Algorithms for proximity problems on colored point sets

Thorsten Graf and Klaus Hinrichs

FB 15 - INFORMATIK, Westfälische Wilhelms-Universität, Einsteinstr.62, D-48149 Münster

1 Introduction

Closest pair (CP) and *all nearest neighbors* (ANN) are fundamental problems in computational geometry. It is well known that these problems can be solved in time $O(n \log n)$ which is optimal in the algebraic decision tree model of computation.
Consider the following modifications of the CP and the ANN problems: Let S be a set of *colored* points. In the CFP (closest foreign pair) problem one has to find a closest foreign pair, i.e. a bichromatic pair of points which are closest. In the ANFN (all nearest foreign neighbors) problem one has to find for each point in the configuration S a nearest neighbor with different color. Since algorithms which solve the CFP or the ANFN problem also solve the CP or the ANN problem, respectively, for configurations that do not contain two points with the same color, the problems CFP and the ANFN are in $\Omega(n \log n)$. [AERT 89] present an optimal algorithm for the ANFN problem with respect to the L^2-metric making use of several Voronoi diagrams; see also [Ya 82] and [Va 84]. The ANFN problem for a constant number of colors and with respect to the Euclidean L^2-metric is mentioned in [HNS 92].
In this paper we present plane sweep algorithms solving the CFP and ANFN problem with respect to the L^∞- and the L^1-metric in optimal time $O(n \log n)$ and $O(n)$ space.
For two points $p, q \in I\!R^2$ their L^∞- and L^1-distances are given by $d_\infty(p,q) := \max\{|p.x - q.x|, |p.y - q.y|\}$ and $d_1(p,q) := |p.x - q.x| + |p.y - q.y|$. The d_1 and d_∞ metrics are relevant to various applications, such as modelling of arm movements in disc transport mechanisms [LW 80] and in integrated circuit layout. Obviously distances depend upon the location of the coordinate axes. The L^1- and L^∞- metrics have a useful relationship: Consider the transformation $\tau : (x,y) \rightarrow (x+y, y-x)$. It is easy to verify that for two points $p, q \in I\!R^2$ and the corresponding transformed points $\tau(p)$ and $\tau(q)$ we have $d_\infty(\tau(p), \tau(q)) = d_1(p,q)$. It follows that an algorithm that solves the CFP or the ANFN problem with respect to the L^∞-metric can also be used to solve the same problem with respect to the L^1-metric by adding a preprocessing step in which the points are transformed by τ. Therefore we concentrate on the problems with respect to the L^∞-metric.

2 The algorithm for the CFP problem

In this section we consider the *closest foreign pair problem:*

> Given a finite set S of points in $I\!R^2$, $|S| = n$, $S = \cup_{i=1}^N S_i$ with $S_i \cap S_j = \emptyset$ for $i,j \in \{1,\dots,N\}$, $i \neq j$, determine two points $p \in S_i$ and $q \in S_j$ $(i \neq j)$ with $d_\infty(p,q) = \min\{d_\infty(\tilde{p},\tilde{q}) : \tilde{p} \in S_k, \tilde{q} \in S_h, k,h \in \{1,\dots,n\}, k \neq h\}$.

Assign each point set S_i $(i \in \{1,\dots,N\})$ a unique color, and let $c(p)$ denote the color of a point $p \in S$. We will consider the following reformulation of the problem: Determine a *bichromatic closest pair* in S.
Our algorithm 'PSCFP' (Plane Sweep Closest Foreign Pair) uses the well known plane-sweep principle sweeping the plane from left to right with a vertical line (front, or cross-section), stopping at every transition point (event) of the geometric configuration to update the cross-section, i.e. to maintain the *sweep invariants* which have to hold for the points being encountered so far. All processing is done at this moving front, without any backtracking, with a look-ahead of only one point.
The *event queue* is initialized with the points of the configuration S sorted increasingly with respect to the lexicographic order

$$\forall p \neq q \in I\!R^2 : p \leq^x q :\Longleftrightarrow (p <_x q) \vee ((p =_x q) \wedge (p <_y q))$$

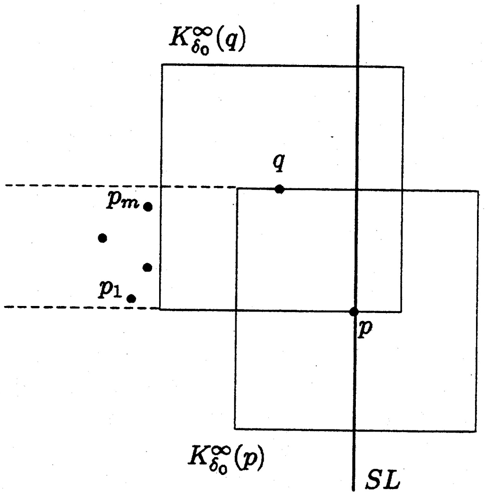
Throughout this paper we write $p \leq_x q$, $p \leq_y q$ and $p =_y q$ instead of $p.x \leq q.x$, $p.y \leq q.y$ and $p.y = q.y$, respectively. Denote by S_L the set of points which have already been encountered, i.e. the set of points seen so far, and by S_R its complement in S.
In the following we distinguish between *active* and *deactivated* points in S_L. When a point p is encountered by the

sweep line SL we wish to know whether this point forms a new closest foreign pair with one of the points in S_L. We keep a pair of closest foreign points found so far, along with their distance δ. Therefore for all candidates points q in S_L which may form a new closest foreign pair with the newly encountered point p we have that $q.x + \delta$ is to the right of the sweep line. A point q in S_L is called *active* if $q.x + \delta$ is to the right of SL, and if it can form a new closest pair with a point in S_R; otherwise the point q is called *deactivated*. Obviously two active points cannot have the same y-coordinate regardless of whether their colors are different or the same. Hence the active points can be stored in the y-table with respect to $<_y$ in increasing order.
During the sweep from left to right we maintain the following *sweep invariants*:

1) For each point p in the y-table $p.x + \delta$ is to the right of SL.
2) If p_1, p_2 are neighbors in the y-table with respect to $<_y$ and their colors are different, i.e. $c(p_1) \neq c(p_2)$, then $d(p_1, p_2) \geq \delta$.

The first invariant is maintained by removing points p from the y-table for which $p.x + \delta$ is on or to the left of the sweep line SL. In order to maintain the second invariant we have to compute distances of points with different colors which become neighbors with respect to $<_y$ in the y-table and update δ, if necessary. We obtain such new bichromatic neighbor pairs after inserting a new point into the y-table or after removing a point from the y-table. It seems to be surprising that our algorithm PSCFP finds a bichromatic closest pair by just testing bichromatic pairs of points which become neighbors in the y-table with respect to $<_y$.

Let $CFP = (p, q)$ be a closest foreign pair chosen in such a way that S_L does not contain another closest foreign pair when SL encounters the larger of the points p and q with respect to $\leq^x$. Let $\delta_0 := d_\infty(p, q)$.



W.l.o.g. we assume $(p \geq^x q) \wedge (q \geq_y p)$. Since $c(p) \neq c(q)$ no other point of S can be contained in $K^\infty_{\delta_0}(p) \cap K^\infty_{\delta_0}(q)$, where $K^\infty_\eta(r)$ denotes the L^∞-circle with radius η and center r. If p and q become neighbors in the y-table when SL encounters p, the second sweep invariant implies that a closest foreign pair is determined correctly and nothing remains to be shown. In particular this happens if $p =_y q$ and we therefore assume $q >_y p$ in the following.
Assume that after processing p there are active points $p_1, \ldots, p_m$ $(m \geq 1)$ separating p and q in the y-table, i.e.

$$p <_y p_1 <_y \cdots <_y p_m <_y q \tag{2.1}$$

Since $p_i \notin K^\infty_{\delta_0}(p) \cap K^\infty_{\delta_0}(q)$ for all $i = 1, \ldots, m$, we have $p_i.x < p.x - \delta_0$. Together with (2.1) this implies that

$$|p_i.x - p.x| > |p_i.y - p.y| \qquad \forall i = 1, \ldots, m \tag{2.2}$$

If the points p and p_1 have different colors, i.e. $c(p) \neq c(p_1)$, their distance $d_\infty(p, p_1)$ has been calculated and therefore $\delta \leq d_\infty(p, p_1)$. Together with (2.2) this implies $p_1.x + \delta \leq p.x$ which shows that p_1 must have been deactivated in contradiction to our assumption that p_1 is still active. Hence we have $c(p) = c(p_1)$.
Consider two points p_i and p_{i+1} $(1 \leq p < m - 1)$ with different colors, i.e. $c(p_i) \neq c(p_{i+1})$. Their distance has been calculated and therefore $\delta \leq d_\infty(p_i, p_{i+1})$. Note that p has the maximum x-coordinate occuring in S_L; hence

$$\begin{aligned}\min_{k=i,i+1} p_k.x + \delta &\leq \min_{k=i,i+1} p_k.x + d_\infty(p_i, p_{i+1}) = \min_{k=i,i+1} p_k.x + \max\{|p_i.x - p_{i+1}.x|, |p_i.y - p_{i+1}.y|\} \\ &\leq \max\{\max_{k=i,i+1} p_k.x, \min_{k=i,i+1} p_k.x + \delta_0\} \leq p.x\end{aligned}$$

This implies that one of the points p_i and p_{i+1} must have been dactivated. To avoid a contradiction to our assumption that p_i and p_{i+1} are active they must have the same color, i.e. $c(p_i) = c(p_{i+1})$. Together with $c(p_1) = c(p)$ this yields $c(p_i) = c(p)$ for all $i = 1, \ldots, m$ and in particular $c(p_m) \neq c(q)$. Hence the distance $d_\infty(p_m, q)$ of p_m and q has been calculated and therefore $\delta \leq d_\infty(p_m, q)$. Since $p_m \notin K^\infty_{\delta_0}(q)$ we have $d_\infty(p_m, q) = q.x - p_m.x$, and hence $p_m.x + \delta \leq p_m.x + d_\infty(p_m, q) = q.x \leq p.x$ which shows that p_m must have been dactivated in contradiction to our assumption. Hence there cannot be active points $p_1, \ldots, p_m$ separating p and q after the point p has been processed, i.e. p and q become neighbors in the y-table, their distance is computed and PSCFP finds a closest foreign pair correctly.

The initialization of the event queue, i.e. sorting the points in S with respect to $\leq^x$, can be accomplished in $O(n \log n)$ worst-case time. PSCFP computes the distances of at most $3(n-2)+1 = 3n-5$ $(n \geq 3)$ pairs of points implying that the cost for all operations performed on the y-table during the sweep sums up to $O(n \log n)$ since each point is inserted into and deleted from the y-table exactly once. Clearly PSCFP requires $O(n)$ storage.

3 The algorithm for the ANFN problem

In this section we consider the *all nearest foreign neighbors problem:*

> Given a finite set S of points in the plane $\mathbb{R}^2$, $|S| = n$, $S = \cup_{i=1}^{N} S_i$ with $S_i \cap S_j = \emptyset$ for $i, j \in \{1, \ldots, N\}$, $i \neq j$. For each $i \in \{1, \ldots, N\}$ and each $p \in S_i$ determine a point $q \in S \setminus S_i$ with $d_\infty(p,q) = \min\{d_\infty(p,r) : r \in S \setminus S_i\}$.

As in section 2 we assign each of the sets S_i a unique color and reformulate the problem as follows: Determine for each point $p \in S$ a nearest neighbor in S having a color different from $c(p)$.
For a point $p \in \mathbb{R}^2$ the two diagonal lines (with slopes ± 1) through p subdivide the plane into four quadrants. Let us denote these quadrants as follows: $QR(p) := \{q >_x p : |p.x - q.x| \geq |p.y - q.y|\}$, $QL(p) := \{q <_x p : |p.x - q.x| \geq |p.y - q.y|\}$, $QB(p) := \{q <_y p : |p.x - q.x| < |p.y - q.y|\}$ and $QT(p) := \{q >_y p : |p.x - q.x| < |p.y - q.y|\}$. For a point $p \in S$ denote by $nn(p)$ a nearest foreign neighbor of p found so far, and by $\delta(p)$ the distance between p and $nn(p)$. Furthermore let $NN(p)$ be the set of all nearest foreign neighbors of p in S.
Our algorithm 'PSANFN' (Plane Sweep All Nearest Foreign Neighbors) applies the plane sweep technique described in the previous section. PSANFN uses four sweeps: from left to right, from right to left, from top to bottom and from bottom to top. We only describe the left-to-right sweep, the other sweeps work similarly. In the left-to-right sweep we find a nearest foreign neighbor for all those points $p \in S$ for which there exists a nearest foreign neighbor $q \in NN(p) \cap QR(p)$. In the three remaining sweeps we find a nearest foreign neighbor q satisfying $q \in QL(p)$, $q \in QB(p)$ or $q \in QT(p)$ if a nearest foreign neighbor of p has not already been found before.
During the left-to-right sweep PSANFN maintains for each point $p \in S$ the smallest distance $\delta(p)$ detected so far between p and one of the other points in S_L. The y-table stores the *active* points $p \in S_L$ which still can have a nearest foreign neighbor among the points of S_R. In particular we know that $p.x + \delta(p)$ is to the right of SL for such an active point p. Obviously two active points with different colors cannot have the same y-coordinate.
For ease of presentation we assume that no two points with the same color have the same y-coordinate. Then the active points can be stored in the y-table with respect to $<_y$. Let the functions pred(r) and succ(r) return the predecessor and successor point of r in the y-table.
An active point p is *deactivated* and therefore removed from the y-table if the position of the sweep line is at or to the right of $p.x + \delta(p)$. This may happen either if the sweep line proceeds to the right or if $\delta(p)$ becomes smaller. Since $\delta(p)$ can be different for different active points, the deactivation events cannot be processed in the order given by the points' x-coordinates. Hence we have to deal with a dynamic processing of deactivation events. [Sch 91] shows how to support an efficient delete operation in a heap if the location of the the element to be deleted is known. Repeated shrinking of $\delta(p)$ for an active point p requires left shifts of its deactivation event.
It is easy to see that it is not sufficient to calculate only distances of points having different colors that become neighbors in the y-table with respect to $<_y$. Points with a color different from $c(q)$ for which the computation of their distance to q does not lead to an immediate deactivation, i.e. at $q.x$, cannot be contained in $QL(q)$. Therefore in PSANFN the y-table further supports the operations pred*(r) and succ*(r) which return for a point r its predecessor and successor point with respect to $<_y$ in $QL(r) \cap y$-table.
During the left-to-right sweep we maintain the following *sweep invariants*:

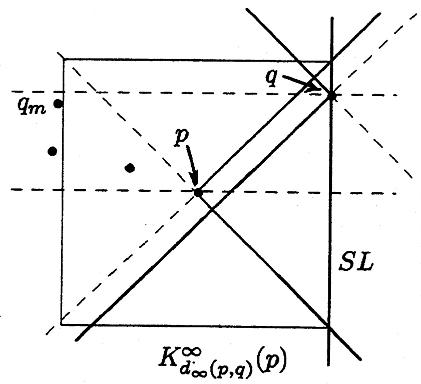
1) For each point p in the y-table $p.x + \delta(p)$ is to the right of SL.

2) If p_1, p_2 are neighbors in the y-table with respect to $<_y$ and their colors are different, i.e. $c(p_1) \neq c(p_2)$, then $d_\infty(p_1, p_2) \geq \delta(p_1)$ and $d_\infty(p_1, p_2) \geq \delta(p_2)$.

3) For a newly encountered point p either pred*$(p) = nil$ or $c(p) = c(\text{pred*}(p))$, and either succ*$(p) = nil$ or $c(p) = c(\text{succ*}(p))$.

The first two invariants do not differ from the sweep invariants in PSCFP except for the individual δ-values of the points. The third sweep invariant is maintained similarly as the second invariant for a new point encountered by the sweep line. If pred*(p) exists and its color is different from $c(p)$ then $\delta(\text{pred*}(p))$ has to be updated; then the first invariant implies that pred*(p) has to be deactivated. This process is repeated until we find either pred*$(p) = nil$ or

$c(p) = c(\text{pred}^*(p))$. In an analog way we treat the successors $\text{succ}^*(p)$. Let $p \in S$ for which there exists a nearest foreign neighbor $q \in NN(p) \cap QR(p)$. We prove that after q has been processed $\delta(p) = d_\infty(p,q)$, and therefore a nearest foreign neighbor (not necessarily q) of p has been found. W.l.o.g. we assume $q \geq_y p$.
Since $q \in NN(p)$ no other point $r \in S$ with $c(r) \neq c(p)$ can be contained in $K^\infty_{d_\infty(p,q)}(p)$. Assume that after processing q, i.e. maintaining the sweep invariants, $\delta(p) > d_\infty(p,q)$. Then $p \notin \{\text{pred}(q), \text{pred}^*(q)\}$. This implies that there are points $q_1, \ldots, q_m \in QL(q)$ $(m \geq 1)$ among the active points lying between p and q in the y-table, i.e. $p <_y q_1 <_y \cdots <_y q_m <_y q$ and $\text{pred}^*(q) = q_m$ as shown in the following figure:



Since $q_m \in QL(q)$ and q_m is still active it follows $c(q_m) = c(q)$. Let k $(1 \leq k \leq m)$ be the unique index with $c(q) = c(q_m) = \ldots = c(q_k)$ and $c(q_{k-1}) \neq c(q)$. If $k = 1$ then set $q_{k-1} := p$. Now consider the point $\tilde{q} = \text{pred}(q_k)$. Note that $\tilde{q} \neq q_{k-1}$ if $\tilde{q} \notin QL(q)$.
If $c(\tilde{q}) = c(q)$ then $\tilde{q} \notin K^\infty_{d_\infty(p,q)}(p)$, and therefore $\tilde{q} \in QL(q)$ and we get $q_{k-1} = \tilde{q}$. This is a contradiction to our choice of k. Hence $c(\tilde{q}) \neq c(q)$.

Now the second sweep invariant implies that the distance of $\tilde{q}$ and q_k has been calculated. Hence

$$\begin{aligned}\min\{\tilde{q}.x + \delta(\tilde{q}), q_k.x + \delta(q_k)\} &\leq \min\{\tilde{q}.x, q_k.x\} + d_\infty(\tilde{q}, q_k) \leq \min\{\tilde{q}.x, q_k.x\} + \max\{|\tilde{q}.x - q_k.x|, |\tilde{q}.y - q_k.y|\} \\ &\leq \max\{\max\{\tilde{q}.x, q_k.x\}, \min\{\tilde{q}.x, q_k.x\} + d_\infty(p,q)\} \leq q.x\end{aligned}$$

which implies that one of the points $\tilde{q}$ and q_k has already been deactivated. This contradicts our assumption that all points q_i $(i = 1, \ldots, m)$ and $\tilde{q}$ are still active. Therefore no such active points $q_1, \ldots, q_m$ can exist, and we obtain $\text{pred}^*(q) = p$. This implies that our assumption $\delta(p) > d_\infty(p,q)$ was wrong, i.e. a nearest foreign neighbor for p has been found.

The operations $\text{pred}(p)$, $\text{succ}(p)$, $\text{pred}^*(p)$ and $\text{succ}^*(p)$ can be performed in $O(\log n)$ time each (see section 4). PSANFN computes the distance of less than $3(n-2)+1+3n$ pairs of points. This implies that the cost for all operations performed on the y-table during the left-to-right sweep and hence the cost for the four sweeps sums up to $O(n \log n)$. Clearly PSANFN requires $O(n)$ space (see section 4).

It remains to show how to support the y-table with the operations $\text{pred}^*(p)$ and $\text{succ}^*(p)$. In the following we restrict ourselves to the description of $\text{succ}^*(p)$, the operation $\text{pred}^*(p)$ can be performed similarly. Let $q = \text{succ}^*(p)$, then q is the unique point in the y-table with $(q.x + q.y < p.x + p.y)$ and $(q >_y p)$ for which $q.y$ is minimal. We apply the transformation $\pi : (x,y) \longrightarrow (x+y, y)$ to all points contained in the y-table and store these points in a data structure QPST (quadrant priority search tree) which supports the following three operations:

1) *insert(p)*: Insert point p into the QPST.
2) *delete(p)*: Delete point p from the QPST.
3) *YMinInQuadrant(p)*: For a point p contained in the QPST find the unique (possibly non-existent) point q in the QPST with the properties $(q <_x p)$, $q >_y p$ and minimal y-value $q.y$.

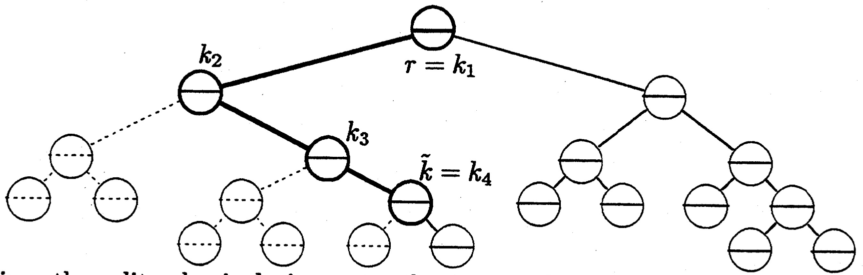
The operation YMinInQuadrant($\pi(p)$) on the transformed data is equivalent to $\text{succ}^*(p)$ on the original data. A similar QPST is built using the transformation $\tilde{\pi} : (x,y) \rightarrow (y-x, y)$.

4 The quadrant priority search tree

The QPST is based on the priority search tree ([Mc 85], [IKO 90]). The skeleton of the QPST is a *half balanced tree* ([Ol 82]). For each node v denote by l_v the number of edges in the longest path from v to a leaf and by s_v the number of edges in such a shortest path. Half balanced trees have the balance property $l_v \leq 2s_v$ for each inner node. This balance property can be maintained after an insertion or deletion operation with at most three rotations ([Ol 82]). The QPST is a 0-2 binary tree, i.e. each inner node has exactly two sons, and a leaf search tree for the y-values, i.e. for every y-value there exists one leaf in the tree. Every node contains the maximum y-value of its left subtree as a split value and space to store a point, possibly the nil-point. The points are stored according to the following three conditions:

1) Each point p lies on the root-to-leaf path to $p.y$.
2) The x-values of the points stored along an arbitrary root-to-leaf path are in increasing order.
3) If a node contains a point then its father does, too.

The y-coordinate $p.y$ of a point p (except for the maximal one) is the split-value of the node encountered after the first right turn on the leaf-to-root path starting at the leaf containing $p.y$. A point p is called *proper* with respect to p_0 if $p.x < p_0.x$ and $p.y > p_0.y$. Let LST(k) denote the left subtree and RST(k) denote the right subtree of a node k. A point p is *inserted* into the QPST by first inserting $p.y$ and performing the necessary rebalancing operations, and then sifting the point down the tree according to its weight, i.e. $p.x$: Compare the x-coordinates of p and the point stored in the root node, store the one with the smaller x-coordinate and continue this operation with the other point, exploring the root of the left or right subtree according to $p.y$, until an empty node is reached.
To *delete* a point p search for it, delete it and fill the gap by sifting successor points up the tree without changing the tree structurally. The x-heap property is maintained by pushing up that point contained in the son nodes which has the smaller x-coordinate. Finally delete the leaf containing $p.y$, update the split values and rebalance the tree if necessary.
It is easy to see that each rotation may cause one sift down and one sift up operation. Since we use a half balanced tree as a skeleton of the QPST, only constant many such operations have to be performed when rebalancing the tree.
In the following denote by r the root node of the QPST. Perform the operation *YMinInQuadrant*(p) as follows: By walking down a root-to-node path $K = \{r = k_1, \ldots, k_m = \tilde{k}\}$ search for the node $\tilde{k}$ with split value $p.y$. During this walk all points stored along this path are examined.
Let LT(k) be true iff $k_{\nu+1}$ is the left son of k_ν and RT(k) be true iff $k_{\nu+1}$ is the right son of k_ν for all nodes $k_\nu (\nu \in \{1, \ldots, n-1\})$. For $\tilde{k}$ define RT($\tilde{k}$) to be false and LT($\tilde{k}$) to be true. Consider a node $k = k_\nu$ with RT(k) or $k = \tilde{k}$. Since we performed a binary search for $p.y$, the split value in node k is less than $p.y$ for $k \neq \tilde{k}$ and equals $p.y$ if $k = \tilde{k}$. The properties of the QPST imply that the y-coordinates of all points stored in LST(k) are less than $p.y$. Therefore these points need not be considered. Since these points are not of interest we only consider a skeleton of the QPST consisting of the path K and the right subtrees of those nodes k_ν of K with LT(k_ν). A node k_ν in K with RT(k) is called a ***dead node***, and a node k_ν with LT(k_ν) is called a ***branch node***. The following figure shows an example with branch nodes $r = k_1$, $\tilde{k} = k_4$ and dead nodes k_2, k_3.



Fix a branch node k_ν. Since the split value in k_ν is greater than or equal to $p.y$, the y-coordinates of all points stored in RST(k_ν) are greater than $p.y$.
Let k_μ and k_ν be branch nodes with $\mu < \nu$, and let q be a proper (not necessarily y-minimal) point, which is either stored in a node of RST(k_ν) or in a node of K between k_μ and k_ν. Since K branches off to the left in k_μ, the proper point q is contained in LST(k_μ) and therefore has a y-value less than the y-values of all points stored in RST(k_μ). Hence the points in RST(k_μ) need not be considered.
The following procedure finds the y-minimal proper point p in the subtree RST(k_ν) of a branch node k_ν or detects the non-existence of such a point in time $O(1)$: Start in the right son node k of k_ν. If the point q stored in k has an x-value greater than $p.x$, the x-heap property of the QPST implies that this is also true for all other points in RST(k_ν). The non-existence of a proper point in RST(k_ν) is detected and we continue the process as described above. If the point q stored in node k has an x-value smaller than $p.x$ then q is proper. We continue the process in the left son node of k if the point it stores is proper; actually this point is better than all proper points stored in the right subtree of k. If the point in the left son node of k is non-proper, we continue in its right son node. We stop the process if we cannot choose a son node containing a proper point any more.
In the following we describe how to process the path K in reverse order: Starting in node $\tilde{k}$ determine for each branch node k_ν the y-minimal proper point in RST(k_ν). Our considerations above show that we can stop climbing the path

K if the search is successful. Otherwise climb up path K and continue with the predecessor branch node k_μ of k_ν if no proper point is stored along K between k_ν and k_μ. After processing K we output the y-minimal proper point or the nil point.
Now we show that YMinInQuadrant(p) finds the y-minimal proper point $\hat{p}$ stored in the node $\hat{k}$. Obviously $\hat{k}$ can not be a node in the subtree LST(k_ν) of a dead node k_ν. If $\hat{k}$ is a node on the path K, the point $\hat{p}$ is found when constructing K, and nothing remains to be shown.
Therefore let us assume that $\hat{p}$ is stored in one of the subtrees RST(k_ν) of a branch node k_ν. If the process terminates before the branch node k_ν with $\hat{k} \in$ RST(k_ν) is reached, a proper point would have been found in LST(k_ν) contradicting the y-minimality of $\hat{p}$. All nodes on the path from the root to $\hat{k}$ store points having x-coordinates less than $p.x$. While searching for the y-minimal proper point in RST(k_ν) we first try to walk left; we walk right if this is not possible, i.e. the point stored in the left son node is non-proper or the nil point. Consider a node k on the path from k_ν to $\hat{k}$: If $\hat{k}$ is contained in LST(k) the left son node of k stores a proper point and we continue in LST(k). If $\hat{k}$ is contained in RST(k) the left son node of k cannot contain a proper point since this would lead to a contradiction to the y-minimality of the proper point $\hat{p}$. Therefore we continue in RST(k). Hence the node $\hat{k}$ is examined and the point $\hat{p}$ is found when processing the subtree RST(k_ν).

The height of a half balanced tree is bounded by $2\log(n+2) - 2 \in O(\log n)$ ([Ol 82]) which implies the same bound for the QPST. Hence sifting up and sifting down require $O(\log n)$ time each. Clearly insertions and deletions can be performed in $O(\log n)$ time. Since the length of the path K is bounded by the height of the QPST, at most $O(\log n)$ many nodes are considered when processing K, i.e. performing the operation YMinInQuadrant(p). Clearly a QPST requires $O(n)$ space if n is the number of points it stores.

Until now we have assumed that all points have different y-coordinates. However, the QPST can be modified so it can handle multiple points with the same y-coordinate. We first try to store a point p with $p.y = y_0$ in a node on the path from the root to the leaf k containing y_0. If this is not possible then p is stored in an overflow structure assigned to leaf k, i.e. a balanced binary tree which stores points with y-coordinate y_0 according to their x-coordinate. The QPST operations can be easily modified to deal with the overflow structures, their time complexities do not deteriorate.

References

[AERT 89] A.Aggarwal, H.Edelsbrunner, P.Raghavan and P.Tiwari: Optimal time bounds for some proximity problems in the plane, *Information Processing Letters* 42, 55-60 (1992).

[HNS 92] K.Hinrichs, J.Nievergelt, P.Schorn: An all-round sweep algorithm for 2-dimensional nearest-neighbor problems, *Acta Informatica*, 29(4), 383-394 (1992).

[IKO 90] Ch.Icking, R.Klein, Th.Ottmann: Priority search trees in secondary memory, in H. Göttler und H.J. Schneider (eds.), *Graphtheoretic Concepts in Computer Science* (WG '87), *LNCS* 314, 84-93, Springer-Verlag, New York, 1987.

[LW 80] D.T. Lee and C.K. Wong: Voronoi diagrams in L_1 (L_∞) metrics with 2-dimensional storage applications, *SIAM Journal on Computing* 9(1), 200-211 (1980).

[Mc 85] E.M.McCreight: Priority search trees, *SIAM Journal on Computing* 14(2), 257-276 (1985).

[Ol 82] H.J.Olivie: A new class of balanced search trees: Half-balanced binary search trees, *R.A.I.R.O. Informatique Theorique* 16, 51-71 (1982).

[Sch 91] P. Schorn: Robust algorithms in a program library for geometric computation, PhD Dissertation No. 9519, ETH Zürich, Switzerland, 1991.

[Va 84] P.M. Vaidya: A fast approximation algorithm for minimum spanning trees in k-dimensional space, *Proc. 25th Annual IEEE Symposium on Foundations of Computer Science*, 403-407, (1984).

[Ya 82] A.C. Yao: On constructing minimum spanning trees in k-dimensional space and related problems, *SIAM Journal on Computing* 11(4), 721-737 (1982).