Approximation Algorithms for Min-Sum k-Clustering and Balanced k-Median[*]

Babak Behsaz[†] Zachary Friggstad[‡] Mohammad R. Salavatipour[§]
Rohit Sivakumar[¶]

Department of Computing Science
University of Alberta
Edmonton, Alberta T6G 2E8, Canada

Abstract

We consider two closely related fundamental clustering problems in this paper. In *Min-Sum k-Clustering*, one is given n points in a metric space and has to partition them into k clusters while minimizing the sum of pairwise distances between points in the same cluster. In the *Balanced k-Median* problem, the instance is the same and the objective is to obtain a partitioning into k clusters $C_1, \ldots, C_k$, where each cluster C_i is centered at a point c_i, while minimizing the total assignment cost of the points in the metric; the cost of assigning a point j to a cluster C_i is equal to $|C_i|$ times the distance between j and c_i in the metric.

In this article, we present an $O(\log n)$-approximation for both these problems. This is an improvement over the $O(\epsilon^{-1} \log^{1+\epsilon} n)$-approximation (for any constant $\epsilon > 0$) obtained by Bartal, Charikar, and Raz [STOC '01]. We also obtain a quasi-PTAS for Balanced k-Median in metrics with constant doubling dimension.

As in the work of Bartal et al., our approximation for general metrics uses embeddings into tree metrics. The main technical contribution in this paper is an $O(1)$-approximation for Balanced k-Median in hierarchically separated trees (HSTs). Our improvement comes from a more direct dynamic programming approach that heavily exploits the properties of standard HSTs. In this way, we avoid the reduction to special types of HSTs that were considered by Bartal et al., thereby avoiding an additional $O(\epsilon^{-1} \log^{\epsilon} n)$ loss.

1 Introduction

One of the most ubiquitous problems encountered in computing science is clustering. At a high level, a clustering problem arises when we want to aggregate data points into groups of similar objects. Often, there are underlying metric distances $d(u, v)$ between data points u, v that quantify their similarities. Ideally, we want to cluster the objects into few clusters while ensuring that the distances within a cluster are small.

This paper focuses on two closely related problems, which are referred to in the literature as *Min-Sum k-clustering* (MSkC) and *Balanced k-Median* (BkM). In both problems, we are given a

[*]A preliminary version of this paper appeared in proceedings of ICALP 2015.

[†]Email: `behsaz@ualberta.ca`.

[‡]Email: `zacharyf@cs.ualberta.ca`.

[§]Supported by NSERC. Email: `mreza@cs.ualberta.ca`.

[¶]Email: `rohit2@ualberta.ca`.

metric space over a set of n points, V, which we assume is provided as a graph $G = (V, E)$ with metric distances $d(u, v)$ between any two vertices $u, v \in V$.

The goal of MSkC is to partition the points V into k clusters $C_1, \ldots, C_k$ to minimize the sum of pair-wise distances between points assigned to the same cluster: $\sum_{i=1}^{k} \sum_{\{j,j'\} \subseteq C_i} d(j, j')$. This problem was first introduced by Sahni and Gonzalez [14] and is the complement of the Max k-Cut problem. Bartal et al. [4] gave an $O(\epsilon^{-1} \log^{\epsilon} n)$-approximation for any constant $\epsilon > 0$, for the case of Hierarchically Separated Trees (HSTs), which in turn (using the $O(\log n)$ bound for approximating metrics using HSTs [8]), gives an $O(\epsilon^{-1} \log^{1+\epsilon} n)$-approximation for general metrics. To do this, Bartal et al. introduced BkM, where the input is the same as MSkC, but the goal is to select k points $c_1, \ldots, c_k \in V$ as the centers of the clusters, and partition the nodes V into clusters $C_1, \ldots, C_k$ to minimize $\sum_{i=1}^{k} |C_i| \sum_{v \in C_i} d(v, c_i)$. The multiplier $|C_i|$ in the objective function penalizes clusters for being too large, hence the term *balanced*. As observed by [4], it is easy to show that an α-approximation for either MSkC or BkM implies a 2α-approximation for the other problem in metric graphs. The approximation of [4] for MSkC was obtained by presenting such an approximation for BkM.

1.1 Related Work

The facility location interpretation of BkM leads to a natural generalization of the problem in which we are given a set of clients $C \subseteq V$ and a set of facilities $F \subseteq V$. We need to choose k facilities from F to open and the clients in C must be served by these k facilities. In other words, the set of clients must be partitioned into k clusters and the center assigned to each partition must be chosen from F. Note that C and F can have vertices in common. The special case that $C = F = V$ is the original problem we defined. Going forward, we will use the term "facility" to refer to the center of a cluster in BkM, and the points assigned to this center are referred to as the "clients" that are served by this facility.

The $O(\epsilon^{-1} \log^{1+\epsilon} n)$-approximation [4] was the best approximation for both MSkC and BkM for almost fourteen years. In the same article, the authors also describe a bicriteria $O(1)$-approximation for BkM that uses $O(k)$ clusters. Fernandez de la Vega et al. [9] devised a $(1 + \epsilon)$-approximation for MSkC with a running time of $O(n^{3k} 2^{\epsilon^{-k^2}})$.

BkM and MSkC have been further studied in more restricted settings. BkM can be solved in time $n^{O(k)}$ by "guessing" the location of the centers along with their capacities, and then finding a minimum-cost assignment from the clients to these centers [10]. This yields a 2-approximation for MSkC when k is regarded as a constant. Furthermore, Indyk gave a PTAS [11] for MSkC when $k = 2$.

The factor-2 reduction between BkM and MSkC fails to hold when the distances are not in a metric space. Indeed, one can still solve non-metric instances of BkM in $n^{O(k)}$ time, however for non-metric MSkC, no $n^{2-\epsilon}$-approximation is possible for any constant $\epsilon > 0$ and any $k \geq 3$ [12] unless $\mathbb{P} = \mathbb{NP}$. An $O(\sqrt{\log n})$-approximation for the non-metric MSkC is known for $k = 2$, as this is just a reformulation of the Minimum Uncut problem [1].

These problems have been studied in geometric spaces as well. For point sets in $\mathbb{R}^d$ and a constant k, Schulman [15] gave an algorithm for MSkC that either outputs a $(1+\epsilon)$-approximation, or a solution that agrees with the optimum clustering on $(1 - \epsilon)$-fraction of the points but may have a much larger than optimum cost. Finally, Czumaj and Sohler [7] have developed a $(4 + \epsilon)$-approximation algorithm for MSkC for the case where $k = o(\log n / \log \log n)$ and $\epsilon > 0$ is a constant.

Perhaps the most well studied related problem is the classical k-median where one has to find a partition of the points into k sets $C_1, \ldots, C_k$, each having a center c_i while minimizing the total sum of distances of the points to their respective centers. There is a long line of research on this problem. Some of the most recent results are [13, 5], which bring down the approximation ratio to $2.675 + \epsilon$. It is worth pointing out that both MSkC and BkM seem significantly more difficult than the classical k-median problem. For instance, for the case of k-median if one is given the set of k centers the clustering of the points is immediate as each point will be assigned to the nearest center point; this has been used in a simple local search algorithm that is proved to have approximation ratio $3 + \epsilon$ [2]. However, for the case of BkM, even if one is given the location of k centers it is not clear how to cluster the points optimally.

1.2 Results and Techniques

Our two primary results are $O(\log n)$-approximation algorithms for both BkM and MSkC, improving over their previous $O(\epsilon^{-1} \log^{1+\epsilon} n)$-approximations for any constant $\epsilon > 0$ [4], and a quasi-polynomial time approximation scheme (QPTAS) for BkM in metrics with constant doubling dimension (a.k.a. doubling metrics). Note that this includes Euclidean spaces of constant dimension. Before this work, there were no results known for Euclidean metrics apart from what was known about general metrics and the result in [15] which relies on k being a constant.

Similar to the approximation in [4], our improved $O(\log n)$-approximation for general metrics uses Hierarchically Separated Trees (HSTs), defined formally in Section 2. Specifically, we give a deterministic constant-factor approximation for BkM on HSTs. As is well-known, an arbitrary metric can be probabilistically embedded into an HST with the expected stretch of each edge being $O(\log n)$ [8], thus our algorithm immediately leads to a randomized polynomial time algorithm that computes a solution with expected cost $O(\log n)$ times the optimum solution cost. In fact, our method gives a PTAS for the problem on μ-HST's for any constant μ. To do this, we heavily exploit some specific properties of HSTs in a dynamic programming algorithm that were not used in the previous $O(\epsilon^{-1} \log^{1+\epsilon} n)$-approximation.

The approximation in [4] relied on slightly non-conventional HSTs where the diameters of the subtrees drop by a factor of $O(\log^{\epsilon} n)$, instead of the usual $O(1)$ factor. One can obtain such HSTs with $O(\frac{1}{\epsilon} \log n / \log\log n)$ height which was necessary in order to ensure that their algorithm runs in polynomial time. Our dynamic programming approach is quite different and requires a few observations about the structure of optimal solutions in 2-HSTs. In this way, we avoid dependence on the height of the tree in the running time of our algorithm, thereby obtaining a polynomial-time, constant-factor approximation for 2-HSTs and ultimately, a $O(\log n)$-approximation in general metrics.

Our second result, which is a QPTAS for BkM, is essentially a dynamic programming algorithm which builds on the hierarchical decomposition of a metric space with constant doubling dimensions. We start this by presenting a QPTAS for BkM for the case of a tree metric and show how this can be extended to metrics with constant doubling dimension. This result strongly suggests that the problem is not APX-hard and therefore should have a PTAS.

We also show (see Appendix A) that if the metric is a simple path then BkM can be solved exactly by using some structural properties of an optimal solution on such instances. While the result by itself seems very restricted since the metric is simple, we leverage on some interesting structural properties that hold in general metrics too. Although we were unable to use these structural results to obtain better algorithms for general metrics or improvements in other specific settings (e.g. turning the QPTAS on doubling metrics into a PTAS), these might be useful toward such goals.

As pointed out earlier, some of the standard methods used to obtain constant approximation algorithms for classical k-median seem difficult to adapt for BkM. For instance, even if we are given the location of the k centers, it is not clear how to employ local search to find an approximately optimal assignment of the clients to these centers. We also tried to use LP relaxation for BkM with the hope of obtaining an $O(1)$-approximation for general metrics. One possible way to approach this is to extend the bicriteria approximation of [4]. This bicriteria approximation relies on the correspondence between BkM and a variant of capacitated k-median on a semi-metric space. They then used a Lagrangian relaxation and a primal-dual method to solve the capacitated k-median; the end result though opens $O(k)$ centers. Chuzhoy and Rabani [6] presented a better approximation for capacitated k-median where there are at most k open centers while up to $O(1)$ centers may be open at each location. Adapting their algorithm to work for the semi-metric space resulting from the work of [4] does not seem to work: a key step in bounding the cost of the bipoint solution seems to break down and, further, it seems difficult to combine two solutions obtained from the primal-dual method with $k_1 < k < k_2$ number of centers. If one could overcome this technical difficulty then it could lead to a $O(1)$-approximation for MSkC and BkM on general metrics.

For our algorithms we consider a special case of the BkM problem in which each cluster has a *type* based on rounding up the size of the cluster to the nearest power of $(1+\epsilon)$ for some given constant $\epsilon > 0$; we call this the *ϵ-Restricted Balanced k-median* (RBkM) problem. Here each cluster has one of the types $0, 1, \ldots, \lceil \log_{1+\epsilon} n \rceil$, where n denotes the number of clients, i.e., $n = |C|$. A cluster that is of type i can serve at most $(1+\epsilon)^i$ clients and the cost of serving each client j in a type i cluster with center (facility) c is $(1+\epsilon)^i \cdot d(c,j)$ (regardless of how many clients are served by the facility). We sometimes refer to $(1+\epsilon)^i$ as the capacity or the multiplier of the center (facility) of the cluster. We also say that the center of the cluster and all the clients of that cluster are of type i. It is not hard to see that an α-approximation algorithm for this version results in a $((1+\epsilon)\alpha)$-approximation algorithm for the BkM problem.

Section 2 outlines our approach for the general $O(\log n)$-approximation, including specific definitions of the HSTs we use. The dynamic programming approach for BkM on HSTs is illustrated in Section 3. We present the QPTAS for BkM in doubling metrics in Section 4.

2 An $O(\log n)$-Approximation for General BkM

As noted earlier, our $O(\log n)$-approximation uses embeddings into tree metrics. In particular, we use the fact that an arbitrary metric can be probabilistically approximated by Hierarchically Separated Trees with a distortion of $O(\log n)$. We begin by listing some properties of μ-HSTs that we use in our algorithm.

Definition 1 *For $\mu > 1$, a μ-*Hierarchical Well Separated Tree *(μ-HST) is a metric space defined on the leaves of a rooted tree T. Let the level of an internal node in the tree be the number of edges on the path to the root. Let Δ denote the diameter of the resulting metric space. For a vertex $u \in T$, let $\Delta(u)$ denote the diameter of the subtree rooted at u. Then the tree has the following properties:*

- *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 and v with a least common ancestor w are at a distance of exactly $\Delta(w)$ from each other. If T is a μ-HST then we denote the distance between u and v in T using the notation $d_T(u,v)$. It follows from [8] that for any integer $\mu > 1$, a metric can be probabilistically embedded into μ-HSTs with a stretch of $O(\mu \cdot \log_\mu n)$. Furthermore, a μ-HST from this distribution can be sampled at random in polynomial time.

For an instance of BkM on the μ-HST T, only the leaf nodes of T correspond to clients or cluster centers. We use this property to get a randomized $O(\log n)$-approximation for BkM and MSkC on general metrics.

Note: Our techniques guarantee a PTAS for μ-HSTs for any constant μ by solving the ϵ-RBkM problem exactly for appropriately small values of ϵ, but it is enough to describe a 2-approximation for BkM in 2-HSTs to get an $O(\log n)$-approximation in general metrics. Thus, we focus on this case for simplicity.

3 Dynamic Programming for BkM in 2-HSTs

Recall that in ϵ-RBkM, the capacity of each facility (or the size of each cluster) is rounded up to the nearest power of $1+\epsilon$. For ease of exposition, we focus on the 1-RBkM problem (i.e. where all cluster sizes are powers of two) and present an exact algorithm for this problem on 2-HSTs. Clearly, this implies a 2-approximation for the BkM problem on such graphs. In this section we simply use RBkM to refer to 1-RBkM. We prove the following:

Theorem 1 *RBkM instances in 2-HSTs can be solved in polynomial time.*

To solve RBkM exactly on 2-HSTs using dynamic programming, we start by demonstrating the existence of an optimal solution with certain helpful properties. Let $T = (V, E)$ be a 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. 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 its corresponding vertex in the tree belongs to 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 say that a facility at location v_f serves a client at location v_c if v_c belongs to the cluster centered at v_f. We emphasize that only the leaf nodes of the 2-HST are clients, and facilities can be opened only at leaf nodes. We say that a facility at v_f is of type i if it is opened with a capacity of 2^i. Thus, a client v which is served by v_f is served with a cost of $2^i \cdot d(v_f, v)$.

The following two lemmas are helpful in narrowing our search for the optimum solution.

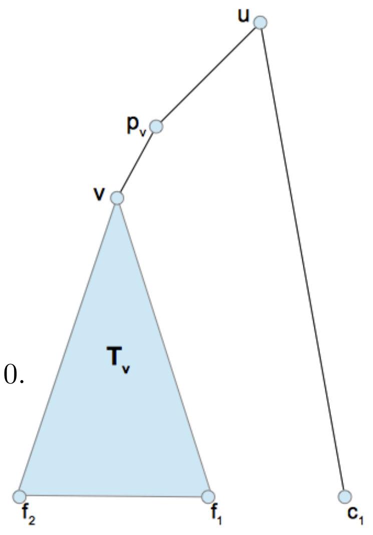
Lemma 1 *In every optimal solution, each opened facility serves its collocated client.*

Proof. Suppose there exists an optimal RBkM solution that opens a facility at location $v_f \in V$ but v_f does not serve the client located at v_f. Let w be the deepest node 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. 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. ■

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

Proof. Let v be a non-root vertex of T 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. Furthermore, suppose f_1 is serving a client $c_1 \notin T_v$.

Note by Lemma 1 that the facility f_2 is serving its collocated client. Let u denote $\mathrm{lca}(f_1, c_1)$ and note that u lies above v in the tree. Suppose, for the sake of contradiction, that $i_2 < i_1$. Consider the modified solution that has f_2 serving c_1 and f_1 serving the client collocated with f_2 (and all other client assignments stay the same). Note that both f_1 and f_2 serve the same number of clients. The change in the solution cost is bounded as follows (see the figure):



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

The first inequality uses $i_2 < i_1$ and the second uses the fact that u is at a strictly higher level than v so, by the property of 2-HSTs, $\Delta(v) \leq \Delta(u)/2$. Finally, since f_2 is no longer serving its collocated client then we get a strictly cheaper solution by moving f_2 to its nearest client (as in the proof of Lemma 1), contradicting optimality of the 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 strictly cheaper (after moving f_1 to its next nearest client). ■

We mention a few more simple observations before describing our recurrence.

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 serving c_u and f_v serving c_v to get a cheaper solution.

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 pairing u with f_u and w with f_w is the same as the cost of pairing u with f_w and w with f_u.*

This is because for every vertex $v \in T$, all clients and facilities in T_v are equidistant from v by Definition 1. For the next observation, recall that all the leaves in T are located at the same level.

Observation 3 *For a facility with multiplier 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 $2 \cdot m_f \cdot d(v_f, v_{lca})$.*

This will be helpful in our 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 $m_f \cdot d(v_f, v_c)$.

Definition 2 *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 3 *In a partial assignment of clients to facilities, the* slack *of a facility f is the difference between its capacity and the number of clients assigned to f. The slack of a subtree T_v rooted at a v is the total slack of facilities in T_v.*

We first present our dynamic programming algorithm under the assumption that the 2-HST is a full binary tree. This cannot be assumed in general, but 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 3.2.

3.1 The Special Case of Full Binary Trees

For ease of understanding, this section provides an overview of the dynamic program for full binary 2-HSTs. Although the 2-HSTs we obtain by metric embedding are not guaranteed to be binary, the idea of the DP is easier to describe for these special cases and we show in the next section, a way to extend the DP to general 2-HSTs. 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 2). 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 table entry 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 feasible solutions with parameters k_v, t_v, u_v, d_v, o if the cell is a non-pessimal state (defined below).

Observation 3 in the previous section provides 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 of 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 for every vertex v in a bottom-up fashion (from leaf to root). For vertices in the same level, ties are broken arbitrarily.

3.1.1 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. An entry is also said to be *infeasible* for one of the following reasons:

- 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.
- If $o > 0$ yet $t_v = -1$; there cannot be any slack in T_v if no facilities are open.

We call an entry of the table *pessimal* when it is either infeasible or trivially suboptimal, or all of its subproblems are pessimal. To be more precise on the recursive part of this statement, if in the recurrence (1) we present below for a particular state at least one of the two subproblems invoked is pessimal, then this state is also pessimal. It is easy to determine the pessimal states in the DP table at the leaf level of the tree. 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 have a collocated facility opened. 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 1 (and)

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

The value stored in these entries are zero.

3.1.2 The Recurrence

If the 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:

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

where the subproblems in the above equation satisfy the following "consistency constraints":
Type consistency: We consider two cases for the type t_v assuming that $u_v > 0$. If $u_v = 0$, the problem boils down to the case where $t_v = -1$.

1. 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 within 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, $\min(t_{v_1}, t_{v_2}) = t_v = -1$.

2. 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. Without loss of generality, if one of the two subtrees, say T_{v_1} has a 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 2 implies that $\min(t_{v_1}, t_{v_2}) = t_v$.

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, $o = o_1 + o_2$.

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

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 $u_{2a}^* + u_{2b}^* = u_2^*$. Also, by accounting for the clients in $T \backslash T_v$ served by facilities in T_v we see

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

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,

$$d_1^* + d_2^* = d_v + u_{1b}^* + u_{2b}^* \tag{3}$$

Summing up the Equations (2) and (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 (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 2, this value is equal to the difference, $cost_{T_v}^{in} - (cost_{T_{v_1}}^{in} + cost_{T_{v_2}}^{in})$.

The optimal RBkM solution is the minimum value from among the entries $\mathcal{A}[r, k, -1, 0, 0, o]$ for all values of o. Note that the number of different values each parameter can take is bounded by the number of nodes in the tree (we assume k is at most the number of leaves, or else the problem is trivial) and the number of recursive calls made to compute a single entry is also polynomially-bounded, so these values can be computed in polynomial time using dynamic programming.

3.2 A Generalized Dynamic Programming Solution for 2-HSTs

In this section, we extend the dynamic program to accommodate generalized 2-HSTs. In HSTs that are not necessarily binary, we still computes the values $\mathcal{A}$ as described in the binary case. However, computing these values for subproblems rooted at a vertex v with multiple children $u_1, \ldots, u_\ell$ requires a more sophisticated approach. For this, we use an "inner" dynamic programming algorithm that, for each $0 \le i \le k$, tracks the movement of clients between $\{T_{u_1}, \ldots, T_{u_i}\}$ and $\{T_{u_{i+1}}, \ldots, T_{u_\ell}\}$, as well as movement in and out of T_v. Using observations like in the binary case, we only have to keep track of the number of clients from a constant number of types.

For a vertex $v \in T$ with $\ell(v)$ children, we maintain an arbitrary total order of these and denote them as $v_1, v_2, \cdots, v_{\ell(v)}$. 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 2, the t_v values are unique for 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:

$$\mathcal{DP}[v, i, k_v, t_v, u_v, d_v, r_v, l_v, o] \tag{4}$$

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.

3.3 High Level Overview

Our algorithm for RBkM populates the table for each subtree $T_{v,i}$ 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, or states where all terms in the recurrence described in Section 3.5 involve at least one pessimal state. 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.

We denote the root of T by v_r. It is clear that in any feasible solution to RBkM on T, the k opened facilities serve all the clients in the tree. However, as we have discretized the values of the multipliers on the facilities into $\mathcal{O}(\log n)$ buckets, there can be facilities in T which are assigned to fewer clients in T, as compared to the value of their multiplier. 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.

3.4 Base Cases

At the leaf level of a 2-HST, every vertex is a client node, although some of these clients are at locations where collocated facilities are opened. At this stage, the only sub-problems capable of being a part of an optimal solution are:

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

(b) Client nodes which have subproblems of the form $\mathcal{DP}[v, 0, 0, -1, 0, 1, 0, 0, 0]$ indicating that there is one vertex to be covered by facilities located in $T \backslash T_v$.

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

3.5 Computing Table Entries

Assume that we are now considering the subproblem $T_{v,i}$ with appropriate values of the other parameters. i.e we want 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 sub-problems 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 in consideration, 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 2, 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 possible 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:

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

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

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 sub-tree.

(c) **Type Constraint for** T_{v_i} **:** We know from Lemma 2 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 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 2, this value is equal to the difference, $cost^{in}_{T_{v_i}} - (cost^{in}_{T_{v_{i-1}}} + cost^{in}_{T_{v_i}})$.

As with the full binary case, the number of possible values each parameter in the recurrence can take is bounded by the number of leaves in the tree. Furthermore, the recurrence makes a polynomial number of calls to smaller subproblems (with either a smaller i or being rooted at a child vertex), so we can compute the optimal solution in polynomial time using dynamic programming.

4 QPTAS for Doubling Metrics

In this section we consider the generalization of BkM where C and F are not necessarily equal and present a QPTAS for it when the input metric has constant doubling dimension. We present an exact quasi polynomial time algorithm for ϵ-RBkM, which implies a $(1+\epsilon)$-approximation for BkM. For simplicity of exposition, we first describe an exact quasi polynomial time algorithm for tree metrics and then extend it to doubling metrics.

4.1 A QPTAS for Tree Metrics

In this section, we present an exact quasi-polynomial time algorithm for the ϵ-RBkM problem on trees. Without loss of generality, we assume the tree is rooted at an arbitrary vertex r. We repeatedly remove leaves with no client or facility until there is no such leaf in the tree. We also repeatedly remove internal vertices of degree two with no client or facility by consolidating their incident edges into one edge of the total length. Also, it is not hard to see that by introducing dummy vertices and zero length edges, we can convert this modified rooted tree into an equivalent binary tree[1] in which the clients and facilities are only located on *distinct* leaves. In other words, each leaf has either a client or a facility. The number of vertices and edges in this binary tree remains linear in the size of the original instance.

Let $p = \lceil \log_{1+\epsilon} n \rceil$. In a solution for the ϵ-RBkM problem, we say a client or facility has type i if it belongs to a type i cluster for some $0 \le i \le p$. We first observe a structural property in an optimal solution of an instance of ϵ-RBkM. Imagine that the clients go to the facilities (the centers of clusters) to recieve some service. We prove that there is an optimal solution in which type i clients either do not enter or do not leave any particular subtree. In other words, in this solution, there are no two clients of the same type such that one enters the subtree to receive service from a facility inside and one leaves the subtree to receive service from a facility outside. To see this, let T_v be the subtree rooted at an arbitrary vertex v, and assume clients j_1 and j_2 have the same type, j_1 is not in T_v but enters this subtree to be served by facility i_1, and client j_2 is in T_v but leaves this subtree to be served by a facility i_2. Then, it is not hard to see that because j_1 and j_2 have the same type, if we send j_1 to i_2 and j_2 to i_1, we get another feasible clustering with no greater cost. Therefore, starting from an optimal solution, one can transform it to a new optimal solution satisfying the above property. We now present a dynamic programming algorithm to compute an optimal solution for the given instance of ϵ-RBkM in quasi-polynomial time.

[1]A tree in which every node other than the leaves has precisely two children

4.1.1 The Table

The table in our dynamic programming algorithm captures "snapshots" of solutions in a particular subtree which includes the information of how many clients of each type either enter or leave this subtree. The subproblems have the form $(v, k', \mathbf{Q})$, where v is a vertex of the tree, $k' \leq k$, and $\mathbf{Q}$ is a vector of length $p+1$ of integers; we describe these parameters below. We want to find the minimum cost solution to cover all the clients in T_v, the subtree rooted at v, such that:

1. There are at most $0 \leq k' \leq k$ open facilities in T_v. These facilities serve clients inside or outside T_v.

2. The clients in T_v are covered by the facilities inside T_v or outside T_v.

3. $\mathbf{Q}$ is a $p+1$ dimensional vector. The ith component of this vector q_i determines the number of type i clients that enter or leave T_v. When $0 \leq q_i \leq n$, q_i is the number of type i clients that enter T_v and when $-n \leq q_i \leq 0$, $|q_i|$ is the number of type i clients that leave T_v.

In a partial solution for the subproblem, the types of clients in T_v and the at most k' facilities to be opened in T_v must be determined. Each client must be assigned to an open facility of the same type in T_v or sent to v to be serviced outside, and each client shipped from outside to v must be assigned to a facility of its type inside T_v. The cost of a partial solution accounts for the cost of sending a client in T_v to a facility inside or to v (i.e., distance to the facility of v times $(1+\epsilon)^i$ where i is the type client) plus, for the clients shipped from outside of T_v to v, the cost of sending them from v to their designated facility in T_v (i.e. the distance from v or the facility times $(1+\epsilon)^i$ where i is the type client). We keep the value of a minimum cost partial solution in table entry $A[v, k', \mathbf{Q}]$. After filling this table, the final answer will be in the entry $A[r, k, \mathbf{0}]$ where $\mathbf{0}$ is a vector with $p+1$ zero components.

Base Case 1: There is a client on v. Then, for each $0 \leq j \leq p$, we do as follows. We form a vector $\mathbf{Q}$ with $p+1$ components such that the ith component $q_i = 0$ for all $i \neq j$ and $q_i = -1$ for $i = j$. Then, we set $A[v, 0, \mathbf{Q}] = 0$. We set all other entries of the form $A[v, ., .]$ to infinity.
Base Case 2: There is a facility on v. Then, for each type $0 \leq j \leq p$ and for each integer $(1+\epsilon)^{j-1} < t \leq (1+\epsilon)^j$, we do as follows. We form a vector $\mathbf{Q}$ with $p+1$ components such that the ith component $q_i = 0$ for all $i \neq j$ and $q_i = t$ for $i = j$. We set $A[v, 1, \mathbf{Q}] = 0$ and all other entries of the form $A[v, ., .]$ to infinity.
Recursive Case: Consider a subtree rooted at a vertex v with two children v_1 and v_2. We say the subproblem corresponding $(v, k', \mathbf{Q})$ is *consistent* with subproblems $(v_1, k'_1, \mathbf{Q}_1)$ and $(v_2, k'_2, \mathbf{Q}_2)$ if $k'_1 + k'_2 \leq k'$ and $\mathbf{Q}_1 + \mathbf{Q}_2 = \mathbf{Q}$.

To find the value of a subproblem $(v, k', \mathbf{Q})$, we initialize $A[v, k', \mathbf{Q}] = \infty$ and enumerate over all subproblems for its children v_1 and v_2. For each pair of consistent subproblems $(v_1, k'_1, \mathbf{Q}_1)$ and $(v_2, k'_2, \mathbf{Q}_2)$, we update the entry to the minimum of its current value and:

$$\sum_{i=1}^{2}(A[v_i, k'_i, \mathbf{Q}_i] + \sum_{j=0}^{p}|q_j^{(i)}| \cdot (1+\epsilon)^j \cdot d(v_i, v)),$$

where $q_j^{(i)}$ is the jth component of $\mathbf{Q}_i$.

Note that the size of the DP table is $O(n^{p+3})$ and we can compute each entry in time $n^{O(p)}$, therefore:

Theorem 2 *There is a QPTAS for the BkM problem on tree metrics.*

4.2 Extension to Doubling Metrics

In this section, we extend the above ideas to give a QPTAS for BkM in doubling metrics by presenting a $(1+\epsilon)$-approximation for ϵ-RBkM. Consider a metric (V, d) defined on a set of vertices V along with distance function d between vertices. Let $B(v, r)$ be the ball of radius r around v, i.e., $B(v, r) = \{u : d(v, u) \leq r\}$. The doubling dimension of (V, d) is the smallest κ such that any $B(v, 2r)$ is contained in the union of at most 2^κ balls of radius r. A metric is called a doubling metric if κ is a constant. For example, a constant dimensional Euclidean metric is a doubling metric.

We need to do some preprocessing steps before running our algorithm. First, we need to bound the aspect ratio of the metric. The aspect ratio of a metric, Δ, is the ratio of the largest distance to the smallest non-zero distance, i.e. $\Delta = \frac{\max_{u,v \in V} d(u,v)}{\min_{u,v \in V, d(u,v) \neq 0} d(u,v)}$. We use a standard scaling technique to bound the aspect ratio. Let OPT be the optimum value of the given instance. We can guess an approximate value λ for OPT such that $\text{OPT} \leq \lambda \leq (1+\epsilon')\text{OPT}$ (for an appropriate ϵ' that depends on ϵ and will be revealed later). We remove all the edges heavier than λ because they are not part of any optimal solution and find the minimum cost clustering for each connected component for all values of $k' \leq k$. With a standard dynamic programming approach the solution of these connected components can be merged into a solution for the instance of ϵ-RBkM at hand.

We show that at a small increase in approximation ratio, the instance defined on each connected component can be transformed to an instance with polynomially bounded aspect ratio. Remember that the distance function d is the shortest path distance in the input graph G between vertices. Let $w(u, v)$ be the weight on an edge between vertices u and v. We change the weights as follows: $w'(u, v) = \lfloor \frac{w(u,v)}{\epsilon' \lambda / n^3} \rfloor$. Since $w(u, v) \leq n\lambda$, we have $\Delta \leq n^4/\epsilon'$. We define the metric based on these new weights and find an approximate solution for it. If we use the original weights $w(u, v)$, the distance of each client to its facility in the approximate solution increases by at most $n \cdot \epsilon' \lambda / n^3$ and because the facility multiplier is at most n, the total increase in cost of a client is $n \cdot \epsilon' \lambda / n^2$. Therefore, the cost of this approximate solution increases by at most $\epsilon' \lambda$ and we only lose a $(1+\epsilon')$ factor by using w'.

Also, we reduce the metric to vertices having a facility or a client located at them. In other words, we remove vertices with no facility or client on them. Then, if a vertex has more than one facility or client located at it, we add some new vertices and zero length edges such that each vertex has either a client or a facility located at it. The above properties help us to simplify the dynamic programming stage of our algorithm.

After these preprocessing steps, we can assume the metric (V, d) is a doubling metric with polynomially bounded aspect ratio Δ. We employ the hierarchical decomposition of metric spaces by means of probabilistic partitioning. The decomposition is essentially the one introduced by Bartal [3], and subsequently used by others, most notably in [8], and most relevantly, for TSP and other problems in doubling metrics, in [16].

The hierarchical decomposition of V is a sequence of partitions of V, where each partition is finer than the previous one. The decomposition can be represented as a tree T (called the *split-tree*), where each node A of T corresponds to a subset of V. The root node corresponds to the single set $\{V\}$ and the leaf nodes correspond to singleton sets $\{\{v\}\}_{v \in V}$. The children of each node A of T correspond to a partition of the set corresponding to A. The union of all subsets corresponding to the vertices at each level in this split-tree constitutes a partition of V.

We give a dynamic programming algorithm that solves subproblems restricted to the points in each set corresponding to a node of T by combining the solutions to the sets corresponding to its children in the split tree. In doing so, we represent a subset of points in each set as portals and require the solution to cross each set of nodes through its portals. This goes along the lines of [16]. More specifically, for two sets S_i and S_j (corresponding to two nodes of T) and two vertices $u \in S_i$ and $v \in S_j$, if a client located at u is assigned to a facility location at v in the solution then we break this into a path which goes from u to a portal of S_i, then to a portal of S_j, and then to v. The approximation scheme follows by showing that: 1) The objective function deteriorates by a factor of at most $1+\epsilon'$, if we require the solution to be portal-respecting, and 2) that the dynamic program can be performed in quasi-polynomial time to find the best portal-respecting solution for an instance of ϵ-RBkM problem.

We use the following theorem on the hierarchical decomposition of doubling metrics:

Theorem 3 ([16]) *There is a hierarchical decomposition of V, i.e. a sequence of partitions $\mathcal{P}_0$, $\mathcal{P}_1, \ldots, \mathcal{P}_h$, where $\mathcal{P}_{i-1}$ is a refinement of $\mathcal{P}_i$, $\mathcal{P}_h = \{V\}$, and $\mathcal{P}_0 = \{\{v\}\}_{v \in V}$. The decomposition has the following properties:*

1. *$\mathcal{P}_0$ corresponds to the leaves and $\mathcal{P}_h$ corresponds to the root of the split-tree T, and height of T is $h = \delta + 2$, where $\delta = \log \Delta$ and Δ is the aspect ratio of metric.*

2. *Each set of nodes S corresponding to a node of T at level i, $S \in \mathcal{P}_i$, has diameter at most 2^{i+1}.*

3. *The branching factor b of T is at most $2^{O(\kappa)}$.*

4. *For any $u, v \in V$, the probability that they are in different sets corresponding to nodes in level i of T is at most $O(\kappa) \cdot \frac{d(u,v)}{2^i}$.*

For each set of nodes S, we introduce a subset $P(S) \subseteq S$ of portals, and require edges to go through the portals to enter and exit S. Talwar [16] shows how one can choose $m = (4\kappa\delta/\epsilon')^\kappa$ portals for each set S such that the expected cost of a portal respecting path between two vertices u and v is at most $(1+\epsilon')d(u,v)$. In other words, if we use these portals, the expected cost of the best portal respecting solution is within $(1+\epsilon')$ factor of optimum. For a constant κ and ϵ', because Δ is polynomially bounded, the number of portals, m, is polylogarithmic which enables us to devise a quasi-polynomial time dynamic programming for finding the best portal respecting solution.

4.2.1 Dynamic Programming

The dynamic programming is similar to the dynamic programming for trees. We define the subproblems on the subtrees rooted at nodes of the split-tree. The table in our dynamic programming algorithm captures "snapshots" of solutions in a particular subtree of T which includes the information of how many clients of each type either enter or leave the portals of this subtree. Using the same argument as in the case of trees, we can show that there is an optimal solution in which the clients of a particular type either enter or leave the portal. The subproblems have the form $(U, k', \mathbf{Q_1}, \mathbf{Q_2}, \ldots, \mathbf{Q_m})$ where U is a set (of the partition) in some $\mathcal{P}_i$. We want to find the minimum cost solution to cover all the clients in U such that there are at most $0 \leq k' \leq k$ open facilities in U and all clients in U are covered, and $\mathbf{Q_i}$ is a $p+1$ dimensional vector that keeps the information of how many of each client type enter or leave the ith portal of U for $1 \leq i \leq m$. The components of each $\mathbf{Q_i}$ have the same meaning as in the case of tree.

In a partial solution for a subproblem, the types of clients in U and the at most k' facilities to be opened in U must be determined. Each client must be assigned to an open facility of the same type in U or to a portal of U to be shipped outside, and each client shipped from outside to a portal of U must be assigned to a facility of its type inside. The cost of a partial solution accounts for the cost of sending a client in U to a facility inside or to a portal of U plus the cost of sending clients shipped from outside of U from the portal of U they have arrived to to their designated facility in U. We keep the value of a minimum cost partial solution in table entry $A[v, k', \mathbf{Q_1}, \mathbf{Q_2}, \ldots, \mathbf{Q_m}]$. After filling this table, the final answer will be in the entry $A[V, k, \mathbf{0}, \ldots, \mathbf{0}]$ where $\mathbf{0}$ is a vector with $p+1$ zero components.

Base case: In the base case, we have a singleton in a leaf of the split-tree. For a singleton, the corresponding set has only one portal and we only need to set $\mathbf{Q_1}$ and we can set the remaining $\mathbf{Q_i}$ to $\mathbf{0}$. Because each singleton has either a facility or client, filling A for base case is almost identical to the case of trees.

Recurrence: For the recursive case, consider the children of a node U and let them be $U_1, \ldots, U_b$ where b is at most $2^{O(\kappa)}$, which is a constant. We say the subproblem corresponding to $(U, k', \mathbf{Q_1}, \mathbf{Q_2}, \ldots, \mathbf{Q_m})$ is *consistent* with subproblems $(U_1, k'_1, \mathbf{Q_1^{(1)}}, \mathbf{Q_2^{(1)}}, \ldots, \mathbf{Q_m^{(1)}})$ through $(U_b, k'_b, \mathbf{Q_1^{(b)}}, \mathbf{Q_2^{(b)}}, \ldots, \mathbf{Q_m^{(b)}})$ if:

1. $k'_1 + \cdots + k'_b \le k'$;
2. $\sum_{i=1}^{m} \sum_{j=1}^{b} \mathbf{Q_i^{(j)}} = \sum_{i=1}^{m} \mathbf{Q_i}$.

To find the value of a subproblem $(U, k', \mathbf{Q_1}, \mathbf{Q_2}, \ldots, \mathbf{Q_m})$, we initialize it with infinity and enumerate all consistent subproblems for its children $U_1, \ldots, U_b$. For each set of consistent subproblems, we update the entry to the minimum of its current value and the minimum value corresponding to this particular set of subproblems. In case of tree, we could compute this value with a closed form expression, but here finding this value is more complicated.

We want to find the value of a consistent set of subproblems of $U_1, \ldots, U_b$ with a subproblem of U. This value is the sum of two main contributors: the total value of subproblems of $U_1, \ldots, U_b$ that can be extracted from table A, and the cost of moving clients between portals of $U_1, \ldots, U_b$ and U. Since clients of type i do not interact with clients of other types, we can find the minimum value for moving clients of each type separately and we use a minimum cost perfect matching algorithm to do that. Consider a particular type i. Create a bipartite graph with bipartitions X_E and X_L. If a portal u has t clients entering it, create t vertices in X_E and if t clients leave this portal, create t vertices in X_L. Set the cost of an edge between any two vertrices corresponding to portals u and v to their distance $d(u, v)$. By the second requirement of consistency, the number of type i clients leaving the portals is equal to the number of type i clients that enter the portals. Therefore, the size of X_E is equal to size of X_L and there is a perfect matching between them. If we find the minimum cost perfect matching in this graph, it gives the best way to move the type i clients between portals of $U_1, \ldots, U_b$ and U.

Analysis: Since the aspect ration Δ is polynomially bounded and $\delta = \log \Delta$, the number of portals $m = (4\kappa\delta/\epsilon')^{\kappa}$ is polylogarithmically bounded for any fixed κ. Therefore, the size of the DP table is quasi-polynomial. Also to compute each entry we have to consider quasi-polynomially many other subproblems. Therefore, the whole algorithm runs in quasi-polynomial time. Considering the factors that we lose in preprocessing, this gives a $(1+\epsilon')^3$-approximation for ϵ-RBkM. We set $\epsilon' = \epsilon/4$ to get a $(1+\epsilon)$-approximation. Combined with the reduction from BkM to ϵ-RBkM it implies:

Theorem 4 *There is a QPTAS for the BkM problem on doubling metrics.*

5 Conclusion

In this paper, we have given an $O(\log n)$-approximation for BkM and MSkC in general metrics, and also a quasi-PTAS for BkM in doubling metrics. Of course, the most natural open problem is to determine if either of these problems admits a true constant-factor approximation in arbitrary metric spaces. A PTAS for BkM in Euclidean metrics or, more generally, doubling dimension metrics seems quite plausible but even obtaining a constant-factor approximation in such cases is an interesting open problem.

Perhaps one direction of attack would be to consider LP relaxation for the problem. It can be shown that the most natural configuration based LP (where we would have a variable $x_{i,C}$ for every possible facility location i and a set C of clients assigned to it) is equivalent to the natural LP relaxation. One of the difficulties of using LP for BkM is that most of the standard rounding techniques that have been used successfully for facility location or the k-median problem (such as filtering, clustering, etc) do not seem to work for the BkM due to the multiplier of cluster sizes. It would be interesting to see if the standard LP relaxation has a constant integrality gap.

References

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

[2] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, V. Pandit. Local Search Heuristics for k-Median and Facility Location Problem. SIAM Journal on Computing, 33:544-562, 2004

[3] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic application. In Proc. of FOCS, 1996.

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

[5] J. Byrka, T. Pensyl, B. Rybicki, A. Srinivasan, and K. Trinh. An Improved Approximation for k-median, and Positive Correlation in Budgeted Optimization. In Proc. of SODA, 2015.

[6] J. Chuzhoy and Y. Rabani. Approximating k-median with non-uniform capacities In Proc. of SODA, 2005.

[7] A. Czumaj and C. Sohler. Small space representations for metric min-sum k-clustering and their applications. In Proc. STACS, 2007.

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

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

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

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

[12] 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.

[13] S. Li and O. Svensson. Approximating k-median via pseudo-approximation. In Proc. of STOC, 2013.

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

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

[16] K. Talwar, Bypassing the embedding: algorithms for low dimensional metrics. In Proc. of STOC, 2004.

A Exact Algorithm for Line Metrics

In this section we present a polynomial time algorithm for BkM when the metric is a simple path (or a line). Recall that V is the set of vertices. A *line distance function* is a distance function $d : V \times V \to \mathbb{Q}_{\geq 0}$ such that $d(v_i, v_j) = \sum_{\ell=i}^{j-1} d(v_\ell, v_{\ell+1})$ for all $1 \leq i \leq j \leq n$. A set of vertices V together with a line distance function defines a line metric. Without loss of generality, assume that the points $v_1, v_2, \ldots, v_n$ are ordered from left to right. We give an exact dynamic programming algorithm for BkM on the line metric. We first need to prove some structural properties.

Assume that the cost of connections (which includes the size multipliers) to facilities are charged to clients. Assume i and i' are two facilities. Among all the feasible solutions where i and i' serve exactly n_i and $n_{i'}$ clients respectively, consider two solutions such that a client j is served by i in one solution and by i' in the other. Let $g_j^{i,i'}$ be the *gain of client* j if we consider the solution in which it is served by facility i' comparing to the solution in which it is served by facility i. In other words, $g_j^{i,i'} = n_i \cdot d(i,j) - n_{i'} \cdot d(i', j)$. Note that in this definition, we only see the cost change from the point of view of client j.

We have the following lemmas about the g-values in *general metrics*:

Lemma 3 *Consider an optimal solution for an instance of the BkM problem in general metrics. Let i and i' be two facilities opened in this solution serving a set of clients C_i and $C_{i'}$, respectively. Then, for any two clients $j \in C_i$ and $j' \in C_{i'}$, we have $g_j^{i,i'} \leq g_{j'}^{i,i'}$.*

Proof. The contribution of these two clusters to the optimal value is

$$\sum_{j \in C_i} |C_i| d(i,j) + \sum_{j' \in C_{i'}} |C_{i'}| d(i', j').$$

We add and subtract the term $\sum_{j' \in C_{i'}} |C_i| d(i, j')$ to this contribution:

$$\sum_{j \in C_i \cup C_{i'}} |C_i| d(i,j) - \sum_{j' \in C_{i'}} (|C_i| d(i,j') - |C_{i'}| d(i', j')) = \sum_{j \in C_i \cup C_{i'}} |C_i| d(i,j) - \sum_{j' \in C_{i'}} g_{j'}^{i,i'}.$$

Note that the first term only depends on the clients in $C_i \cup C_{i'}$ and is independent of how these clients are assigned to i and i'. If for two clients $j \in C_i$ and $j' \in C_{i'}$ $g_j^{i,i'} > g_{j'}^{i,i'}$, we can swap the assignment of j and j' to decrease the second term which is a contradiction. ■

Lemma 4 *Consider any optimal solution to an instance of the BkM problem in general metrics. Let i and i' be two facilities opened by this solution, serving n_i and $n_{i'}$ clients respectively such that $n_i \geq n_{i'}$. If j and j' are two arbitrary clients such that j lies on the shortest path of j' from i, then $g_j^{i,i'} \leq g_{j'}^{i,i'}$.*

Proof. Let $d(i, j') - d(i, j) = \ell$. Because j is on the shortest path of j' from i, we have $\ell \geq 0$ and $d(j, j') = \ell$. By the triangle inequality, $|d(i', j) - d(i', j')| \leq d(j, j') = \ell$ where $|.|$ is absolute value function. We show that $g_j^{i,i'} - g_{j'}^{i,i'} \leq 0$.

$$\begin{aligned} g_j^{i,i'} - g_{j'}^{i,i'} &= n_i d(i,j) - n_{i'} d(i',j) - (n_i d(i,j') - n_{i'} d(i',j')) \\ &= n_i (d(i,j) - d(i,j')) - n_{i'} (d(i',j) - d(i',j')) \\ &\leq -n_i \ell + n_{i'} |d(i',j) - d(i',j')| \leq (n_{i'} - n_i)\ell \leq 0, \end{aligned}$$

where we used the fact that $\ell \geq 0$ and $n_i \geq n_{i'}$. ■

Corollary 1 *For any two facilities i and i' in a solution of the BkM problem in tree metrics, the $g^{i,i'}$ values do not decrease along any simple path starting from i.*

The above corollary together with Lemma 3 result in an interesting structural property in line metrics. We call the segment that connects a client to its facility an *arm.* Let ℓ and r be the indices of the leftmost and the rightmost clients served by a facility respectively. We call the set $\{v_\ell, v_{\ell+1}, \ldots, v_r\}$, the *span* of this facility.

Lemma 5 *To every instance of the BkM problem on line metrics, there lies an optimal solution in which, for any two opened facilities i and i', if the number of clients served by i, n_i, is greater than or equal to the number of clients served by i', $n_{i'}$, then facility i' does not serve any client in span of i.*

Proof. Consider any optimal solution and order the open facilities in this solution as follows. A facility serving more clients than another facility precedes it. If two facilities serve the same number of clients, the facility with lower index precedes the other facility. These two rules define a unique order. Assume facility i precedes facility i' in this order. By Lemma 3, facility i must serve clients with n_i smallest $g^{i,i'}$-values among the clients served by i and i'. Among all optimal solutions having the same set of open facilities and same number of clients served by each one of these facilities, pick the one in which whenever there is a choice between two clients having the same $g^{i,i'}$-value, the one with smaller index is assigned to i. We claim this solution has the desired property.

Consider two arbitrary facilities i and i' such that $n_i \geq n_{i'}$. There are two cases: either $n_i > n_{i'}$ or $n_i = n_{i'}$. Assume that $n_i > n_{i'}$. By Corollary 1, the $g^{i,i'}$-values do not decrease as we move away from i. Therefore, facility i' does not serve any client in span of i, because otherwise, facility i serves a client with $g^{i,i'}$-value less than or equal to a client with greater index served by i, which is a contradiction to the way we picked the optimal solution. Now assume that $n_i = n_{i'}$ and without loss of generality, assume that i has a smaller index than i'. In this case , the n_i clients assigned to i are the clients with n_i smallest indices from among the clients served by i and i' (because they have the n_i smallest $g^{i,i'}$-values). Therefore, the spans of these facilities are disjoint. ∎

Using a dynamic programming algorithm, we can find an optimal solution having the property of Lemma 5. Let $A(k', i, j, f, n_f, \ell)$ be the minimum cost to serve $\{v_i, v_{i+1}, \ldots, v_j\}$ by opening k' new facilities such that $f = \perp$ or f is a facility. If $f = \perp$, we have $n_f = \ell = 0$. If f is a facility and $\ell > 0$, among all facilities that their spans contain $\{v_i, v_{i+1}, \ldots, v_j\}$, f must have the smallest size span such that

1. f serves n_f clients;

2. f serves ℓ clients in $\{v_i, v_{i+1}, \ldots, v_j\}$.

The optimum value can be found in $A[k, 1, n, \perp, 0, 0]$. We use the following recurrence to fill A. In what follows, by *guessing* a parameter, we mean that we try all *polynomially* many possible values of that parameter and if one of them results in a feasible solution, we update the value of the current subproblem (i.e. , take the minimum of the current value and the newly found value).
Base Case. Set $A(k', i, i, f, n_f, \ell) = n_f \cdot d(f, i)$ and $A(k', i, i, f, n_f, 0) = \infty$ for all k', i, f, n_f, and $\ell > 0$.
Recursive Case. To find $A(k', i, j, f, n_f, \ell)$, there are two cases. Guess whether f covers i or not. If f covers i, we must have $f \neq \perp$, $\ell > 0$, and:

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

If f does not cover i, guess the facility f' that covers i, guess $n_{f'}$ the number of clients f' serves, guess j' the rightmost vertex f' serves and guess k'' the number of facilities that only serve clients in $\{v_i, v_{i+1}, \ldots, v_{j'}\}$. We have

$$A(k', i, j, f, n_f, \ell) = A[k'', i, j', f', n_{f'}, n_{f'}] + A[k' - k'' - 1, j' + 1, j, f, n_f, \ell],$$

where we assume $j' < j$. If $j' = j$, we must have $\ell = 0$ and the second term must be omitted.

It is not hard to see that we should fill the elements of A in order of increasing $j - i$. After filling the table A, an optimal solution can be constructed in the standard way from the values in these tables. This results in the following theorem:

Theorem 5 *There is an exact algorithm for the BkM problem on line metrics.*