

2003

# Faster Algorithms for $k$ -Median Problem on Trees with Smaller Heights

Rahul Shah

Report Number:  
03-030

---

Shah, Rahul, "Faster Algorithms for  $k$ -Median Problem on Trees with Smaller Heights" (2003). *Computer Science Technical Reports*. Paper 1579.  
<http://docs.lib.purdue.edu/cstech/1579>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact [epubs@purdue.edu](mailto:epubs@purdue.edu) for additional information.

**FASTER ALGORITHMS FOR K-MEDIAN PROBLEM  
ON TREES WITH SMALLER HEIGHTS**

**Rahul Shah**

**Department of Computer Sciences  
Purdue University  
West Lafayette, IN 47907**

**CSD TR #03-030  
September 2003**

# Faster algorithms for $k$ -median problem on trees with smaller heights

Rahul Shah\*

Purdue University

**Abstract.** The  $k$ -median problem is one of the classical problems in location theory and clustering. It has been well-studied for over the decades. Given a distance metric on  $n$  nodes, the task is to find the location of  $k$  median points (or facilities) so that the sum of the distances from each node to its nearest median is minimized. We apply some recent techniques in improving tree dynamic programming to the case of unweighted and weighted  $k$ -median problem on trees. The previous known best bound is  $O(kn^2)$  by Tamir [1]. We show that improved complexity bounds for the problem are possible when  $k$  and  $h$  (the height of the tree) are asymptotically smaller than  $n$ . For the unweighted case, we obtain the complexity bound of  $O(k^2n(h + \log n))$ . In the particular case of random trees this implies  $O(k^2n^{3/2})$ . In the case of balanced trees, we achieve  $O(k^2n \log(n/k))$ . For the weighted case, we show that similar bounds are possible if we allow a small error in the optimal objective value. Also, in the case of balanced trees, we achieve an  $O(n \text{polylog}(n))$  time algorithm for the weighted case, if we assume  $k$  to be a fixed constant. This is the first known result which achieves better bounds than previously known  $O(kn^2)$  for many particular cases and promises to be sub-quadratic in  $n$ .

## 1 Introduction

*Formulation.* The  $k$ -median problem is a classical problem in location theory as well as clustering. The  $k$ -median problem is formulated as follows: We are given a graph  $G = (V, E)$ ,  $V = \{v_1, v_2, \dots, v_n\}$ , with a nonnegative length (distance) on each edge. The length of a path in  $G$  is the sum of lengths of its edges. For each pair of nodes  $v_i, v_j$  the distance between them is the length of shortest path between them and is denoted by  $d(v_i, v_j)$ . The problem is to select a subset  $S$  of  $k$  nodes (medians) that will minimize the sum of all the nodes to their respective nearest median in  $S$ ,

$$\sum_{v_i \in V} \min_{v_j \in S} d(v_i, v_j)$$

In the weighted case, we have a nonnegative weight  $w_v$  on each node  $v$ . The goal is then to minimize the weighted sum  $\sum_{v_i \in V} \min_{v_j \in S} w_{v_i} d(v_i, v_j)$ . In the context of location theory, the nodes  $v_i \in V$  are seen as customers and nodes in  $v_j \in S$  are considered as service centers. Each customer  $v$  has a demand of  $w_v$  and the cost of service from  $v_j$  to customer  $v_i$  is  $d(v_i, v_j)$  per each unit of demand making the total cost of service to  $v_i$  as  $w_{v_i} d(v_i, v_j)$ .

*Related Work.* The  $k$ -median problem has been a subject of study for several decades. The general case was shown to be NP-hard [6, 17]. Most of the work was focussed on efficient heuristics. Recently, there have been many approximation algorithms given for the problem when graph  $G$  is

---

\* Dept. of CS, Purdue University, rahul@cs.purdue.edu; Part of the work was done when author was at Rutgers University

a metric [14, 15]. The problem on trees was already shown to be polynomially solvable by Kariv and Hakimi [6]. Bartal's [24] result on approximating arbitrary metric with a tree metric yielded a randomized polynomial time approximation algorithm using the polynomial solution on trees. This was derandomized by Charikar et al [13]. A series of approximation algorithms have followed [14–16]. There are many other variants considered like  $k$ -median in euclidean metric [11] and  $k$ -median as a data stream clustering problem [20].

The case when  $G$  is a tree has also been extensively studied. In 1979 Kariv and Hakimi [6] gave a dynamic programming algorithm with running time  $O(k^2n^2)$ . Hsu[18] gave an algorithm with running time  $O(kn^3)$ . In 1996, Tamir [1] showed by a tighter analysis that the dynamic programming algorithm of Kariv and Hakimi runs in  $O(kn^2)$  time. Similar analysis was also done by Halldorsson et al[2]. This is the best known bound to date. Some other special cases have also been investigated. For the case of  $k = 2$  on trees, Gavish and Sridhar [10] presented an algorithm which runs in  $O(n \log n)$  time. Their algorithm runs in linear time for the unweighted case. When the tree is a path, Hassin and Tamir [12] gave an  $O(kn)$  algorithm.

Earlier work in improvement of dynamic programming was in the case where these problems involved recurrences in one or two dimensions. Many of these, particularly the one dimensional problems (like dynamic programming on line or path), exploit the concavity, convexity and sparsity of involved functions and/or properties of Monge array searching [26, 27]. The dynamic programming improvements on trees generalizes some of the methods used line problems.

There has been some recent work in improving the time complexity of dynamic programming algorithms on trees. The main idea is to represent the dynamic programming functions in term of real parameter value (the so called “undiscretization” of [4, 9] or depth-based dynamic programming as in [8, 7]). Shah, Langerman and Lodha [9] applied the idea of expressing the dynamic programming functions as piecewise linear functions in real valued parameter to the facility location problem on directed trees. Shah and Farach-Colton [4] further extended this method to facility location problem on trees. Their algorithms run in  $O(n \log h)$  and  $O(n \log n)$  respectively for two problems improving from the previous bounds of  $O(nh)$  and  $O(n^2)$  respectively. Independently, a similar approach was deployed by Chrobak et al. [8, 7] for the  $k$ -median problem on *directed* trees. (where facilities only serve downwards on a rooted tree). The previous known (traditional dynamic programming) bound for the problem was  $O(kn^2)$ . [8] first obtained  $O(n \log n)$  and  $O(n \log^2 n)$  for the case  $k = 2$  and  $k = 3$  respectively. They then generalized it to an  $O(n \text{polylog}(n))$  algorithm when  $k$  is assumed to be a fixed constant on directed trees<sup>1</sup>. Their algorithm indeed yields  $O(n \log^2 n)$  algorithm for the unweighted case of  $k$ -median on directed trees with fixed  $k$ . Also, the technique by Gulia et al. [19] for approximate dynamic programming on streams can be applied to the weighted case. It implies that if we allow a small room for error we can achieve similar bounds for the weighted case as in the unweighted case.

We explore similar methods for the  $k$ -median problem on trees. While it is not obviously apparent how to obtain strong complexity improvements for this problem, we nevertheless obtain faster algorithm when height of the tree is small. e.g. in the case of balanced trees.

*Our Results.* We redefine the dynamic programming functions as given by Tamir [1] and apply “undiscretization” to them. This way, the functions on a subtree don't depend on the nodes of the tree outside the given subtree. In many cases, we show that this is indeed a concise representation of these functions. This implies that they need lesser time to be manipulated and hence result in faster algorithms. In the weighted case, this may be a concise representation if  $k$  and  $h$  are very small or if the linearity conjecture (sec 3.2) is true. Otherwise, we obtain the concise representation by allowing small error. We summarize the main complexity bounds achieved as follows:

<sup>1</sup> All the tree algorithms in the literature are for weighted case unless specified as unweighted

1. For the unweighted  $k$ -median on trees, we achieve an  $O(k^2 P)$  algorithm in general, where  $P$  is the path length of the (appropriately rooted and binarized) tree. This implies  $O(k^2 n(h + \log n))$  upper bound. In case of random trees this implies  $O(k^2 n^{3/2})$  algorithm by analysis similar to [5, 25]. In case of balanced binary trees, we achieve  $O(k^2 n \log(n/k))$  bound by a tighter analysis. In general, the algorithm is faster than or as fast as Tamir's  $O(kn^2)$  algorithm. These bounds would hold for the weighted case if the linearity conjecture is true. Nevertheless, these bounds hold for the case where all the weights are nonnegative integers bounded by a constant.
2. For the weighted case, this yields  $O(nk^2 h^{k+1})$  algorithm in general. Which implies, if  $k$  is considered to be a constant and if the tree is balanced, then the algorithm runs in  $O(n \text{polylog}(n))$  time.
3. For the weighted case, in general we can achieve  $1 + \epsilon$  approximation algorithm on trees which runs in  $O(\epsilon^{-1} k^2 n(h + \log n)(\log D + \log W + \log n))$  time, where  $D$  and  $W$  are the sum of all distances and sum of all weights respectively. Typically,  $D$  and  $W$  are bounded by a polynomial in  $n$ . This, then implies  $O(\epsilon^{-1} k^2 n \log n(h + \log n))$  algorithm.

*Map.* The rest of the paper is organized as follows: Section 2 describes the dynamic programming algorithm. Section 3 proves the properties of dynamic programming functions and gives the analyses of running times of the algorithm. Section 4 gives the  $(1 + \epsilon)$  approximation algorithm and presents the analysis. We conclude in section 5 with remarks and future work.

## 2 Dynamic Programming Algorithm

We shall regard the tree  $T = (V, E)$  as a rooted binary tree. If it is a non-binary tree it can be converted into a binary tree in linear time using techniques similar to [1] with a small change that whenever we binarize a high degree node we replace it by a balanced binary tree. This makes sure that the overall height doesn't increase more than  $\log n$ . We choose the root  $R$  as a 1-median of a unit edge distance tree. This will ensure the minimum path length and this can also be done in linear time. Now in this rooted tree, let  $T_v$  denote the subtree rooted at the vertex  $v$  and let  $s_v$  denote the size of  $T_v$  which is the number of nodes in  $T_v$ . Let  $h_v$  denote height of  $T_v$  and let  $d_v$  denote the depth of node  $v$  in  $T$  i.e. the number of edges in the unique path from  $R$  to  $v$ . Let  $W = \sum_{v \in V} w_v$  and  $D = \sum_{e \in E} d(e)$ .

Let  $G_v^p(x)$  be the minimum objective function value of the subproblem defined on the subtree  $T_v$  such that there is at least one selected median location within distance  $x$  from  $v$  and given that total of  $p$  median are allowed to be selected in  $T_v$ . Let  $F_v^p(x)$  be the minimum objective function value of the subproblem defined on  $T_v$  such that the nearest median in  $T - T_v$  from  $v$  is exactly at distance  $x$  from  $v$  given that  $p$  medians are allowed in  $T_v$ . We wish to find  $G_R^k(\text{inf})$  (which is the same as  $F_R^k(\text{inf})$ ).

Let  $l$  and  $r$  be the left and right children of  $v$  separated by distance  $x_l$  and  $x_r$  respectively from  $v$ . Let  $x_1, x_2, \dots, x_{s_v}$  be the distances of points in  $T_v$  from  $v$  in the increasing order. The following table describes the dynamic programming recurrences in terms of undiscretized parameter  $x$ .

```

if  $v$  is a leaf then
     $G_v^0(x) = \infty$  and  $F_v^0(x) = w_v x$ 
     $G_v^p(x) = 0$  and  $F_v^p(x) = 0$  for  $p \geq 1$ 
else
     $G_v^p(0) = \min_{i=0..p-1} F_l^i(x_l) + F_r^{p-1-i}(x_r)$ 
     $G_v^p(x_m) = \min(G_v^p(x_{m-1}), \min_{i=0}^p G_l^i(x_m - x_l) + F_r^{p-i}(x_m + x_r) + w_v x_m)$ 
    when  $x_m$  is a distance to vertex in  $T_l$ 
     $G_v^p(x_m) = \min(G_v^p(x_{m-1}), \min_{i=0}^p G_r^i(x_m - x_r) + F_l^{p-i}(x_m + x_l) + w_v x_m)$ 
    when  $x_m$  is a distance to vertex in  $T_r$ 

     $G_v^p(x) = G_v^p(x_m)$ 
    when  $x_m < x < x_{m+1}$ 
     $F_v^p(x) = \min(G_v^p(\infty), \min_{i=0}^p F_l^i(x + x_l) + F_r^{p-i}(x + x_r) + w_v x)$ 
end if

```

The previous known algorithm for  $k$ -median on trees due to Tamir [1] uses a similar dynamic programming recurrence but in the discrete sense. The values for  $F_v^p(x)$  and  $G_v^p(x)$  are maintained as an array for the values of  $x$  corresponding to the distances of  $v$  from every other point in  $T$ . As described by the equations above, every particular value in  $F_v^p(x)$  or  $G_v^p(x)$  can be computed by trying at most  $k$  combinations of previously computed values. There are totally  $O(kn^2)$  values to be computed:  $n$  different values of  $v$ ,  $n$  of  $x$  and  $k$  of  $p$ . This makes the algorithm run in time  $O(k^2n^2)$ . However, the analysis due [1] shows that this bound is actually  $O(kn^2)$ . This is due to the fact that at smaller subtrees  $T_v$ , not all  $k$  values of  $p$  are required and if subtree of one of the left or right child is small then not all  $k$  combinations are required to compute the values.

Our algorithms shall benefit from the idea of storing these functions in geometric or 'undiscretized' form. Hence, instead of storing for  $n$  values of  $x$ , we shall store the function in continuous variable  $x$  which will yield a smaller storage space and hence, faster processing.

We shall describe the faster algorithms and analysis in the next two sections.

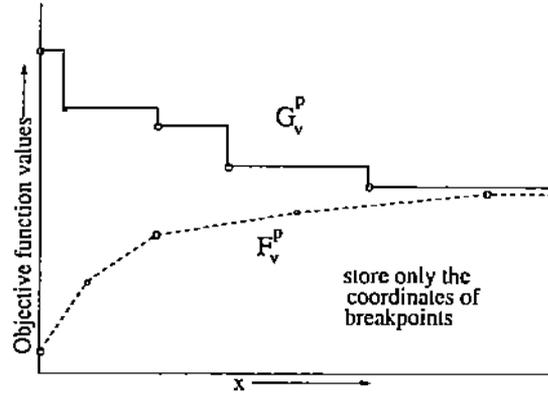
### 3 Function Representation and Analysis

#### 3.1 Unweighted $k$ -median

This case corresponds to the case when all the node weights  $w_v$  are unit. The analyses also holds if the all the weights are nonnegative integers bounded by a constant.

**Lemma 1** *For the unweighted  $k$ -median problem,  $G_v^p(x)$  (as a function of  $x$ ) is a piecewise decreasing step function and  $F_v^p(x)$  is piecewise linear non-decreasing concave function. The sizes of these functions (number of linear pieces) are bounded above by  $s_v$ . i.e. if  $|F|$  denotes the size of piecewise linear function  $F$ , then  $|F_v^p| \leq s_v$  and  $|G_v^p| \leq s_v$ .*

**Proof :** As a function of  $x$ , the value of  $G_v^p$  only changes when  $x$  equals the distance of some node in  $T_v$  from  $v$  (this corresponds to the situation when the nearest median location from  $v$  in  $T_v$  is at distance  $x$ ). Since the restriction on the placement (by the definition of  $G_v^p(x)$  is that there must be a median within distance  $x$ ) is only relaxed when the value of  $x$  increases,  $G_v^p$  can only decrease. As regards  $F_v^p(x)$ , at any point  $x$  the slope of  $F_v^p$  is the number of nodes in  $T_v$  which are served by the median location outside  $T_v$  which is at distance  $x$  from  $v$ . This slope only changes when there is some rearrangement of median locations within  $T_v$ . Due to the minimality of  $F_v^p$  this rearrangement only occurs when the slope of  $F_v^p$  can be decreased. Hence,  $F_v^p$  is a slope-decreasing (i.e. concave) function with possible values of slopes being integers ranging from  $\{s_v, s_v - 1, \dots, 2, 1\}$ . Therefore, the number of linear pieces in  $F_v^p$  is at most  $s_v$ .  $\square$



We store  $G_v^p(x)$  and  $F_v^p(x)$  as an array of tuples. Each tuple contains the  $x$  and  $y$  coordinates of the breakpoints in these functions. Probe to each function can be done by a binary search. At values of  $x$  other than breakpoints, the function value can be easily obtained from values at the two nearest breakpoints. Hence, the storage at each node  $v$  is  $O(ks_v)$  as against  $O(kn)$  as in the previous dynamic programming algorithms [1, 6]. Each operation involving additions of functions  $F_l^p(x)$ ,  $G_l^p(x)$ ,  $F_r^p(x)$  and  $G_r^p(x)$  can be done in  $O(s_l + s_r)$  time i.e.  $O(s_v)$ . At each node, we have to maintain at most  $k$  (one for each value of  $p$ ) such functions and for constructing each we have to take the pointwise minimum of at most  $k$  functions obtained by additions of functions at child nodes. This can be done in  $O(ks_v)$  time. That makes the total computation at node  $v$  at most  $O(k^2s_v)$ . Summing this up over all the nodes in the tree we get the upper bound of  $k^2 \sum_{v \in V} s_v$ . Note that this is same as  $O(k^2P)$  achieved for directed trees by Vigneron et al [5] where  $P$  is the path length of the rooted tree. Following their argument, we get  $O(k^2n^{3/2})$  for random trees. In general, this improves the traditional dynamic programming algorithm bound obtained by Kariv and Hakimi of  $O(k^2n^2)$  to at most  $O(k^2n(h + \log n))$ . Note that  $h + \log n$  is an upper bound on the height of the binarized tree and  $\sum_{v \in V} s_v$  is bounded above by  $n$  times the height of the binarized tree. Hence, we conclude the following theorem:

**Theorem 1** *The unweighted  $k$ -median on trees can be computed in  $O(k^2n(h + \log n))$  time.* □

Although, we have  $k^2$  term (as against  $k$  in [1]) in our analysis we can show that this algorithm never performs worse than the  $O(kn^2)$ . Tamir's analysis of the traditional dynamic programming algorithm of  $O(kn^2)$  is obtained due to the fact that at the tree nodes with subtree sizes less than  $k$  we don't need to maintain the functions for all  $k$  values. The total running time of Tamir's algorithm was analyzed as  $\sum_{v \in V} n \cdot \min(k, s_v) \cdot \min(k, s_l, s_r)$ . A similar analysis is true for our algorithm also. Strictly speaking the running time of our algorithm is  $\sum_{v \in V} s_v \cdot \min(k, s_v) \cdot \min(k, s_l, s_r)$ . Just by comparing the terms in summation, we can infer that our algorithm is never slower than  $O(kn^2)$  and in fact faster in many cases, since  $s_v \leq n$ .

**Balanced Trees.** Particularly, for balanced trees, we obtain  $O(k^2n \log(n/k))$  time algorithm. For simplicity, we first show how this is obtained in a complete binary tree. For top  $\log(n/k)$  levels, where subtree size of each node is  $\geq k$ , the total computation time for an entire level is  $O(k^2n)$ . Hence for these top levels the total time required is  $O(k^2n \log(n/k))$ . For the bottom  $\log k$  levels the total computation is  $n(1^2 + 2^2 + 4^2 + 8^2 + \dots + k^2)$  which is bounded above by  $O(k^2n)$ . Thus, the total complexity is  $O(k^2n \log(n/k))$ .

In general, for balanced trees, we decompose the tree for the sake of analysis to vertices with  $s_v > k$  and others. The firsts form a top part of the tree and the rest is bottom forests. Again,

since it is a balanced tree the top part has  $O(\log(n/k))$  levels and the total computation on top part is  $O(k^2 n \log(n/k))$ . The rest of the nodes (in bottom part) can be grouped by their respective  $s_v$  values. All the nodes  $v$  with  $s_v$  within some range will fall in the same group. For grouping of  $s_v$  values we take into account the balance factor  $c > 1$  (also  $c < 2$ ) which is the least ratio of  $s_v$  to  $s_l$  or  $s_r$  over all  $v$ . Then, the groups for  $s_v$ 's will be  $[1, c], [c, c^2], [c^2, c^3], \dots$  and so on. Note that no two nodes within the same group of  $s_v$  values can have ancestral relationship (i.e. they form an antichain). Hence,  $\sum_{v|s_v \in [c^l, c^{l+1}]} s_v \leq n$  for all  $l \geq 0$ . The computation at each node is proportional to  $s_v$  at that node. Hence, the total computation over all these nodes is again  $O(n(1 + (c^2)^2 + (c^3)^2 + \dots + (c^l)^2))$  where  $l$  is the least power of  $c$  which is greater than  $k$ . This sum is  $O(k^2 n)$  which is less than  $O(k^2 n \log(n/k))$  and hence we achieve the required complexity of  $O(k^2 n \log(n/k))$ .

### 3.2 Weighted $k$ -median

In case of weighted  $k$ -median, lemma 1 is no longer completely true. Although  $|G_v^p| \leq s_v$ , function  $F_v^p$  can have more number of linear pieces. i.e.  $|F_v^p| \leq s_v$  may be no longer true. In the worst case  $F_v^p$  could have  $\binom{s_v}{p}$  different slopes which is a trivial upper bound on  $|F_v^p|$ . However, we conjecture that the number of slopes is substantially smaller, in fact just linear in  $s_v$ .

**Conjecture 1** *The Linearity Conjecture: For weighted  $k$ -median problem,  $|F_v^p|$  is  $O(s_v)$ .*

If this conjecture is true, we would achieve the same bounds for weighted  $k$ -median problem as in unweighted case. However, here in this subsection we shall provide an upper bound tighter than  $O(\binom{s_v}{p})$  on  $|F_v^p|$  following the analysis similar to [7]. This would mean that in case of balanced trees and if  $k$  is a small fixed constant we would achieve  $O(\text{npolylog}(n))$  bound for the algorithm.

**Lemma 2** *For the weighted  $k$ -median problem,  $|F_v^p|$  is  $O(s_v h_v^p)$ .*

**Proof :** We shall bound the size of  $F_v^p$  by induction. Note that  $F_v^p$  is constructed using the last line of the dynamic programming recurrence table. Note that for two piecewise linear functions  $A$  and  $B$ ,  $A+B$  and  $\min(A, B)$  are also piecewise linear. Also,  $|A+B| \leq |A|+|B|$  and  $|\min(A, B)| \leq |A|+|B|$ . Then, by the last line of the recurrence table,  $|F_v^p|$  is bounded by the recurrence

$$|F_v^p| \leq 1 + \sum_{i=0}^{p-1} |F_l^i| + \sum_{i=0}^{p-1} |F_r^i|$$

i.e.

$$|F_v^p| \leq |F_l^p| + |F_r^p| + (1 + \sum_{i=0}^{p-1} |F_l^i| + \sum_{i=0}^{p-1} |F_r^i|)$$

This means  $|F_v^p|$  is upper bounded by the recurrence  $S_v^p$  given by

$$S_v^p = S_l^p + S_r^p + S_v^{p-1}$$

$$S_v^0 = 1$$

$$S_v^p = 2 \quad \text{for any leaf } v \text{ when } p > 0$$

It is easy to check that  $2s_v h_v^p$  is an upper bound for  $S_v^p$  from the recurrence, by plugging in values and using the inequalities  $h_l, h_r \leq h_v - 1$  and  $s_l + s_r \leq s_v$ .  $\square$

We can derive the following corollary from the above lemma:

**Corollary 1** *In the case of balanced trees, where  $h$  is  $c \log n$  for some constant  $c$ , we obtain  $O(k^2 n c^{k+1} \log^{k+1} n)$  algorithm. This means if  $k$  is considered to be a fixed constant we achieve  $O(\text{npolylog}(n))$  algorithm.*

#### 4 Fast $(1 + \epsilon)$ Approximation for Weighted $k$ -median

Here, we allow an arbitrarily small error, in the objective  $k$ -median value and show that by taking this liberty we can ensure that  $|F_v^p|$  is small. That is, to show that even if the function  $F_v^p$  at some node  $v$  has many breakpoints not all of them store a significant amount of information, in fact a very few do. Thus many breakpoints can be ignored and eliminated, if we allow the relative error of  $(1 + \delta)$  in function  $F_v^p(x)$ . We shall call this procedure of eliminating the break points as  $\delta$ -trimming.

Consider a piecewise linear concave function  $F$ . The  $\delta$ -trimming of function  $F$  denoted by  $\hat{F}$ .  $\hat{F}$  is obtained by following procedure: Visit each breakpoint of  $F$  in increasing  $x$  order (this is also increasing  $y$  order where  $y = F(x)$ ). Let the sequence of  $y$  in  $F$  values be  $y_1, y_2, y_3, \dots$ . We maintain  $y_1$ . Now let  $y_i$  be the breakpoint which is most recently visited and maintained. For  $j = i + 1, i + 2, \dots$  in increasing order eliminate  $y_j$  if  $y_{j+1} \leq (1 + \delta)y_i$  else maintain  $y_j$  and  $y_{j+1}$  and set  $i = j + 1$ . Now  $\hat{F}$  is a piecewise linear concave function obtained by joining these maintained breakpoints. It is easy to see that  $F(x) \geq \hat{F}(x) \geq F(x)/(1 + \delta)$ . i.e. the cost represented by this function is always less than the original cost but original cost is no more than  $(1 + \delta)$  times that represented by  $\hat{F}(x)$ .

**Lemma 3**  $|\hat{F}_v^p|$  is  $O(\frac{\log n + \log D + \log W}{\log(1 + \delta)})$  for any  $v, p$ .

**Proof :** Since the maximum value of  $y$  in any function  $F$  involved in dynamic programming is bounded above by  $n^2DW$  the number of breakpoints in  $\hat{F}$  is bounded above by  $2 \log_{1 + \delta}(n^2DW)$ . This is because every odd breakpoint in the sequence of  $F$  has value at least  $(1 + \delta)$  times the previous odd breakpoint.  $\square$

Now, for the algorithm, we fix  $\delta$ . We proceed with the bottom up dynamic programming. Now let  $\hat{F}, \hat{G}$  be the functions in this  $\delta$ -trimmed dynamic programming. At each node when we construct the function  $\bar{F}_v$  from  $\hat{F}_l$  and  $\hat{F}_r$  by dynamic programming, and then we apply  $\delta$ -trimming to it to get  $\hat{F}_v$ .  $\hat{G}$  does not need  $\delta$ -trimming. The computation time for  $\hat{G}$  is still upper bounded by theorem 1. We still call it  $\hat{G}$  instead of  $G$  because the values of  $\hat{G}$  are different due to their dependency on  $\hat{F}$ .

**Lemma 4** For any  $v, p$  and  $x$ ,

$$F_v^p(x) \geq \hat{F}_v^p(x) \geq \frac{F_v^p(x)}{(1 + \delta)^{h_v}}$$

and

$$G_v^p(x) \geq \hat{G}_v^p(x) \geq \frac{G_v^p(x)}{(1 + \delta)^{h_v}}$$

**Proof :** We prove this by induction on height. It is trivially true for the base cases. Now, let  $\bar{F}_v$  be the function which is constructed by dynamic programming from  $\hat{F}_l$  and  $\hat{F}_r$ . By induction,  $F_l(x) \geq \hat{F}_l \geq F_l(x)/(1 + \delta)^{h_v - 1}$ , and a similar condition is true for  $\hat{F}_r$  (and also  $\hat{G}_l, \hat{G}_r$ ). This implies  $F_v(x) \geq \bar{F}_v(x) \geq \frac{F_v(x)}{(1 + \delta)^{h_v - 1}}$ . Since  $\hat{F}_v$  is a  $\delta$ -trimming of  $\bar{F}_v$ , the result follows.  $\square$

The lemma indicates that the  $k$ -median objective value achieved by the placement according to  $\hat{F}$  achieves the placement which is at most  $(1 + \delta)^h$  times the minimum objective value desired (here  $h = h_R$  is the height of the binarized tree). If we choose  $\delta \leq (1 + \epsilon)^{1/h} - 1$  then this gives us  $(1 + \epsilon)$  approximation for  $k$ -median problem. Note that  $\delta = \epsilon/2h$  suffices this condition for values of  $\epsilon < 1$ . Also,  $\log(1 + \delta) = \log(1 + \epsilon)/h > \epsilon/(2h)$ . By doing this, the size of functions involved becomes  $O(\epsilon^{-1}h(\log n + \log D + \log W))$ . If  $D$  and  $W$  are bounded by polynomial in  $n$  then the running time of the algorithm becomes  $O(\epsilon^{-1}k^2hn \log n)$ .

**Theorem 2** We can compute  $(1+\epsilon)$  approximate weighted  $k$ -median on trees in time  $O(\epsilon^{-1}k^2n(h+\log n)(\log n + \log D + \log W))$ . For balanced binary trees with  $W, D$  bounded by polynomial in  $n$ , this algorithm runs in  $O(k^2n \log^2 n/\epsilon)$  time. □

## 5 Remarks and Future Work

In facility location and related one-dimensional tree problems [4], the speed-up achieved by undiscretized dynamic programming is due to two key facts – Concise representation of the functions involved and quicker update operations on these functions to create new functions in the bottom up fashion on the tree. The quicker updates can be attributed to the fast-merging algorithm for merging two sorted lists. The fast merging data structure, by merging to unequal sized lists, gives a balancing effect to the unbalanced trees. Here, in the case of  $k$ -median, we achieve the concise representation but we don't achieve quicker updates because the operation involved in constructing new function are more complex. If this was possible the factor of height  $h$  would be replaced by  $\log n$  in the final complexity bounds achieved. This might be possible by some good two-dimensional data structure.

Another way of converting height  $h$  into  $\log n$  is by the “decomposition tree” idea of [7]. The decomposition tree is a superstructure over the tree which has the effect of converting unbalanced tree into a balanced tree on which dynamic programming takes place. In the case of directed trees as considered by [8, 7] the dynamic programming functions are simpler and they work on the decomposition trees. This indicates a possibility of a new kind of dynamic recurrences for  $k$ -median which might allow the tree to be balanced as in decomposition trees.

Also, the linearity conjecture remains an interesting open problem in the weighted case.

## References

- [1] A. Tamir, “An  $O(pn^2)$  algorithm for the  $p$ -median and related problems on tree graphs”, *Operations Research Letters*, 19:59-94, 1996.
- [2] M. Halldorsson, K. Iwano, N. Katoh and T. Tokuyama, “Finding subsets maximizing minimum structures”, SODA 1995.
- [3] G. Cornuejols, G.L. Nemhauser and L.A. Wosley, “The uncapacitated facility location problem”, in P.B. Mirchandani and R.L. Francis(eds), *Discrete Location Theory*, Wiley, New York, 1990, pp. 119-171.
- [4] R. Shah and M. Farach-Colton, “Undiscretized dynamic programming: Faster algorithms for facility location and related problems on trees”, *In Proc. thirteenth annual Symposium on Discrete Algorithms (SODA)*, 2002.
- [5] A. Vigneron, L. Gao, M. Golin, G. Italiano and B. Li, “An algorithm for finding a  $k$ -median in a directed tree”, *Information Processing Letters*, vol. 74 (2000), 81-88
- [6] O. Kariv and S. Hakimi, “An algorithmic approach to network location problems, part II:  $p$ -medians”, *SIAM Journal of Applied Mathematics*, 37, 539-560, 1979.
- [7] M. Chrobak, L. L. Larmore and W. Rytter, “A fast algorithm for computing  $k$ -medians in directed trees”, *manuscript*, 2002.
- [8] M. Chrobak, L. Larmore and W. Rytter, “The  $k$ -median problem for directed trees”, *In Proc. 26th International symposium on mathematical foundations of computer science (MFCS)*, 2001.
- [9] R. Shah, S. Langerman and S. Lodha, “Algorithms for efficient filtering in content-based multicast”, *In Proc. of 9th annual european symposium on algorithms (ESA)*, 2001.
- [10] R. Gavish and S. Sridhar, “Computing the 2-median on tree networks in  $O(n \log n)$  time”, *Networks*, 26:305-317, 1995.
- [11] S. Arora, P. Raghavan and S. Rao, “Approximation schemes for euclidean  $k$ -medians and related problems”, *In Proc. 30th annual ACM symposium on theory of computing (STOC)*, 1998.

- [12] R. Hassin and A. Tamir, "Improved complexity bounds for location problems on the real line", *Operation research letters*, 10:395-402, 1991.
- [13] M. Charikar, C. Chekuri, A. Goel and S. Guha, "Rounding via trees: Deterministic approximation algorithms for group steiner trees and  $k$ -median", STOC 1998.
- [14] K. Jain and V. Vazirani, "Primal-dual approximation algorithms for metric facility location and  $k$ -median problems. In *Proc. 40th annual IEEE symposium on foundations of computer science (FOCS)*, 1999.
- [15] M. Charikar and S. Guha, "Improved combinatorial algorithms for facility location and  $k$ -median problems", In *Proc. 40th annual IEEE symposium on foundations of computer science (FOCS)*, 1999.
- [16] V. Arya, N. Garg, R. Khandekar, K. Munagala and V. Pandit, "Local search heuristic for  $k$ -median and facility location problems. STOC 2001.
- [17] M. R. Garey and D. S. Johnson, "Computers and intractability: a guide to the theory of NP-completeness", W. H. Freeman and Co., 1979.
- [18] W. L. Hsu, "The distance-domination numbers of trees", *Operations research letters*, 1:96-100, 1982.
- [19] S. Guha, N. Koudas and K. Shim, "Data streams and algorithms", In *Proc. 33rd annual ACM symposium on theory of computing (STOC)*, 2001.
- [20] S. Guha, N. Mishra, R. Motwani and L. O'Callaghan, "Clustering data streams", In *Proc. IEEE symposium on foundations of computer science (FOCS)*, 2000.
- [21] S. Guha and K. Munagala, "Generalized Clustering" In *Proc. 13th annual symposium on discrete algorithms (SODA)*, 2002.
- [22] B. Li, M. Golin, G. Italiano, X. Deng and K. Sohraby, "On the optimal placement of web proxies in the internet", In *IEEE InfoComm*, 1999.
- [23] R. Shah, F. Aujum and R. Jain, "Efficient dissemination of personalized information using content-based multicast", In *IEEE InfoComm*, 2002.
- [24] Y. Bartal, "Probabilistic approximation of metric spaces and its algorithmic applications", In *Proc. 37th IEEE symposium on foundations of computer science (FOCS)*, 1996.
- [25] R. Sedgewick and P. Flajolet, "An Introduction to the analysis of algorithms", Addison-Wesley, 1996.
- [26] A. Aggarwal and J. Park, "Improved algorithms for economic lot size problem", *Operations Research*, 41(3):549-571, 1993.
- [27] Z. Galil and K. Park, "Dynamic programming with convexity, concavity and sparsity", *Theoretical Computer Science*, 92:49-76, 1992.