Theoretical Computer Science 592 (2015) 39–48

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Group nearest-neighbor queries in the L_1 plane ☆

Wanbin Son [a], Sang Won Bae [b], Hee-Kap Ahn [c,*]

[a] MADALGO, Department of Computer Science, Aarhus University, Aarhus, Denmark
[b] Department of Computer Science, Kyonggi University, Suwon, Republic of Korea
[c] Department of Computer Science and Engineering, Pohang University of Science and Technology, Republic of Korea

ARTICLE INFO

Article history:
Received 12 December 2014
Received in revised form 29 April 2015
Accepted 9 May 2015
Available online 14 May 2015
Communicated by G.F. Italiano

Keywords:
k-nearest-neighbor query
Plane sweep algorithm
Orthogonal range searching
Segment dragging query
Geometric data structure

ABSTRACT

Let P be a set of n points in the plane. The k-nearest-neighbor (abbreviated as k-NN) query problem is to preprocess P into a data structure that quickly reports k closest points in P for a query point q. This paper addresses a generalization of the k-NN query problem to a query set Q of points, namely, the *group* k-nearest-neighbor query problem, in the L_1 plane. More precisely, a query is assigned with a set Q of at most m points and a positive integer k with $k \leq n$, and the distance between a point p of P and a query set Q is defined as the sum of L_1 distances from p to all $q \in Q$. The maximum number m of query points Q is assumed to be known in advance and to be at most n. In this paper, we propose two algorithms, one based on the range tree and the other based on a data structure for segment dragging queries, and obtain the following complexity bounds: (1) a group k-NN query can be handled in $O(T_{\min} \log n + (k+m^2)(\log\log n + \log m))$ time after preprocessing P using $O(m^2 n \log^2 n)$ space, where $T_{\min} = \min\{k+m, m^2\}$, or (2) a group k-NN query can be handled in $O((k+m)\log^2 n + m^2(\log^\epsilon n + \log m))$ time after preprocessing P using $O(m^2 n)$ space, where $\epsilon > 0$ is an arbitrarily small constant. We also show that our approach can be applied to the weighted group k-nearest-neighbor query problem and the group k-farthest-neighbor query problem.

1. Introduction

The *nearest-neighbor query* problem, also known as the proximity query or closest-point query problem, is one of the fundamental problems in computer science. The problem is, for a set P of points in a metric space M, to preprocess P such that given a query point $q \in M$, one can find the closest point of q in P quickly. Many areas in computer science including computational geometry, databases, machine learning, and computer vision use the nearest-neighbor query as one of the most primitive operations. Various solutions to the nearest-neighbor query problem have been proposed. A straightforward algorithm for this problem is the sequential search. Several tree-based data structures [1–3] have been proposed to increase the efficiency of the nearest-neighbor query. Approximate nearest-neighbor algorithms have also been studied [4–6].

The k-nearest-neighbor (abbreviated as k-NN) query problem is a generalization of the nearest-neighbor query problem whose goal is to find the k closest points of the query in the data set P. For the k-NN query problem, several disk-based in-

☆ Work by Wanbin Son was supported by Center for Massive Data Algorithmics, a center of the Danish National Research Foundation. Work by Hee-Kap Ahn was supported by NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea. Work by Sang Won Bae was supported by the Contents Convergence Software Research Center funded by the GRRC Program of Gyeonggi Province, South Korea.

* Corresponding author.
E-mail addresses: wson@cs.au.dk (W. Son), swbae@kgu.ac.kr (S.W. Bae), heekap@postech.ac.kr (H.-K. Ahn).

http://dx.doi.org/10.1016/j.tcs.2015.05.011
0304-3975/

dex structures [7–9] have been proposed. Roussopoulos et al. [8] presented an R-tree-based algorithm, and Jagadish et al. [7] showed a B^+-tree-based algorithm for the k-NN query problem. Arya et al. [5] proposed an approximation algorithm for the k-NN query problem.

In this paper, we focus on a generalized version of the k-NN query problem, namely, the *group k-nearest-neighbor query* problem, where a query is associated with a set Q of more than one query point and a positive integer k, and the k closest data points in P with respect to Q are to be reported. The distance (or, the closeness) of a data point $p \in P$ with respect to the query set Q is defined as the sum of distances between p and each $q \in Q$, that is, $\sum_{q \in Q} \mathrm{dist}(p, q)$ where $\mathrm{dist}(p, q)$ is the distance between two points. We shall call this quantity the *sum-of-distance of p with respect to Q*. The goal is, for a given set P of n data points in the plane, to preprocess P into a data structure that efficiently handles group k-NN queries.

To our best knowledge, the group nearest-neighbor query problem has been first studied and coined by Papadias et al. [10]. They proposed a heuristic method in the Euclidean plane and showed several applications of the group nearest-neighbor query and the sum-of-distance function in GIS (Geographic information system), clustering, and outlier detection. Later, Papadias et al. [11] considered the problem under aggregate distance functions. Recently, Agarwal et al. [12] studied the expected NN queries, where the location of each input point and/or query point is specified as a probability density function. Wang and Zhang [13] improved the result of Agarwal et al. on the expected NN queries for the case of an uncertain query point in the L_1 plane. Li et al. [14] presented an approximation algorithm with factor 3 for the group nearest-neighbor query problem when the aggregation function is the sum-of-distance function. They showed that their algorithm can be extended to the group k-NN query problem. Yiu et al. [15] studied the group k-NN query problem in road networks.

In this paper we study the group k-NN query problem in the L_1 plane, that is, the distance of a data point is defined as the sum of the L_1 distances. The L_1 distance, also known as the Manhattan distance, reflects well-connected road networks in metro areas such as Manhattan [16]. The L_1 distance is also appropriate for the automated design of VLSI circuits [17].

A brute-force way to handle a group k-NN query computes the sum-of-distances of all the data points in $O(nm)$ time and finds the k closest points among them using a selection algorithm in $O(n)$ time. It is unlikely that we achieve a $o(n)$-time algorithm without any preprocessing because of the lower bound for the selection problem [18]. To avoid $\Omega(n)$ query time, especially when k is much smaller than n, we consider the following approach. We first compute the region that has the minimum sum-of-distance with respect to Q. If the region contains k or more points of P, we just report k of them. Otherwise, we report the data points in the region and then expand the region by increasing the sum-of-distance value. During the expansion, we report every data point that is newly included to the expanding region until we have k points reported. In this case, we may not need to consider all the points in P.

Our results We present two efficient algorithms for the group k-NN query problem in the L_1 plane: RngAlgo and SgmtAlgo. We assume that the maximum number m of query points is known in advance and $m \leq n$. In many applications, this is a reasonable assumption. We also assume that no two query points have the same x-coordinate or y-coordinate. We can easily handle such degenerate cases by using the lexicographic order. The model of computation in this paper is a unit-cost RAM with word size logarithmic in n.

- RngAlgo answers a group k-NN query in $O(T_{\min} \log n + (k + m^2)(\log\log n + \log m))$ time after preprocessing P into a data structure using $O(m^2 n \log^2 n)$ space,[1] where $T_{\min} = \min\{k + m, m^2\}$.
- SgmtAlgo answers a group k-NN query in $O((k + m)\log^2 n + m^2(\log^{\epsilon} n + \log m))$ time after preprocessing P into a data structure using $O(m^2 n)$ space,[1] where $\epsilon > 0$ is an arbitrary small constant.

Note that both of our query algorithms spend $o(n)$ time to answer a query when k and m are reasonably small. Moreover, our approach can be used for the same problem in the L_∞ plane. We also show that the approach can be applied to the weighted group k-NN query problem and the group k-farthest-neighbor query problem in the L_1 plane.

2. Observations on the sum-of-distance function

In this section we investigate the sum-of-distance function $\mathrm{sumdist}_Q : \mathbb{R}^2 \to \mathbb{R}$ with respect to a query set Q. The function is defined as $\mathrm{sumdist}_Q(p) = \sum_{q \in Q} \mathrm{dist}(p, q)$, where $\mathrm{dist}(p, q)$ is the L_1 distance between two points p and q in the plane. For a point p, let $x(p)$ and $y(p)$ denote the x-coordinate and y-coordinate of p, respectively. For a compact set S, let ∂S denote the boundary of S.

We first observe that $\mathrm{sumdist}_Q$ is a convex function. Note that the L_1 distance function dist is convex and piecewise linear. Since $\mathrm{sumdist}_Q$ is the sum of these distance functions, it is also convex [19]. For any real number $c \in \mathbb{R}$, let $\mathcal{E}_Q(c) := \{x \in \mathbb{R}^2 \mid \mathrm{sumdist}_Q(x) \leq c\}$ be the sublevel set of function $\mathrm{sumdist}_Q$. The sublevel set of a convex function is also convex by definition [19]. We thus get the following lemma.

Lemma 1. *The sum-of-distance function $\mathrm{sumdist}_Q$ is convex, and therefore its sublevel set $\mathcal{E}_Q(c)$ is convex for any $c \in \mathbb{R}$.*

[1] We do not specify the preprocessing times for building data structures because they are not important issues for online query problems. All the data structures can be built in polynomial time.

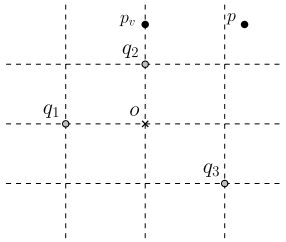

Fig. 1. The grid constructed from $Q = \{q_1, q_2, q_3\}$ and two data points p_v and p. The point o minimizes sumdist_Q.

2.1. The set of points minimizing sumdist_Q

Consider the problem of identifying the region of points that minimize the function sumdist_Q. This problem is well known as the Fermat–Weber problem [20], and a result in the L_p metric was given in 1964 [21]. We present a simple proof for the problem in the L_1 plane in the following.

For each query point, draw a horizontal line and a vertical line passing through the point. See Fig. 1. We denote by $G(Q)$ the grid of Q constructed in this way. Let x_v be the median among the x-coordinates of the query points when $|Q|$ is odd, where $|Q|$ denotes the cardinality of Q.

Lemma 2. *For any two distinct points $p = (x, y)$ and $p_v = (x_v, y)$, we have $\text{sumdist}_Q(p) > \text{sumdist}_Q(p_v)$ if $|Q|$ is odd.*

Proof. See Fig. 1 for an illustration of the proof. Without loss of generality, we assume that $x > x_v$. Let Q_1 denote the set of query points whose x-coordinates are at most x_v, and $Q_2 = Q \setminus Q_1$. So we always have $|Q_1| > |Q_2|$. For any $q \in Q_1$ and $q' \in Q_2$, we have the followings.

$$\text{dist}(p, q) = \text{dist}(p_v, q) + |x - x_v|, \quad \text{dist}(p, q') \geq \text{dist}(p_v, q') - |x - x_v|.$$

Because $|Q_1| > |Q_2|$, we have

$$\begin{aligned}
\text{sumdist}_Q(p) &= \sum_{q \in Q_1} \text{dist}(p, q) + \sum_{q' \in Q_2} \text{dist}(p, q') \\
&\geq \sum_{q \in Q_1} (\text{dist}(p_v, q) + |x - x_v|) + \sum_{q' \in Q_2} (\text{dist}(p_v, q') - |x - x_v|) \\
&> \sum_{q \in Q} \text{dist}(p_v, q) = \text{sumdist}_Q(p_v). \qquad \square
\end{aligned}$$

Let y_v be the median among the y-coordinates of the query points when $|Q|$ is odd. Then, the point (x_v, y_v), called *the median point*, minimizes sumdist_Q over all the points in $\mathbb{R}^2$ when $|Q|$ is odd.

When $|Q|$ is even, the *median cell*, denoted by g_{med}, of $G(Q)$ is defined by two query points q_x and $q_{x'}$ whose x-coordinates are the median among the x-coordinates of the query points in Q, and two query points q_y and $q_{y'}$ whose y-coordinates are the median among the y-coordinates of the query points in Q. The median cell is the cell of $G(Q)$ bounded by two vertical lines, one passing through q_x and one passing through $q_{x'}$, and two horizontal lines, one passing through q_y and $q_{y'}$. The following lemma can be proved in a way analogous to Lemma 2.

Lemma 3. *If $|Q|$ is odd, the median point (x_v, y_v) minimizes the function sumdist_Q. If $|Q|$ is even, any point in the median cell g_{med} of $G(Q)$ minimizes sumdist_Q.*

2.2. Properties of cells of $G(Q)$

Let g be any cell of $G(Q)$. By the definition of $G(Q)$, every interior point of g has the same set of query points lying to the left of it; the same claim holds on the query points lying to the right of, above, and below the interior point. Let $Q_l(g)$, $Q_r(g)$, $Q_t(g)$ and $Q_b(g)$ denote the sets of query points that are to the left, right, above, and below of any point in the interior of g, respectively.

Lemma 4. *The function sumdist_Q is linear on a cell of $G(Q)$.*

Proof. Let $r = (x, y)$ be a point in a cell g. Then we have

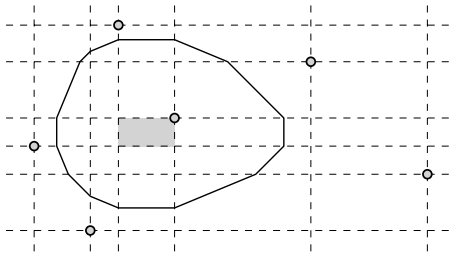

Fig. 2. The grid $G(Q)$ (dashed lines), the median cell (gray region) that minimizes function sumdist_Q, and the boundary of a sublevel set $\mathcal{E}_Q(c)$ that has the same sumdist_Q value (solid line segments).

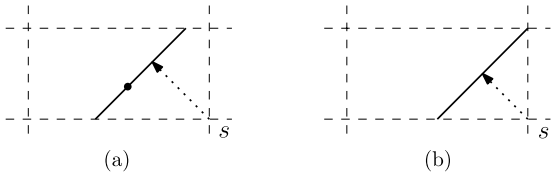

Fig. 3. Two types of events: (a) point event, (b) corner event.

$$\text{sumdist}_Q(r) = \sum_{q \in Q_l(g)} (x - x(q)) + \sum_{q \in Q_r(q)} (x(q) - x) + \sum_{q \in Q_t(g)} (y(q) - y) + \sum_{q \in Q_b(g)} (y - y(q)),$$

which is a sum of linear functions. Therefore, function sumdist_Q is linear on any cell of $G(Q)$. □

It is not difficult to see that function sumdist_Q is continuous on two neighboring cells of $G(Q)$, and therefore the function is continuous in its domain.

Corollary 1. *The function* sumdist_Q *is continuous and piecewise linear.*

By Lemmas 1 and 4, and Corollary 1, we know that $\mathcal{E}_Q(c)$ is a convex polygon for any fixed c unless it is empty or degenerate. Such a polygon has $O(m)$ edges, each of which is contained in a cell of $G(Q)$.

For instance, Fig. 2 shows the grid subdivision of the plane by six query points and the boundary of a sublevel set $\mathcal{E}_Q(c)$ consisting of points whose sumdist_Q values are at most $c > 0$. All the points on $\partial\mathcal{E}_Q(c)$ have the same sumdist_Q value which is c, and therefore they form a 6-ellipse in the L_1 plane. Observe that the shape of $\partial\mathcal{E}_Q(c)$ is determined solely by the query points and the value c. In general, it is not symmetric as seen in Fig. 2.

3. Algorithms for group *k*-nearest-neighbor queries

In this section we present two algorithms that compute the k nearest neighbors of P for a given query set Q and k. We assume that $|Q|$ is even from now on. The other case can also be handled in a similar way.

We start with describing our algorithm as follows. Given a query composed of Q and k, we first compute g_{med} and set $c := \text{sumdist}_Q(r)$ for any $r \in g_{\text{med}}$. Then, we have $\mathcal{E}_Q(c) = g_{\text{med}}$. If there are k or more points of P in $\mathcal{E}_Q(c)$, report any k of them. Otherwise, we expand it by increasing c. During the expansion, we report every data point in P that is newly included in $\mathcal{E}_Q(c)$ until we report k data points. To do this efficiently, we preprocess P so that we consider only $O(k + m^2)$ distinct values of c in the increasing order of sumdist_Q value and report the k nearest neighbors.

More precisely, we encounter the following events during the expansion.

- *Point event*: An edge of $\mathcal{E}_Q(c)$ hits a data point (Fig. 3(a)).
- *Corner event*: An edge of $\mathcal{E}_Q(c)$ hits a corner of a cell of $G(Q)$ (Fig. 3(b)).

We maintain these events in an *event queue* $\mathcal{Q}$, which is a priority queue indexed by the sumdist_Q value of its associated point or corner. After initializing the event queue, we insert four corners of the median cell as corner events to $\mathcal{Q}$. We then process the events of $\mathcal{Q}$ one by one as follows. For a point event e, we report the data point associated with the event. If this is the k-th point that we have reported, we are done. Otherwise, we find the next event in the cell containing the point and insert it to $\mathcal{Q}$. For a corner event e, we find the next event in the cell that is associated with e and insert it to $\mathcal{Q}$. We also consider each unvisited cell having e as a corner and find the next event e' in the cell.

We skip the process of finding point events when the cell under consideration contains no data point. We use range emptiness queries to check whether a cell of $G(Q)$ contains no data point. Chan et al. [22] proposed efficient data structures

for 2-D orthogonal range emptiness queries on points in rank space where points have coordinates on the integer grid $[n]^2 = \{0, \ldots, n-1\}^2$. To use this data structure, we reduce P and the cells of $G(Q)$ to points and query rectangles in rank space, respectively. A range searching problem in $\mathbb{R}^d$ can be reduced to a range searching problem in rank space by a standard technique [23]. Reducing P can be done by sorting coordinates of points in P in each dimension. Reducing a range query can be done by using binary search for the sorted coordinates of P in each dimension. We have $O(m^2)$ range queries, but they are composed of $O(m)$ x-coordinates and y-coordinates. So $O(m)$ binary searches suffice for reducing all the range queries.

The correctness of our algorithm immediately follows from the convexity of $\mathcal{E}_Q(c)$ and the fact that the events are processed in the increasing order of sumdist_Q value. That is, when we process a point event e, the data point of e has the minimum sumdist_Q value among all the unreported data points, so our algorithm reports the k nearest neighbors in P with respect to Q correctly.

Let us now analyze the complexity of our algorithm. We preprocess P to construct the orthogonal range emptiness query structure. We construct the grid $G(Q)$ in $O(m^2)$ time, and then check the emptiness of each cell of $G(Q)$. We reduce all cells to query rectangles in rank space in $O(m^2 + m \log n)$ time by $O(m)$ binary searches for each dimension. Orthogonal range emptiness for all cells can be determined in $O(m^2 \log\log n)$ total time using $O(n \log\log n)$ space, or in $O(m^2 \log^{\epsilon} n)$ total time using $O(n)$ space after preprocessing P [22].

Since $\mathcal{E}_Q(c)$ is convex, the number of cells intersected by $\partial\mathcal{E}_Q(c)$ is $O(m)$ for any fixed c. Until we report k nearest neighbors, $O(m^2)$ corner events and $k + O(m)$ point events can be inserted to $\mathcal{Q}$, so it takes $O((k+m^2)\log m)$ time to insert and delete them from the event queue.

The corner events in each cell can be found easily. The only remaining part of the algorithm is to process the data points contained in a cell g of $G(Q)$ to support the operation of finding next point events in g efficiently. We will describe two methods for this and analyze their time complexities in the following.

3.1. Detecting point events in a cell

We propose two methods to find events in a cell in the increasing order of sumdist_Q value. As aforementioned in Lemma 4, function sumdist_Q is linear on a cell g of $G(Q)$. Let $\ell(g)$ be a line such that for any two points r and r' on $g \cap \ell(g)$, $\text{sumdist}_Q(r) = \text{sumdist}_Q(r')$. We sweep g by $\ell(g)$ from the corner with the minimum sumdist_Q value over all the points in g in the direction orthogonal to $\ell(g)$. Let $\text{slp}(\ell(g))$ be the slope of $\ell(g)$. Recall that we denote by $Q_l(g)$, $Q_r(g)$, $Q_t(g)$ and $Q_b(g)$ the sets of query points that are to the left, right, above, and below of any point in the interior of g, respectively.

Lemma 5. *For a cell g of $G(Q)$, the slope $\text{slp}(\ell(g))$ of $\ell(g)$ is $\frac{|Q_r(g)|-|Q_l(g)|}{|Q_b(g)|-|Q_t(g)|}$.*

Proof. Let $r = (x, y)$ and $r' = (x + \delta_x, y + \delta_y)$ be two arbitrary points on $g \cap \ell(g)$ with $\delta_y/\delta_x = \text{slp}(\ell(g))$. Then by the following equations, the lemma holds.

$$\begin{aligned}\text{sumdist}_Q(r') &= \text{sumdist}_Q(r) + \delta_x(|Q_l(g)| - |Q_r(g)|) + \delta_y(|Q_b(g)| - |Q_t(g)|)\\ &= \text{sumdist}_Q(r) + \delta_x(|Q_l(g)| - |Q_r(g)|) + \delta_x \cdot \text{slp}(\ell(g))(|Q_b(g)| - |Q_t(g)|)\\ &= \text{sumdist}_Q(r). \qquad \square\end{aligned}$$

Since m is given in advance, we know all possible slopes of cells before Q is given. Let S denote the set of the possible slopes. There are $O(m^2)$ slopes in S by Lemma 5. We preprocess P for all the slopes in S as described in the following subsections.

3.1.1. Orthogonal range query based algorithm

The first algorithm, RngAlgo, is based on an orthogonal range query structure, namely, the range tree [24]. Recently Rahul et al. [25] introduced a data structure based on the orthogonal range query structure such that the first k points from n weighted points in $\mathbb{R}^d$ can be reported in sorted order in $O(\log^{d-1} n + k \log\log n)$ query time using the data structure. This structure can be constructed using $O(n \log^d n)$ space in the preprocessing phase.

Consider a slope $s \in S$. For each data point p, we set the weight of p to the y-intercept value of the line of slope s passing through p. Then we construct the data structure of Rahul et al. for the weighted data points. We do this for every slope of S and maintain one data structure of Rahul et al. for each slope. Let $\mathcal{R}$ denote the set of these data structures. By using $\mathcal{R}$, we can get the first point event from each cell in $O(\log n)$ time, and then we spend $O(\log\log n)$ time for finding the next point event in each cell.

Until we report k nearest neighbors, we try to find a point event only from $O(\min\{k+m, m^2\})$ cells. Therefore, it takes $O(k \log\log n + \min\{k+m, m^2\}\log n)$ time to find $k + O(m)$ point events. The following theorem summarizes the complexities of the algorithm RngAlgo.

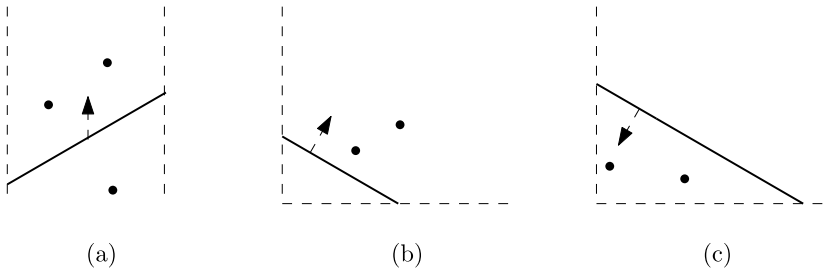

Fig. 4. Three types of segment dragging: (a) dragging by parallel tracks, (b) dragging out of a corner, (c) dragging into a corner.

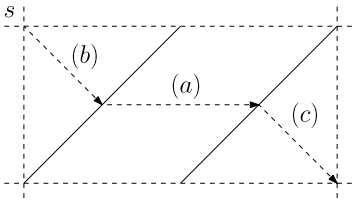

Fig. 5. We sweep a cell using a segment in three different types of dragging.

Theorem 1. *The algorithm* RngAlgo *reports the k nearest neighbors in* $O(T_{\min} \log n + (k+m^2)(\log\log n + \log m))$ *time after preprocessing P into a data structure using* $O(m^2 n \log^2 n)$ *space, where* $T_{\min} = \min\{k+m, m^2\}$.

Proof. The correctness of the algorithm follows from the discussion above. Until we report k nearest neighbors, $k + O(m)$ point events and $O(m^2)$ corner events can be inserted to $\mathcal{Q}$. The time to maintain $\mathcal{Q}$ is $O((k+m^2)\log m)$. Orthogonal range emptiness for all cells can be determined in $O(m^2 \log\log n)$ time using $O(n \log\log n)$ space. To find the point events, RngAlgo spends $O(\min\{k+m, m^2\}\log n + k\log\log n)$ time using $O(m^2 n \log^2 n)$ space. Therefore the theorem holds. □

3.1.2. Segment dragging query based algorithm

The second algorithm, SgmtAlgo, uses the segment dragging query structure [26–28] to find the events in a cell of $G(Q)$. The segment dragging query, informally speaking, is to determine the next point hit by the given query segment $\overline{st}$ of orientation θ when it is dragged to $\overline{s't'}$, with s in direction ϕ_s and t in direction ϕ_t. There are three types of segment dragging queries: (a) dragging by parallel tracks, (b) dragging out of a corner, and (c) dragging into a corner (see Fig. 4).

Chazelle [27] and Mitchell [28] showed that one can preprocess a set P of n points into a data structure of size $O(n)$ that answers the segment dragging queries of type (a) and (b) in $O(\log n)$ time. Bae et al. [26] proposed a data structure of size $O(n)$ that answers segment dragging queries of type (c) in $O(\log^2 n)$ time.

We sweep a cell using a segment in one of three different types (see Fig. 5). From the corner s with the minimum sumdist_Q value, we sweep the cell using a segment dragging of type (b) 'dragging out of a corner' until the segment hits another corner. We then apply a segment dragging of type (a) 'dragging by parallel tracks' from the corner until it hits the third corner. Afterwards, we apply a segment dragging of type (c) 'dragging into a corner' until it hits the last corner.

If we adopt the segment dragging query structure, it takes $O(\log^2 n)$ time to find each point event. Therefore, it takes $O((k+m)\log^2 n)$ time in total to report $k + O(m)$ point events.

Theorem 2. *The algorithm* SgmtAlgo *computes the k nearest neighbors in* $O((k+m)\log^2 n + m^2(\log^\epsilon n + \log m))$ *time after preprocessing P into a data structure using* $O(m^2 n)$ *space, where* $\epsilon > 0$ *is an arbitrary small constant.*

Proof. The correctness of the algorithm follows from the discussion above. Until we report k nearest neighbors, $k + O(m)$ point events and $O(m^2)$ corner events can be inserted to $\mathcal{Q}$. The time to maintain $\mathcal{Q}$ is $O((k+m^2)\log m)$. Orthogonal range emptiness for all cells can be determined in $O(m^2 \log^\epsilon n)$ total time using $O(n)$ space. To find the point events, SgmtAlgo spends $O((k+m)\log^2 n)$ time using $O(m^2 n)$ space. Therefore the theorem holds. □

4. Weighted group *k*-nearest-neighbor queries

Imagine that you prefer some query points to the others. A good and typical way of finding the group k-NN that reflects your preference well is to assign weights to query points and use a distance metric modified by weights of query points. In this section, we consider a weighted version of the group k-NN query problem.

The problem is defined under the same settings as the group k-NN query problem except that each query point has a weight and the distance between a data point and a query point is multiplied by the weight of the query point. Let $w(q)$

be the weight of a query point q. In the weighted group k-NN query problem, we define the weighted distance function wsdist_Q of a point p with respect to Q and the sublevel set of function wsdist_Q as follows.

$$\text{wsdist}_Q(p) = \sum_{q \in Q} w(q) \cdot \text{dist}(p, q),$$

$$\mathcal{E}_Q^*(c) := \{x \in \mathbb{R}^2 \mid \text{wsdist}_Q(x) \leq c\}.$$

In this paper, we consider a simple version of the problem in which $w(q)$ is an integer and the upper and lower bounds of $w(q)$ are known in advance. We will explain how to extend our algorithms to solve the problem without these two assumptions later.

Let c_w be a fixed positive value. We first consider the case when $w(q)$ is a positive integer between 1 and c_w. Then we continue with the case when $w(q)$ is an integer between $-c_w$ and c_w.

4.1. Group k-nearest-neighbor queries with positive integer weights

Consider the case that all the weights are positive integers between 1 and c_w. The group k-NN query problem with positive integer weights is conceptually same to the group k-NN query problem with $\sum_{q \in Q} w(q)$ query points: $w(q)$ query points at the position of q. It means that the properties of the group k-NN query problem hold in this problem, and therefore the algorithm in Section 3 works. However, the duplicated query points make the algorithm inefficient, so we refine the properties and the algorithm for the group k-NN query problem to avoid such inefficiency.

Clearly, wsdist_Q and $\mathcal{E}_Q^*$ are convex. We use an approach similar to the one for the group k-NN query problem in Section 3. We first compute the set of points r such that $\text{wsdist}_Q(r) = \min_{x \in \mathbb{R}^2} \text{wsdist}_Q(x)$. This set forms either a point or a region in the plane. We initially set $c = \text{wsdist}_Q(r)$ and then expand $\mathcal{E}_Q^*(c)$ by increasing c. During the expansion, we report the data points hit by $\mathcal{E}_Q^*(c)$ until we report the k-th point.

Consider the problem of identifying the region of points that minimizes wsdist_Q. We define a weighted x-median among all the x-coordinates of the query points as follows.

Definition 1. For a set Q of weighted query points, a query point $q_v \in Q$ is a *weighted x-median* of Q if and only if

$$\sum_{q \in Q_r(q_v)} w(q) \leq \sum_{q \in Q_l(q_v)} w(q) + w(q_v) \quad \text{and}$$

$$\sum_{q \in Q_l(q_v)} w(q) \leq \sum_{q \in Q_r(q_v)} w(q) + w(q_v),$$

where $Q_l(q_v) = \{q \in Q \mid x(q) < x(q_v)\}$ and $Q_r(q_v) = \{q \in Q \mid x(q) > x(q_v)\}$.

A weighted y-median among all the y-coordinates of the query points can be defined similarly. By definition, there can be at most two weighted x-medians and at most two y-medians. The following lemma describes the region that minimizes wsdist_Q is. The lemma can be proven analogously as in the proof of Lemma 2.

Lemma 6. *A point $r \in \mathbb{R}^2$ has the minimum wsdist_Q value among all the points in the plane if and only if $x(r)$ is between the smaller and larger x-coordinates of weighted x-medians and $y(r)$ is between the smaller and larger y-coordinates of weighted y-medians of query points.*

Let us consider the algorithm. We compute the region that minimizes wsdist_Q by using Lemma 6. The weighted x-medians and y-medians can be computed by checking whether each query point satisfies the condition or not in order. Initially, let $c = \text{wsdist}_Q(r)$, where r is a point that minimizes wsdist_Q. If there are k or more points in $\mathcal{E}_Q^*(c)$, then we report k points among them. Otherwise, we expand $\mathcal{E}_Q^*(c)$ by increasing c. To find data points hit by $\mathcal{E}_Q^*$ during the expansion, we use the same methods as the ones in Section 3 after we redefine the definition $\text{slp}(\ell(g))$ in Lemma 5 as follows. We denote by $w_l(g)$, $w_r(g)$, $w_t(g)$, and $w_b(g)$ the sums of weights of query points that are to the left, right, above, and below of a point p contained in the interior of a cell g of $G(Q)$, respectively. We redefine $\text{slp}(\ell(g))$ for this problem in the following lemma, which can be proven analogously as in the proof of Lemma 5.

Lemma 7. *For a cell g of $G(Q)$, the slope $\text{slp}(\ell(g))$ is $\frac{w_r(g) - w_l(g)}{w_b(g) - w_t(g)}$.*

Now let us analyze the complexities of the algorithms. Let RngAlgo$^+$ and SgmtAlgo$^+$ be two algorithms that are based on RngAlgo and SgmtAlgo, respectively. Their complexities are roughly the same to those of RngAlgo and SgmtAlgo. The only exception is the number of possible slopes of cells $\text{slp}(\ell(g))$. Consider the number of possible slopes. Because $\sum_{q \in Q} w(q) \leq c_w m$, the numerator and the denominator of $\text{slp}(\ell(g))$ are integers between $-c_w m$ and $c_w m$, so there are

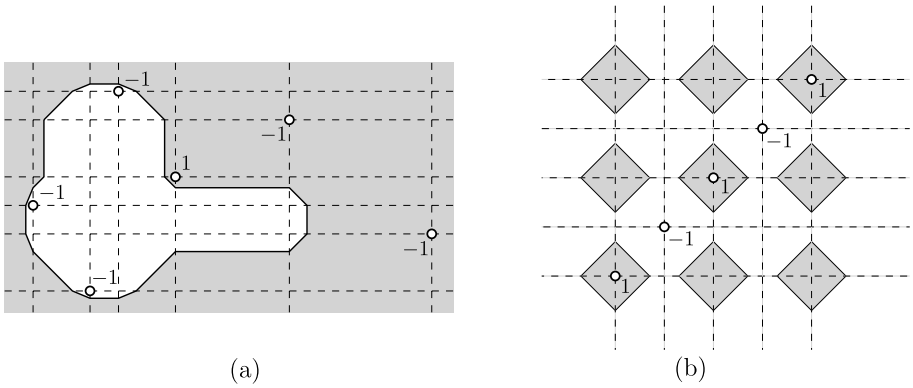

Fig. 6. $\mathcal{E}_Q^*$ (gray regions) for (a) six query points and (b) five query points on a unit grid with $c_w = 1$.

$O((c_w m)^2)$ possible slopes. Because the remaining parts of the analysis are the same as the ones in Section 3, the complexities of our algorithms are as follows.

Theorem 3. *The algorithm* RngAlgo$^+$ *reports the positive weighted group k nearest neighbors in* $O(T_{\min} \log n + (k+m^2)(\log\log n + \log m))$ *time after preprocessing P into a data structure using* $O((c_w m)^2 n \log^2 n)$ *space, where* $T_{\min} = \min\{k+m, m^2\}$.

Proof. The correctness of the algorithm follows from the discussion above. Until we report k nearest neighbors, $k + O(m)$ point events and $O(m^2)$ corner events can be inserted to $\mathcal{Q}$. The time to maintain $\mathcal{Q}$ is $O((k+m^2)\log m)$. Orthogonal range emptiness for all cells can be determined in $O(m^2 \log\log n)$ time using $O(n \log\log n)$ space. To find the point events, RngAlgo$^+$ spends $O(\min\{k+m, m^2\}\log n + k \log\log n)$ time using $O((c_w m)^2 n \log^2 n)$ space. Therefore the theorem holds. □

Theorem 4. *The algorithm* SgmtAlgo$^+$ *computes the positive weighted group k nearest neighbors in* $O((k+m)\log^2 n + m^2(\log^\epsilon n + \log m))$ *time after preprocessing P into a data structure using* $O((c_w m)^2 n)$ *space, where* $\epsilon > 0$ *is an arbitrary small constant.*

Proof. The correctness of the algorithm follows from the discussion above. Until we report k nearest neighbors, $k + O(m)$ point events and $O(m^2)$ corner events can be inserted to $\mathcal{Q}$. The time to maintain $\mathcal{Q}$ is $O((k+m^2)\log m)$. Orthogonal range emptiness for all cells can be determined in $O(m^2 \log^\epsilon n)$ total time using $O(n)$ space. To find the point events, SgmtAlgo$^+$ spends $O((k+m)\log^2 n)$ time using $O((c_w m)^2 n)$ space. Therefore the theorem holds. □

4.2. When negative integer weights are allowed

Now we consider the case that negative integer weights are allowed: for any $q \in Q$, $w(q)$ is an integer between $-c_w$ and c_w. Unlike the problem allowing positive weights only, $\mathcal{E}_Q^*$ and wsdist_Q is not necessarily convex. It may even be unbounded or consist of more than one connected component. Fig. 6(a) shows an example of $\mathcal{E}_Q^*$ that is unbounded. The example of Fig. 6 (b) shows that $\mathcal{E}_Q^*$ consists of multiple connected components. The grid points contained in $\mathcal{E}_Q^*$ have the minimum sum-of-distance 4, so $\mathcal{E}_Q^*$ intersects every cell of $G(Q)$ for any $c > 4$.

Therefore we are going to consider all cells of $G(Q)$ simultaneously. The function wsdist_Q has the following property.

Lemma 8. *The function* $\text{wsdist}_Q : \mathbb{R}^2 \to \mathbb{R}$ *with* $\text{wsdist}_Q(p) = \sum_{q\in Q} w(q) \cdot \text{dist}(p,q)$ *is linear on a cell of* $G(Q)$.

Proof. Let $r = (x, y)$ be a point in a cell g. Then we have

$$\text{wsdist}_Q(r) = \sum_{q\in Q_l(g)} w(q)(x - x(q)) + \sum_{q\in Q_r(q)} w(q)(x(q) - x) + \sum_{q\in Q_t(g)} w(q)(y(q) - y) + \sum_{q\in Q_b(g)} w(q)(y - y(q)),$$

which is a sum of linear functions. Therefore, function wsdist_Q is linear on any cell of $G(Q)$. □

Because wsdist_Q is linear in a cell and $\mathcal{E}_Q^*$ is the sublevel set of function wsdist_Q, the following corollary holds by definition [19].

Corollary 2. *For a cell* g *of* $G(Q)$, $\mathcal{E}_Q^*(c) \cap g$ *is convex for any* $c > 0$.

By Lemma 8, one corner of a cell has the minimum wsdist_Q value over all the points in the cell. Clearly, Lemma 7 still holds in this problem. By using these properties our algorithm works as follows.

We first construct $G(Q)$, and then for each grid point of $G(Q)$ compute its wsdist_Q value. To compute wsdist_Q for the grid points, we compute the wsdist_Q value of one grid point first. Then, the wsdist_Q values of adjacent grid points can easily be computed by using the linearity of the function wsdist_Q within a grid cell. For each grid cell g of $G(Q)$, we find the corner that has the minimum wsdist_Q. From the corner with the minimum wsdist_Q, we expand $\mathcal{E}_Q^*(c)$ by increasing c. This can be done by using the methods in Section 3.1. During the expansion, we encounter a point event where a data point is hit by the boundary of $\mathcal{E}_Q^*(c)$. We insert the first point event of every grid cell to the event queue $\mathcal{Q}$ that is a priority queue indexed by wsdist_Q value. We then process point events one by one in order from $\mathcal{Q}$. For a point event e, we report the data point p associated with e. If this point is the k-th reported point, we are done. Otherwise, we find the next point event in the cell containing p and insert it to $\mathcal{Q}$. We repeat the process until reporting k points. The correctness of the algorithm immediately follows from Lemma 8 and Corollary 2.

Let us analyze the complexities of the algorithm. There are $O(m^2)$ cells and $O((c_w m)^2)$ possible slopes as the analysis in Section 4.1. We use two methods in Section 3.1 to sweep each cell. Until we report k nearest neighbors, $O(k+m^2)$ events are inserted to $\mathcal{Q}$. Because $\mathcal{Q}$ maintains at most $O(m^2)$ events at any moment, the insertion and deletion operations take $O((k+m^2)\log m)$ time in total. Let RngAlgo$^-$ and SgmtAlgo$^-$ be two algorithms that are based on the orthogonal range query and the segment dragging query, respectively. As a result, we get the following results.

Theorem 5. *The algorithm* RngAlgo$^-$ *reports the weighted group k nearest neighbors in $O(m^2 \log n + k(\log\log n + \log m))$ time after preprocessing P into a data structure using $O((c_w m)^2 n \log^2 n)$ space.*

Proof. The correctness of the algorithm follows from the discussion above. Until we report k nearest neighbors, $k + O(m^2)$ point events can be inserted to $\mathcal{Q}$. The time to maintain $\mathcal{Q}$ is $O((k+m^2)\log m)$. To find the point events, RngAlgo$^-$ spends $O(m^2 \log n + k \log\log n)$ time using $O((c_w m)^2 n \log^2 n)$ space. Therefore the theorem holds. □

Theorem 6. *The algorithm* SgmtAlgo$^-$ *computes the weighted group k nearest neighbors in $O((m^2+k)\log^2 n)$ time after preprocessing P into a data structure using $O((c_w m)^2 n)$ space, where $\epsilon > 0$ is an arbitrary small constant.*

Proof. The correctness of the algorithm follows from the discussion above. Until we report k nearest neighbors, $k + O(m^2)$ point events can be inserted to $\mathcal{Q}$. The time to maintain $\mathcal{Q}$ is $O((k+m^2)\log m)$. To find the point events, SgmtAlgo$^-$ spends $O((m^2+k)\log^2 n)$ time using $O((c_w m)^2 n)$ space. Therefore the theorem holds. □

Remarks. Our algorithms in this section can be easily extended to solve the weighted version of the group k-NN query problem without any assumptions about the weights of query points. Consider a sorted list of n points with respect to an arbitrary linear function. There are infinitely many linear functions, but the number of different sorted lists is $O(n^2)$ because there are $\binom{n}{2}$ lines that pass pairs of the points; for any two lines of which slopes are in between consecutive slopes of the lines, the sorted lists of the points with respect to each of them are the same. It means that we only need to consider $O(n^2)$ slopes to preprocess points in P. This method does not change the time complexities of the algorithms, but changes the term $(c_w m)$ in the space complexities to n.

5. Group farthest neighbor queries

The group farthest-neighbor query problem is a generalization of the farthest-neighbor query problem where more than one query point are given at the same time, and the distance of a data point is measured with respect to the query points. The group k-farthest neighbor (abbreviated as k-FN) query problem is analogous to the group k-NN query problem.

There have been a few previous works on the farthest neighbor query problem. For the group k-FN query problem, Gao et al. [29] presented heuristic algorithms to solve the problem in the Euclidean space. The all-pairs farthest neighbor problem has been studied under various settings [30–32]. Agarwal et al. [30] proposed an algorithm for computing all-pairs farthest neighbors in $\mathbb{R}^3$. Cheong et al. [31] studies the all-pairs farthest neighbor problem for a set of points lying on a convex polytope in $\mathbb{R}^3$. Katoh and Iwano [32] proposed an algorithm for finding k-farthest pairs.

We consider the group k-FN query problem in the L_1 plane. The basic idea is similar to the one for the group k-NN query problem in Section 3. For the group k-NN query problem, we expand $\mathcal{E}_Q(c)$ until k data points are reported. In the group k-FN query problem, we sweep the plane by shrinking $\mathcal{E}_Q(c)$ from $c = \infty$ until it contains only $n-k$ data points.

We preprocess P in the same way as we do for the group k-NN query problem. Given a query with Q and k, we construct $G(Q)$ and then sweep all the unbounded cells of $G(Q)$ in the descending order of the sumdist_Q value. To handle the point and corner events, we construct a max-heap structure. We handle each event in the same way as we do for the group k-NN query problem in Section 3. We repeat this process until we report k farthest neighbors of Q among points in P. Since the boundary of $\mathcal{E}_Q(\infty)$ intersects the unbounded cells of $G(Q)$ only and we handle the events in order, the algorithm reports the k farthest neighbors correctly. The running time and space complexity of this algorithm are the same as those of the algorithms for the group k-NN query problem. Similarly, the weighted group k-FN query problem can also be solved with the same time and space complexities as those of the algorithms for the weighted group k-NN query problem.

6. Conclusions

In this paper, we propose two algorithms, RngAlgo and SgmtAlgo, to solve the group nearest-neighbor query problem in the L_1 plane. Our approach can be easily extended for the problem in the L_∞ metric by rotating all the data points and query points by $\pi/4$. We also show that the weighted group k-NN query problem and the group k-FN query problem in the L_1 metric can be solved similarly.

As aforementioned, we can solve this problem in $O(nm)$ time in a straightforward way. Without any preprocessing we cannot avoid $\Omega(n)$ time because of the lower bound for the selection problem [18]. For RngAlgo, if $m = o(\sqrt{\frac{n}{\log n}})$ and $k = o(\frac{n}{\log n})$, then the query time of RngAlgo is $o(n)$. For SgmtAlgo, if $m = o(\sqrt{\frac{n}{\log n}})$ and $k = o(\frac{n}{\log^2 n})$, then the query time of SgmtAlgo is $o(n)$. While SgmtAlgo uses less space than RngAlgo, RngAlgo outperforms SgmtAlgo for the query time.

References

[1] J.L. Bentley, Multidimensional binary search trees used for associative searching, Commun. ACM 18 (9) (1975) 509–517.
[2] A. Beygelzimer, S. Kakade, J. Langford, Cover trees for nearest neighbor, in: Proceedings of the 23rd International Conference on Machine Learning, ACM, 2006, pp. 97–104.
[3] R. Sproull, Refinements to nearest-neighbor searching in k-dimensional trees, Algorithmica 6 (1991) 579–589.
[4] S. Arya, D.M. Mount, Approximate nearest neighbor queries in fixed dimensions, in: Proceedings of the Fourth Annual ACM–SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 1993, pp. 271–280.
[5] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, A.Y. Wu, An optimal algorithm for approximate nearest neighbor searching fixed dimensions, J. ACM 45 (6) (1998) 891–923.
[6] P. Indyk, R. Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality, in: Proceedings of the 13th Annual ACM Symposium on Theory of Computing, ACM, 1998, pp. 604–613.
[7] H.V. Jagadish, B.C. Ooi, K.-L. Tan, C. Yu, R. Zhang, Idistance: an adaptive b+-tree based indexing method for nearest neighbor search, ACM Trans. Database Syst. 30 (2) (2005) 364–397.
[8] N. Roussopoulos, S. Kelley, F. Vincent, Nearest neighbor queries, SIGMOD Rec. 24 (2) (1995) 71–79.
[9] T. Seidl, H.-P. Kriegel, Optimal multi-step k-nearest neighbor search, SIGMOD Rec. 27 (2) (1998) 154–165.
[10] D. Papadias, Q. Shen, Y. Tao, K. Mouratidis, Group nearest neighbor queries, in: Proceedings of the 20th International Conference on Data Engineering, IEEE, 2004, pp. 301–312.
[11] D. Papadias, Y. Tao, K. Mouratidis, C.K. Hui, Aggregate nearest neighbor queries in spatial databases, ACM Trans. Database Syst. 30 (2) (2005) 529–576.
[12] P.K. Agarwal, A. Efrat, S. Sankararaman, W. Zhang, Nearest-neighbor searching under uncertainty, in: Proceedings of the 31st Symposium on Principles of Database Systems, ACM, 2012, pp. 225–236.
[13] H. Wang, W. Zhang, The L_1 top-k nearest neighbor searching with uncertain queries, arXiv:1211.5084, 2013.
[14] Y. Li, F. Li, K. Yi, B. Yao, M. Wang, Flexible aggregate similarity search, in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, ACM, 2011, pp. 1009–1020.
[15] M. Yiu, N. Mamoulis, D. Papadias, Aggregate nearest neighbor queries in road networks, IEEE Trans. Knowl. Data Eng. 17 (6) (2005) 820–833.
[16] W. Son, S. won Hwang, H.-K. Ahn, MSSQ: Manhattan spatial skyline queries, Inform. Sci. 40 (0) (2014) 67–83.
[17] R.H.J.M. Otten, R.K. Brayton, Planning for performance, in: Proceedings of the 35th Annual Design Automation Conference, ACM, 1998, pp. 122–127.
[18] S.W. Bent, J.W. John, Finding the median requires 2n comparisons, in: Proceedings of the 17th Annual ACM Symposium on Theory of Computing, ACM, 1985, pp. 213–216.
[19] R.T. Rockafellar, Convex Analysis, Princeton University Press, 1996.
[20] R. Durier, C. Michelot, Geometrical properties of the Fermat–Weber problem, European J. Oper. Res. 20 (3) (1985) 332–343.
[21] C. Witzgall, Optimal Location of a Central Facility: Mathematical Models and Concepts, National Bureau of Standards, 1964.
[22] T.M. Chan, K.G. Larsen, M. Pătraşcu, Orthogonal range searching on the RAM, revisited, in: Proceedings of the 27th Annual Symposium on Computational Geometry, ACM, 2011, pp. 1–10.
[23] S. Alstrup, G. Brodal, T. Rauhe, New data structures for orthogonal range searching, in: Proceedings of the 41st Annual Symposium on Foundations of Computer Science, IEEE, 2000, pp. 198–207.
[24] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry: Algorithms and Applications, 3rd edition, Springer, 2008.
[25] S. Rahul, P. Gupta, R. Janardan, K.S. Rajan, Efficient top-k queries for orthogonal ranges, in: Proceedings of the 5th International Conference on WALCOM: Algorithms and Computation, Springer, 2011, pp. 110–121.
[26] S.W. Bae, M. Korman, T. Tokuyama, All farthest neighbors in the presence of highways and obstacles, in: Proceedings of the 3rd International Workshop on Algorithms and Computation, Springer, 2009, pp. 71–82.
[27] B. Chazelle, An algorithm for segment-dragging and its implementation, Algorithmica 3 (1) (1988) 205–221.
[28] J. Mitchell, L_1 shortest paths among polygonal obstacles in the plane, Algorithmica 8 (1) (1992) 55–88.
[29] Y. Gao, L. Shou, K. Chen, G. Chen, Aggregate farthest-neighbor queries over spatial data, in: Proceedings of the 16th International Conference on Database Systems for Advanced Applications: Part II, Springer, 2011, pp. 149–163.
[30] P.K. Agarwal, J. Matoušek, S. Suri, Farthest neighbors, maximum spanning trees and related problems in higher dimensions, Comput. Geom. 1 (4) (1992) 189–201.
[31] O. Cheong, C.-S. Shin, A. Vigneron, Computing farthest neighbors on a convex polytope, Theoret. Comput. Sci. 296 (1) (2003) 47–58.
[32] N. Katoh, K. Iwano, Finding k farthest pairs and k closest/farthest bichromatic pairs for points in the plane, Internat. J. Comput. Geom. Appl. 5 (1–2) (1995) 37–51.