An optimal algorithm for dynamic post-office problem in R_1^2 and related problems

(Extended abstract)

Sergei N. Bespamyatnikh
Department of Mathematics and Mechanics,
Ural State University,
51 Lenin St., Ekaterinburg 620083, Russia.
e-mail: Sergei.Bespamyatnikh@usu.ru.

Abstract

Let S be a set of n points in $\mathbf{R}^k$. It is shown that a fair split tree can be used to find an L_∞-nearest neighbor in S of any query point, in $O(\log^{k-1} n)$ time. This data structure has size $O(n \log^{k-2} n)$ and an update time of $O(\log^{k-1} n)$. For $k = 2$ this algorithm is optimal and can be used for Manhattan metric. For $k \geq 3$ the update time can be reduced to $O(\log^{k-2} n \log\log n)$ applying the fractional cascading.

This result is used to solve the dynamic $(1+\varepsilon)$-approximate L_2-nearest neighbor problem and the dynamic all-nearest-neighbors problem within the same bounds. This improves previous bounds by $\log n$ factor.

1 Introduction

In this paper we address to the well-known post-office problem. The post-office problem is stated as follows [18].

The post-office problem: Given a set S of n points in $\mathbf{R}^k$, store it in a data structure such that for any query point $p \in \mathbf{R}^k$, we can efficiently find its *nearest neighbor*, i.e., a point $p^* \in S$ that is closest to p,

$$d(p, p^*) = \min\{d(p, q) : q \in S\}.$$

In the dynamic version of problem the set S is dynamically changed by insertions and deletions of points. It is an open problem [17, 21] if there exists a dynamic data structure for the planar problem having size $O(n \log^{O(1)} n)$ and polylogarithmic query and update times.

We consider two weaker versions of the problem. First, we consider the post-office problem for "simple" metrics, namely L_1- or L_∞-metric. The Table 1 contains the previous results for this problem in plane. We give an optimal algorithm for solving the planar version of the problem, i.e. with $O(\log n)$ update and query time using (n) space. The algorithm is based on the fair split tree [11, 12, 9] and the dynamic trees [5, 20]. We extend this algorithm to higher-dimensional space. The algorithm has $O(\log^{k-1} n)$ query and update time and $O(n)$ space. Applying fractional cascading for $k \geq 3$ we can obtain an algorithm with $O(\log^{k-2} n \log\log n)$ query time and $O(\log^{k-2} n \log\log n)$ amortized update time.

Second, we consider the variant in which we do not have to find the *exact* nearest neighbor p^* of the query point p, but are satisfied with an *approximate* neighbor, i.e., a point $q \in S$ such that $d(p, q) \leq (1 + \varepsilon)\, d(p, p^*)$, for some positive constant ε. There are two variant of the approximate neighbor problem. In the first case, ε is the variable, i.e. query input consists of a point p and an approximation bound ε. Arya et al. [3] gave an algorithm with $O((1 + \frac{1}{\varepsilon})^k \log n)$ query

update time	w/a	query time	w/a	space	reference
$\log n \log\log n$	w	$\log n \log\log n$	a	$n \log n$	[17]
$\log^3 n \log\log n$	w	$\log^3 n \log\log n$	w	n	[7]
$\log^2 n$	w	$\log^2 n$	a	$n \log^2 n$	[14][1]
$\log n$	w	$\log n$	w	n	this paper

Table 1. The data structures for the dynamic post-office problem in $\mathbf{R}_1^k$. All bounds are "big-oh". The update times are either worst-case (w) or amortized (a).

time and $O(\log n)$ update time, using $O(n)$ space. Bespamyatnikh [10] obtained an algorithm with $O((1+\frac{1}{\varepsilon})^{k-1}+\log n)$ query time and $O(\log n)$ update time, using $O(n)$ space.

In the second case, ε is a constant. Applying range tree techniques and fractional cascading Kapoor and Smid [17] obtained a data structure of size $O(rn\log^{k-1} n)$ that allows to answer to a query in $O(r\log^{k-1} n \log\log n)$ time and to update S in $O(r\log^{k-1} n \log\log n)$ amortized time. r is the number of range trees or coordinate systems. Chan and Snoeyink [14] noted that r is $O((1+\frac{1}{\varepsilon})^{\frac{k-1}{2}})$. Applying the approach of Kapoor and Smid [17] to our dynamic L_∞-post-office algorithm we solve the approximate L_t-neighbor problem with a query time of $O(\log^{k-2} n \log\log n)$ and an amortized update time of $O(\log^{k-2} n \log\log n)$, using $O(n\log^{k-2} n)$ space.

Finally we consider the all-nearest-neighbors problem. Vaidya [22, 23] has given an optimal $O(n\log n)$ algorithm that computes the L_t-neighbor for each point of S. Kapoor and Smid [17] gave a data structure of size $O(n\log^{k-1} n)$ that maintains for each point in S its L_∞-neighbor in $O(\log^{k-1}\log\log n)$ amortized time per update. We presented an optimal algorithm for dynamic all-nearest-neighbors problem in plane under L_∞ (L_1) metric, i.e. with $O(\log n)$ update time and $O(n)$ space. In higher dimensions we give an algorithm with the same complexity bounds as the bounds of our dynamic L_∞-post-office algorithm. In fact we prove that any algorithm for solving the dynamic post-office problem can be used for solving the dynamic all-nearest-neighbors problem with the same complexity bounds.

[1]The data structure of T. Chan and J. Snoeyink can be used to solve the weighted L_1-problem.

2 The fair split tree

In this Section we describe a fair split tree as in [10]. The fair split tree is a hierarchical subdivision of space into boxes. We define a box to be the product $[a_1, a_1') \times \ldots \times [a_d, a_d')$ of d semiclosed intervals. i-th side of this box is the interval $[a_i, a_i')$. If all sides have the same length, we say that the box is a *d-cube*.

Let s be a separator [10]. The separator is at least Golden Ratio, i.e $s \geq \frac{\sqrt{5}+1}{2} \approx 1.62$

Definition 2.1 Let $[a, a')$ be an interval in $\mathbf{R}$ and b be a point in this interval. The split of the interval into the intervals $[a, b)$ and $[b, a')$ is *fair split* if $\frac{b-a}{a'-b} \in [\frac{1}{s}, s]$.

Definition 2.2 Let $B = [a_1, a_1') \times \ldots \times [a_d, a_d')$ be a box and $c_i \in (a_i, a_i')$ be a real number for some i. The split of B by the hyperplane $x_i = c_i$ into the boxes $B \cap \{x | x_i < c_i\}$ and $B \cap \{x | x_i \geq c_i\}$ is *fair split* of B if the split of the interval $[a_i, a_i')$ by c_i is fair split.

The fair-split operation generates a relation on the set of boxes.

Definition 2.3 Let A and B be d-dimensional boxes. The box A is said to be a *s-sub-box* of B if A can be constructed from B by applying a (possibly empty) sequence of fair cuts. We shall write $B \rightsquigarrow A$. For $d = 1$, we shall say that A is *s-sub-interval* of B.

The relation of s-sub-box is the product of s-sub-interval relation.

We do not include the condition of almost cubical boxes into the definition of the fair split of boxes although we shall apply fair split only for such boxes. The almost cubical boxes can be obtained from cubes by applying repeatedly a fair

split by a hyperplane perpendicular to one of the longest side of box.

Definition 2.4 Let B be a box with sides $s_1, \ldots, s_k$. The box B is said to be a *c-box* if, for any $i, j \in \{1, \ldots, k\}$, $\frac{s_i}{s_j} \in [\frac{1}{1+s}, 1+s]$.

The fair split tree is the binary tree T. With each node v of the tree T, we store a box $B(v)$ and a shrunken box $SB(v)$. The boxes satisfy the following conditions.

1. For any node v, the boxes $B(v)$ and $SB(v)$ are c-boxes.
2. For any node v, the box $SB(v)$ is a s-sub-box of $B(v)$.
3. For any node v, $SB(v) \cap S = B(v) \cap S$.
4. If w has two children u and v, then boxes $B(u)$ and $B(v)$ are the results of an fair split of the box $SB(w)$.
5. If v is a leaf, then $|S \cap B(v)| = 1$ and $SB(v) = S \cap B(v)$.

For a point $p \in S$ corresponding to the leaf v, let $B(p)$ denotes the box $B(v)$.

Let $parent(v)$, $lson(v)$, and $rson(v)$ denote parent, left son, and right son of the node v of T.

We use $d_{min}(X, Y)$ to denote the distance between two sets $X, Y \subset \mathbf{R}^k$, i.e. $d_{min}(X, Y) = \inf\{dist(x, y) | x \in X, y \in Y\}$. $d_{max}(X, Y)$ denotes the maximal distance between two sets $X, Y \subset \mathbf{R}^k$, i.e. $d_{max}(X, Y) = \sup\{dist(x, y) | x \in X, y \in Y\}$.

3 The dynamic post-office algorithm in the plane

In this Section we modify the dynamic tree and show that L_∞-neighbor queries can be answered in $O(\log n)$ time.

At first we consider the degenerate case when the path tree contains only one node. For single node v of such solid path P, we store four additional pointers

$x_{\min}(v)$ = a point $p \in B(v)$ that minimizes p_1,
$x_{\max}(v)$ = a point $p \in B(v)$ that maximizes p_1,
$y_{\min}(v)$ = a point $p \in B(v)$ that minimizes p_2,
$y_{\max}(v)$ = a point $p \in B(v)$ that maximizes p_2.

Let v be an internal node of the fair split tree and u, w be the sons of v. We say that u has type UP if the box $SB(v)$ is split by a line $y = const$ and the box $B(u)$ lies above it. In this case we write $type(v) = UP$. Also we define types $DOWN$, $LEFT$ and $RIGHT$. Clearly, there exist only four types.

At each internal node v of a path tree, we store eight auxiliary pointers $up\_x_{\min}$, $up\_x_{\max}$, $down\_x_{\min}$, $down\_x_{\max}$, $left\_y_{\min}$, $left\_y_{\max}$, $right\_y_{\min}$ and $right\_y_{\max}$. Let $v_1, \ldots, v_k$ be a subpath corresponding to v, i.e.

$$\begin{aligned} v_k &= bhead(v), \\ v_{k-1} &= parent(v_k), \\ &\ldots, \\ v_1 &= parent(v_2) = btail(v). \end{aligned}$$

Let $S(v) \subset S$ be a set of points such that the point locations for them pass through v. In other words,

$$S(v) = \begin{cases} S \cup B(v_1), \text{ if no solid edge enters } v_k \\ S \cup (B(v_1) \setminus B(w)), \text{ if the solid} \\ \qquad\qquad \text{edge } (w, v_k) \text{ enters } v_k \end{cases}$$

Define $UP(v)$. The point $p \in S(v)$ is included into $UP(V)$ if there is a node u of the fair split tree such that $btail(v)$ is ancestor of u and $type(u) = UP$. Now we define the pointers

$$\begin{aligned} up\_x_{\min}(v) &= \text{a point } p \in UP(v) \text{ that} \\ &\qquad \text{minimizes } p_1 \\ up\_x_{\max}(v) &= \text{a point } p \in UP(v) \text{ that} \\ &\qquad \text{maximizes } p_1 \end{aligned}$$

The next six pointers can be defined in similar way.

$$\begin{aligned} down\_x_{\min}(v) &= \text{a point } p \in DOWN(v) \text{ that} \\ &\qquad \text{minimizes } p_1 \\ down\_x_{\max}(v) &= \text{a point } p \in DOWN(v) \text{ that} \\ &\qquad \text{maximizes } p_1 \\ left\_y_{\min}(v) &= \text{a point } p \in LEFT(v) \text{ that} \\ &\qquad \text{minimizes } p_2 \\ left\_y_{\max}(v) &= \text{a point } p \in LEFT(v) \text{ that} \\ &\qquad \text{maximizes } p_2 \\ right\_y_{\min}(v) &= \text{a point } p \in RIGHT(v) \text{ that} \\ &\qquad \text{minimizes } p_2 \\ right\_y_{\max}(v) &= \text{a point } p \in RIGHT(v) \text{ that} \\ &\qquad \text{maximizes } p_2 \end{aligned}$$

3.1 The nearest neighbor searching

Now we describe the algorithm for finding nearest neighbor. For simplicity we assume that additional pointers are the coordinates of corresponding points.

Given a query point q. We apply the technique similar to the finding the sets E_p and $A(v)$ [8, 9]. The search uses a set V of nodes. We store V in a queue (a queue functions in a first-in, first-out manner). An element of V is a node of the fair split tree or a node of a path tree. With each node $v \in V$ we associate a domain $D(v) \subset \mathbf{R}^k$ [10]. To define the domain $D(v)$ we consider three cases.

Case 1. v is a node of fair split tree. $D(v) = SB(v)$.

In the next two cases v is a node of a path tree PT.

Case 2. v is a leaf of a path tree. v corresponds to a node v' of T. We replace v in V with a node of T which is determined as follows. Note that v' is an internal node of T. Let u and w be sons of v'. If u and w are linked to v' by dashed edges then we replace v with v'. If the edge (v', u) is dashed and the edge $(v'w)$ is solid then we replace v with u.

Case 3. v is an internal node of a path tree. The node v covers nodes $v_1, \ldots, v_k$ of solid path where $parent(v_{i+1}) = v_i$ for $i = 1, \ldots, k-1$. If v_k is the bottommost node of path tree then $D(v) = SB(v_1)$ $(v_1 = btail(v))$. Otherwise v_k has a son v_{k+1} linked to it by a solid edge. The domain $D(v) = SB(v_1) \setminus SB(v_{k+1})$.

To process domains $D(v)$ for internal nodes of path trees we store two boxes $B_{out}(v)$ and $B_{in}(v)$ (possibly empty) such that $D(v) = B_{out}(v) \setminus B_{in}(v)$. This information allows us, for a node $v \in V$, compute $D(u)$ and $D(w)$ where u and w appear when we traverse the node v (and the boxes $B_{out}(v)$ and $B_{in}(v)$).

The search algorithm uses the pointer *pcand* to a candidate for nearest neighbor of q and *dcand* for the distance from q to *pcand*. Initially, we set *pcand* := *null* and *dcand* := ∞. When the algorithm stops *pcand* point to the nearest neighbor of q. The initial set V contains one node $pt\_root(root(T))$. The algorithm proceed the search step while $V \neq \emptyset$. Let v be a node of V. We show how to process this node. As was explained above we traverse v and obtain nodes u and w. Delete node v from queue V.

Case 1. The query point q lies in the domain $D(v)$. Add the nodes u and w to the queue V and stop processing of v.

Case 2. Consider the lines $l_1 = \{p | p_1 - q_1 = p_2 - q_2\}$ and $l_2 = \{p | p_1 + q_1 = p_2 + q_2\}$. These lines partition the plane into four quarters. In the Case 2 the domain $D(v)$ is contain in one quarter of plane. Wlog $D(v)$ lies in the quarter $\{p | p_1 - q_1 \geq |p_2 - q_2|\}$. We can compute the neighbor of q in $D(v)$ in $O(1)$ time.

Case 3. The domain $D(v)$ intersects both lines l_1 and l_2. Add the nodes u and w to the queue V and stop processing of v.

Case 4. The domain $D(v)$ intersects one of the lines l_1 and l_2. Wlog $D(v)$ intersect the ray $ray = \{p | p_1 - q_1 = p_2 - q_2 > 0\}$. If the domain $D(u)$ $(D(w))$ does not intersect the ray then

- compute the nearest neighbor of q in $D(u)$ and update *pcand*,
- add w to V, and stop processing of v.

Now $D(u)$ and $D(w)$ intersect ray. We shall show that search shall traverse only one of two nodes u or w (in fact other node can contain the nearest neighbor). Compute $d(q, D(u))$ and $d(q, D(w))$. Without loss of generality $d(q, D(u)) \leq d(q, D(w))$. Let p be the point which lies on the ray ray at distance $d(q, D(w))$ from query point q.

The following special case demonstrate the main idea of one node exclusion. Let $D(u)$ and $D(w)$ are the boxes (hence u and w are the nodes of fair split tree) and a vertical line split the box $B(v)$ into these boxes. If $y_{\min}(u) \leq p_2$ then there exists a point $r \in D(u)$ such that

- $r_2 = y_{\min}(u)$ and
- $d(q, r) \leq \max\{r_1 - q_1, r_2 - q_2\} \leq \max\{p_1 - q_1, p_2 - q_2\} = d(q, D(w))$.

Hence $\mathbf{R}^2 \setminus D(w)$ contains a nearest neighbor of q and we can remove w. Add u to the queue Q.

If $y_{\min}(u) > p_2$ then $d(q, D(u)) = y_{\min}(u) - q_2$. We take into account the nearest neighbor of q in $D(u)$. Add the node w to the queue Q.

Now consider the general case. Let

$$db = \min(left\_y_{\min}(u) - q_2, down\_x_{\min}(u) - q_1).$$

If $db \leq d(p,q)$ then remove w and add the node u to the queue Q. Otherwise take into account a point $p' \in D(u)$ such that $d(p,p') = db$, and add the node w to the queue Q.

We omit the proof of the following Theorem.

Theorem 3.1 *The above search algorithm finds a nearest neighbor of a query point in* $O(\log n)$ *time.*

3.2 The maintenance of modified tree

Note that an insertion (deletion) of point in (from) S causes $O(\log n)$ creations and updates of nodes. We can modify the insertion (and deletion) algorithm to mark these nodes. The number of marked nodes is $O(\log n)$. The parent of any marked node is marked. We have to compute the auxiliary pointers (four or eight) for marked nodes. We apply the *breadth-first search* on the search tree. It is easy to show that, for a node v, the auxiliary pointers of v can be computed in $O(1)$ time if these pointers are known for the sons of v.

Theorem 3.2 *The auxiliary pointers can be maintained in* $O(\log n)$ *time per update of the point set.*

4 The dynamic post-office algorithm in higher dimensions

In this Section, we apply the range tree technique to generalize the dynamic post-office algorithm of previous Section to higher dimensions. For a point p in $\mathbf{R}^k$, we denote by p' the point $(p_2, \ldots, p_k)$ in $\mathbf{R}^{k-1}$). For a set A of points in $\mathbf{R}^k$, we define $S' = \{p' \mid p \in S\}$). For the dimension $k = 2$, we use the data structure of previous Section.

For the dimension $k > 2$, the data structure is constructed as follows. The balanced binary tree stores the point of S in its leaves, sorted by their first coordinates. For each internal node v, let S_v be the set of points that are stored in its subtree. For v, we store $(k-1)$-dimensional data structure for the set S'_v. We apply dynamic fractional cascading [13, 19] as in [17]. This proves the following result.

Theorem 4.1 *For the dynamic post-office problem in k-dimensional space,* $k \geq 3$, *under* L_∞*-metric, there is a data structure of size* $O(n \log^{k-2} n)$ *having* $O(\log^{k-2} n \log\log n)$ *query time and* $O(\log^{k-2} n \log\log n)$ *amortized update time.*

5 The dynamic all-nearest-neighbors problem

In this Section we show that our dynamic post-office algorithm can be used for solving the dynamic all-nearest-neighbors problem with the same complexity bounds.

Theorem 5.1 *Let DS be any data structure for the dynamic post-office problem. Let* $S(n)$, $Q(n)$ *and* $U(n)$ *denote the size and query and update time of DS, respectively. For the dynamic all-nearest-neighbors problem there is a data structure of size* $S(n)$ *having* $Q(n) + U(n)$ *update time.*

Corollary 5.2 *Using our dynamic post-office algorithm we obtain the data structure for the dynamic all-nearest-neighbors problem with the following complexity bounds.*

- *For planar case, the data structure has size* $O(n)$ *and an update time of* $O(\log n)$.
- *For the dimension* $k \geq 3$, *the data structure has size* $O(n \log^{k-2} n)$ *and an amortized update time of* $O(\log^{k-2} n \log\log n)$.

We omit the proof of Theorem 5.1.

References

[1] S. Arya and D. M. Mount. *Approximate Nearest-Neighbor Queries in Fixed Dimensions.* Proceedings 4th Annual Symposium on Discrete Algorithms, 1993, pp. 271–280.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. *An Optimal Algorithm for Approximate Nearest-Neighbor*

Searching. Proceedings 5th Annual Symposium on Discrete Algorithms, 1994, pp. 573–582.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. *An Optimal Algorithm for Approximate Nearest-Neighbor Searching.* (revised version), 1994.

[4] S. Arya and D. M. Mount. *Approximate Range Searching.* Proc. 11th Annual ACM Symp. Comput. Geom., 1995, pp. 172–181.

[5] S. W. Bent, D. D. Sleator and R.E. Tarjan *Biased Search Trees.* SIAM Journal of Computing, 1985, 14, pp. 545-568

[6] M. Bern. *Approximate closest-point queries in high dimensions.* Inform. Proc. Lett., 45, 1993, pp. 95–99.

[7] S. N. Bespamyatnikh. *The Region Approach for Some Dynamic Closest-Point Problems.* Proc. 6th Canadian Conf. Comput. Geom., 1994.

[8] S. N. Bespamyatnikh. *An optimal algorithm for closest pair maintenance.* Proc. 11th Annual ACM Symp. Comput. Geom., 1995, pp. 152–161.

[9] S. N. Bespamyatnikh. *An optimal algorithm for closest pair maintenance.* manuscript (final version).

[10] S. N. Bespamyatnikh. *Dynamic algorithms for approximate neighbor searching.* manuscript.

[11] P. B. Callahan and S. R. Kosaraju. *A Decomposition of Multi-Dimensional Point-Sets with Applications to k-Nearest-Neighbors and n-Body Potential Fields.* Proc. 24th Annual AMC Symposium on the Theory of Computing, 1992, pp. 546–556.

[12] P. B. Callahan and S. R. Kosaraju. *Algorithms for Dynamic Closest Pair and n-Body Potential Fields.* Proc. 6th Annual ACM-SIAM Symp. on Discrete Algorithms, 1995.

[13] B. Chazelle and L.J. Guibas. *Fractional cascading I: A data structuring technique.* Algorithmica 1 (1986), pp. 133-162.

[14] T.M. Chan and J. Snoeyink. *Algorithms for approximate nearest-neighbor queries.* Manuscript, 1995.

[15] K. L. Clarkson. *A Randomized Algorithm for Closest-Point Queries.* SIAM Journal on Computing, 17(4), 1993, pp. 830–847.

[16] K. L. Clarkson. *An Algorithm for Approximate Closest-Point Queries.* Proc. 10th Annual ACM Symp. Comput. Geom., 1994, pp. 160–164.

[17] S. Kapoor and M. Smid. *New Techniques for Exact and Approximate Dynamic Closest-Point Problems.* Proc. 10th Annual ACM Symp. Comput. Geom., 1994, pp. 165–174.

[18] D.E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Addison-Wesley, Reading, MA, 1973.

[19] K. Mehlhorn and S. Näher. *Dynamic fractional cascading.* Algorithmica 5 (1990), pp. 215-241.

[20] D. D. Sleator and R. E. Tarjan. *A Data Structure for Dynamic Trees.* Journal of Computer and System Sciences, 26, 1983.

[21] M. Smid. *Closest Point Problems in Computational Geometry.* to appear in the Handbook on Computational Geometry, edited by J.-R. Sack and J. Urrutia.

[22] P. M. Vaidya. *An Optimal Algorithm for All-Nearest-Neighbors Problem.* Proceedings 27th Annual Symposium Found. Comp. Sc., 1986, pp. 117–122.

[23] P. M. Vaidya. *An $O(n \log n)$ Algorithm for All-Nearest-Neighbors Problem.* Discrete Comput. Geom., 1989, pp. 101–115

[24] A. C. Yao and F. F. Yao. *A General Approach to D-Dimensional Geometric Queries.* Proc. 17th Annu. ACM Sympos. Theory Comput., 1985, pp. 163–168.