The Two-Median Problem on Manhattan Meshes

Mordecai J. Golin and Yan Zhang
Department of Computer Science and Engineering, Hong Kong University of Sciences and Technology, Clear Water Bay, Kowloon, Hong Kong, People's Republic of China

We investigate the two-median problem on a mesh with M columns and N rows ($M \geq N$), under the Manhattan (L_1) metric. We derive exact algorithms with respect to m, n, and r, the number of columns, rows, and vertices, respectively, that contain requests. Specifically, we give an $O(mn^2 \log m)$ time, $O(r)$ space algorithm for general (nonuniform) meshes (assuming $m \geq n$). For uniform meshes, we give two algorithms both using $O(MN)$ space. One is an $O(MN^2)$ time algorithm, while the other is an algorithm running in $O(MN \log N)$ time with high probability and in $O(MN^2)$ time in the worst case assuming the weights are independent and identically distributed random variables satisfying certain natural conditions. These improve upon the previously best-known algorithm that runs in $O(mn^2 r)$ time. © 2007 Wiley Periodicals, Inc. NETWORKS, Vol. 49(3), 226–233 2007

Keywords: two-median; mesh; probabilistic analysis

1. INTRODUCTION

In the *k-median* problem we are given a graph $G = (V, E)$ with nonnegative edge costs. We want to choose k vertices (the *medians*) from V to minimize the sum of the distances between each vertex and its closest median. As motivation, the vertices can be thought of as *customers*, the medians as *service centers*, and the distance between a customer and a service center as the cost of servicing the customer from that center. In this view, the k-median problem is about choosing a set of k service centers that minimizes the total cost of servicing all customers. In a parallel computing scenario, the vertices can be thought of as clients and the medians as servers; the k-median problem then corresponds to a form of load-balancing.

The k-median problem is often extended so that each customer (vertex) has a weight, corresponding to the *amount* of service requested. The distance between a customer and its closest service center (median) then becomes the cost of providing *one unit* of service, that is, the cost of servicing a customer will then be the weight of the customer-vertex times its distance from the closest service center.

Lin and Vitter [13] proved that, in general graphs, even finding an $o(\ln n)$-approximate solution to the k-median problem is NP-hard. They were able to show, though, that it is possible in polynomial time to achieve a cost within $O(1+\epsilon)$ of optimal if one is allowed to use $(1+1/\epsilon)(\ln n+1)k$ medians. Guha and Khuller [8] proved that this problem is still MAX-SNP hard if restricted to metric spaces. Charikar et al. [3] showed that constant-factor approximations can be computed for any metric space. In the specific case of points in Euclidean space, Arora et al. [1] developed a polynomial time approximation scheme (PTAS). There are some special graph topologies for which fast polynomial time algorithms for finding exact solutions exist, though. For example, this is true for trees [15, 17] and lines [9].

In the literature, the k-median problem is also classified according to the number of medians k. If $k = 1$, the problem is known as the one-median problem. If $k = 2$, the problem is known as the two-median problem. For $k \geq 3$, the problem is known as the k-median problem. In particular, the two-median problem was discussed widely on trees [2, 5, 6, 11].

In this article, we deal with the specific problem of solving the *two-median* problem on an $M \times N$ orthogonal mesh ($M \geq N$) where the distance function is the Manhattan (L_1) metric. This problem was recently introduced by Lau et al. [12], who were motivated by load-balancing on parallel machines with mesh-topologies such as the iWarp system [7]. In [12], they developed an exact algorithm that runs in $O(mn^2 r)$ time, where m, n (assume $m \geq n$) and r are, respectively, the number of columns, rows, and vertices of the mesh containing requests. Because $r \geq m$, their algorithm, therefore, needed at least $O(m^2 n^2)$ time. Recently, Tse and Lau [16] also developed an approximation algorithm with worst case ratio 1.5 and running in $O(m^2 + r \log r)$ time. Both of their algorithms required that the mesh must be a subset of the *uniform mesh*. The main result of this article is an improved exact algorithm that runs in $O(mn^2 \log m)$ time and uses $O(r)$ space. The algorithm also works for arbitrary meshes, dispensing with the requirement that the mesh be a subset of a uniform one. We also give two algorithms for uniform meshes. One algorithm runs in $O(MN^2)$ time in the

Received March 2005; accepted September 2006
Correspondence to: Y. Zhang; e-mail: cszy@cse.ust.hk
Contract grant sponsor: HK CERG; Contract grant numbers: HKUST6082/01E and HKUST6312/04E.
DOI 10.1002/net.20156
Published online in Wiley InterScience (www.interscience.wiley.com).

worst case. The other runs in $O(MN \log N)$ time *with high probability* if the requests of all vertices of the mesh are independent and identically distributed (iid) random variables satisfying certain natural conditions, for example, if the requests from each vertex are bounded by a constant (that is, resources needed by a single vertex do not depend on the size of the mesh) and the number of vertices that have requests is not too small (say, the proportion of the vertices having requests being $\ln N/\sqrt{N}$ is enough). A natural example of such a distribution would be that each vertex requests $i \in \{0, 1, \ldots, B\}$ units of service independently with constant probability p_i, where at least one p_i for $i \geq 1$ is nonzero. Both of the two algorithms for uniform meshes use $O(MN)$ space.

The article is organized as follows. In Section 1, we give the notations and the formal definitions. We begin with some basic observations in Section 2. In Section 3, we show the algorithm for general nonuniform meshes, which is later improved in Section 4 for uniform meshes. In Section 5, we discuss the range query subroutine that is used as a black box in the preceding sections.

1.1. The Parameters

Let $\{x_i : 1 \leq i \leq M\}$ and $\{y_j : 1 \leq j \leq N\}$ be two sets of real numbers sorted increasingly, that is, $x_1 < x_2 < \cdots < x_M$ and $y_1 < y_2 < \cdots < y_N$. The two sets define an $M \times N$ *mesh*, denoted by G, whose vertex set is $\{(x_i, y_j) : 1 \leq i \leq M, 1 \leq j \leq N\}$, and whose edges consist of those between vertices (x_i, y_j) and (x_i, y_{j+1}), for $1 \leq i \leq M$ and $1 \leq j < N$, and those between vertices (x_i, y_j) and (x_{i+1}, y_j), for $1 \leq i < M$ and $1 \leq j \leq N$. In general, G is considered to be *nonuniform*, that is, the values of the x_i's and y_j's can be arbitrary real values. A mesh is *uniform* iff $x_{i+1} - x_i = y_{j+1} - y_j =$ constant for all $1 \leq i < M$ and $1 \leq j < N$.

Each vertex may or may not have requests. The *weight* of a vertex u, denoted by $w(u)$, represents the amount requested from u. The values of the weights can be arbitrary nonnegative real values. Denote by r the number of vertices that have requests, that is, the vertices with nonzero weights, and by m and n, respectively, the number of columns and rows that have requests, that is, at least one vertex on it has requests. We also call these columns and rows *nonempty*. Without loss of generality, we assume $m \geq n$.

The input consists of the values x_i and y_j, and a list of vertices with their nonzero weights. All vertices that are not given in the list have no requests. In general (i.e., nonuniform meshes), we would like to bound the algorithm by a function of m, n and r, because we can throw away empty columns and rows. But for uniform meshes we cannot, because throwing away columns or rows in this way will destroy the uniform property (in general), so the best we can hope for is to bound the algorithm by a function of M, N and possibly r.

1.2. Problem Definitions

In this section we specify the two-median problem considered in this article. First, the distance $d(u, v)$ between vertices $u = (x_u, y_u)$ and $v = (x_v, y_v)$ is defined as the Manhattan, or L_1, distance, which is $|x_u - x_v| + |y_u - y_v|$. For a vertex set S, denote by $d(u, S)$ the distance between u and the closest vertex in S to u, that is, $d(u, S) = \min_{v \in S} d(u, v)$. A *region* is just a subset of the mesh. We define the *cost* of vertices in a region R w.r.t. S, denoted by $cost_S(R)$, as the weighted sum of the distances from each vertex in R to S, that is, $cost_S(R) = \sum_{u \in R} w(u) \cdot d(u, S)$. If S contains only one vertex, say $S = \{s\}$, we write $cost_s(R)$ instead of $cost_{\{s\}}(R)$. The k-median problem is to find a set S with $|S| = k$ that minimizes $cost_S(G)$. In this article, we consider the case $k \leq 2$. An example of the two-median problem is shown in Figure 1.

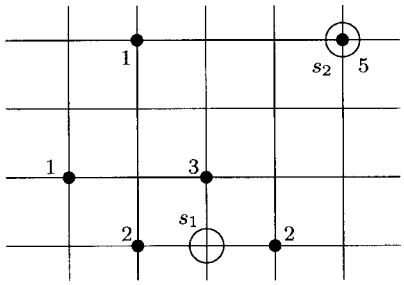

FIG. 1. An example of the two-median problem on a uniform mesh. Here, $M = 5$ and $N = 4$. The small black circles indicate the six vertices that have requests; their labels indicate their weights. Big empty circles indicate the optimal two-median $\{s_1, s_2\}$. In this example, the optimal cost is 13; s_2 services itself and the other vertex in its row, while s_1 services the other four requests.

Throughout the article, we use "point" to denote an arbitrary point in the plane, and "vertex" for a vertex of the mesh. For polygonal regions, we use "corner" to denote a vertex of the polygon, which can either be chosen from the vertices of the mesh or not.

2. PRELIMINARIES

2.1. The One-Median Problem

We start with the one-median problem. Note that in the optimal k-median set, each median s_i is an optimal one-median for the set of vertices whose nearest median is s_i.

Let $W_X(x_a, x_b)$, $W_X(x_a, x_b]$, $W_X[x_a, x_b)$, $W_X[x_a, x_b]$ be the sum of weights of vertices (x, y) with $-\infty < y < +\infty$ and, respectively, $x_a < x < x_b$, $x_a < x \leq x_b$, $x_a \leq x < x_b$, $x_a \leq x \leq x_b$. Similarly, let $W_Y(y_a, y_b)$, $W_Y(y_a, y_b]$, $W_Y[y_a, y_b)$, $W_Y[y_a, y_b]$ be the sum of weights of vertices (x, y) with $-\infty < x < +\infty$ and, respectively, $y_a < y < y_b$, $y_a < y \leq y_b$, $y_a \leq y < y_b$, $y_a \leq y \leq y_b$. Denote by W the total weight of all vertices in G. We have the following straightforward lemma from [12].

Lemma 1 (Lemma 3.1 of [12]). *Vertex $s = (x_s, y_s)$ is an optimal one-median iff*

$$W_X(-\infty, x_s) \leq W/2 \leq W_X(-\infty, x_s] \tag{1}$$

and

$$W_Y(-\infty, y_s) \leq W/2 \leq W_Y(-\infty, y_s]. \tag{2}$$

Using Lemma 1, we can find all optimal one-medians in $O(r)$ time. Note that there might be more than one optimal one-median for some special problem instances: if $W_X(-\infty, x_i] = W_X[x_j, +\infty) = W/2$ for some columns x_i and x_j, any $x_s \in [x_i, x_j]$ will satisfy Inequality (1). This is also true for the y-direction. In such cases, the set of all optimal one-medians forms an axis-parallel rectangle and we can always choose the one with the smallest x- and y-coordinates (lower left corner of the rectangle). Thus, some optimal one-median is always a vertex of the mesh, and always on a column and a row that have requests. This means that allowing the one-median to be an arbitrary point in the plane instead of being restricted to vertices of the mesh will not improve the solution. The same situation holds for the k-median problem: recall that, as mentioned before, each median s_i in an optimal k-median set is an optimal *one-median* for the set of vertices whose nearest median is s_i. Hence, we have the following result.

Lemma 2. *The cost of an optimal k-median of mesh G, where the medians S are restricted to be the vertices of the mesh, is the same as that of an optimal k-median where S can take arbitrary points in the plane, that is,*

$$\min_{|S|=k,\ S\subseteq G} cost_S(G) = \min_{|S|=k,\ S\subseteq \mathbb{R}^2} cost_S(G)$$

In the sequel we therefore always assume that the k-medians are vertices of the mesh.

2.2. Basic Regions

Denote the two medians by $s_1 = (x_{s_1}, y_{s_1})$ and $s_2 = (x_{s_2}, y_{s_2})$. We can assume $x_{s_1} \le x_{s_2}$ throughout the article. Denote $l = |x_{s_1} - x_{s_2}|$ and $d = |y_{s_1} - y_{s_2}|$. We will focus on how to compute an optimal two-median under the constraints $l \ge d$ and $y_{s_1} \le y_{s_2}$. Everything is symmetric when $l \ge d$ and $y_{s_1} > y_{s_2}$, so we will not discuss it. The case $l < d$ is similar, and we will discuss it briefly before Lemma 10.

Define the *Voronoi cell* of s_i to be the set of points for which the nearest median is s_i. If a point (x, y) has equal distance to both s_1 and s_2, we can assign it arbitrarily to either cell, but in this article, we will always assign it to the Voronoi cell of s_1 if $x \le x_{s_2}$ and assign it to the Voronoi cell of s_2 otherwise. We call the boundary curve between two Voronoi cells the *bisector*. Note that by the tie-breaking rule, the bisector always belongs to the Voronoi cell of s_1 (see Fig. 2). The following lemma describes the shape of the bisectors under the L_1 distance.

Lemma 3. *Assume $l \ge d$ and $y_{s_1} \le y_{s_2}$. The bisector between s_1 and s_2 consists of the following three segments: the vertical radial from $p_1 = \left(x_{s_2} - \frac{l-d}{2}, y_{s_1}\right)$ downward, the vertical radial from $p_2 = \left(x_{s_1} + \frac{l-d}{2}, y_{s_2}\right)$ upward, and the line segment, whose slope is always -1, connecting p_1 and p_2.*

Lemma 3 implies that the Voronoi cells under the L_1 metric can be partitioned into very simple shapes, as follows.

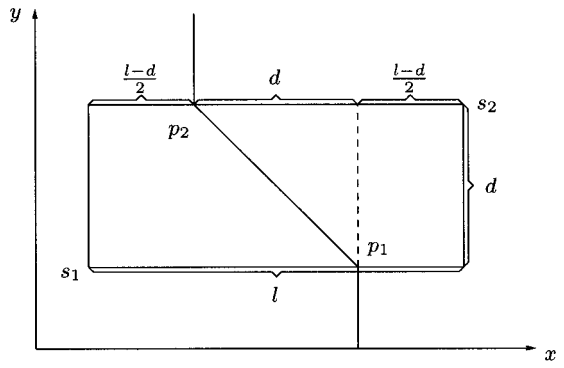

FIG. 2. The bisector between s_1 and s_2. We assume $l \ge d$ and $y_{s_1} \le y_{s_2}$.

Define R to be a *basic region* iff R is either an axis-parallel rectangle (possibly unbounded) or an axis-parallel triangle with a hypotenuse slope -1. Furthermore, we call a region R *separable* from a point $s = (x_s, y_s)$ iff $x_s \ge \max_{u\in R} x_u$ or $x_s \le \min_{u\in R} x_u$, and $y_s \ge \max_{u\in R} y_u$ or $y_s \le \min_{u\in R} y_u$. That is, R is separable from s if R falls in exactly one of the four quadrants around s.

Property 1. *The Voronoi cell of s_i can be partitioned into $O(1)$ basic regions that are separable from s_i.*

Proof. We can partition the Voronoi cell in such a way that the line segment connecting p_1 and p_2 is the hypotenuse of the triangular basic region. The rest of the area can be partitioned easily into four rectangular basic regions that are separable from s_i. ∎

The reason for introducing separability is that the cost of a region w.r.t. any separable point can be computed efficiently. Denote by $w(R)$ the total weight of vertices in R. Define $c_x(R) = \sum_{u\in R} w(u) \cdot x_u$ and $c_y(R) = \sum_{u\in R} w(u) \cdot y_u$.

Property 2. *If region R is separable from a point s, the value of $cost_s(R)$ can be computed in $O(1)$ time given the values of $w(R)$, $c_x(R)$ and $c_y(R)$.*

Proof. $cost_s(R) = \sum_{u\in R} w(u) \cdot |x_u - x_s| + \sum_{u\in R} w(u) \cdot |y_u - y_s|$. We will only show how to compute $\sum_{u\in R} w(u) \cdot |x_u - x_s|$. The computation of $\sum_{u\in R} w(u) \cdot |y_u - y_s|$ is similar. Because R is separable from s, either $x_s \ge \max_{u\in R} x_u$ or $x_s \le \min_{u\in R} x_u$. If $x_s \ge \max_{u\in R} x_u$, we have $|x_u - x_s| = x_s - x_u$ for all vertices $u \in R$. Then $\sum_{u\in R} w(u) \cdot |x_u - x_s| = w(R) \cdot x_s - c_x(R)$. If $x_s \le \min_{u\in R} x_u$, then $|x_u - x_s| = x_u - x_s$ and $\sum_{u\in R} w(u) \cdot |x_u - x_s| = c_x(R) - w(R) \cdot x_s$. ∎

Combining Properties 1 and 2, if we can compute $w(R)$, $c_x(R)$, and $c_y(R)$ of any basic region R efficiently, we can compute $cost_S(G)$ efficiently.

It turns out that the complexity of calculating w, c_x, and c_y will depend upon the model in which we are working, for example, uniform versus nonuniform meshes and the data structures that we are willing to introduce. We therefore postpone these computations to Section 5 and, for the time being,

introduce a parameter Q to denote the calculation time. More formally see the following.

Definition 1. *Let $Q = Q(M, N, m, n, r)$ be a function depending on the various parameters, such that the values of $w(R)$, $c_x(R)$, and $c_y(R)$ of any basic region R can be computed in $O(Q)$ time.*

Lemma 4. *We can compute $cost_S(G)$ for any given two-median set S in $O(Q)$ time.*

Proof. We first partition the Voronoi cells into separable basic regions as we did in Property 1, and then we compute the cost of each basic region w.r.t. the corresponding median. The total cost $cost_S(G)$ is simply the sum of the costs of all the basic regions. This approach takes $O(Q)$ time because the number of basic regions is constant. ∎

3. ALGORITHMS FOR NONUNIFORM MESHES

In this section, we consider general (nonuniform) meshes. As previously mentioned, in nonuniform meshes, we can throw away empty columns and rows. So, in this section, we will assume the size of the given mesh is $m \times n$ and all columns and rows are nonempty. The following lemma generalized from [12] is a key to the two-median problem. We repeat the proof here in our notation because we will need it later.

Lemma 5 (Lemma 4.2 of [12])**.** *Suppose $s_1 = (x_{s_1}, y_{s_1})$ and $s_2 = (x_{s_2}, y_{s_2})$ form an optimal two-median set under the constraint $l \geq d$. Then (we can assume $x_{s_1} \leq x_{s_2}$)*

$$W_X(-\infty, x_{s_1}) + W_X(x_{s_2}, +\infty) \leq W/2 \leq W_X(-\infty, x_{s_1}] + W_X[x_{s_2}, +\infty). \quad (3)$$

Proof. Recall the shape of the bisector from Lemma 3. Vertices (x, y) with $x \in (-\infty, x_{s_1}]$ are in the Voronoi cell of s_1 and those with $x \in [x_{s_2}, +\infty)$ are in the Voronoi cell of s_2. Denote by W_i the sum of the weights of vertices in the Voronoi cell of s_i, for $i = 1, 2$. Because s_i is an optimal one-median for the vertices in the Voronoi cell of s_i, using Lemma 1, we have

$$W_X(-\infty, x_{s_1}) \leq W_1/2 \leq W_X(-\infty, x_{s_1}] \quad (4)$$

and

$$W_X(x_{s_2}, +\infty) \leq W_2/2 \leq W_X[x_{s_2}, +\infty). \quad (5)$$

The lemma follows by summing Inequalities (4) and (5). ∎

We call $\langle x_{s_1}, x_{s_2} \rangle$ a *candidate* column-pair if it satisfies Inequality (3). Lemma 5 says we only need to consider the two-medians on the candidate column-pairs to find an optimal one. Lemma 2 tells us that there exists an optimal two-median set on some columns of the mesh. So we will only consider the candidate column-pairs on the columns of the mesh. As mentioned in [12], there are only $O(m)$ such candidate column-pairs. We write this as a lemma and give a formal proof (recall that r is the total number of vertices with nonzero requests).

Lemma 6 (Lemma 4.4 of [12])**.** *There are $O(m)$ candidate column-pairs, and they can be found in $O(r)$ time.*

Proof. We first bound the number of candidate column-pairs. As discussed above, we assume we have thrown away the empty columns and rename the nonempty columns as $x_1, \ldots, x_m$.

For each column x_i, $1 \leq i \leq m$, note that Inequality (3) implies that if $i < i_1 < i_2$ and both $\langle x_i, x_{i_1} \rangle$ and $\langle x_i, x_{i_2} \rangle$ are candidate column-pairs then, for every i', $i_1 < i' < i_2$, $\langle x_i, x_{i'} \rangle$ is also a candidate column-pair. Now let $[l_i, r_i]$, where $i < l_i \leq r_i$, be the interval of the columns that can form candidate column-pairs with x_i, that is, $\langle x_i, x_{i'} \rangle$ is a candidate column-pair iff $l_i \leq i' \leq r_i$. From Inequality (3),

$$W_X(-\infty, x_i) + W_X(x_{r_i}, +\infty) \leq W/2 \leq W_X(-\infty, x_i] + W_X[x_{r_i}, +\infty). \quad (6)$$

Because $W_X(-\infty, x_i] = W_X(-\infty, x_{i+1})$ and $W_X[x_{r_i}, +\infty) = W_X(x_{r_i - 1}, +\infty)$,

$$W/2 \leq W_X(-\infty, x_{i+1}) + W_X(x_{r_i - 1}, +\infty), \quad (7)$$

which means $l_{i+1} \geq r_i - 1$. So the total number of candidate column-pairs is $\sum_{i=1}^{m} (r_i - l_i + 1) \leq \sum_{i=1}^{m} (r_i - r_{i-1} + 2) = O(m)$.

To compute these candidate column-pairs, we first perform an $O(r)$ time preprocessing step such that, given $\langle x_i, x_{i'} \rangle$, we can check in constant time whether it is a candidate column-pair. Then, finding $[l_1, r_1]$ requires $O(m)$ time by scanning through all columns. After that, we can find each $[l_i, r_i]$, from $i = 2$ to $i = m$, in $O(r_i - r_{i-1} + 1)$ time by scanning the columns from $r_{i-1} - 1$ to $r_i + 1$. So the scanning steps take $O(m)$ time in total. Together with the preprocessing step, we have found all $O(m)$ candidate column-pairs in $O(r)$ time. ∎

Lemma 6 leads to the following algorithm.

Lemma 7. *An optimal two-median of an $m \times n$ nonuniform mesh can be computed in $O(mn^2 Q)$ time.*

Proof. We first show how to compute the optimal two-median under the constraint $l \geq d$. We calculate the $O(m)$ candidate column-pairs in $O(r)$ time as in Lemma 6. For each candidate column-pair, we simply compute the costs of all $O(n^2)$ possible two-medians on those columns. So the running time for the case $l \geq d$ is $O(mn^2 Q)$.

By symmetry, the optimal two-median under the constraint $l < d$ can be computed in a similar way. There are $O(n)$ candidate row-pairs, and for each candidate row-pair, there are $O(mn)$ possible two-medians because $l < d \leq n$. So the running time for the case $l < d$ is also $O(mn^2 Q)$. ∎

In Section 5 we will prove in Lemma 13 that, for nonuniform meshes, $Q = O(\log m)$. Combining Lemma 7 with Lemma 13, we have

Theorem 1. *An optimal two-median on an $m \times n$ nonuniform mesh can be computed in $O(mn^2 \log m)$ time in the worst case, using $O(r)$ space.*

4. ALGORITHMS FOR UNIFORM MESHES

In this section, we consider uniform meshes. The algorithm in Lemma 7 runs in $O(MN^2Q)$ time for an $M \times N$ uniform mesh. From Lemma 12, we have $Q = O(1)$ for uniform meshes, which immediately gives

Theorem 2. *An optimal two-median on an $M \times N$ uniform mesh can be computed in $O(MN^2)$ time in the worst case, using $O(MN)$ space.*

We will improve Lemma 7 and Theorem 2 in the probabilistic case. We assume the request weights of *all* the MN vertices are iid random variables with the same distribution X. Denote $\mu = E(X)$, the *expectation* of X. We will show that our algorithm runs in $O(MNQ \log N)$ time with high probability if X satisfies certain natural conditions; at the same time the algorithm will also run in $O(MN^2Q)$ time in the worst case.

The technique we use is a simple binary search. Assume $l \geq d$ and fix a candidate column-pair $\langle x_{s_1}, x_{s_2} \rangle$. We consider all possible two-medians on those two columns that arise by allowing all possible values of y_{s_1} and y_{s_2}. In Lemma 7, we simply tried all $O(N^2)$ possible two-medians in arbitrary order. However, if we group the $O(N^2)$ possibilities according to the vertical distance $d = |y_{s_1} - y_{s_2}|$, we can use a binary search, as follows.

Consider the set of two-medians with fixed vertical distance $d \geq 0$ on the candidate column-pair $\langle x_{s_1}, x_{s_2} \rangle$. We assume $y_{s_1} + d = y_{s_2}$ (the case $y_{s_1} > y_{s_2}$ is symmetric). In this case, the locations of the two-medians are completely determined by the value of y_{s_1}. From the proof of Lemma 5, we only need to compute costs for those two-medians satisfying Inequality (4). Recall that W_i is the sum of the weights of the vertices in the Voronoi cell of s_i, for $i = 1, 2$. Note from Figure 2 that W_1 increases when the two-medians move upward, that is, y_{s_1} increases, while all other values in Inequality (4) are fixed. Computing W_1 for a given two-median takes $O(Q)$ time. So a binary search can, in $O(Q \log N)$ time, find the upper and lower bounds of the value of y_{s_1} for which the corresponding two-median sets satisfy Inequality (4). We call these two-medians *valid* and the range of y_{s_1} in which the two-medians are valid the *valid interval*. Denote the valid interval by $[y_{\min}, y_{\max}]$, where $y_{\min}$ and $y_{\max}$ are the minimum and maximum values of y_{s_1} over all the valid two-medians. From the argument above we find that, instead of checking all n possibilities for y_{s_1}, as was essentially done in Lemma 7, we only need to examine the valid two-medians to find the optimal one. What remains is to bound the *length* of the valid interval. In the *worst case* this could be as bad as $\Omega(N)$ but, in the *random case*, it will usually be much smaller.

Lemma 8. *Assume $l \geq d \geq 1$. We consider the two-medians with fixed vertical distance d on a fixed candidate column-pair. Let $[y_{\min}, y_{\max}]$ be the valid interval of these two-medians. If $X \leq B$ for some constant B, then for any $c \geq 2$,*

$$\Pr(y_{\max} - y_{\min} \geq 8cN/d) \leq 4e^{-2cN\mu^2/B^2} \tag{8}$$

Proof. From Inequality (4), we see that, for a fixed column-pair $\langle x_{s_1}, x_{s_2} \rangle$, the maximum difference among the possible W_1 values of valid two-medians is at most $2(W_X(-\infty, x_{s_1}] - W_X(-\infty, x_{s_1})) = 2W_X[x_{s_1}, x_{s_1}]$, which has the same distribution as $2Y$, where Y is the sum of N independent copies of X.

We will only consider the case $y_{s_1} \leq y_{s_2}$, because the case $y_{s_1} > y_{s_2}$ is symmetric. Refer to Figures 2 and 3. Note that if we move the two medians upward by one row, the Voronoi cell of s_1 will contain at least d new vertices (recall from Lemma 3 that all the vertices on the bisector belong to the Voronoi cell of s_1). Denote by $W_1(y)$ the sum of weights of vertices in the Voronoi cell of s_1 when $y_{s_1} = y$. Let $Z_t = W_1(y_{\min} + t) - W_1(y_{\min})$. Therefore, Z_t is the sum of $d \cdot t$ independent copies of X. Note that Y and Z_t are *independent* since Y only depends upon requests in column x_{s_1} while Z_t only depends upon requests in columns to the *right* of column x_{s_1}. Our goal is to prove that when $t = 8cN/d$, $\Pr(Z_t \leq 2Y) \leq 4e^{-2cN\mu^2/B^2}$.

Let $X' = X/B$, so that $X' \in [0, 1]$. Denote $\mu' = E(X') = \mu/B$. Similarly, let $Y' = Y/B$ and $Z'_t = Z_t/B$. From the discussion above, $E(Y') = N\mu'$ and $E(Z'_t) = dt\mu' = 8cN\mu'$ when $t = 8cN/d$. Because Y and Z_t are independent, Y' and Z'_t are also independent. So we have

$$\begin{aligned}
&\Pr(Z_t \leq 2Y) \\
&\quad = \Pr(Z'_t \leq 2Y') \\
&\quad = \Pr(Y' \geq 2cN\mu', Z'_t \leq 2Y') + \Pr(Y' < 2cN\mu', Z'_t \leq 2Y')
\end{aligned}$$

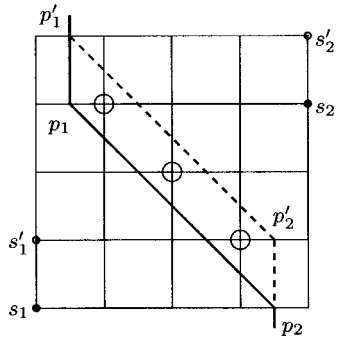

FIG. 3. The change of the Voronoi cell when the two-median moves upward by one row. We can see d new vertices (with big circles on them) are added to the Voronoi cell of s_1 when the two-median $\{s_1, s_2\}$ moves to $\{s'_1, s'_2\}$.

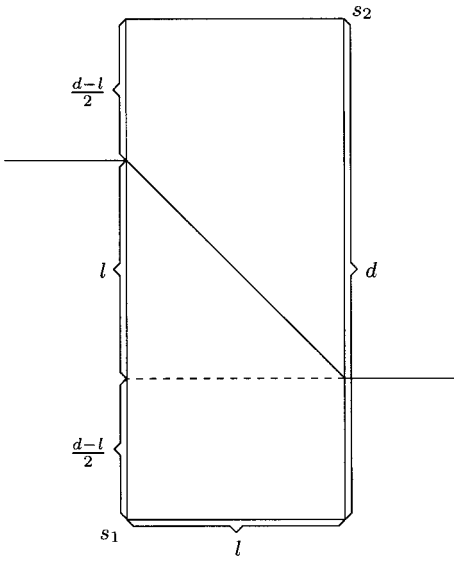

FIG. 4. The bisector between s_1 and s_2 when $l < d$.

$$
\begin{aligned}
&\le \Pr(Y' \ge 2cN\mu') + \Pr(Y' < 2cN\mu') \cdot \Pr(Z'_t \le 4cN\mu') \\
&\le \Pr(Y' \ge 2cN\mu') + \Pr(Z'_t \le 4cN\mu') \\
&\le \Pr(Y' - E(Y') \ge cN\mu') + \Pr(Z'_t - E(Z'_t) \le -4cN\mu') \\
&\le 2e^{-2c^2N\mu'^2} + 2e^{-4cN\mu'^2} \\
&\le 4e^{-2cN\mu'^2} \\
&= 4e^{-2cN\mu^2/B^2}
\end{aligned}
$$

The fourth inequality comes from the Hoeffding bound [10], which says $\Pr(|S_n - E(S_n)| \ge \delta n) \le 2e^{-2n\delta^2}$, where $S_n = \sum_{k=1}^{n} X_k$ and $X_k \in [0,1]$ are mutually independent. ∎

Lemma 8 will be used in Lemma 10 to bound the length of valid intervals. Before continuing, we quickly look at the symmetric case $l < d$. Figure 4 shows the bisector and the Voronoi cells when $l < d$. As stated in the proof of Lemma 7, when $l < d$, there are $O(n)$ candidate row-pairs, and on each candidate row-pair, there are $O(mn)$ possibilities for the two-medians. In the probabilistic algorithm, we group these $O(MN)$ possible two-medians according to the horizontal distance l, and for each group, we use a binary search to find the valid interval $[x_{\min}, x_{\max}]$, where $x_{\min}$ and $x_{\max}$ are the minimum and maximum values of x_{s_1} over all the valid two-medians. Similar to Lemma 8, we also have the following lemma to bound the length of the valid intervals when $l < d$.

Lemma 9. *Assume $1 \le l < d$. We consider the two-medians with fixed horizontal distance l on a fixed candidate row-pair. Let $[x_{\min}, x_{\max}]$ be the valid interval of these two-medians. If $X \le B$ for some constant B, then for any $c \ge 2$,*

$$\Pr(x_{\max} - x_{\min} \ge 8cM/l) \le 4e^{-2cM\mu^2/B^2} \tag{9}$$

The proof of Lemma 9 is very similar to that of Lemma 8, and is therefore omitted. With Lemma 8 and Lemma 9, we have the following algorithm.

Lemma 10. *An optimal two-median of an $M \times N$ uniform mesh can be computed in $O(MNQ \log N)$ time with probability at least $1 - 16MNe^{-2cN\mu^2/B^2}$, if the weight values of all vertices are iid random variables with some distribution X, where $X \le B$ for some constant B and $E(X) = \mu$.*

Proof. We first consider the case $l \ge d$. Fix a candidate column pair. For $d = 0$, there are $O(N)$ possible two-medians on a fixed candidate column-pair, and we simply compute the costs of all of them in $O(NQ)$ time. For $d \ge 1$, we partition the set of two-medians on the fixed candidate column-pair into $2N$ groups classified according to the vertical distance d, where $1 \le d \le N$, and whether $y_{s_1} \le y_{s_2}$ or not. For each group, finding the valid interval takes $O(Q \log N)$ time by binary search, and according to Lemma 8, we can compute the costs of all the valid two-medians in at most $8cNQ/d = O(NQ/d)$ time with probability at least $1 - 4e^{-2cN\mu^2/B^2}$. Summing over all $2N$ groups and the $d = 0$ case, the time to find an optimal two-median on a fixed candidate column-pair is $O(NQ) + O(NQ \log N) + O\left(\sum_{d=1}^{N}(NQ/d)\right) = O(NQ \log N)$ with probability at least $1 - 8Ne^{-2cN\mu^2/B^2}$. So for all M candidate column-pairs, computing an optimal two-median under the constraint $l \ge d$ requires $O(MNQ \log N)$ time with probability at least $1 - 8MNe^{-2cN\mu^2/B^2}$.

Next we consider the case $l < d$. Fix a candidate row-pair. For $l = 0$, there are $O(M)$ possible two-medians, and we compute the costs of all of them in $O(MQ)$ time. For $l \ge 1$, we classify the set of two-medians on the fixed candidate row-pair into $2N$ groups according to the horizontal distance l, where $1 \le l \le N$, and whether $x_{s_1} \le x_{s_2}$ or not. For each group, finding the valid interval takes $O(Q \log M)$ time, and by Lemma 9, computing the costs takes $O(MQ/l)$ time with probability at least $1 - 4e^{-2cM\mu^2/B^2}$. So a fixed candidate row-pair takes $O(MQ) + O(NQ \log M) + O(\sum_{l=1}^{N} MQ/l) = O(MQ \log N)$ time with probability at least $1 - 8Ne^{-2cM\mu^2/B^2}$. Thus, all N candidate row-pairs take $O(MNQ \log N)$ time with probability at least $1 - 8N^2e^{-2cM\mu^2/B^2}$.

Therefore, the running time of the algorithm is $O(MNQ \log N)$ with probability at least $1 - 16MNe^{-2cN\mu^2/B^2}$, as the lemma states. ∎

Before proceeding to Lemma 11, we clarify some notations used below. We say an event happens *with high probability* iff the probability is $1 - O(M^{-c^*})$ for some constant $c^* \ge 0$. To bound the algorithm with high probability, we also need that the value of M is polynomially bounded by the value of N, that is, $M = O(N^\beta)$ for some constant $\beta \ge 1$. In what follows, $g(N) = \omega(f(N))$ denotes that $g(N)/f(N) \to \infty$ as $N \to \infty$.

Lemma 11. *Assume $M = O(N^\beta)$ for some constant $\beta \ge 1$. An optimal two-median of an $M \times N$ uniform mesh can be computed in $O(MNQ \log N)$ time with high probability if the*

weight values of all vertices are iid random variables with some distribution X, where $X \le B$ for some constant B and $E(X) = \omega(\sqrt{\ln N/N})$.

Proof. Substituting $M = O(N^\beta)$ and $\mu \ge c_1\sqrt{\ln N/N}$, where c_1 is a constant, into Lemma 10, the running time of the algorithm is $O(MNQ \log N)$ with probability at least $1 - M^{1+1/\beta-2cc_1^2/\beta B^2}$, which satisfies the definition of high probability by taking a large enough value of c. ∎

In the next section we will see in Lemma 12 that, for uniform meshes, $Q = O(1)$. Combining Lemma 11 with Lemma 12, we have the following.

Theorem 3. *Assume $M = O(N^\beta)$ for some constant $\beta \ge 1$. An optimal two-median on an $M \times N$ uniform mesh can be computed in $O(MN \log N)$ time with high probability if the weights of all vertices of the mesh are iid random variables with some distribution X, where $X \le B$ and $E(X) = \omega(\sqrt{\ln N/N})$. The algorithm also runs in $O(MN^2)$ time in the worst case and uses $O(MN)$ space.*

Intuitively, the requirement $X \le B$ is the natural condition that the request from any single vertex does not depend on the size of the network, that is, the requests cannot get unlimited when the network becomes large. The requirement $E(X) = \omega(\sqrt{\ln N/N})$ implies that the vertices having requests can not be too small, for example, if the requests come in integral units and the probability of some nonzero request is always greater than some constant $p > 0$, then $E(X) \ge p = \omega(\sqrt{\ln N/N})$ and the condition is satisfied.

An example of using the probabilistic algorithm is as follows. Assume $M = N$, so we have an $N \times N$ square mesh, and suppose the requests are mutually independent unit requests, that is, $X = 1$ with probability $p = \ln N/\sqrt{N}$, and $X = 0$ otherwise. Then $E(X) = \omega(\sqrt{\ln N/N})$ and the conditions of Theorem 3 are satisfied. Note that for these parameters, with high probability, every row and every column of the uniform mesh will contain at least one request, that is, $m = M$ and $n = N$. Therefore, with high probability, applying the worst-case algorithm of Theorem 2 would require $O(MN^2)$ time. On the other hand, with high probability, the algorithm of Theorem 3 requires only $O(MN \log N)$ time.

5. RANGE QUERIES ON MESHES

In this section we will discuss appropriate values for Q, the time for computing $w(R)$, $c_x(R)$, and $c_y(R)$ of a basic region R. Because all the three values w, c_x, and c_y are calculated in some commutative groups, this problem is the classic range searching problem. In particular, all the three operations of the groups are simply ordinary arithmetic addition. For example, $c_x(R) = \sum_{u \in R} w(u) \cdot x_u$. The value $c_x(u) = w(u) \cdot x_u$ can be regarded as a property of u represented by a constant, and the value of $c_x(R)$ is just the arithmetic sum of the values of $c_x(u)$ for all $u \in R$.

The basic rectangular regions are orthogonal ranges. See, for example, [4]. The basic triangular regions can also be treated as orthogonal ranges, as follows. Because the slopes of the hypotenuses are always a constant, we could apply an affine transformation such that all the hypotenuses become parallel to some axis. For example, as we will see in the next paragraph, in our specific problem, we will need to query vertices (x, y) in the two-sided triangular region $x \ge c_1$ and $x + y \le c_2$. We could transform each vertex (x, y) to a new coordinate (x, z) where $z = x + y$, and then apply the orthogonal range searching $x \ge c_1$ and $z \le c_2$ under the new coordinate.

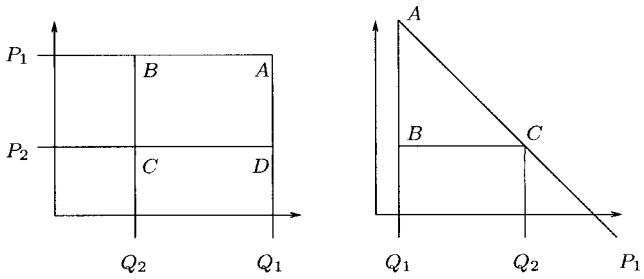

FIG. 5. The decomposition of rectangular and triangular regions into two-sided regions. P_1, P_2, Q_1, Q_2 are points at infinity of the corresponding directions.

Our problem only need counting-type queries, that is, we do not need to *list* all vertices in the query range. In the counting case, orthogonal range searching is equivalent to solving the two-sided case, also known as dominance search, where two of the four boundaries of the search region is infinity. Refer to Figure 5. The basic rectangular region $ABCD$ (the left figure) can be decomposed into the sum/difference of four two-sided rectangle regions P_1AQ_1, P_1BQ_2, P_2DQ_1 and P_2CQ_2. If we can make the two-sided queries efficiently, we can make the four-sided query as well up to a constant factor. Similarly, a basic triangular region ABC (the right figure) can be decomposed into two two-sided triangular regions P_1AQ_1 and P_1CQ_2 plus a three-sided rectangular region Q_1BCQ_2. The three-sided rectangles can be further decomposed into the difference of two two-sided rectangles. So we only need to find efficient ways to query two-sided triangular regions, as discussed in the previous paragraph.

Therefore, we have reduced our problem to the two-sided orthogonal range counting problem. The range counting algorithm depends on whether the mesh is uniform or not. We first deal with the uniform case. Range counting on a uniform mesh is easy. A simple "table lookup" approach can do everything optimally in constant time.

Lemma 12. *In an $M \times N$ uniform mesh, the values of $w(R)$, $c_x(R)$ and $c_y(R)$ of any basic region R can be computed in $O(1)$ time using $O(MN)$ preprocessing time and $O(MN)$ additional space.*

Proof. We can always shrink a basic region such that it contains the same set of vertices of the mesh and the corners of the basic region are on some vertices of the mesh. So we only need to query the regions whose corners are the vertices of the mesh.

Let the corner A in Figure 5 be the location of a vertex u. We denote by $R_1(u)$ the two-sided rectangular region P_1AQ_1 in the left figure, and by $R_2(u)$ the two-sided triangular region P_1AQ_1 in the right figure. The table lookup approach is to simply precompute the values of $w(u)$, $c_x(u)$ and $c_y(u)$ for both regions $R_1(u)$ and $R_2(u)$ for each vertex u. It is easy to do this in $O(MN)$ time; hence, the lemma follows. ∎

The table lookup approach does not work well on nonuniform meshes. If we use table lookup, we need to precompute $O(mn)$ two-sided rectangular regions and $O(mb^-)$ two-sided triangular regions, where b^- is the number of slope -1 diagonals that have requests. The value of b^- is bounded by r, which results in a preprocessing time larger than the bound in Lemma 7. Moreover, the query time is still $O(\log m)$ because we need a binary search.

There are many results on range searching on meshes, but to the best of the authors' knowledge, none of them can do better than $O(\log m)$ in counting-type queries while at the same time keeping the preprocessing time low enough compared to the bound in Lemma 7 to be useful to us. Hence, we will therefore use the classic priority search tree [14] data structure. This data structure is not only practical, but also allows arbitrary real values instead of only integers.

Lemma 13 ([14]). *In an $m \times n$ nonuniform mesh, the values of $w(R)$, $c_x(R)$ and $c_y(R)$ of any basic region R can be computed in $O(\log m)$ time using $O(r)$ preprocessing time and $O(r)$ additional space.*

Finally, as we did in Section 3 and 4, combining Lemma 13 with Lemma 7 gives Theorem 1. Combining Lemma 12 with Lemma 7 gives Theorem 2, while combining Lemma 12 with Lemma 11 gives Theorem 3.

REFERENCES

[1] S. Arora, P. Raghavan, and S. Rao, Approximation schemes for Euclidean k-medians and related problems, Proc. 30th Ann ACM Symp Theory Comput, 1998, pp. 106–113.

[2] R.E. Burkard, E. Cela, and H. Dollani, 2-medians in trees with pos/neg weights, Discrete Appl Math 105 (2000), 51–71.

[3] M. Charikar, S. Guha, E. Tardos, and D.B. Shmoys, A constant-factor approximation algorithm for the k-median problem, J Comput Syst Sci 65 (2002), 129–149.

[4] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, Computational geometry—Algorithms and applications, Springer-Verlag, Berlin, 2000, 2nd ed.

[5] E. Erkut, R.L. Francis, and T.J. Lowe, Locating two facilities on a tree subject to distance constraints, Transportation Sci 22 (1988), 199–208.

[6] B. Gavish and S. Sridhar, Computing the 2-median on tree networks in $O(n \lg n)$ time, Networks 26 (1995), 305–317.

[7] T. Gross and D.R. O'Hallaron, iWarp: Anatomy of a parallel computing system, MIT Press, Cambridge MA, 1998.

[8] S. Guha and S. Khuller, Greedy strikes back: Improved facility location algorithms, Proc. 9th Ann ACM-SIAM Symp Discr Algorithms, 1998, pp. 649–657.

[9] R. Hassin and A. Tamir, Improved complexity bounds for location problems on the real line, Oper Res Lett 10 (1991), 395–402.

[10] W.J. Hoeffding, Probability inequalities for sums of bounded random variables, J Am Stat Assoc 58 (1963), 713–721.

[11] S.C. Ku, C.J. Lu, B.F. Wang, and T.C. Lin, Efficient algorithms for two generalized 2-median problems on trees, Proc. 12th Ann Int Symp Algorithms Computation, 2001, pp. 768–778.

[12] F.C.M. Lau, P.K.W. Cheng, and S.S.H. Tse, An algorithm for the 2-median problem on two-dimensional meshes, Comput J 44 (2001), 101–108.

[13] J.H. Lin and J.S. Vitter, ϵ-approximations with minimum packing constraint violation, Proc. 24th Ann ACM Symp Theory Comput, 1992, pp. 771–782.

[14] E.M. McCreight, Priority search trees, SIAM J Comput 14 (1985), 257–276.

[15] A. Tamir, An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs, Oper Res Lett 19 (1996), 59–64.

[16] S.S.H. Tse and F.C.M. Lau, An approximation solution for the 2-median problem on two-dimensional meshes, Proc. 19th Int Conference Advanced Informat Networking Appl, 2005, Vol. 2, pp. 457–460.

[17] A. Vigneron, L. Gao, M.J. Golin, G.F. Italiano, and B. Li, An algorithm for finding a k-median in a directed tree, Informat Process Lett 74 (2000), 81–88.