Information Processing Letters 114 (2014) 256–263

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

On reporting the L_1 metric closest pair in a query rectangle

Ananda Swarup Das [a], Prosenjit Gupta [b,*], Kishore Kothapalli [c,*], Kannan Srinathan [c,*]

[a] IBM India Research Labs, New Delhi, India
[b] Heritage Institute of Technology, Kolkata, India
[c] International Institute of Information Technology, Hyderabad, India

ARTICLE INFO

Article history:
Received 6 May 2013
Received in revised form 11 December 2013
Accepted 12 December 2013
Available online 13 December 2013
Communicated by R. Uehara

Keywords:
Computational geometry
Range queries
Closest pair
Manhattan metric

ABSTRACT

In this work, we consider the problem of finding the closest pair (in L_1 metric) of points in an orthogonal query rectangle. Given a set of n static points on a $U \times U$ grid, we preprocess these points into a data structure of size $O(mf(m)\log^2 m)$ that can be queried in time $O((g(m)+\log\log m)\log^3 m)$, for $m = O(n \log U)$ and (i) $f(m) = O(1)$ and $g(m) = O(\log^{\varepsilon} m)$; (ii) $f(m) = O(\log\log m)$ and $g(m) = O(\log\log m)$; (iii) $f(m) = O(\log^{\varepsilon} m)$ and $g(m) = O(1)$. Here ε: $0 < \varepsilon < 1$ is a small but arbitrary constant.

1. Introduction

Range searching is one of the most fundamental and well studied problems in computational geometry. The objective of any range searching query is to preprocess a set S of geometric objects (points, lines, etc.) such that given an orthogonal query rectangle, we can report the subset S' of objects intersecting the query rectangle. However, more often than not, an *informative* summary of the output is of more interest to a user. Such informative summary can be obtained by applying appropriate aggregate functions like *counting, maximal points, closest pair* (see [1,3,6]). These aggregate functions find applications in tools like databases, tools for design rule checking in electronic design automation, decision making, etc. In this work, we consider the closest pair in L_1 metric to be our aggregate function and design data structures to support range-aggregate queries. The problem finds its application in tools like EDA-design rule checking where the use of L_1 metric is quite common. The problem may also be important for air-traffic control where a controller may wish to determine the closest pair of aircrafts in any query window to avoid possible collisions. For such a tool, the positions of the aircrafts can be considered as points (see [6]).

As stated in [6], the problem of finding the closest pair in a query rectangle is not decomposable. This means that if S_1 and S_2 are two subsets of S' such that $S' = S_1 \cup S_2$, then the closest pair information for S_1, S_2 respectively do not help us to find the closest pair in S'. Of course, one can first find all the points in the query rectangle and then try to find the distance between each pair of points to return the closest pair. But the solution is prohibitive if the number of points is too large. In this work, we therefore design a data structure of size $O(mf(m)\log^2 m)$ that can be queried in time $O((g(m)+\log\log m)\log^3 m)$, for $m = O(n \log U)$ and (i) $f(m) = O(1)$ and $g(m) = O(\log^{\varepsilon} m)$; (ii) $f(m) = O(\log\log m)$ and $g(m) = O(\log\log m)$; (iii) $f(m) = O(\log^{\varepsilon} m)$ and $g(m) = O(1)$. Here ε is a small but arbitrary constant. We assume our model of computation to be word RAM with size of each word being $\theta(\log U)$. All

* Corresponding authors.
E-mail addresses: anandaswarup@gmail.com (A.S. Das), prosenjit_gupta@acm.org (P. Gupta), kkishore@mail.iiit.ac.in (K. Kothapalli), srinathan@mail.iiit.ac.in (K. Srinathan).

0020-0190/$ – see front matter
http://dx.doi.org/10.1016/j.ipl.2013.12.006

the storage spaces mentioned in this work are in terms of words unless specified otherwise.

1.1. Our contribution

The formal definition for the problem that we study in this work is as follows:

Problem 1. Given a set S of n points on a $U \times U$ grid, preprocess S into a data structure such that given an axes-parallel query rectangle q, the closest pair (in the L_1 metric) in $S \cap q$ can be reported efficiently.

We assume that the coordinates of the points are integers and no two points are on the same horizontal or vertical line. It should be noted that the distance between two points p, r in the L_1 metric is equal to the half of the perimeter of the axes-parallel rectangle formed by taking p, r as diagonally opposite corners of the rectangle.

Let $Query(q, S)$ denote the operation of reporting the closest pair in L_1 metric for a set of points from S in the query rectangle q. Also, let ε be an arbitrary but small constant. For Problem 1, we have the following result

Theorem 1. *A set S of n points on a $U \times U$ grid can be preprocessed into a data structure of size $O(mf(m)\log^2 m)$ that supports Query(q, S) in time $O((g(m) + \log\log m)\log^3 m)$, for $m = O(n \log U)$ and* (i) *$f(m) = O(1)$ and $g(m) = O(\log^{\varepsilon} m)$;* (ii) *$f(m) = O(\log\log m)$ and $g(m) = O(\log\log m)$ and* (iii) *$f(m) = O(\log^{\varepsilon} m)$ and $g(m) = O(1)$.*

2. Preliminaries

Rectangle visibility We define two points (p, r) of S to be rectangle visible to each other, if the axes parallel rectangle defined by p, r as diagonally opposite corners does not contain any other point from S.

Range tree Suppose, we are given a set S of n points in $\mathbb{R}^2$ and we need to preprocess these points into a data structure such that given an axes parallel query rectangle, we can efficiently report the points of the set S that are in the query rectangle. Range tree is known to be a very efficient data structure for this problem [4]. Broadly speaking, the range tree needed to solve the problem is a two-layer balanced binary search tree which is constructed as follows: build a primary tree T_x, the leaf nodes of which store the x-coordinates of the points of S in non-decreasing order. Each internal node of the tree T_x is assigned to an interval $int(\mu)$ which is equal to the union of the discrete intervals associated with the leaf nodes of the subtree rooted at μ. Each internal node μ maintains an auxiliary array A_μ which stores the x and the y coordinates of the points present in the subtree rooted at μ. The array A_μ is sorted in nondecreasing order of the y-coordinates of the points stored. Given a query rectangle, $[a, b] \times [c, d]$, the segment $[a, b]$ is allocated to the node ϕ if the interval $int(\phi) \subset [a, b]$ but $int(parent(\phi)) \not\subseteq [a, b]$. As is known from [4], the segment $[a, b]$ is thus allocated to $O(\log n)$ canonical nodes. Next, at each such canonical node ϕ, the corresponding auxiliary array A_ϕ is searched to find the smallest index i such that $c \leqslant A_\phi[i]$ and the largest index j such that $A_\phi[j] \leqslant d$. All the points with y-coordinates in $A_\phi[i, \ldots, j]$ are then reported. The query time needed by this search technique is $O(\log^2 n + k)$ where k is the size of the output. The storage space needed by the range tree is $O(n \log n)$. The query time can be improved to $O(\log n + k)$ by using a *pointer jumping* technique known as *fractional cascading*. For more details on the query search as well as fractional cascading, one can refer to [4]. The range tree data structure can be extended for higher dimensions with a penalty of $O(\log n)$ on the storage space as well as on the query time.

Compact range tree As stated above, a standard range tree for a set of n static points in $\mathbb{R}^2$ uses $O(n \log n)$ space. However, in [2] the authors have shown that the space requirement for the standard range tree can be reduced by storing compact representation of the auxiliary arrays A_μ. More precisely, the array A_μ is stored as a bit vector where the ith index of bit vector stores a *zero* if the point whose y-coordinate is at position i in $A_\mu[1, n]$ is present in the left subtree rooted at μ. Else $A_\mu[i]$ stores a one. As stated in [8], we need to support two operations on the compact range tree, namely: (a) *noderange*(c, d, v): given a range $[c, d]$ and a node v, *noderange*(c, d, v) returns two indices i, j such that for any point $p = (p(x), p(y))$, if $p(x) \in int(v)$ and $p(y) \in [c, d]$, then $p(y) \in A_v[i, \ldots, j]$; (b) *point*$(v, i)$: given an index i and a node v, *point*(v, i) returns the coordinates of the point in $A_v[i]$. For these two operations, the following result is known from [2,8].

Lemma 1. *There exists a compact range tree that uses $O(nf(n))$ space and supports operations point(v, i) and noderange(c, d, v) in $O(g(n))$ and $O(g(n) + \log\log n)$ time, respectively, for* (i) *$f(n) = O(1)$ and $g(n) = O(\log^{\varepsilon} n)$,* (ii) *$f(n) = O(\log\log n)$ and $g(n) = O(\log\log n)$ and* (iii) *$f(n) = O(\log^{\varepsilon} n)$ and $g(n) = O(1)$.*

Reporting points in a query rectangle using compact range tree Given a query rectangle $q = [a, b] \times [c, d]$, the segment $[a, b]$ is allocated to a node μ in the compact tree if $int(\mu) \subset [a, b]$ but $int(parent(\mu)) \not\subseteq [a, b]$. For each of the $O(\log n)$ nodes μ to which the segment $[a, b]$ is allocated, we run the operation *noderange*(c, d, μ) and find the two indices i, j for the auxiliary array A_μ. Next, for each l: $i \leqslant l \leqslant j$, we run the operation *point*(μ, l) to find the coordinates of the point whose y-coordinate is a rank l in A_μ. Thus, the operation *point*(μ, l) for each l: $i \leqslant l \leqslant j$ has to be executed for $|j - i|$ times as each of these points are in q. The exact query time of the algorithm presented in this section is $O(\log n \log^{\varepsilon} n + k \log^{\varepsilon} n)$. However, this can be improved to $O((1 + k)\log^{\varepsilon} n)$ using the techniques of [2] without increasing the storage space. Recently, the authors of [7] have shown a technique to report the points in a query rectangle in increasing order of their x-coordinates in time $O(\log^{\varepsilon} n + k \log^{\varepsilon} n)$ using a compact range tree. Here k is the output size.

Range minima/maxima query (RMQ) As defined in [5], Range Minima (respectively Maxima) query is defined as follows: given an array $A[1,n]$ of elements from a totally ordered set, a range minima (respectively maxima) query $RMQ_A[l,r]$ returns the index of the smallest (respectively largest) element in $A[l,r]$. It is known from [5] that for the RMQ queries, there exists a data structure of size $2n+o(n)$ bits and the data structure answers the queries in constant time without accessing the values of the array A.

Reporting the most weighted point Suppose, we are given a set S of n weighted static points in $\mathbb{R}^2$ and we need to preprocess these points into a data structure such that given an axes parallel query rectangle q, we can efficiently report the maximum weighted point of the set S that is in q. For this problem, we can have the following result.

Lemma 2. *For a set S of n weighted static points in $\mathbb{R}^2$, there exists a data structure of size $O(nf(n))$ that can be queried with an axes parallel rectangle q to find the most weighted point in the query rectangle in time $O((g(n)+\log\log n)\log n)$, for* (i) $f(n)=O(1)$ *and* $g(n)=O(\log^{\varepsilon} n)$, (ii) $f(n)=O(\log\log n)$ *and* $g(n)=O(\log\log n)$ *and* (iii) $f(n)=O(\log^{\varepsilon} n)$ *and* $g(n)=O(1)$.

Proof. As the points of the set S are static, we map the x and the y coordinates of the points in the set S to a rank space which is a grid of size $n\times n$. We then build a compact range tree T_x, the leaves of which store the x-coordinates of the points in the set S. At each internal node μ, we associate a compact representation of the auxiliary array A_μ. Next, for the point p whose y-coordinate is at position i in A_μ, we store its corresponding weight $w(p)$ in the ith index of a separate array B_μ. We then construct a range maxima data structure of [5] for the array B_μ. The range maxima data structure is denoted by RMQ_μ. Once RMQ_μ is constructed, the array B_μ is deleted. Given a query rectangle $[a,b]\times[c,d]$, we allocate the segment $[a,b]$ to a canonical node ϕ, if and only if $int(\phi)\subset[a,b]$ but $int(parent(\phi))\not\subseteq[a,b]$. The segment $[a,b]$ is allocated to $O(\log n)$ canonical nodes of T_x. For each such canonical node ϕ, we find the indices (i,j) by running the query $noderange(c,d,\phi)$. Next, we identify the index m with maximum weight by running $RMQ_\phi[i,j]$ and find the coordinates and the weight of the corresponding point by running $point(\phi,i)$. We need to collect $O(\log n)$ such values, one from each canonical node. We then return the point with the maximum weight among these $O(\log n)$ points. RMQ at any node takes $O(1)$ time. Thus, the running time of this query algorithm is dependent on the performance of $noderange(c,d,\phi)$ and $point(\phi,i)$. From Lemma 1, we know that the operations $noderange(c,d,\phi)$ and $point(\phi,i)$ can be supported in $O(g(n))$ and $O(g(n)+\log\log n)$ time using a storage space of $O(nf(n))$, for (i) $f(n)=O(1)$ and $g(n)=O(\log^{\varepsilon} n)$, (ii) $f(n)=O(\log\log n)$ and $g(n)=O(\log\log n)$, and (iii) $f(n)=O(\log^{\varepsilon} n)$ and $g(n)=O(1)$. Since, we have to repeat $noderange(c,d,\phi)$ and $point(\phi,i)$ for $O(\log n)$ nodes, the total query time we need is $O((g(n)+\log\log n)\log n)$. It should be noted that the correctness of the technique discussed here follows from the correctness of the operations $noderange(c,d,\phi)$, $point(i,\phi)$ and the $RMQ[i,j]$. □

Weighted rectangle visible points Let $(r_1,r_2)\in S$ be two rectangle visible points. We form a 4-d point $p=(r_1(x), r_2(x), r_1(y), r_2(y))$ to which we assign a weight $w(p)$. The weight $w(p)$ is equal to the half of the perimeter of the axes parallel rectangle realized by (r_1,r_2) with (r_1,r_2) being two diagonally opposite corners of the rectangle. In other words, $w(p)=|r_1(x)-r_2(x)|+|r_1(y)-r_2(y)|$. In the rest of this work, we will call such a point as weighted 4-d rectangle visible point.

3. Proposed solution for range aggregate closest pair in L_1 metric

3.1. An overview of the algorithm

In this section, we first provide an overview of our algorithm without providing the details of the data structures and preprocessing of data. For this section, the readers may consider the data structures to be black boxes and assume that the preprocessing is taken care of by some oracle.

For the n 2-d points in the set S, we first create $O(n\log U)$ pairs of weighted 4-d rectangle visible points and then preprocess these points into a data structure denoted by D. We also preprocess the coordinates of the n 2-d points in the set S into a compact range tree denoted by RA. Given a query rectangle $q=[a,b]\times[c,d]$, we first search the compact range tree RA and start reporting the points in the query rectangle. There are two possible cases which we enumerate next.

1. If the number of reported points is less than or equal to four, then
   - we find all pairs of rectangle visible points realizable with the reported points and return the pair with minimum Manhattan distance in between. It must be noticed that the axes parallel rectangle realized by two reported points with the two points being diagonally opposite corners must be completely contained in the query rectangle. Thus, to find if the rectangle thus realized is empty, all we have to do is to check for its emptiness with the other reported points.
2. If the number of points reported exceeds four, then
   - we stop reporting the points in the query rectangle. Instead, we search D with $q'=[a,b]\times[a,b]\times[c,d]\times[c,d]$ and find the 4-d point with smallest weight in q'.

Road map In Section 3.2, we provide the details of how to create $O(n\log U)$ 4-d weighted points. In Section 3.3, we discuss the data structure D. The query algorithm is provided in details in Section 3.5. Finally, the reason for reporting only four points in the query rectangle is provided in the Proof of Lemma 5.

3.2. Selecting the 4-d weighted points

For each point $p\in S$, find the rectangle visible points for p in its northeast quadrant that is $NE(p)$. The rectangle

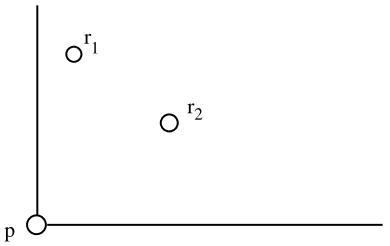

Fig. 1. The point p and two rectangle visible points for p in its northeast quadrant.

visible points for $p = (p(x), p(y))$ are actually the set or chain of points in $NE(p)$ that do not dominate any other point in $NE(p)$. This set of points are monotonically increasing in their x-coordinates while monotonically decreasing in their y-coordinates. Denote such a chain as C. Next, consider the topmost and the second topmost points $r_1 = (r_1(x), r_1(y))$ and $r_2 = (r_2(x), r_2(y))$ of the chain C. In the rest of the section, we denote by $|r(y) - p(y)|$ (respectively $|r(x) - p(x)|$) the distance of the projections of the points r and p on the y-axes (respectively on the x-axes).

Consider Fig. 1. Let us first consider the following claim.

Lemma 3. *Let r_1, r_2 be two points in S that are rectangle visible from p in $NE(p)$, $r_1(y) > r_2(y)$ and $h = |r_1(y) - p(y)|$. Now, if r_2 is closer to p than r_1, then $|r_2(y) - p(y)| < \frac{h}{2}$.*

Proof. For contradiction, let us assume that $|r_2(y) - p(y)| \geqslant \frac{h}{2}$.

Then, we have

$$\big|r_1(y) - r_2(y)\big| \leqslant \frac{h}{2}. \tag{1}$$

It is evident from Fig. 1 that

$$\big|r_2(x) - r_1(x)\big| < \big|r_2(x) - p(x)\big|. \tag{2}$$

Let $d_{r_2,p}$ and d_{r_1,r_2} be the respective distances between the points (r_2, p) and (r_2, r_1). Clearly, $d_{r_2,p} = |r_2(x) - p(x)| + |r_2(y) - p(y)|$ and $d_{r_2,r_1} = |r_2(x) - r_1(x)| + |r_1(y) - r_2(y)|$.

By using Eqs. (1), (2) and our assumption that $|r_2(y) - p(y)| \geqslant \frac{h}{2}$, we get

$$d_{r_2,p} > d_{r_2,r_1}. \tag{3}$$

But, this is a contradiction to the assumption that r_2, p are the closest. □

Thus, for each point $p \in S$, we do the following:

1. We find the points which are rectangle visible from p in its northeast quadrant. Let C be the chain of such points. Consider these points in order of their decreasing y-coordinates and denote these points as $r_1 \ldots r_{|C|}$.
2. *YSWEEP*: Consider the topmost point r_1 in the chain C. Form a 4-d point $(p(x), r_1(x), p(y), r_1(y))$. Also, set $crux = r_1$. In the subsequent section, we will denote the coordinates of the point stored in $crux$ as $(crux(x), crux(y))$. Let $h_{threshold} = |p(y) - crux(y)|$.
3. For any point r_i for $i = 2, \ldots, |C|$, let $h = |p(y) - r_i(y)|$. If $h < \frac{h_{threshold}}{2}$, we
   (i) form a 4-d point $(p(x), r_i(x), p(y), r_i(y))$
   (ii) set $crux = r_i$ and
   (iii) set $h_{threshold} = |p(y) - r_i(y)|$.
4. Repeat the above steps until all the points of the chain C are visited.
5. *XSWEEP*: Next, we form a 4-d point with the last point of the chain that is $r_{|C|}$. Notice that this is the point with maximum x-coordinate in the chain C. We set $crux = r_{|C|}$. Let $h'_{x,threshold} = |p(x) - crux(x)|$.
6. Now, consider the points in chain C in decreasing order of their x coordinates. For any point r_i starting from the second rightmost point to the leftmost point, let $h'_x = |p(x) - r_i(x)|$. If $h'_x < \frac{h'_{x,threshold}}{2}$, we
   (i) form a 4-d point $(p(x), r_i(x), p(y), r_i(y))$
   (ii) set $crux = r_i$ and
   (iii) set $h'_{x,threshold} = |p(x) - r_i(x)|$, respectively.
7. Continue till all the points starting from the second rightmost point to the leftmost point of the chain C are visited.
8. Repeat similar steps in $NW(p), SE(p), SW(p)$. The step 3 (respectively 6) for the point p will start from the point $p' = (p'(x), p'(y))$ for which the horizontal distance $|p'(x) - p(x)|$ (respectively the vertical distance $|p'(y) - p(y)|$) is minimum.

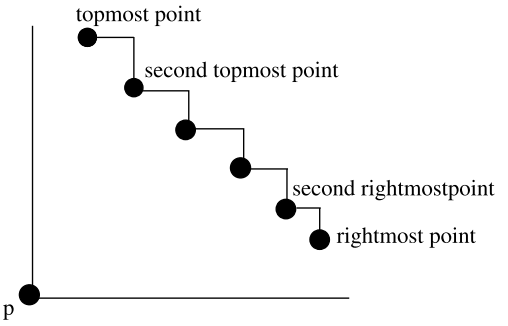

Fig. 2. The point p and the visible points for p in its northeast quadrant.

Lemma 4. *With a set of n points on a $U \times U$ grid, the number of 4-d points thus formed is $O(n \log U)$.*

Proof. Consider the chain C of the rectangle visible points for the point p in its northeast quadrant. See Fig. 2. The topmost point (respectively the rightmost point) of the chain will surely form a 4-d point with p. The maximum distance of the projection of the topmost point and the point p on y-axes (respectively the rightmost point and the point p on x-axes) is U. Let this distance be denoted as h. The distance is set as a threshold. By the step 3 *if condition* (respectively step 6 *if condition*), the second point from the top (respectively from the right) in the chain which forms a 4-d point with p has its projection on the y-axes (respectively on the x-axes) at a distance $< \frac{h}{2}$ from the projection of p on the y-axes (respectively on the x-axes) and if the point forms a 4-d point with p, then the distance of its projection on the y-axes (or x-axes) is set as the new threshold. Similarly, for any subsequent point z to form a 4-d point with p, the distance of its projection from the projection of p on the y-axes (respectively on the

x-axes) has to be less than half of the current threshold. Hence the claim. □

3.3. Construction of data structure D

1. Let S' be the set of $m = O(n \log U)$ 4-d weighted points, each representing a pair of rectangle visible points in S. For any point $p \in S'$: $p = (p_1(x), p_2(x), p_1(y), p_2(y), w(p))$, $(p_1(x), p_2(x), p_1(y), p_2(y))$ are the coordinates of the point p and $w(p)$ is the weight assigned to the point p. As stated earlier, $w(p) = |p_1(x) - p_2(x)| + |p_1(y) - p_2(y)|$.
2. Construct a height balanced binary search tree T_x, the leaves of which store in increasing order, the values $p_1(x)$ for the points in S'. Each internal node $\mu \in T_x$ is assigned an interval $int(\mu)$ which is equal to the union of the discrete intervals associated with the leaf nodes of the subtree rooted at μ.
3. For each internal node $\mu \in T_x$ construct a set S'' such that $p' = (p_2(x), p_1(y), p_2(y), w(p)) \in S''$ if $p_1(x) \in int(\mu)$ for $p = (p_1(x), p_2(x), p_1(y), p_2(y), w(p))$ and $p \in S'$.
4. At each internal node $\mu \in T_x$, associate an auxiliary height balanced binary search tree $T_{\mu,x}$, the leaves of which store in increasing order, the values of $p_2(x)$. Each internal node $\phi \in T_{\mu,x}$ is assigned an interval $int(\phi)$ which is equal to the union of the discrete intervals associated with the leaf nodes of the subtree rooted at ϕ.
5. At each internal node $\phi \in T_{\mu,x}$, construct a set S_ϕ such that $p'' = (p_1(y), p_2(y), w(p)) \in S_\phi$ if $p_2(x) \in int(\phi)$ for $p' = (p_2(x), p_1(y), p_2(y), w(p))$ and $p' \in S''$.
6. Thus, at each node $\phi \in T_{\mu,x}$ we have a set of 2-d weighted points. These points are stored in the set S_ϕ. Hence, we construct an instance of the data structure of Lemma 2 at the node ϕ. We denote this data structure as W_ϕ. While storing the point $p'' = (p_1(y), p_2(y), w(p))$ in W_ϕ, also store the point $p = (p_1(x), p_2(x), p_1(y), p_2(y), w(p))$ along with it.

3.4. Construction of RA

As stated earlier, the data structure RA is a compact range tree built on the 2-d static points of the set S such that RA supports reporting points in a query rectangle in $O(\log^\epsilon n + k \log^\epsilon n)$ time. Here k is the output size.

3.5. Query algorithm

1. Given an axes-parallel query rectangle $q = [a, b] \times [c, d]$, we first query RA with q and start reporting the points in $S \cap q$. While reporting the points, we maintain a *counter* to count the number of points thus reported.
2. While $counter \leqslant 4$, continue reporting points.
   (a) If $|S \cap q| \leqslant 4$, find all pairs of empty axes parallel rectangles realizable by the reported points with two points being diagonally opposite and return the pair which has the smallest Manhattan distance in between.

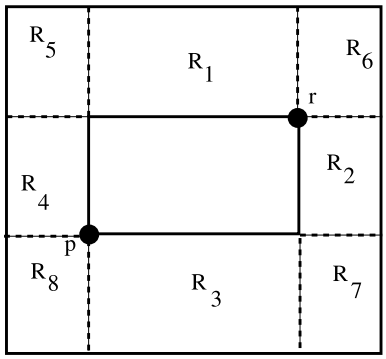

Fig. 3. The point p and r in q.

3. If $counter == 5$, we stop reporting. Instead, we search the data structure D in the following way:
   (a) given the axes-parallel query rectangle $[a, b] \times [c, d]$, we convert it into a 4-d orthogonal rectangular box $[a, b] \times [a, b] \times [c, d] \times [c, d]$. Next, we find the canonical nodes $\mu \in T_x$ such that $int(\mu) \subset [a, b]$ but $int(parent(\mu)) \not\subseteq [a, b]$. There will be $O(\log m)$ such canonical nodes.
   (b) For each such canonical node $\mu \in T_x$, we search the associated secondary tree $T_{\mu,x}$ and find the canonical nodes $\phi \in T_{\mu,x}$ such that $int(\phi) \subset [a, b]$ but $int(parent(\phi)) \not\subseteq [a, b]$.
   (c) Thus, there are $O(\log^2 m)$ canonical nodes. At each such node ϕ, we search the data structure W_ϕ with the query rectangle $[c, d] \times [c, d]$ and find the smallest weighted point in the rectangle $[c, d] \times [c, d]$. Let $|W_\phi|$ be the number of points stored in the data structure W_ϕ. From the proof for Lemma 2, we know that after searching W_ϕ at the node ϕ, there will be $\log(|W_\phi|)$ candidate points. Select the one with the smallest weight.
   (d) Thus, we get $O(\log^2 m)$ 4-d weighted rectangle visible points. We then return the point with smallest weight among these $O(\log^2 m)$ points.

Lemma 5. *Given an axes parallel query rectangle $q = [a, b] \times [c, d]$, let $p = (p(x), p(y))$ and $r = (r(x), r(y))$ be the two points that are closest in the query rectangle. However the rectangle visible point $(p(x), r(x), p(y), r(y))$ is not present in D. Then, the query rectangle cannot have more than four points of S.*

Proof. Let $Rect(p, r)$ be the axes parallel rectangle realized by the points (p, r). See Fig. 3. With reference to $Rect(p, r)$, the query rectangle q is divided into eight regions denoted by $R_1, \ldots, R_8$. Our proof is divided into three parts. Precisely, we show that (i) there cannot be any point in the regions R_1, R_2, R_3, R_4; (ii) there cannot be any point in the regions R_6 and R_8; (iii) no more than one point can be present in the region R_7 (respectively in R_5).

We assume all the coordinates to be positive.

Part (i): Let $z = (z(x), z(y))$ be a point in the region R_2. See Fig. 4. If there are more than one point in the region R_2, then consider z to be the one with the smallest

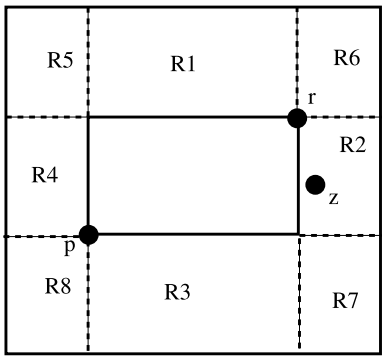

Fig. 4. The point z is in region R_2.

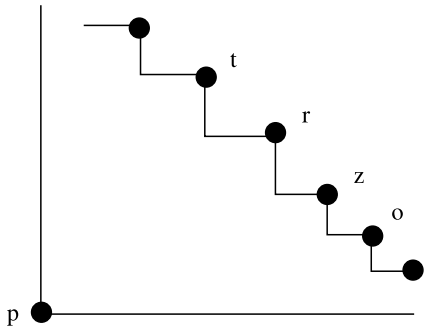

Fig. 5. The chain of rectangle visible points for p in its northeast quadrant.

x-coordinate. It should be noted that z will also be a rectangle visible point for p as z has the smallest x coordinate in the region R_2 which means that any point dominated by z will also be dominated by r. Since r is a rectangle visible point, it does not dominate any point in $NE(p)$. Hence, z will be a rectangle visible point for p.

It must be evident from Fig. 3 that

$$\big|p(y) - r(y)\big| > \big|z(y) - r(y)\big|. \tag{4}$$

Next, see Fig. 5. Since (p, r) do not form a 4-d point, by step 8 of Section 3.2 (selecting the 4-d points), YSWEEP has to start from a point above r in the chain C of rectangle visible points for p in $NE(p)$. This is because YSWEEP starts from a point $p' \in C$: $p' = (p'(x), p'(y))$ such that $|p'(x) - p(x)|$ is minimum.

Let t be the last point above r that forms a 4-d point with p during YSWEEP. Also, let o be the last point to the right of r that forms a 4-d point with p during XSWEEP. By step 6(iii) of Section 3.2, $h'_{x,threshold}$ is set to $|p(x) - o(x)|$ once o forms a 4-d point with p.

Since z and r do not form 4-d points with p during XSWEEP, by step 6 of Section 3.2, we have

$$\big|p(x) - z(x)\big| > \frac{h'_{x,threshold}}{2}. \tag{5}$$

Similarly,

$$\big|p(x) - r(x)\big| > \frac{h'_{x,threshold}}{2}. \tag{6}$$

Thus, $|o(x) - z(x)| < \frac{h'_{x,threshold}}{2}$ and $|o(x) - r(x)| < \frac{h'_{x,threshold}}{2}$.

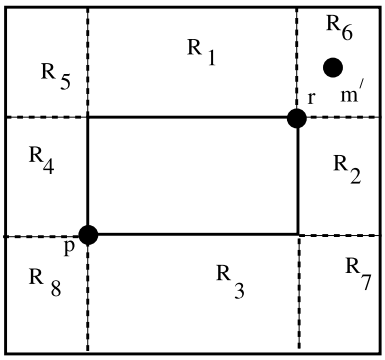

Fig. 6. The point m' is in region R_6 in q.

From Fig. 5, we can see that $r(x) < z(x) < o(x)$. Thus,

$$\big|r(x) - z(x)\big| < \frac{h'_{x,threshold}}{2}. \tag{7}$$

Therefore, $|r(x) - z(x)| + |r(y) - z(y)| < |r(x) - p(x)| + |r(y) - p(y)|$. Clearly this is a contradiction to the assumption that (p, r) are the closest in the query rectangle.

Part (ii): See Fig. 6. Using arguments similar to the ones used for the proof of Part (i), it can be shown that for any point m' in the region R_6,

$$\big|r(x) - m'(x)\big| < \big|r(x) - p(x)\big|. \tag{8}$$

See Fig. 5. Let t be the last point above r that forms a 4-d point with p during YSWEEP. By step 3(iii) of Section 3.2, $h_{threshold}$ is set to $|t(y) - p(y)|$ once t forms a 4-d point with p. As the point r does not form a 4-d point with p, from step 3 of Section 3.2, we have

$$\big|r(y) - p(y)\big| > \frac{h_{threshold}}{2}. \tag{9}$$

Hence, $|t(y) - r(y)| < |r(y) - p(y)|$.

Also, from Fig. 5, we have $p(x) < t(x) < r(x)$. Therefore, $|t(x) - r(x)| < |p(x) - r(x)|$. Notice that the point t cannot be in q. This is because, if t is in q, then $|t(x) - r(x)| + |t(y) - r(y)| < |p(x) - r(x)| + |p(y) - r(y)|$. In other words, (t, r) are closer than (p, r) which is a contradiction to our assumption.

Next, as the point m' is in R_6, $t(y) > m'(y) > r(y)$. Therefore, $|m'(y) - r(y)| < |t(y) - r(y)| < |r(y) - p(y)|$. Hence, $|m'(y) - r(y)| + |m'(x) - r(x)| < |r(y) - p(y)| + |r(x) - p(x)|$. This is clearly a contradiction to the assumption that (p, r) are closest in the query rectangle.

Part (iii): See Fig. 7. Let z, z' be two points in the region R_7. Using arguments similar to the proof of Part (i), it can be shown that

$$\big|z(x) - z'(x)\big| < \big|p(x) - r(x)\big|. \tag{10}$$

See Fig. 8. Consider the chain of rectangle visible points for r in its southwest quadrant. As p does not form a 4-d point with r, by step 8 of Section 3.2, there has to be a point below p which forms a 4-d point with r during YSWEEP. This is because, in any quadrant for the point r, YSWEEP starts from a point p' in the chain of rectangle visible points such that $|p'(x) - r(x)|$ is minimum. Let t be the last point below p which forms a 4-d point with r.

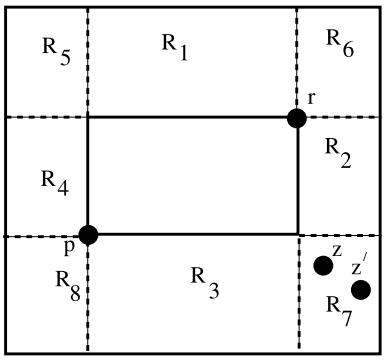

Fig. 7. The points z, z' are in region R_7.

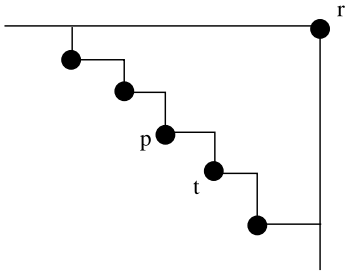

Fig. 8. The chain of rectangle visible points for r in its southwest quadrant.

Once t forms a 4-d point with r, $h_{threshold}$ is set to $|t(y) - r(y)|$. As (p, r) do not form a 4-d point,

$$\big|p(y) - r(y)\big| > \frac{h_{threshold}}{2}. \tag{11}$$

Thus, $|p(y) - t(y)| < |p(y) - r(y)|$. Notice that the point t cannot be in q. This is because, if t is in q, then $|t(x) - p(x)| + |t(y) - p(y)| < |p(x) - r(x)| + |p(y) - r(y)|$. In other words, (t, p) are closer than (p, r), a contradiction to our assumption.

Let $\max(a, b)$ (respectively $\min(a, b)$) denote the maximum (respectively the minimum) of the two values. Since z, z' are in region R_7, $p(y) > \max(z(y), z'(y))$. Also $\min(z(y), z'(y)) > t(y)$ as the point t is not in q. Thus,

$$\big|z(y) - z'(y)\big| < \big|t(y) - p(y)\big| < \big|r(y) - p(y)\big|. \tag{12}$$

Hence, z, z' are closer than p, r which is a contradiction. Thus, if p, r are the closest points in the query rectangle, then there cannot be any point in the regions R_1, R_2, R_3, R_4, R_6 and R_8 and there can be at most one point in the region R_7 (respectively R_5). □

Lemma 6. *Given an axes parallel query rectangle $q = [a, b] \times [c, d]$, let $p = (p(x), p(y))$ and $r = (r(x), r(y))$ be the two points that are closest in the query rectangle. Then, the query algorithm returns the pair (p, r).*

Proof. We divide the proof into two parts.
Part (i): Let the point $p' = (p(x), r(x), p(y), r(y), w(p'))$ be not present in D. Then, by Lemma 5, there can be at most four points in the query rectangle and by step 2 of Section 3.5 (Query Algorithm), our query algorithm finds all the points and return the pair (p, r) that realizes the empty axes parallel rectangle with smallest Manhattan distance in between.
Part (ii): Let the point $p' = (p(x), r(x), p(y), r(y), w(p'))$ be present in D. By construction, the data structure D comprises of a primary tree T_x (step 2 of Section 3.3). Each internal node $\mu \in T_x$ is associated with an auxiliary secondary tree $T_{\mu,x}$ (step 4 of Section 3.3). Each internal node $\phi \in T_{\mu,x}$ has an instance of the data structure of Lemma 2 and is denoted by W_ϕ. Since the point p is present in the query rectangle and the point $(p(x), r(x), p(y), r(y), w(p'))$ is present in D, by step 2 of Section 3.3 and step 3a of Section 3.5, it is ensured that there exists a node $\mu \in T_x$ such that $p(x) \in int(\mu)$ and $int(\mu) \subset [a, b]$ but $int(parent(\mu)) \nsubseteq [a, b]$. By step 3 of Section 3.3, the node μ has a set S'' where the point $(r(x), p(y), r(y), w(p'))$ is stored. Similarly, by step 4 of Section 3.3 and step 3b of Section 3.5, it is ensured that there exists a node $\phi \in T_{\mu,x}$ such that $r(x) \in int(\phi)$ and $int(\phi) \subset [a, b]$ but $int(parent(\phi)) \nsubseteq [a, b]$. By step 5 of Section 3.3, the node ϕ has a set S_ϕ where the point $(p(y), r(y), w(p'))$ is stored. At the node ϕ, we search the data structure W_ϕ with the query $[c, d] \times [c, d]$ to find the smallest weighted 2-d point. By step 6 of Section 3.3, the data structure W_ϕ is built on the weighted points of the set S_ϕ. The correctness of the data structure W_ϕ which follows from Lemma 2 ensures that the point $(p(y), r(y), w(p'))$ is reported. Along with the point $(p(y), r(y), w(p'))$, the point $(p(x), r(x), p(y), r(y), w(p'))$ is stored. Thus, the query algorithm returns the pair (p, r). □

Lemma 7. *Finding the canonical nodes in the primary tree T_x needs $O(\log m)$ time for $m = O(n \log U)$ time.*

Proof. Given a query rectangle $[a, b] \times [c, d]$, finding the two leaves $leaf_1$ and $leaf_2$ respectively storing the smallest value $\geqslant a$ and the largest value $\leqslant b$ can be found in $O(\log m)$ time. Finding the least common ancestor (lca) for $leaf_1$ and $leaf_2$ takes $O(\log m)$ time. Let π_1 (respectively π_2) be the path from lca to $leaf_1$ (respectively to $leaf_2$). For any node $\mu_1 \in \pi_1$ (respectively in π_2), if it is a left child (respectively right child) of its parent, allocate the segment $[a, b]$ to its sibling. Since the length of the paths π_1 and π_2 can be at most $O(\log m)$, finding the canonical nodes in the primary tree T_x needs $O(\log m)$ time. □

Lemma 8. *The above data structure needs $O(mf(m) \log^2 m)$ storage space and can be queried in time $O((g(m) + \log \log m) \times \log^3 m)$ for $m = O(n \log U)$ and* (i) *$f(m) = O(1)$ and $g(m) = O(\log^\varepsilon m)$;* (ii) *$f(m) = O(\log \log m)$ and $g(m) = O(\log \log m)$ and* (iii) *$f(m) = O(\log^\varepsilon m)$ and $g(m) = O(1)$.*

Proof. The height of the primary tree T_x is $O(\log m)$. Let $le(T_x^\mu)$ be the count of leaf nodes present in the subtree of T_x rooted at the internal node $\mu \in T_x$. At the node μ, there is a secondary tree $T_{\mu,x}$ associated with the node. The height of the tree $T_{\mu,x}$ is $O(\log(le(T_x^\mu)))$. At any internal node $\phi \in T_{\mu,x}$, we have associated an instance of the data structure of Lemma 2 which is denoted by W_ϕ. W_ϕ needs a storage space of $O(le(T_{\mu,x}^\phi) f(le(T_{\mu,x}^\phi)))$. Thus,

the total storage space needed by all the W_ϕs across all the internal nodes ϕs at a particular level h in $T_{\mu,x}$ is $\sum O(le(T^\phi_{\mu,x}) f(le(T^\phi_{\mu,x}))) = O(le(T^\mu_x) f(le(T^\mu_x)))$. This is because $T_{\mu,x}$ is the secondary tree associated with the node μ in the primary tree T_x. Therefore, the total storage space needed by all the W_ϕs across all the levels in $T_{\mu,x}$ is $O(le(T^\mu_x) f(le(T^\mu_x)) \log(le(T^\mu_x))$. This is the storage space needed by the secondary tree $T_{\mu,x}$. Thus, the sum of storage space needed by all the secondary trees at a particular level l in T_x is $\sum O(le(T^\mu_x) f(le(T^\mu_x)) \log(le(T^\mu_x))$. Since $\sum le(T^\mu_x) = O(m)$ for all the internal nodes $\mu \in T_x$ at a particular level h in T_x, $\sum O(le(T^\mu_x) f(le(T^\mu_x)) \times \log(le(T^\mu_x)) = O(mf(m)\log(m))$. Since the height of the primary tree T_x is $O(\log m)$, the total storage space needed by all the secondary trees across all the levels of the tree T_x is $O(mf(m)\log^2 m)$.

By step 3c of Section 3.5, finding the closest pair in L_1 metric for the points in the query rectangle involves identifying $O(\log^2 m)$ canonical nodes. This involves finding $O(\log m)$ canonical nodes μ in the primary tree T_x and for each such canonical node μ, finding $O(\log m)$ canonical nodes ϕ in the secondary tree $T_{\mu,x}$. By Lemma 7, finding the $O(\log m)$ canonical nodes μ in the primary tree needs $O(\log m)$ time. Finding the $O(\log m)$ canonical nodes ϕ in the secondary tree $T_{\mu,x}$ needs another $O(\log m)$ time. Thus, finding all the $O(\log^2 m)$ canonical nodes ϕ needs $O(\log^2 m)$ time. At each canonical node ϕ, the data structure W_ϕ is queried to find the point with smallest weight. Let the number of point stored in W_ϕ is denoted by $|W_\phi|$. While querying W_ϕ, $O(\log|W_\phi|)$ nodes of W_ϕ are searched. From the proof of Lemma 2, we can see that this step needs $O(g(|W_\phi|) + \log\log(|W_\phi|))$ time per node and hence $O(g(|W_\phi|) + \log\log(|W_\phi|)\log(|W_\phi|))$ time in total. Since the number of points stored in W_ϕ is less than or equal to $O(m)$, the total time needed at the node ϕ to search W_ϕ is $O((g(m) + \log\log m)\log m)$. Since there are $O(\log^2 m)$ canonical nodes, the total time needed is $O((g(m)+\log\log m)\log^3 m)$. Also, the data structure *RA* supports reporting points in a query rectangle in $O(\log^\varepsilon m + k\log^\varepsilon m)$ time where k is the output size. In our case, k is at most five. Therefore, the total time needed to query *RA* is $O(\log^\varepsilon m)$. Hence the claim. □

4. Conclusion

Combining Lemma 6 and Lemma 8, we conclude to Theorem 1.

References

[1] P. Bose, M. He, A. Maheshwari, P. Morin, Succinct orthogonal range search structures on a grid with applications to text indexing, in: WADS, in: Lecture Notes in Computer Science, vol. 6552, Springer, 2007, pp. 52–63.

[2] T.M. Chan, K.G. Larsen, M. Patrascu, Orthogonal range searching on the ram, revisited, in: Symposium on Computational Geometry, 2011, pp. 1–10.

[3] A.S. Das, P. Gupta, A.K. Kalavagattu, J. Agarwal, K. Srinathan, K. Kothapalli, Range aggregate maximal points in the plane, in: WALCOM, 2012, pp. 52–63.

[4] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry: Algorithms and Applications, second edition, Springer-Verlag, 2000.

[5] J. Fischer, Optimal succinctness for range minimum queries, in: LATIN, 2010, pp. 158–169.

[6] R. Janardan, P. Gupta, Y. Kumar, M.H.M. Smid, Data structures for range-aggregate extent queries, in: CCCG, 2008, pp. 7–10.

[7] G. Navarro, Y. Nekrich, Top-k document retrieval in optimal time and linear space, in: SODA, 2012, pp. 1066–1077.

[8] Y. Nekrich, G. Navarro, Sorted range reporting, in: SWAT, 2012, pp. 271–282.