$, the union of whose closed neighborhoods, $\bigcup\limits_{i \in 1,\cdots,k} N_G(d_i)$ span the entire graph. By choosing the pair $(d_i, N'_G(d_i))$ in G' as the centers and clusters respectively with ties broken arbitrarily, we get a feasible BkM solution of cost at most $\sum\limits_{i=1}^{k} |N_{G'}(d_i)| \sum\limits_{j \in N_{G'}(d_i)} 1 = \sum\limits_{i=1}^{k} |N_{G'}(d_i)|^2 \leq \sum\limits_{i=1}^{k} n^2 \leq n^3$.

Alternatively, if G does not have a dominating set of size k, then any choice of k centers for BkM in G' will contain a vertex $v \in G'$ at a distance of $(\alpha+1)n^3$ from every open center, incurring a cost equal to at least this value in the optimal BkM solution. This proves the our claim.

As the problem of finding the existence of a dominating set of size k is NP-hard, it is NP-hard to decide whether the optimal BkM solution in G' has a cost less than n^3 or greater than $(\alpha+1)n^3$, thereby proving the inexistence of an α-approximation. ■

Since it is NP-hard to find any α-approximate BkM solution for $\alpha \geq 1$, it follows as a weaker statement of the above theorem that BkM is NP-hard. Moreover, as the cost of any given BkM solution can be verified in polynomial time in a straight-forward way, this problem is clearly in NP, resulting in the following corollary.

Corollary 1 *The decision version of the Balanced k-Median problem is NP-complete.*

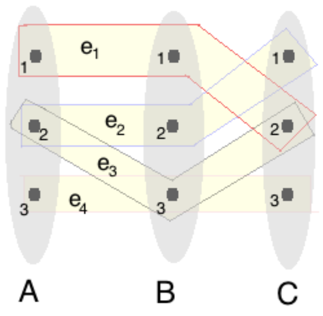
Having established the APX-hardness of BkM, it would be interesting to know if the metric Balanced k-Median problem is APX-Hard (or NP-Hard). To the best of our knowledge, this still remains an open problem. However, we define a new problem called the metric Generalized Balanced k-Median and prove its APX-Hardness in Theorem 2.

In the metric Generalized Balanced k-Median problem (GBkM), we are given a graph $G(V, E)$ with metric edge weights $d(u, v)$ between vertices $u, v \in V$, a parameter k and two sets $F \subseteq V$ and $D \subseteq V$ such that $F \cup D = V$ denoting the set of facilities (or centers) and clients, respectively. The goal of this problem is to partition the set D into k clusters $C_1, C_2, \cdots, C_k$ with each cluster centered at a vertex $c_i \in F$, $i \in \{1, 2, \cdots, k\}$ to minimize the value of $\sum_{i=1}^{k} |C_i| \sum_{j \in C_i} d(c_i, j)$. This problem is identical to BkM if $F = D = V$.

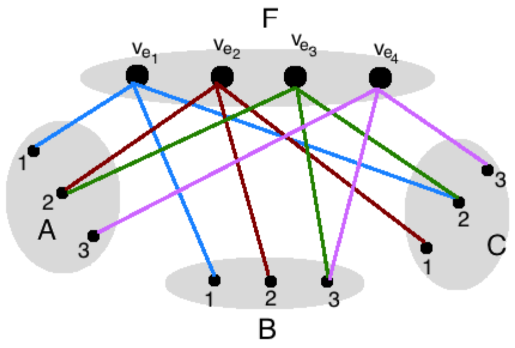
Theorem 2 *Metric GBkM cannot be approximated within a factor $\frac{148}{147}$ unless $P = NP$.*

Proof. We prove this theorem by a reduction from the 3D matching problem. Given a 3-uniform hypergraph $H(V, E)$ on *independent sets* of vertices, A, B and C, each containing n nodes, we construct a new metric graph $G(V', E')$ on $|V| + |E|$ vertices as follows. We copy each vertex in H into G and denote this set of new vertices by D. We also add a set F of new vertices, v_{e_i}, for every edge $e_i \in E$. For each hyperedge $e_i = (a, b, c) \in E$ such that $a \in A$, $b \in B$ and $c \in C$, we add three unit weighted edges (v_{e_i}, a'), (v_{e_i}, b') and (v_{e_i}, c') to the graph G, where a', b' and c' are copies of a, b and c in G, respectively. For each two nodes in F or two nodes in D, we add an edge of weight 2, and for any two non-adjacent nodes, $u \in F$ and $v \in D$, we add an edge of weight 3. Note that the edge weights we assigned form a metric. We consider the GBkM problem on G with $k = n$, by setting F to be the set of locations where facilities can be opened and D to be the set of clients to be served.

Claim 1 *There exists an optimal solution to GBkM on the graph G with $k = n$ of cost $9n$ if the hypergraph H has a 3D matching saturating all its vertices.*



(a) Hypergraph H with 4 labeled edges



(b) Unit-weighted edges of graph G formed by reduction from Figure 2.1a

Figure 2.1: Proof of Theorem 2

Furthermore, if the size of the maximum 3D matching in H is at most αn, then the optimal GBkM solution on G has a cost of at least $15n - 6\alpha n$.

When α is set to be the hardness-gap of the 3D matching problem ($\frac{97}{98}$) [7, 8], this results in a hardness gap (of $\frac{148}{147}$) for BkM.

Proof of Correctness: Suppose that H has a perfect 3D matching $\mathcal{M}$. For each edge $e_i = (a_i, b_i, c_i) \in M$, we open a cluster with the vertex v_{e_i} as center in G and have it serve copies of the clients a_i, b_i and c_i in G. Thus, each cluster in this solution has a capacity of 3 and the total cost of this solution is $\sum\limits_{e=(a,b,c)\in\mathcal{M}} 3(d(v_e, a) + d(v_e, b) + d(v_e, c)) = 9n$.

Proof of Soundness: We claim that there exists a 3D matching of size greater than αn in H if G has a GBkM of cost less than $15n - 6\alpha n$.

In any feasible GBkM solution, $\mathcal{O}$, on the graph G, every client in D is served by exactly one facility, and among all the clients that an open facility $f \in F$ serves, at most three clients located at unit distance from f. Let $\beta > \alpha n$. If $\mathcal{O}$ contains β open facilities, which serve all the three clients at unit distance from it, then, by our construction, the hyperedges in H corresponding to these β facilities form a 3D matching of size β in H. Therefore, if H has a maximum 3D matching of size at most αn, then any feasible solution to GBkM on G can have at most αn open facilities which serve all the three clients at unit distance from it.

In the rest of this proof, we aim to lower bound the cost of the optimal GBkM on G, given that at most αn open facilities can serve all the 3 clients at unit distance from it. We call this new constraint as the *hardness constraint*. Firstly, observe that for two vertices, $f \in F$ and $d \in D$, the weight of the edge (f, d) in G is equal to either 1 or 3, by construction. Every cluster $\mathcal{C}$, in a feasible solution to GBkM, is represented by a duple (p_1, p_3), called its *type*, where p_1 and p_3 represent the number of clients in $\mathcal{C}$ at a distance of 1 and 3 from its center, respectively. The cost of such a cluster type is the contribution of $\mathcal{C}$ towards the cost of the GBkM solution, which equals $(p_1 + p_3)(p_1 + 3 \cdot p_3)$.

A multiset of n type values, $(p_1^1, p_3^1), \cdots, (p_1^i, p_3^i), \cdots, (p_1^n, p_3^n)$ is *permissible* if the following constraints hold.

(a) $p_1^i \leq 3$, for $1 \leq i \leq n$,

(b) $\sum\limits_{i:p_1^i=3} 1 \leq \alpha n$, and

(c) $\sum\limits_{i=1}^{n}(p_1^i + p_3^i) = 3n$,

where the first constraint represents that an open facility can serve at most three clients at unit distance from it, the second constraint denotes the hardness constraint, and the third constaint denotes that every client is served.

It is clear that every feasible solution to GBkM on G, with the hardness constraint, gives us a permissible multiset of n type values. Therefore, the minimum value of $\sum_{i=1}^{n}(p_1^i + p_3^i)(p_1^i + 3 \cdot p_3^i)$ over all permissible type values, lower bounds the optimal cost of GBkM with the hardness constraint. In what follows, we find an optimal multiset of permissible type values to minimize the above expression.

To this end, we start with an arbitrary feasible multiset of type values and prune this set by applying the following rules one after another. Each rule is applied several times to the multiset until the point it cannot be applied any further. Each of these rules, both preserve the permissibility of the multiset and reduce the cost of the objective function. This can be verified easily by computing the difference in cost of the affected type values before and after a rule is applied.

Rule 1 For a type T of the form (p_1, p_3):

- If $p_1 + p_3 \leq 2$, replace T by $(p_1 + p_3, 0)$.
- If $p_1 + p_3 \geq 3$ and $p_1 < 2$, replace T by $(2, p_1 + p_3 - 2)$.

After applying rule 1, notice that the remaining types in our set can be divided into six groups. These are:

(A) $(1, 0)$

(B) $(2, 0)$

(C) $(2, 1)$

(D) $(3, 0)$

(E) $(3, p_3)$, s.t. $p_3 > 0$.

(F) $(2, p_3)$, s.t $p_3 > 1$.

Observe that type values, (p_1, p_3), of the form A and B have $p_1 + p_3 < 3$, while $p_1 + p_3 = 3$ for type values of the form C and D, and $p_1 + p_3 > 3$ for type values of the form E and F. Moreover, since the mean value of p_1+p_3 is equal to 3 in every permissible multiset of type values as $\sum_{i=1}^{n}(p_1^i+p_3^i) = 3n$ and there are n elements in the multiset, elements of type E or F exist if and only if elements of type A or B can be found in our multiset.

Rule 2 For each element of type A and an element (p_1, p_3) of type E or F, replace these by two new elements, $(2, 0)$ and $(p_1, p_3 - 1)$.

Now, we have removed elements of type A completely, while elements of types B through F still remain in our multiset.

Rule 3 For each element of type B and an element (p_1, p_3) of type E or F, replace these by a set of two new elements, $(2, 1)$ and $(p_1, p_3 - 1)$.

After applying Rule 3, elements of type A and B are not present in our solution anymore, which implies that elements of type E and F don't exist either. Moreover, each of the above rules maintain a permissible multiset of type values and reduce the cost of the feasible solution we started with. As the cost of an element of type C is equal to 15, the cost of a type D element is equal to 9, and there are at most αn elements of type D in the new multiset, the optimal cost of permissible type values is at least $15n - 6\alpha n$.

Since this value is a lower bound on the optimal cost of GBkM with the hardness constraint, GBkM on G has a cost of at least $15n - 6 \cdot \alpha n$ if the size of the maximum 3D matching in H is at most αn. This proves both Claim 1 and Theorem 2. ■

2.4 Relation between BkM and MSkC

The two clustering problems, MSkC and BkM, are so closely related that an approximation algorithm for an instance of one of these problems guarantees an approximation algorithm for the other. More specifically, Bartal et al. [1] show that every α-approximate cluster to BkM is a 2α-approximation for

MSkC on the same graph. A stronger statement of this result is contained in the following lemma.

Lemma 2 (Bartal et al.[1]) *Given a graph $G(V,E)$, there exists a feasible partition of the vertices into clusters with a total cost of C under the MSkC objective if and only if G contains a partition $C_1, C_2, \cdots, C_k$ of clusters with centers $c_1, c_2, \cdots, c_k$ such that*

$$\frac{1}{2}\sum_{i=1}^{k}|C_i|\sum_{j\in C_i} d(c_i,j) \leq C \leq \sum_{i=1}^{k}|C_i|\sum_{j\in C_i} d(c_i,j)$$

Proof. From triangle inequality, we know that $d(u,v) \leq d(u,w) + d(w,v)$ for any three vertices $u, v, w \in V$. Therefore,

$$C = \sum_{i=1}^{k}\sum_{u,v\in C_i} d(u,v) \leq \sum_{i=1}^{k}\sum_{u,v\in C_i}(d(u,c_i)+d(c_i,v)) \leq \sum_{i=1}^{k}|C_i|\sum_{j\in C_i} d(c_i,j)$$

If we set $c_i = \underset{u\in C_i}{\operatorname{argmin}} \sum_{v\in C_i} d(u,v)$, we have that

$$\begin{aligned}\frac{1}{2}\cdot C = \frac{1}{2}\sum_{i=1}^{k}\sum_{u,v\in C_i} d(u,v) &= \sum_{i=1}^{k}\sum_{u\in C_i}\sum_{v\in C_i} d(u,v)\\ &\geq \sum_{i=1}^{k}\sum_{c_i\in C_i}\sum_{v\in C_i} d(c_i,v) = \sum_{i=1}^{k}|C_i|\sum_{v\in C_i} d(c_i,v)\end{aligned}$$

■

Using the above result, the focus of research (in [1, 3, 6]) has been on using approximation algorithms for BkM to get algorithms for the MSkC problem. In the next section, we illustrate the techniques used by [1] to achieve this goal.

2.5 An existing approximation for BkM

Since both Bartal et al.'s $O(\frac{1}{\epsilon}\log^{1+\epsilon} n)$-approximation [1] and our $O(\log n)$ approximation [3] for BkM rely heavily on the metric embedding of the given input graph into a special tree structure called Hierarchically Separated Trees (HSTs), we use this section to explain the concept of probabalistic embeddings and HSTs and give a high level overview of the algorithm in [1].

2.5.1 Hierarchically Separated Trees

Definition 1 *[Hierarchically Well Separated Trees [4]] For $\mu > 1$, a μ-*Hierarchical Well Separated Tree *(μ-HST) is a metric space defined on the leaves of a rooted tree T of diameter Δ. Let the level of an internal node in the tree be the number of edges on its path to the root and, for a vertex $u \in T$, let $\Delta(u)$ denote the diameter of the metric restricted to the leaves of the subtree rooted at u. This metric is a μ-HST if the following properties hold:*

- *All edges at a particular level have the same weight.*
- *All leaves are at the same level.*
- *For any internal node u at level i, $\Delta(u) = \Delta \cdot \mu^{-i}$.*

By this definition, any two leaf nodes u, v with least common ancestor w are at distance exactly $\Delta(w)$ from each other. If T is a μ-HST then we let $d_T(u, v)$ denote the distance between u and v in T.

Definition 2 *[Probabilistic Embedding [4]] Given a metric over vertices V of a graph $G(V, E)$ with distances $d(u, v)$ between two vertices $u, v \in V$, we say that the metric is probabilistically embedded into μ-HSTs with a stretch of $f(n)$ if there exists a probability distribution over a family of μ-HSTs where the leaves of each HST are the vertices V and $d(u, v) \leq d_T(u, v)$ for every pair of vertices $u, v \in V$ and for each tree T in the family. Furthermore, $\mathrm{E}_T[d_T(u, v)] \leq f(n) \cdot d(u, v)$, where the expectation is over all the trees T sampled from this distribution.*

The following Theorem is a result from [5].

Theorem 3 *[5] For any integer $\mu > 1$, any metric can be probabilistically embedded into μ-HSTs with stretch $O(\mu \cdot \log_\mu n)$. Furthermore, we can sample a μ-HST from this distribution in polynomial time.*

Fakcharoenphol, Rao and Talwar [5] obtained this result by a randomized hierarchical decomposition of the input metric graph into smaller subgraphs of decreasing diameter. This approach, in $\log_2 \Delta$ iterations, where Δ is the diameter of the input metric results in 2-HSTs. The intermediate nodes in the 2-HST represent the subgraphs obtained by successive decompositions by the algorithm. This procedure, along with Bartal's [4, 1] algorithm to probabilistically embed a 2-HSTs into μ-HSTs with an expected stretch of $O(\frac{\mu}{\log \mu})$ serves as a proof to the above theorem.

The authors of [1] use dynamic programming on HSTs to achieve their approximation guarantee. At a high-level, their algorithm works as follows.

Since all the facilities and clients are located at the leaves of the μ-HST, for a pair u, v of vertices, where u is a facility and v is a client, it suffices to know the height of the least common ancestor of u and v and guess the capacity of u to accurately determine the $u - v$ connection cost. Recall that the *level of a node* in the HST is the number of edges on its path to the root. For every level ℓ node v in the graph, an entry in the dynamic programming table $\mathcal{D}(v, k, s, n_1, n_2, \cdots, n_\ell)$ stores the value of the optimal solution for the subtree of the HST rooted at v, T_v, such that k leaves of T_v are chosen as centers, at most s clients of T_v are connected by facilities outside T_v, and the facilities in T_v serve n_i clients through a least common ancestor located at level i in the tree, for $i \in \{1, 2, \cdots, \ell\}$.

By designing a dynamic program in which the value stored at every cell is consistent with that stored in its neighboring cells (the details of which we omit in this thesis), this dynamic program computes the BkM optimally in time $n^{O(\log_\mu n)}$ because each of the $\log_\mu n$ different entries $n_1, n_2, \cdots, n_\ell$ can take a value of up to n.

By discretizing the values of n_i by rounding them up to their nearest power of α, one can construct a similar dynamic program as above that returns an α-approximate BkM solution on the μ-HST. For each value of v and k, the algorithm computes $(\log_\alpha n)^{\log_\mu n}$ different values, which is equal to $n^{O(\frac{1}{\epsilon})}$ by setting $\mu = O(\log^{\epsilon} n)$ and $\alpha = 1 + \frac{1}{c \cdot \log^2 n}$. The dynamic program now has a polynomial size, and a subset of these are visited before computing the value of each cell, giving an α approximation for BkM on μ-HSTs. When combined with Theorem 3, this result gives an $O(\frac{1}{\epsilon} \log^{1+\epsilon} n)$ approximation for BkM on general metrics. This result is due to Bartal et al. [1].

Chapter 3

Balanced k-Median on the line metric

Despite the fact that Balanced k-Median is NP-Hard in general metrics, it can be solved in polynomial time on the line metric. In the first half of this chapter, we prove structural results for BkM and in the latter half, we show how to exploit these structures to solve BkM exactly on line metrics.

In what follows, we assume that we are given a set of n nodes on a line and denote the vertices as $v_1, v_2, \cdots, v_n$ based on their ordering from left to right on the metric.

3.1 Structural results and laminarity

In this section, we prove some results about the structure of an optimal BkM solution on the line. Note that, Claim 2 and Lemma 3 (which follow) hold even if the given instance is not a metric. As a warm-up, we look at an easier case of the problem where there are only two facilities, whose locations and capacities are known.

Given a graph with n clients, and the locations and capacities of two open facilities, we consider the problem of assigning every client to one of the given facilities to optimize the Balanced k-Median objective. Known techniques such as min-cost flow can solve this problem even if we have up to n facilities with their capacities known. In this thesis, we start by showing that a simple greedy technique gives us an optimal solution for the case of 2 facilities. We will introduce structural properties of optimal solutions, which extend to cases where there are more than two facilities, whose locations and capacities are not known.

Let the two given facilities be denoted by f_1 and f_2 with capacities m_{f_1} and m_{f_2}, respectively. Without loss of generality, we assume that $m_{f_1} \geq m_{f_2}$. Our algorithm simply assigns m_{f_1} clients v, with a minimum value of $m_{f_1} \cdot d(f_1, v) - m_{f_2} \cdot d(f_2, v)$ to the facility f_1 and the remaining clients to f_2.

Claim 2 *The aforementioned algorithm returns an optimal Balanced k-Median cluster.*

Proof. Let us denote the clusters centered at f_1 and f_2 as C_1 and C_2, respectively. The proposed solution has a cost of $\sum_{j \in C_1} |C_1| d(j, f_1) + \sum_{j \in C_2} |C_2| d(j, f_2)$. By adding and subtracting the term $\sum_{j \in C_1} |C_2| d(f_2, j)$, we get

$$\sum_{j \in C_1 \cup C_2} |C_2| d(f_2, j) + \sum_{j \in C_1} (|C_1| d(f_1, j) - |C_2| d(f_2, j)).$$

The first term in the above expression is always a constant for a given instance, and our greedy solution minimizes the second expression by picking the $|C_1|$ smallest values of $m_{f_1} \cdot d(f_1, v) - m_{f_2} \cdot d(f_2, v)$ into the solution. This proves that our solution is optimal. ■

Claim 2 holds even when the underlying graph is not a metric. This result can be extended to the following lemma, which states that in any optimal solution to BkM, if we are given the locations and capacities of any two facilities along with the set of clients that the two facilities serve combined, the greedy solution discussed above finds an optimal assignment of the clients to the two facilities. We skip the proof of this lemma as it is identical to the proof of Claim 2.

Lemma 3 *Given the location of two open facilities, f_1 and f_2, and their respective capacities m_{f_1} and m_{f_2}, along with the set of clients these facilities together cover in an optimal solution to BkM, the strategy of picking clusters from the greedy algorithm of Claim 2 returns an optimal assignment of clients to f_1 and f_2.*

Notice again that this lemma holds even if the given graph is not a metric. For two facilities i and i' opened with capacities c_i and $c_{i'}$, respectively, and a client j, the value $c_i \cdot d(i,j) - c_{i'} \cdot d(i,j)$ denotes the gain in the BkM objective with respect to j if j is served by i as compared to a solution where j is served by i'. We call this value as the *gain of client* j with respect to facilities i and i' and notate it as $g_j^{i,i'}$. The following lemma proves that for any two facilities i and i', the values $g_j^{i,i'}$ increase as we move closer to i, irrespective of the location of i'.

Lemma 4 *Let f_1 and f_2 be two open facilities in any optimum solution to metric BkM serving c_1 and c_2 clients, respectively, such that $c_1 \geq c_2$. If j and j' are any two arbitrary clients such that j lies on the shortest path from j' to f_1, then $g_j^{f_1,f_2} \leq g_{j'}^{f_1,f_2}$. In other words, if both f_1 and f_2 serve exactly one client each from j and j', it is always better for f_1 to serve j and f_2 to serve j'.*

Proof. Let the distance between the two clients j and j' be equal to p. As j lies on the shortest path from f_1 to j', $p = d(f_1, j') - d(f_1, j)$. Since the distance between the two clients is equal to p, by triangle inequality, $|d(f_2, j) - d(f_2, j')| \leq d(j, j') = p$.

Now,

$$\begin{aligned}
g_j^{f_1,f_2} - g_{j'}^{f_1,f_2} &= (c_1 \cdot d(f_1,j) - c_2 \cdot d(f_2,j)) - (c_1 \cdot d(f_1,j') - c_2 \cdot d(f_2,j')) \\
&= c_1(d(f_1,j) - d(f_1,j')) + c_2|d(f_2,j') - d(f_2,j)| \\
&\leq -c_1 \cdot p + c_2 \cdot p \\
&\leq 0,
\end{aligned}$$

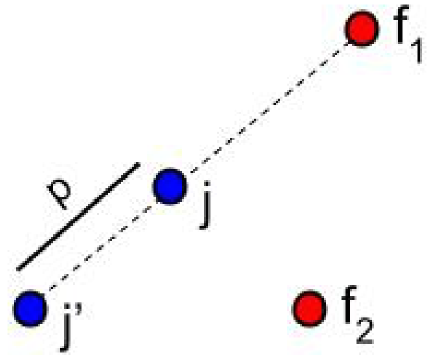


Figure 3.1: Proof of Lemma 4

where the reduction to $-c_1 \cdot p$ uses the fact that $p = d(f_1, j') - d(f_1, j)$ and the reduction to $c_2 \cdot p$ uses the fact that $|d(f_2, j) - d(f_2, j')| \leq p$. Additionally, $-c_1 \cdot p + c_2 \cdot p \leq 0$ because $c_2 \leq c_1$. ■

We can strengthen the above lemma in the case of line metrics in the following way. Consider an optimal solution to BkM on line metrics, where the client assignments conform to the structure observed in Lemma 4. Let f' be an open facility, v_l and v_r be the left most and the right most vertices that f' serves, respectively. We use the term *span(f')* to denote the interval between the clients v_l and v_r, both inclusive. Let f be an open facility with the highest capacity (ties are broken arbitrarily) in the solution. It is clear from Lemma 4 that since f is the facility of the highest capacity, there cannot exist a vertex v_1 between f and v_l which is not served by f. Similarly, there cannot exist a vertex between f and v_r which is not served by f in the optimal solution. Therefore, every client in the interval $[v_l, v_r]$ in the metric is served by f.

From Lemma 4, this structure of the solution recursively follows for the remaining clients and facilities when the facility f along with the clients it serves are removed.

Therefore, no optimal solution for BkM contains two facilities f_1 with capacity c_1 and f_2 with capacity c_2 (with $c_1 \geq c_2$) such that f_2 serves a client in span(f_1).

3.2 An Exact Dynamic Program

Using the results from the previous section, we build a dynamic program to solve BkM exactly on line metrics. In our dynamic programming algorithm, we maintain that the cost of serving a client v by a facility f of capacity c_f, $c_f \cdot d(f, v)$ is charged to the client v.

Each cell of our dynamic programming table is of the form $A[k', i, j, f, c_f, n_f]$ for $1 \leq i \leq j \leq n$. The terms i and j denote the interval between the vertices v_i and v_j. i.e. this includes all vertices $(v_i, v_{i+1}, \cdots, v_j)$. We represent this interval by $[v_i, v_j]$. The cell $A[k', i, j, f, c_f, n_f]$ stores the minimum cost of serving all the clients in $[v_i, v_j]$ either by opening at most k' new facilities that only serve the clients in the interval $[v_i, v_j]$ or by an external facility f of capacity c_f such that f serves at most n_f clients in $[v_i, v_j]$. If there exists no such facility in the subproblem, we replace f by $\perp$ and c_f and n_f by 0. As per our algorithm, f is the facility with the largest capacity among all the open facilities whose span intersects the interval $[v_{i-1}, v_i]$.

We start with the base cases where each interval has exactly one client $i = j$ and fill the table in the increasing order of $j - i$.

Base Cases: $A[k', i, i, f, c_f, n_f]$

- $k' > 0$: We open a new facility at v_i with capacity 1 to serve its collocated client. The value at this cell is equal to 0.
- $k' = 0, f \neq \perp$ and $n_f > 0$: We serve v_i from f. The cost is $c_f \cdot d(v_i, f)$.
- If neither of the above cases hold, then the cost is ∞ because there are no facilities to serve client v_i.

Recursive Cases:
In order to find the value at $A[k', i, j, f, c_f, n_f]$, we look at the following cases and assign this cell the minimum value from among the values set by the respective subproblems.

- $f \neq \perp, n_f > 0$ and f serves v_i:
  If f serves v_i, then f can serve at most $n_f - 1$ clients in the interval $[v_{i+1}, v_j]$. The associated cost follows from the equation:

$$A[k', i, j, f, c_f, n_f] = c_f \cdot d(f, v_i) + A[k', i+1, j, f, c_f, n_f - 1]$$

- If $k' > 0$:
  We guess the location of a new facility f' such that v_i is the left most client it serves, guess $c_{f'}$, the capacity of f', guess j' such that $v_{j'}$ is the right most client that f' serves and guess k'', which is the number of facilities that only serve clients in the interval $[v_{i+1}, v_{j'}]$. In this case, the associated cost of the cell is the minimum cost from all our guesses and is equal to:

$$\begin{aligned} A[k', i, j, f, c_f, n_f] = \min_{f', c_{f'}, j', k''} &(c_{f'} \cdot d(f', v_i) \\ &+ A[k'', i+1, j', f', c_{f'}, n_{f'}] \\ &+ A[k' - k'' - 1, j'+1, j, f, c_f, n_f]) \end{aligned}$$

- If none of the above cases hold then, $k' = 0$ and $n_f = 0$, and the cost is equal to ∞.

At the end of the computation, the optimal cost of BkM is stored in the cell $A[k, 1, n, \perp, 0, 0]$ and the optimal set of clusters along with their centers can be computed by maintaining and iterating through an auxilary data structure which stores the minimum-cost decision at each step of the dynamic program. The only problem with the solution retrieved above is that our *guessing* strategy can lead to multiple facilities being opened at a single vertex. The following paragraph gives a way to remedy this without increasing the cost of the optimal solution.

Given a set of p clients $v'_1, v'_2, \cdots, v'_p$ that are served by a single facility, say f of capacity p, the contribution of the facility f towards the cost of the BkM solution is equal to $p \cdot \sum_{i=1}^{p} d(f, v'_i)$. To find an optimal location for f given the clients, we have to minimize the sum, $\sum_{i=1}^{p} d(f, v'_i)$.

It is easy to see that, if we are given a set of clients in a line, the median of these clients minimizes the sum of absolute deviations from each of the locations. (i.e) f minimizes $\sum_{i=1}^{p} d(f, v'_i)$ if f is chosen to be the median of the nodes $v'_1, v'_2, \cdots, v'_p$.

There exists a non-trivial median if p is odd. However, when p is even, every node f in the interval $[v'_{\frac{p}{2}}, v'_{\frac{p}{2}+1}]$ minimizes the value of $p \cdot \sum_{i=1}^{p} d(f, v'_i)$.

If the DP returns a solution that contains multiple facilities collocated at a single vertex v, we know that at most one of the collocated facilities serves the client at v. Moreover, from the above argument, every facility f at v that does not serve its collocated client serves an even number of clients such that the position of v falls between the 2 medians of the clients served by f. In this case, we simply move the facility f to the location of its nearest collocated client at no additional loss in the cost of the objective function.

3.2.1 Running Time

Each cell of the table is of the form $A[k', i, j, f, c_f, n_f]$. The attributes k', c_f and n_f are restricted to take values between 0 and n while i and j can only take values between 1 and n. The attribute f can either denote a node in the line graph or can be $\perp$; hence f takes $n+1$ different values. The total number of cells in the table is bounded by $O(n^6)$.

For computing the value of each of these $O(n^6)$ cells, the dynamic programming algorithm looks at at most $O(n^4)$ different subproblems: 1 subproblem for the case that f covers i and $O(n^4)$ subproblems for each guess of the variables $f', c_{f'}, k''$ and j' (refer to the algorithm for notations). Therefore, the total running time of the dynamic program is $O(n^{6+4}) = O(n^{10})$, which is clearly polynomial.

The results discussed in this chapter prove the below theorem:

Theorem 4 *The Balanced k-Median problem can be solved in polynomial time on line metrics.*

Chapter 4

BkM on General Metrics

In this chapter, we give an approximation algorithm for the metric Balanced k-Median problem. At a high level, we achieve this goal by first embedding the given metric into 2-HSTs, followed by a dynamic programming algorithm which gives a 2-approximation to BkM on HSTs. The main result of this chapter is the proof of the following theorem.

Theorem 5 *There exists an $O(\log n)$ approximation algorithm to the metric Balanced k-Median problem.*

We begin by introducing a variant of BkM, which we call the Restricted Balanced k-Median (RBkM) problem.

4.1 Reduction to RBkM

In a feasible Balanced k-Median solution, we define the *load* of a cluster C with center v, to be the number of clients C contains. The contribution of the cluster C towards the cost of the BkM solution is equal to $|C| \cdot \sum_{j \in C} d(v, j)$ where the first term, also called *the balancing term* of the objective function is the load of C and the second term represents the sum of distances of the clients in C to its center, v.

We define the Restricted Balanced k-Median (RBkM) to be a variant of Balanced k-Median where *the balancing term* of each cluster's cost function is always a power of 2, i.e. in any optimal RBkM solution, a cluster C with center v contributes a cost of $\mathcal{U}_C \cdot \sum_{j \in C} d(v,j)$ where $\mathcal{U}_C$ is the closest power of 2 greater than or equal to the load of C. Intuitively, the open facility at v is capable of serving $\mathcal{U}_C$ clients, but it only serves $|C|$ clients, where $|C| \leq \mathcal{U}_C$. In what follows, the *capacity* of an RBkM cluster C, denotes the value $\mathcal{U}_C$. The following claim proves how one can convert an approximation algorithm for RBkM into an approximation algorithm for BkM.

Claim 3 *Every α-approximate solution to the minimum cost RBkM is a 2α approximation to BkM on the same graph.*

Proof. Given a metric graph G(V, E) and a parameter k as input, let us denote the optimal cost of RBkM in the given instance by OPT_R and the optimal cost of BkM by OPT_B. Since every feasible solution to RBkM is also a feasible BkM solution, $OPT_B \leq OPT_R$. Furthermore, by taking any optimal BkM solution and rounding up the *balancing term* of its clusters to the nearest power of 2, we get a new feasible RBkM solution of cost at most $2 \cdot OPT_B$. This is because the rounding up operation, in the worst case, doubles the contribution of every cluster's cost.

The existence of a feasible RBkM solution of cost at most $2 \cdot OPT_B$, combined with the fact that $OPT_B \leq OPT_R$ proves the above claim. ■

In section 2.5.1, we saw that any given metric I can be embedded into a μ-HST with an expected stretch of O($\mu \log_\mu n$). This embedding increases the distance between every pair of nodes in I by a factor of $O(\mu \log_\mu n)$, in expectation. Thus, by linearity of expectation, this embedding increases the contribution of an arbitrary cluster C's cost towards the BkM objective in the given metric, $|C| \cdot \sum_{j \in C} d(v,j)$, by a factor of $O(\mu \log_\mu n)$, where v is cluster C's center. Therefore, Fakcharoenphol et al.'s algorithm [5] to approximate the metric I by μ-HSTs increases the cost of the optimal BkM by a factor of at most $O(\log n)$, when μ is regarded as a constant.

Moreover, their algorithm [5] also ensures that the distance between every pair of nodes, u and v in the HSTs obtained by the metric embedding of I is at least equal to the distance between the two points in the metric I. When combined with Claim 3 and the arguments in the preceding paragraph, this proves that, an α approximation algorithm to RBkM on HSTs yields an $\mathrm{O}(\alpha \log n)$ approximation to BkM on the given metric, I.

We show in the rest of this chapter, a dynamic programming framework to solve RBkM exactly on 2-HSTs. We prove some structural results for optimal RBkM solutions in section 4.2. Using these results, we give an intuition to our dynamic program by illustrating its working on full binary 2-HSTs in Section 4.3. This is a hypothetical case modelled to provide the reader with a flavor of the dynamic program since the HST obtained by embedding is not guaranteed to be a full binary tree. The complete algorithm for arbitrary 2-HSTs is described in Section 4.4.

In general, the techniques described in this chapter can be extended to obtain a $\frac{\mu}{\mu-1}$ factor approximation algorithm for BkM in μ-HSTs, by choosing the capacities to be powers of $\frac{\mu}{\mu-1}$. For simplicity of exposition, we confine to the case of 2-HSTs in this thesis.

4.2 Structural Results for RBkM on HSTs

To solve RBkM exactly on 2-HSTs using dynamic programming, we start by demonstrating the existence of an optimal solution with certain helpful structural properties. Let $T = (V, E)$ denote the 2-HST rooted at a vertex $r \in V$. For any vertex $v \in V$, let T_v denote the subtree of T rooted at v and $\Delta(v)$ denote the diameter of T_v. By the properties of HSTs, it is obvious that T_v itself is a 2-HST. A client (or facility) is said to be *located in the subtree* T_v if the vertex in T representing this client (or facility) is a leaf of T_v. In the same vein, a client (or facility) is located outside T_v if it is located in the subtree

$T \backslash T_v$.

We emphasize that all clients are located at the leaf nodes of 2-HSTs and we can only open facilities at leaf nodes. An open facility v_f is of type i if it has a capacity of 2^i. Thus, each client v served by v_f contributes a cost of $2^i \cdot d(v_f, v)$ towards the RBkM solution.

Lemma 5 *In an optimal RBkM solution in HSTs, every facility serves its collocated client.*

Proof. Suppose there exists an optimal RBkM solution that opens a facility at a location $v_f \in V$ but v_f does not serve the client located at v_f. Let w be the deepest node in T such that $v_f \in T_w$ and there is some client $v_c \in T_w$ served by v_f. We close the facility at v_f, open a facility at v_c, and have all the clients previously served by v_f be served by this new facility at v_c. After this operation, the distance from v_c to its serving facility strictly decreases and the distance of all other clients served by this facility remain the same by the properties of 2-HSTs, which contradicts optimality of the solution. ■

We will use the above result to simplify the base cases of our dynamic program. Recall that an open facility has *type* i, if the cluster it serves has a *capacity* of 2^i.

Lemma 6 *In an optimal RBkM solution on* 2*-HSTs and for every vertex v of the tree, there is at most one type of facility in T_v which serves clients located outside T_v. Furthermore, if a facility of type i in T_v serves clients located outside T_v, then every open facility in T_v has type at least i.*

Proof. Let v be a non-root vertex of T (as the lemma is trivial otherwise) and let p_v be its parent node. Suppose there are two open facilities $f_1, f_2 \in T_v$ of types i_1, i_2 respectively such that $i_2 < i_1$. Also, suppose f_1 is serving a client $c_1 \notin T_v$. Note that by Lemma 5, facility f_2 is serving its collocated client. Let u denote the *least common ancestor* of f_1 and c_1. Observe that u lies above v

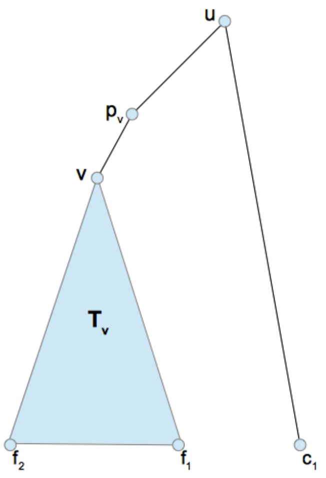


Figure 4.1: Proof of Lemma 6

in the tree. We claim that the modified solution in which f_2 serves c_1 and f_1 serves f_2's collocated client (with other assignments remaining unaffected) has a lower cost. The following inequality formally proves the claim by showing the difference in cost between the modified and initial solutions.

$$\begin{aligned} 2^{i_1} \cdot \Delta(v) + 2^{i_2} \cdot \Delta(u) - 2^{i_1} \cdot \Delta(u) &\leq 2^{i_1} \cdot \Delta(v) - 2^{i_1 - 1} \cdot \Delta(u) \\ &\leq 2^{i_1 - 1} \cdot \Delta(u) - 2^{i_1 - 1} \cdot \Delta(u) \\ &= 0. \end{aligned}$$

The first inequality uses $i_2 + 1 \leq i_1$, and the second one uses the fact that u is at a strictly higher level than v, and by the property of 2-HSTs, $\Delta(v) \leq \Delta(u)/2$. Finally, since f_2 is no longer serving its collocated client, we get a strictly cheaper solution by moving f_2 to its nearest client (as in the proof of Lemma 5), contradicting the optimality of the initial solution. Therefore, any facility that is open in T_v has type at least i_1. The same reasoning also shows that if f_2 is also serving a client $c_2 \notin T_v$ then $i_1 = i_2$. Otherwise, if, say, $i_2 > i_1$ then the new solution that has c_2 served by f_1 and the client collocated with f_1 being served by f_2 would be cheaper (after moving f_1 to its next nearest client). ■

Having proved Lemmas 5 and 6, we now lay out some observations, which are direct results of the above lemmas. These observations are immensely helpful in constructing the dynamic program that we will see in Sections 4.3 and 4.4.

Observation 1 *In an optimal solution to RBkM with two vertices $u, v \in V$ such that T_u and T_v are disjoint, there cannot exist two facilities f_u and f_v and clients c_u and c_v in the subtrees rooted at u and v, respectively, such that f_u serves c_v and f_v serves c_u.*

If this were not the case, we can reduce the cost by swapping the clients and having f_u serve c_u and f_v serve c_v to yield a solution of strictly smaller cost.

Observation 2 *For any feasible solution to RBkM and a vertex v in the tree, if $u, w \in T_v$ are two clients served by two facilities $f_u, f_w \notin T_v$ then the cost of f_u serving u and f_w serving w is the same as the cost of f_u serving w and f_w serving u.*

This is because, for every vertex $v \in T$, all clients and facilities in T_v are equidistant from v by Definition 1.

Observation 3 *For a facility with capacity m_f located at v_f and a client located at v_c, let v_{lca} denote their least common ancestor. Then the cost of serving v_c at v_f is $2 \cdot m_f \cdot d(v_f, v_{lca})$.*

The above observation stems from the fact that $d(v_f, v_{lca}) = d(v_c, v_{lca}) = \frac{d(v_f, v_c)}{2}$, by properties of HSTs. This will be helpful in our dynamic programming algorithm, because, in some sense, it only keeps track of the distance between v_f and v_{lca} for a client v_c served by v_f. For an edge e between v_f and v_{lca}, we call $2 \cdot m_f \cdot d(e)$ the ***actual cost*** of the edge e for the (v_c, v_f) pair, where $d(e)$ is the weight of e in the metric. Note that the sum of the actual costs of edges between v_f and v_{lca} is precisely $m_f \cdot d(v_f, v_c)$. We rely on the actual cost of the edges to find the cost of intermediate solutions in our dynamic program. Below, we give two definitions that will be used in our algorithm for RBkM on HSTs.

Definition 3 *For a subtree T_v of T and any feasible solution to RBkM, we use $cost^{in}_{T_v}$ to refer to the sum of the* actual costs *of edges within T_v accrued due to all the facility-client pairs (v_f, v_c) where $v_f \in T_v$.*

Thus, for any feasible solution to RBkM, $cost^{in}_{T_r}$ is the cost of this solution.

Definition 4 *In any partial assignment of clients to facilities, the slack of a facility f with capacity 2^t is the difference between 2^t and the number of clients that are being served by f. Moreover, for a vertex v in the 2-HST, the* slack *of subtree T_v denotes the sum of slacks of the facilities located in T_v.*

4.3 Dynamic Program for full-binary HSTs

In this section, we present our dynamic programming algorithm under the assumption that the 2-HST obtained by metric embedding is a full binary tree, i.e. each non-leaf node in the HST has exactly two children. As mentioned earlier, this is a hypothetical scenario and we present this first because it is simpler than the general case and still introduces the key ideas behind our algorithm. The general case is more technical and requires two levels of DP; the details appear in Section 4.4. To define a subproblem for the DP, let us consider an arbitrary feasible solution and focus on a subtree T_v, for $v \in T$. We start by defining a few parameters:

- k_v is the number of facilities opened in the subtree T_v.
- t_v denotes the type of the facility, if any, in T_v which serves clients located outside T_v (c.f. Lemma 6). We assign a value of -1 to t_v if no client in $T \backslash T_v$ is served by a facility in T_v.
- u_v is the number of clients in $T \backslash T_v$ that are served by facilities in T_v.
- d_v is the number of clients in T_v that are served by facilities in $T \backslash T_v$ (and)
- o is the slack of T_v.

Each entry in our table is of the form: $\mathcal{A}[v, k_v, t_v, u_v, d_v, o]$. For a vertex $v \in V$, the value stored in this table entry is the minimum of $cost^{in}_{T_v}$ over all possible feasible solutions with parameters k_v, t_v, u_v, d_v, o if the cell denotes a non-pessimal state (defined below) and has a value of ∞ otherwise.

Observation 3 in the previous section gives insight on why it is sufficient to keep track of the d_v values without caring about the type or the location of the facilities outside T_v for calculating the cost of the solution. Our algorithm for RBkM fills the table for all permissible values of parameters v, k_v, t_v, u_v, d_v and o in the decreasing order of the level of the vertex v (from leaf to root). For vertices in the same level, ties are broken arbitrarily.

Pessimal States and Base Cases

An entry of the dynamic programming table is said to be *trivially suboptimal* if it is forced to contain a facility that does not cover its collocated client and is said to be *infeasible* when either the number of clients to be covered or the number of facilities to be opened within a subtree is greater than the total number of nodes in the subtree. We call an entry of the table *pessimal* when it is either infeasible or trivially suboptimal. It is easy to determine the pessimal states in the DP table at the leaf level of the tree. For other subproblems, a cell in the table is pessimal if and only if all its subproblems are pessimal states. For the ease of execution of our DP, we assign a value of ∞ to these cells in our table.

Notice that, at the leaf level of a 2-HST, all the vertices are client nodes. But some of these nodes may also be locations where collocated facilities are opened in a solution. At this stage, the only non-pessimal subproblems are the following:

(a) Facility nodes that correspond to subproblems of the kind $\mathcal{A}[v, 1, t_v, u_v, 0, o]$ satisfying the capacity constraint that $u_v + o + 1 = 2^{t_v}$, where the number 1 indicates the facility's collocated client from Lemma 5 (and)

(b) Client nodes which have subproblems of the form $\mathcal{A}[v, 0, -1, 0, 1, 0]$.

The value stored in these entries is zero.

Recurrence

If a vertex v has two children v_1 and v_2 and the values for the dynamic program are already computed for all subproblems of T_{v_1} and T_{v_2}, then the recurrence we use is given as follows:

$$\begin{aligned} \mathcal{A}[v, k_v, t_v, u_v, d_v, o] = & \min_{k',k'',t_1^*,t_2^*,u_1^*,u_2^*,d_1^*,d_2^*,o_1,o_2} \{\mathcal{A}[v_1, k', t_1^*, u_1^*, d_1^*, o_1] \\ + & \mathcal{A}[v_2, k'', t_2^*, u_2^*, d_2^*, o_2] \\ + & 2 \sum_{i=\{1,2\}, t_i^* \geq 0} 2^{t_i^*} \cdot u_i^* \cdot d(v, v_i)\} \end{aligned} \tag{4.1}$$

where the subproblems in the above equation satisfy the following *consistency constraints*:

1. **Type consistency:** We consider two cases for t_v assuming that $u_v > 0$. If $u_v = 0$, the problem boils down to the case where $t_v = -1$.

   (a) If $t_v = -1$, then no facility in T_v serves clients located in $T \backslash T_v$. Therefore, all the clients served by facilities in T_{v_1} are located either in T_{v_1} or in T_{v_2}. Similarly, for the subtree T_{v_2}, every client served by a facility in T_{v_2} is either located in T_{v_1} or in T_{v_2}. But it is clear from Observation 1 that an optimal solution cannot simultaneously have a facility in T_{v_1} serving a client in T_{v_2} and a facility in T_{v_2} serving a client in T_{v_1}. Hence, $\min(t_{v_1}, t_{v_2}) = t_v = -1$.

   (b) If $t_v \geq 0$, then there exists at least one client in $T \backslash T_v$ that will be served by a facility in T_v. If one of the two subtrees T_{v_1} or T_{v_2}, say T_{v_1} has type $t_{v_1} = -1$, then the type of the other subtree t_{v_2} must be equal to the type of the facility leaving its parent, t_v. Otherwise, if both the values, t_{v_1} and t_{v_2}, are non-negative, Lemma 6 implies that $\min(t_{v_1}, t_{v_2}) = t_v$.

2. **Slack consistency:** The slack of T_v comes from the combined slack of facilities in both its subtrees, T_{v_1} and T_{v_2}. Therefore, $o = o_1 + o_2$.

3. **Consistency in the number of facilities :** k_v is the number of facilities opened in T_v. Since these facilities belong to the subtrees T_{v_1} and T_{v_2}, we must have $k_v = k' + k''$.

4. **Flow consistency:** $u_1^* + u_2^* + d_v = d_1^* + d_2^* + u_v$. This constraint ensures that the subproblems we are looking at are consistent with the u_v and d_v values in hand. More specifically, note that u_1^* is the number of clients in $T \backslash T_{v_1}$ served by facilities in T_{v_1} and that these u_1^* clients can either be located in T_{v_2} or in the subtree $T \backslash T_v$. Let us denote by u_{1a}^*, the number of such clients in $T \backslash T_v$ and by u_{1b}^*, the number of clients in T_{v_2} served by facilities in T_{v_1}. Likewise, let u_{2a}^* be the number of clients in $T \backslash T_v$ and u_{2b}^*, the number of clients in T_{v_1} which are served by facilities in T_{v_2}. It is easy to see that $u_{1a}^* + u_{1b}^* = u_1^*$ and $u_{2a}^* + u_{2b}^* = u_2^*$. Additionally, by accounting for the clients in $T \backslash T_v$ served by facilities in T_v, we have

$$u_v = u_{1a}^* + u_{2a}^* \tag{4.2}$$

Now, out of the d_1^* clients in T_{v_1} and d_2^* clients in T_{v_2} which are served by facilities located outside their respective subtrees, d_v of these clients are served by facilities in $T \backslash T_v$, while the remaining clients $d_1^* + d_2^* - d_v$ must either be served by the u_{1b}^* facilities situated in T_{v_1} and u_{2b}^* situated in T_{v_2}. Hence,

$$d_1^* + d_2^* = d_v + u_{1b}^* + u_{2b}^* \tag{4.3}$$

Summing the Equations (4.2) and (4.3) and from the observation that $u_{1a}^* + u_{1b}^* = u_1^*$ and $u_{2a}^* + u_{2b}^* = u_2^*$, we get the flow constraint stated above.

The last term in Equation (4.1) gives the sum of actual costs of the edges between v and its children for the client-facility pairs where the facility is inside one of the two subtrees T_{v_1} or T_{v_2}. From Definition 3, this value is equal to the difference, $cost^{in}_{T_v} - (cost^{in}_{T_{v_1}} + cost^{in}_{T_{v_2}})$.

The optimal RBkM solution is the minimum value from amongst the entries $\mathcal{A}[r, k, -1, 0, 0, o]$ for all values of o. Note that the values o (the slack variable) can take lie in the interval $[0, n-1]$ in any optimal solution, where n is the number of leaf nodes in T. This is because the slack of any facility is strictly smaller than half its capacity (otherwise we can reduce the type of this facility by 1).

4.4 Dynamic Program for arbitrary HSTs

Now that we have seen a dynamic program for full binary 2-HSTs, we finally look at extending this algorithm to solve RBkM exactly for generic 2-HSTs. For a vertex $v \in T$ with $\ell(v)$ children, we maintain an arbitrary total order of these children and denote them by v_1, v_2, $\cdots$, $v_{\ell(v)}$. For $1 \le i \le \ell(v)$, let $T_{v,i}$ represent the subtree of T induced by the vertices in $\{v\} \cup T_{v_1} \cup T_{v_2} \cup \cdots \cup T_{v_i}$. In addition, let $T_{v,0}$ denote the trivial tree containing the singleton vertex $\{v\}$. In order to define the subproblems of our DP, consider an arbitrary feasible solution and consider an arbitrary v, i and focus on $T_{v,i}$. The following parameters are used to define our subproblems.

- k_v **:** is the number of facilities open in the subtree $T_{v,i}$.
- t_v **:** is the type of facility in the subtree $T_{v,i}$, if any, that serves clients located outside T_v. If there is no such facility, t_v is equal to -1. By Lemma 6, there is a unique value for t_v in every subtree T_v.
- u_v **:** is the number of clients outside the subtree T_v that are covered by type t_v facilities located in $T_{v,i}$.
- d_v **:** Much like the u_v values above, the d_v value represents the number of clients in $T_{v,i}$ that are served by facilities located outside T_v.
- r_v **:** is the number of clients in the subtree $T_v \backslash T_{v,i}$ that are served by facilities located in $T_{v,i}$.

- l_v **:** is the number of clients in the subtree $T_{v,i}$ that are served by facilities located in $T_v \backslash T_{v,i}$.
- o **:** is the slack of $T_{v,i}$.

Each entry in our table is of the form:

$$\mathcal{DP}[v, i, k_v, t_v, u_v, d_v, r_v, l_v, o] \tag{4.4}$$

For a vertex $v \in V$ with $\ell(v) > 0$ children, and any $1 \leq i \leq \ell(v)$, the value stored in this table entry is the minimum of $cost^{in}_{T_{v,i}}$, over all possible feasible solutions with parameters $k_v, t_v, u_v, d_v, r_v, l_v, o$ if the subproblem denotes a non-pessimal state. For a leaf vertex $v \in V$ with $\ell(v) = 0$, the cell $\mathcal{DP}[v, 0, k_v, t_v, u_v, d_v, r_v, l_v, o]$ stores a value of 0 (which is equal to $cost^{in}_{T_{v,0}}$) if the parameters correspond to a non-pessimal state and ∞ otherwise. We discuss this case in detail, later in this section.

High Level Overview

Our algorithm for RBkM populates the table for each subtree $T_{v,i}$, and for every possible value of $k_v, t_v, u_v, d_v, r_v, l_v$ and o, in the decreasing order of the levels of the nodes across levels (from leaf to root) and in the increasing order of the ordering of the children of v for vertices in the same level that share the same parent node. For two vertices in the same level that do not share a common parent, ties are broken arbitrarily. Let the vertex v have $\ell(v)$ children, named as per the total ordering as $v_1, v_2, \cdots, v_{\ell(v)}$.

Note that the number of facilities and clients that are reaching in and out of the tree $T_{v,i}$ will be split across the subtrees $T_{v,i-1}$ and T_{v_i}. Additionally, the only feasible states we will need to look for in the subtree $T_{v,\ell(v)}$ will be the ones with r_v and l_v values of 0.

Pessimal states in the execution of the DP are either infeasible states or trivially suboptimal states, as defined in the case of binary 2-HSTs. These states are dealt with by setting a value of ∞ to the respective cells in our table, in the same vein as our algorithm for the binary case.

If we denote the root of T by v_r, it is clear that in any feasible solution to RBkM on T, all the clients in T are covered by facilities within T. Since we have discretized the values of the capacities on the facilities into $O(\log n)$ buckets, there can be facilities in T which are assigned to fewer clients in T, as compared to their capacities. Therefore, the final solution to RBkM would entail picking the minimum-cost solution from among the values stored in $\mathcal{DP}[v_r, \ell(v_r), k, -1, 0, 0, 0, 0, o]$ for all possible values of o. Again, the value of o is at most $n-1$ because the slack of every facility in any optimal solution is strictly less than half its capacity.

Base Cases

At the leaf level of a 2-HST, all the vertices are client nodes, although some of the clients are also locations where collocated facilities are opened. At this stage, the only subproblems capable of being a part of an optimal solution are:

(a) Facility nodes that correspond to subproblems of the form

$$\mathcal{DP}[v, 0, 1, t_v, u_v, 0, 0, 0, o]$$

satisfying the multiplicity constraint that $u_v + o + 1 = 2^{t_v}$, where the number 1 indicates the facility's collocated client from Lemma 5 (and)

(b) Client nodes which have subproblems of the form

$$\mathcal{DP}[v, 0, 0, -1, 0, 1, 0, 0, 0].$$

These base cases have value zero. Subproblems at the leaf level which do not belong to the above categories are pessimal states.

Computing Table Entries

Assume that we are now considering the subproblem $T_{v,i}$ with appropriate values of the other parameters and we wish to determine the value of $\mathcal{DP}[v, i, k_v, t_v, u_v, d_v, r_v, l_v, o]$, where $1 \leq i \leq \ell(v)$, given that the minimum cost incurred in each of the assignments for all subproblems within T_{v_i} and for various instances of $T_{v,i-1}$ (if $i \geq 2$) are precomputed. We analyze the recursive structure of the dynamic program as two separate cases.

1. **If** $i = 1$ **:** When i equals 1, we are looking at the subtree of T induced by the vertices in $T_{v_1} \cup \{v\}$. As v_1 is the first vertex in the total order of the children of v, the facilities and clients in any feasible solution to the subproblem must come from T_{v_1}. Therefore,

$$\begin{aligned}\mathcal{DP}[v, 1, k_v, t_v, u_v, d_v, r_v, l_v, o] = \mathcal{DP}[&v_1, \ell(v_1), k_v, t^*, u^*, d^*, 0, 0, o] \\ &+ \mathcal{E}_{c_1},\end{aligned}$$

$$\text{where } \mathcal{E}_{c_1} = \begin{cases} 2 \cdot 2^{t^*} \cdot u^* \cdot d(v, v_1) & \text{if } t^* \geq 0 \\ 0 & \text{if } t^* = -1 \end{cases}$$

where t^* denotes the type of the facility in the subtree rooted at v_1 having clients outside this subtree, u^* is the number of clients outside T_{v_1} served by a facility inside T_{v_1} and d^* is the number of clients in T_{v_i} served by facilities located outside this subtree. Moreover, the value of t^*, u^* and d^* can be determined from the following consistency constraints:

   (a) **Facilities located outside** T_{v_1} **:** Consider the set of d^* clients in T_{v_1}, which are served by facilities located outside this subtree. These facilities can be situated either in the subtree $T \backslash T_v$ or in the subtree $T_v \backslash T_{v_1}$. From the table entry considered, we know that there are a total of d_v clients of the former category and l_v clients of the latter. Therefore, $d^* = d_v + l_v$.

   (b) **Clients located outside** T_{v_1} **:**

   - **Case (i):** Suppose $t_v \neq -1$. Then, there must exist u_v clients outside T_v, served by facilities of type t_v located in T_{v_1}. From Lemma 6, there exists a unique type of facility in T_{v_1} serving clients outside this subtree. Also, these facilities serve u_v clients from $T \backslash T_v$ and r_v clients from $T_v \backslash T_{v_1}$. Therefore, $t^* = t_v$ and $u^* = u_v + r_v$.

- **Case (ii) :** If $t_v = -1$, then every client served by a facility in T_{v_1} is either located within this subtree or is located in the subtree $T_v \backslash T_{v_1}$. Hence, $u^* = r_v$.

The last term in the DP recurrence, $\mathcal{E}_{c_1}$ captures the actual cost of the edge (v, v_1) in the feasible solution obtained. This value is equal to $cost^{in}_{T_{v,1}} - cost^{in}_{T_{v_1}}$.

2. **If** $2 \leq i \leq \ell(v)$ **:**

If $i \geq 2$, a feasible solution to the subproblem on $T_{v,i}$ is obtained from feasible solutions to problems on subtrees $T_{v,i-1}$ and T_{v_i}, for various values of the other parameters $k_v, t_v, u_v, d_v, r_v, l_v$ and o. The recurrence, in this case is as follows:

$$\begin{aligned}\mathcal{DP}[v, i, k_v, t_v, u_v, d_v, r_v, l_v, o] = \min_{k', k'', t^*, u'_v, u^*, d'_v, d^*, r'_v, l'_v, o_1, o_2} \{ \\ \mathcal{DP}[v, i-1, k', t_v, u'_v, d'_v, r'_v, l'_v, o_1] \\ + \mathcal{DP}[v_i, \ell(v_i), k'', t^*, u^*, d^*, 0, 0, o_2] \\ + \mathcal{E}_{c_2}\},\end{aligned}$$

$$\text{where } \mathcal{E}_{c_2} = \begin{cases} 2 \cdot 2^{t^*} \cdot u^* \cdot d(v, v_i) & \text{if } t^* \geq 0 \\ 0 & \text{if } t^* = -1 \end{cases}$$

where the k' is the number of open facilities in $T_{v,i-1}$, and k'' is the number of open facilities in the subtree T_{v_i}, subject to the constraint that $k' + k'' = k_v$, t^* denotes the type (can also be -1) of the facility in the subtree rooted at v_i having clients outside this subtree, u^* is the number of clients outside T_{v_i} served by a facility inside T_{v_i}, d^* is the number of clients in T_{v_i} served by facilities located outside this subtree, o_1 and o_2 are the slacks of $T_{v,i-1}$ and T_{v_i} respectively, satisfying $o_1 + o_2 = o$. The other variables in the recurrence statement, u'_v, d'_v, r'_v and l'_v conform to the following consistency constraints:

(a) **Clients located outside** T_v **:** We break the u^* clients outside T_{v_i} that are served by a facility in the subtree into two groups: let u_2^* be the number of clients in $T_v \backslash T_{v_i}$ that are served by a facility inside T_{v_i} and $u_1^* = u^* - u_2^*$ be the rest (which are the clients outside T_v that are served by a facility inside T_{v_i}). Then constraint $u'_v = u_v - u_1^*$ ensures consistency in the number of facilities leaving T_v to serve the associated clients.

(b) **Facilities located outside** T_v **:** As in the above constraint, if we split d^* into d_1^* and d_2^* where d_2^* refers to the number of clients of T_{v_i} served by a facility in $T_v \backslash T_{v,i}$ and d_1^* is the number of the clients of T_{v_i} that are served by a facility outside T_v. Similar to the above constraints, $d'_v = d_v - d_1^*$ is a necessary and sufficient consistency constraint for the facilities located outside $T_{v,i}$ serving clients within the subtree.

(c) **Type Constraint for** T_{v_i} **:** We know from Lemma 6 that every facility in T_v that serves clients in $T \backslash T_v$ is of the same type. Therefore, u_1^* can be greater than zero only when $t^* = t_v$. If $t^* \neq t_v$, then $u_1^* = 0$.

(d) **Flow consistency for the subtree** T_v **:** Consider the total number of facility and client pairs in the trees rooted at the children of v. The condition $r'_v + u_2^* + l_v = l'_v + d_2^* + r_v$ should be met for any feasible solution to RBkM. The proof to this statement is similar to the proof of the flow consistency constraint in the case of full binary 2-HSTs.

The last term in the recurrence, $\mathcal{E}_{c_2}$ gives the sum of actual costs of the edge (v, v_i). From Definition 3, this value is equal to the difference, $cost^{in}_{T_{v_i}} - (cost^{in}_{T_{v_{i-1}}} + cost^{in}_{T_{v_i}})$.

4.4.1 Analysis and Running time

To compute the running time of our algorithm, we start by finding an upper-bound on the size of our table. For a fixed value of v and i, the parameters k_v, d_v, u_v, l_v and r_v cannot exceed n. The slack variable o is at most $n-1$ in any optimal solution. The only remaining variable, t_v, can take a value of at most $\lceil \log_2 n \rceil$, which is by definition. Finally, for any fixed value of v, each i represents a unique index of the children of vertex v from our total ordering of the children of v in T. We know that these children are all in the same level of the tree T. Since the number of nodes in any single level of a rooted tree is at most the number of leaves in the tree, a vertex v in the tree can have at most n children, which is also the number of values i can take.

The number of distinct values of v is equal to the number of nodes in the tree T, which by the algorithm of [5] is at most $n \cdot \log_2 \Delta$, where Δ is the diameter of the original metric. By a losing a factor of $(1+\epsilon)$ on the approximation ratio, standard techniques can scale the distances so that $\Delta = n^2$. Then number of nodes in the tree is at most $2n \cdot \log_2 n = O(n \cdot \log n)$.

From the above arguments, the total number of entries in our DP table is at most $O(n^8 \cdot \log^2 n)$.

Before filling each entry in the DP table, we look at a number of cells based on the values of the variables $k', k'', t^*, u'_v, u^*, d'_v, d^*, r'_v, l'_v, o_1$ and o_2. Moreover, from the consistency constraints, it follows that the variables k' and k'' are dependent. i.e. Setting a value for one variable leaves us with exactly one choice for the other variable to meet the consistency constraints. Therefore, there are at most $O(n)$ guesses needed to set k' and k''. Similarly, the slack o in the dynamic program's recursive equation is divided among o_1 and o_2; thus, a guess on one of these variables gives us exactly one choice to set the other and there are at most $O(n)$ guesses needed to set o_1 and o_2.

Since the type variable t^* can take at most $O(\log_2 n)$ many values, these are the number of guesses that will be needed in the worst case for t^*. As for the other variables ($u'_v, u^*, d'_v, d^*, r'_v$ and l'_v), we infer from the flow constraint ($r'_v + u^*_2 + l_v = l'_v + d^*_2 + r_v$) that these variables are dependent. As each of these six variables are capable of taking a non-negative integral value of at most n, we have n^5 ways to set a value to these variables, while meeting the consistency constraints.

From the above arguments, the total number of pairs of cells in the DP table that we look at to compute a single entry is at most $O(n \cdot n \cdot \log n \cdot n^5) = O(n^7 \log n)$. As there are at most $O(n^8 \cdot \log^2 n)$ entries in the DP table and computing each entry looks at $O(n^7 \log n)$ cells in the table, the running time of the dynamic program is of the order $O(n^{15} \log^3 n)$.

Running Time Improvements

As we just observed, our dynamic programming algorithm has a running time of $O(n^{15} \log^3 n)$. The following lemma aims to reduce both the size of our table and the running time of the DP by a factor of $O(n)$.

Lemma 7 *In each optimal subproblem $\mathcal{DP}[v, i, k_v, t_v, u_v, d_v, r_v, l_v, o]$ of our DP table, one of the two variables r_v and l_v is always equal to* 0 *for an RBkM solution.*

Proof. Let us assume that our HST, T is not binary. Otherwise, the lemma trivially holds from Observation 1.

Let v_1, v_2 and v_3 be three distinct vertices in T that share a common parent v such that the following hold in an optimal RBkM solution:

- The subtree rooted at v_1, T_{v_1}, contains an open facility f_1, which serves a client, c_2, where c_2 is located in T_{v_2}.

- There exists a client c_1 in T_{v_1}, which is served by an open facility, f_3 located in the tree T_{v_3}.

If there do not exist such vertices v_1, v_2 and v_3, then the lemma holds by default. Consider the new feasible solution where the facilities serving c_1 and c_2 are switched. i.e. the facility f_3 serves the client c_2 and the facility f_1 serves client c_1. By Observation 2, the contribution of the cluster centered at f_3 is unaffected by having it serve the client c_2 instead of c_1, but the contribution of the cluster centered at f_1 strictly decreases by at least $2 \cdot d(v_1, v)$, contradicting the optimality of the initial solution.

Therefore, an optimal solution cannot contain a vertex v_1 such that an open facility in T_{v_1} serves a client in $T_v \backslash T_{v_1}$ and a client in T_{v_1} is served by an open facility in $T_v \backslash T_{v_1}$ at the same time. ■

As a direct consequence of the above lemma, the number of guesses on the variables r_v and l_v in the dynamic program reduces from $O(n^2)$ to at most $2 \cdot n$. Instead of storing both r_v and l_v in a DP cell, we gain by storing only the $r_v - l_v$ values, improving the running time of the algorithm to $O(n^{14} \log^3 n)$.

Chapter 5

Conclusion

5.1 Summary

In this thesis, we gave an $O(\log n)$ approximation algorithm for both Min Sum k-Clustering and Balanced k-Median. Intuitively, these results were obtained by repeatedly breaking down MSkC and BkM, and recasting them into other, relatively easier problems at a loss of a bounded approximation factor. In what follows, we enumerate the key steps of the algorithm.

(a) Every instance of metric Min Sum k-Clustering can be transformed into a corresponding instance of metric Balanced k-Median on the same graph, at a loss of an approximation factor of 2, using Lemma 2. Therefore, any approximation algorithm to BkM guarantees an approximation algorithm to MSkC. At this step, we have collapsed the two problems into one single problem, BkM.

(b) By using Theorem 3, we convert an instance of metric BkM into an instance of BkM on 2-HSTs, at a loss of an additional factor of $O(\log n)$ to the approximation ratio. At this stage, an α approximation to BkM on 2-HSTs guarantees an $O(\alpha \log n)$ approximation to both MSkC and BkM on general metrics.

(c) At a further loss of factor 2, we convert the Balanced k-Median problem into Restricted Balanced k-Median (RBkM), a variant of BkM where the capacities of opened facilities are restricted to being powers of 2.

(d) In Chapter 4, we prove that RBkM can be solved exactly on 2-HSTs. Working backwards through the approximation factor we have lost along the way, this gives us an $O(\log n)$ approximation for metric instances of both MSkC and BkM.

This result improves on the previous best $O(\epsilon^{-1}\log^{1+\epsilon} n)$-approximation by Bartal et al. [1] by a factor of $O(\epsilon^{-1}\log^{\epsilon} n)$, where ϵ is any arbitrarily small constant greater than 0.

Besides the above result, we also gave auxiliary results to Balanced k-Median in Chapters 2, 3 and 4. In Chapter 2, we proved that no approximation algorithms exist for Balanced k-Median if the instance is not a metric. We also defined a new problem, the Generalized Balanced k-Median problem and proved that it is APX-Hard, in Theorem 2. In Chapter 3, we showed an exact algorithm for Balanced k-Median on line metrics (cf. Theorem 4). It is interesting to observe that most of the structural results leading to this algorithm hold for general metrics too. Our techniques from Chapter 4, which give a 2-approximation algorithm for BkM on 2-HSTs, can be generalized to give a $\frac{\mu}{\mu-1}$ approximate BkM for μ-HSTs.

5.2 Future Work

In Chapter 2 of this thesis, we proved a generalized version of metric Balanced k-Median (GBkM) to be APX-Hard. Though we believe that the metric Balanced k-Median problem is also APX-Hard, proving this still remains an open problem. Moreover, one can also focus on improving the hardness gap of GBkM.

Our $O(\log n)$ approximation for Balanced k-Median relies heavily on embedding metric instances into HSTs. In fact, we lose a factor of $O(\log n)$ in the approximation ratio at this step, the bound for which was shown to be tight by [5]. Therefore, it is clear that any algorithm that beats the factor of $O(\log n)$

must use vastly different techniques, not involving HSTs.

In Chapter 3, our algorithm for BkM in line metrics leverages on the laminarity of an optimal solution. Although the structural properties leading to an exact algorithm on line metrics hold in the case of arbitrary metric instances as well, these results have been inadequate for us to formulate an approximation algorithm. However, they can be seen as a starting point for future research towards better algorithms for metric BkM.

As with most optimization problems, another avenue towards developing approximation algorithms for these problems is linear programming. Unfortunately, nothing is known about the linear program for BkM. Proving lower and upper bounds on the integrality gap of the linear program as well as developing techniques to round an optimal LP solution are other promising areas for future research.

Bartal et al's [1] approximation technique results in a quasi-polynomial time, $O(\log n)$ approximation algorithm for the Generalized Balanced k-Median problem. However, finding a polynomial time, true approximation for GBkM still remains an open problem. In addition to the above, future work on these problems can also focus on developing lower bounds to approximate BkM and MSkC.

Bibliography

[1] Y. Bartal, M. Charikar and D. Raz. Approximating min-sum k-Clustering in metric spaces. In Proceedings of STOC, 2001.

[2] S. Sahni and T. Gonzalez. P-Complete Approximation Problems. J. of the ACM (JACM), v.23 n.3, p.555-565, July 1976.

[3] B. Behsaz, Z. Friggstad, M. Salavatipour and R. Sivakumar. Approximation Algorithms for Min-Sum k-Clustering and Balanced k-Median. In Proceedings of ICALP, 2015.

[4] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic application. In the Proceedings of the 37^{th} Annual Symposium on Foundations of Computer Science, p. 184-193, 1996.

[5] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In Proceedings of STOC, 2003.

[6] A. Czumaj, C. Sohler. Small Space Representations for Metric Min-Sum k-Clustering and Their Applications. In Proceedings of Symposium on Theoretical Aspects of Computer Science, p. 536-548, 2007.

[7] M. Chlebik, J. Chlebikova. Approximation hardness for small occurrence instances of NP-Hard problems. ECCC TR02-073, 2002.

[8] P. Berman, M. Karpinski. Improved Approximation lower bounds on small occurrence optimization. ECCC TR03-008, 2003.

[9] W. Fernandez de la Vega, M. Karpinski, C. Kenyon and Yuval Rabani. Approximation schemes for clustering problems. In In Proceedings STOC 2003.

[10] A. Frieze and M. Jerrum. Improved approximation algorithms for Max k cCut and Max Bisection. Algorithmica, Volume 18, Issue 1 , pp 67-81, 1997.

[11] V. Kann, S. Khanna, J. Lagergren, and A. Panconessi. On the hardness of max k-cut and its dual. In Israeli Symposium on Theoretical Computer Science, 1996.

[12] N. Guttman-Beck and R. Hassin. Approximation algorithms for min-sum p-clustering. Discrete Applied Mathematics, 89:125-142, 1998.

[13] P. Indyk. A sublinear time approximation scheme for clustering in metric spaces. In Proceedings of FOCS, 1999.

[14] S. Arora, M. Charikar, K. Makarychev and Y. Makarychev. $O(\sqrt{\log n})$-approximation algorithms for Min UnCut, Min-2CNF Deletion, and directed cut problems. In Proceedings of STOC, 2005.

[15] L.J. Schulman. Clustering for edge-cost minimization. In Proceedings of STOC 2000.

[16] M. Charikar, S. Guha, E. Tardos and D.B. Shmoys. A constant-factor approximation algorithm for the k-median problem. In Proceedings of STOC, p 1-10, 1999.

[17] J. Chuzhoy, Y. Rabani. Approximating k-median with non-uniform capacities. In Proceedings of SODA, pp 952-958, 2005.

[18] K. Aardal, P. L. van de Berg, D. Gijswijt and S. Li. Approximation Algorithms for Hard Capacitated k-facility Location Problems. European Journal of Operational Research, 2015.

[19] S. Li. On Uniform Capacitated k-Median beyond the Natural LP Relaxation. SODA, p:696-707, 2015.

[20] S. Li. Approximating capacitated k-median with $(1+\epsilon)k$ open facilities. http://arxiv.org/abs/1411.5630.

[21] J. Byrka, T. Pensyl, B. Rybicki, A. Srinivasan and K. Trinh. An improved approximation for k-median, and positive correlation in budgeted optimization. SODA, p:737-756, 2015.

[22] S. Li and O. Svensson. Approximating k-Median via psuedo-approximation. In the Proceedings of STOC, 2013.

[23] C. Wu, D. Xu, D. Du and Y. Wang. An improved approximation algorithm for k-median problem using a new factor-revealing LP. http://arxiv.org/abs/1410.4161.

[24] T.F. Gonzalez. Clustering to minimize the maximum intercluster distance. Theoretical Computer Science, p:293-306, 1985.