Dynamic Algorithms for Approximating Interdistances

Sergey Bereg* and Michael Segal†

November 22, 2004

Abstract

In this paper we present efficient dynamic algorithms for approximation of k^{th}, $1 \leq k \leq \binom{n}{2}$ distance defined by some pair of points from a given set S of n points in d-dimensional space. Our technique is based on the dynamization of well-separated pair decomposition proposed in [11], computing approximate nearest and farthest neighbors [23, 26] and use of persistent search trees [18].

1 Introduction

Let S be a set of n points in $\mathbb{R}^d, d \geq 1$ and let $1 \leq k \leq \frac{n(n-1)}{2}$. Let $d_1 \leq d_2 \leq \ldots \leq d_{\binom{n}{2}}$ be the L_p-distances determined by the pairs of points in S. In this paper we consider the dynamic version of the following optimization problem:

- **Distance selection.** Compute the k-th smallest Euclidean distance between a pair of points of S.

In the dynamic version of the distance selection problem the points are allowed to be inserted or deleted and given a number k, $1 \leq k \leq \binom{|S|}{2}$ one wants to answer efficiently what is the k-th smallest distance between a pair of points of S (by $|S|$ we denote the cardinality of the current set of points).

The distance selection problem above received a lot of attention during the past decade. The solution to the distance selection problem can be obtained using a parametric searching. The decision problem can be reduced to the following problem. Compute, for a given real r, the sum $\Sigma_{p \in S} |D_r(p) \cap (S - \{p\})|$, where $D_r(p)$ is the closed disk of radius r centered at p. Thus, we can determine, for each point p in S, the number of points that are at distance at

*Department of Computer Science, University of Texas at Dallas, Box 830688, Richardson, TX 75083, USA. E-mail: besp@utdallas.edu

†Communication Systems Engineering Department, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel. E-mail: segal@cse.bgu.ac.il. Preliminary version of this paper has appeared in International Colloquium on Automata , Languages and Programming (ICALP), 2003.

most r from p. By summarizing all these values and comparing the resulted value to k we can answer the original decision problem. Agarwal et al. [1] gave an $O(n^{\frac{4}{3}} \log^{\frac{4}{3}} n)$ expected-time randomized algorithm for the decision problem, which yields an $O(n^{\frac{4}{3}} \log^{\frac{8}{3}} n)$ expected-time algorithm for the distance selection problem. Goodrich [22] derandomized this algorithm, at a cost of an additional polylogarithmic factor in the runtime. Katz and Sharir [27] obtained an expander-based $O(n^{4/3} \log^{2+\varepsilon} n)$-time deterministic algorithm for this problem. By applying a randomized approach Chan [13] was able to obtain an $O(n \log n + n^{2/3} k^{1/3} \log^{5/3} n)$ expected time algorithm for this problem. Bespamyatnikh and Segal [9] considered an approximation version of the distance selection problem. For a distance d determined by some pair of points in S and for any fixed $0 < \delta_1 \leq 1$, $\delta_2 \geq 1$, the value d' is the (δ_1, δ_2)-*approximation* of d, if $\delta_1 d \leq d' \leq \delta_2 d$. They [9] present an $O(n \log^3 n/\varepsilon^2)$ runtime solution for the distance selection problem that computes a pair of points realizing distance d' that is either $(1, 1+\varepsilon)$ or $(1-\varepsilon, 1)$-approximation of the actual k-th distance, for any fixed $\varepsilon > 0$. They also present an $O(n \log n/\varepsilon^2)$ time algorithm for computing the $(1-\varepsilon, 1+\varepsilon)$-approximation of k-th distance and show how to extend their solution in order to answer efficiently the queries approximating k-th distance for a static set of points. Agarwal et al. [1] considers a similar problem, where one want to identify approximate "median" distance, that is, a pair of points $p, q \in S$ with the property that there exist absolute constants c_1 and c_2 such that $0 < c_1 < \frac{1}{2} < c_2 < 1$ and the rank of the distance determined by p and q is between $c_1\binom{n}{2}$ and $c_2\binom{n}{2}$. They [1] showed how to solve this problem in $O(n \log n)$ time. Arya and Mount[4] introduced a *balanced box-decomposition tree* (BBD tree) in order to answer efficiently approximate range searching queries. They obtained $O(\log n + \frac{1}{\varepsilon^d})$ query time for d-dimensional point sets using linear space after $O(n \log n)$ preprocessing time. Their results also can be used to solve the decision version of the distance selection problem with $(1-\varepsilon, 1+\varepsilon)$-approximation in $O(n \log n + \frac{n}{\varepsilon^2})$ runtime.

We call an algorithm an *almost-linear-time approximation scheme with almost logarithmic update time (ALTAS-LOG) of order* (c_1, c_2) if it has a preprocessing time of the form $O(n \log^{l_1} n/\varepsilon^{c_1})$, for some constant $l_1 > 0$ and update time of the form $O(\log^{l_2} n/\varepsilon^{c_2})$, for some constant $l_2 > 0$ and any fixed $\varepsilon > 0$.

In this paper we show an ALTAS-LOG algorithm of order $(2, 2)$ such that given number k, $1 \leq k \leq \binom{|S|}{2}$ it outputs in $O(\log n)$ time a pair of points realizing distance which is the $(1-\varepsilon^2, 1+\varepsilon)$-approximation (or $(1-\varepsilon, 1+\varepsilon^2)$-approximation) of k^{th} distance. More precisely, we show how to construct a data structure in $O(n \log n/\varepsilon^2 + n \log^4 n/\gamma)$ time that dynamically maintains a set of n points in the plane in $O(\log^4 n/\gamma)$ time under insertions and deletions, for any arbitrary fixed $\gamma > 0$, such that given number k, $1 \leq k \leq \binom{|S|}{2}$ one

can compute in $O(\log n)$ time a pair of points realizing distance which is either $(1-\gamma, 1+\varepsilon)$ or $(1-\varepsilon, 1+\gamma)$-approximation of k^{th} distance. We also show how to obtain dynamic $(1-\varepsilon, 1+\varepsilon)$-approximation of k^{th} distance by simpler ALTAS-LOG algorithm of order $(2, 0)$ with slightly faster preprocessing time. It should be noted here that approximating the actual k-th distance within the factor $1+\varepsilon^2$ (or $1-\varepsilon^2$) is considerably harder than getting $1+\varepsilon$ (resp. $1-\varepsilon$) approximation with the same ε dependency in the running time of algorithm. We also generalize our algorithms to work in higher dimensions.

For our best knowledge, the dynamic problem of maintaining exact and approximate k^{th} distance is not studied in literature, except the famous closest pair problem (1^{st} distance selection) with optimal $O(\log n)$ worst-case update time [7] and diameter problem (farthest pair selection) with $O(n^{\varepsilon})$ worst-case update time [19], expected $O(\log n)$ update time [20] and $O(b \log n)$ update time [25] that maintains an approximate diameter (the approximation factor depends on the integer constant $b > 0$). One may find our algorithms useful in parametric searching applications, where a set of candidate solutions is defined by the distances between pairs of points of dynamic set S. For example, Agarwal and Procopiuc [3] (see also [2, 14, 29]) studied various k-center problems in R^d under L_∞ and L_2 metric: combinations of exact and approximate, continuous and discrete, uncapacitated and capacitated versions. Typically an algorithm performs a search (for example, binary search) on the sorted list of interdistances between data points. Our algorithms provide fast implementation of the search if an approximate solution suffices.

The main contribution of this paper is by developing efficient approximating dynamic algorithm for the well known distance selection problem using an approach that based on well separated pairs decomposition introduced by Callahan and Kosaraju [11] (see also [17]), computing approximate nearest and farthest neighbors [23, 26] and persistent binary search trees introduced by Driscoll et al. [18].

This paper is organized as follows. In the next section we briefly describe well-separated pair decomposition. Section 3 is dedicated to the approximating dynamic distance selection problem. Finally we conclude in Section 4.

2 Well-separated pair decomposition

In this section we shortly describe the well-separated pair decomposition proposed by Callahan and Kosaraju [11]. Let A and B be two sets of points in d-dimensional space ($d \geq 1$) of size n and m, respectively. Let s be some constant strictly greater than 0 and let $R(A)$ (resp. $R(B)$) be the smallest axis-parallel bounding box that encloses all the points of A (resp. B). We say that point sets A and B are *well-separated* with respect to s, if $R(A)$

and $R(B)$ can be each contained in d-dimensional ball of some radius r, such that the distance between these two balls is at least sr. One can easily show that for given two well-separated sets A and B, if $p_1, p_4 \in A$, $p_2, p_3 \in B$ then $dist(p_1, p_2) \leq (1 + \frac{2}{s})dist(p_1, p_3)$ and $dist(p_1, p_2) \leq (1 + \frac{4}{s})dist(p_4, p_3)$. (For general L_p metric the inequality may differ by some multiplicative constant.) Let S be a set of d-dimensional points, and let $s > 0$. A *well-separated pair decomposition* (WSPD) for S with respect to s is a set of pairs $\{(A_1, B_1), (A_2, B_2), \ldots, (A_p, B_p)\}$ such that:

(i) $A_i \subseteq S$ and $B_i \subseteq S$, for all $i = 1, \ldots, p$.

(ii) $A_i \cap B_i = \emptyset$, for all $i = 1, \ldots, p$.

(iii) A_i and B_i are well-separated with respect to s.

(iv) for any two distinct points r and q in S, there is exactly one pair (A_i, B_i) such that either $r \in A_i$ and $q \in B_i$ or $r \in B_i$ and $q \in A_i$.

The main idea of the algorithm for constructing WSPD is to build a binary fair split tree T whose leaves are points of S, with internal nodes corresponding to subsets of S. More precisely, split tree of S is a binary tree, constructed recursively as follows. If $|S| = 1$, its unique split tree consists of the node of S. Otherwise a split tree is any tree with root S and two subtrees that are split trees of the subsets formed by a split of S. For any node A in the tree, denote its parent (if exists) by $p(A)$. The outer rectangle of A, denoted by $R(A)$ is either an open d-cube centered at the center of the bounding box of S with the side size that equals to the largest side $l_{max}(S)$ of the bounding box of S (if A is root), or we have a situation when the splitting hyperplane used for the split of $p(A)$ divides $R(p(A))$ into two open rectangles. Let $R(A)$ be the one that contains A. A fair split of A is a split in which the splitting hyperplane is at distance of at least $l_{max}(A)/3$ from each of the two boundaries of $R(A)$ parallel to it. A split tree formed using only fair splits is called a fair split tree.

Each pair (A_i, B_i) in WSPD is represented by two nodes $v, u \in T$, such that all the leaves in the subtree rooted at v correspond to the points of A_i and all the leaves in the subtree rooted at u correspond to the points of B_i. The paper of Callahan and Kosaraju [11] presents an algorithm that implicitly constructs WSPD for a given set S and separation value $s > 0$ in $O(n \log n + s^d n)$ time such that the number of pairs (A_i, B_i) is $O(s^d n)$. Moreover, Callahan [10] showed how to compute a $WSPD$ in which at least one of the sets A_i, B_i of each pair (A_i, B_i) contains exactly one point of S. In this case, the number of pairs increases to $O(s^d n \log n)$.

3 Approximating k-th distance

3.1 Computing WSPD

Our algorithm consists of several stages. At the first stage we compute a WSPD for S with separation constant $s = \frac{6}{\varepsilon}$. From each (A_i, B_i) we take any pair $(a_i, b_i) \in (A_i, B_i), 1 \leq i \leq p$, $p = O(n)$. Our task now is to find the smallest index j in the sorted list of (a_i, b_i) pairs , such that the sum of cardinalities of all pairs (A_i, B_i) that correspond to prefix starting with $i = 1$ and ending at $i = j$ is at least k. Therefore, we sort the distances d'_i between a_i and b_i, $1 \leq i \leq p$. We assume that the pairs (A_i, B_i) are in order of increasing d'_i. Next, for each pair (A_i, B_i), $1 \leq i \leq p, p = O(n)$ we compute the $\alpha_i = |A_i||B_i|$ value, i.e. α_i is the total number of distinct pairs (a, b), $a \in A_i$, $b \in B_i$. Let

- $m_i = \min_{a \in A_i, b \in B_i} dist(a, b)$ and
- $M_i = \max_{a \in A_i, b \in B_i} dist(a, b)$

Let also $l_i, 1 \leq i \leq p$ be a number such that $(1 - \gamma)M_i \leq l_i \leq M_i$, for arbitrary fixed $\gamma > 0$.

As we said above, for a particular k we compute the smallest j such that $\sum_{i=1}^{j} \alpha_i \geq k$. Let $M' = \max_{i=1}^{j} M_i$ and let $l' = \max_{i=1}^{j} l_i$. We claim that l' is the $(1 - \gamma, 1 + \varepsilon)$-approximation of k-th distance.

In what follows we prove the correctness of our algorithm and show how to implement it efficiently.

Lemma 1 $(1 - \gamma)d_k \leq l' \leq (1 + \varepsilon)d_k$.

Proof. We observe that the total number of distances defined by pairs (A_i, B_i), $1 \leq i \leq j$ is at least k because $\Sigma_{i=1}^{j} \alpha_i \geq k$. Since M' is the maximum of these distances $M' \geq d_k$ follows. Thus, from $l' \geq (1 - \gamma)M'$ it follows that $l' \geq (1 - \gamma)d_k$. Our goal now it to prove that $M' \leq (1 + \varepsilon)d_k$. We recall that all possible pairs of points of S are uniquely represented by pairs (A_i, B_i) in WSPD. Consider the set of pairs $D = \{(a, b) | a \in A_i, b \in B_i, i \geq j\}$. There is an index r, $j \leq r \leq p$ such that m_r is the smallest distance defined by pairs of D. The total number of pairs in D is larger than $\binom{n}{2} - k$. Therefore, $d_k \geq m_r$. Let t, $1 \leq t \leq j$ be the index such that $M' = M_t$. From the observation in previous section it follows that $M_t \leq (1 + \frac{2}{s})d'_t = (1 + \varepsilon/3)d'_t$. Thus, $M' \leq (1 + \varepsilon/3)d'_j \leq (1 + \varepsilon/3)d'_r$, since the sequence d'_i, $j \leq i \leq p$ is non-decreasing. It follows that $(1 + \varepsilon/3)d'_r \leq (1 + \varepsilon/3)(1 + \varepsilon/3)m_r \leq (1 + \varepsilon)d_k$. So, $l' \leq M' \leq (1 + \varepsilon)d_k$. ∎

Remark 1. Using a similar approximation scheme with decreasing list of d'_i distances and by taking $M_i = \min_{a \in A_i, b \in B_i} dist(a,b)$ and l_i such that $(1+\gamma)M_i \geq l_i \geq M_i$ we can obtain $(1-\varepsilon, 1+\gamma)$-approximation of the k^{th} distance.

Remark 2. If, instead of computing l_i, we choose d'_j as the value returned by the algorithm, we obtain $(1-\varepsilon, 1+\varepsilon)$-approximation of the k^{th} distance. This is based on fact that $(1+\varepsilon)d'_j = \max_{1 \leq i \leq j}(1+\varepsilon)d'_i \geq \max_{1 \leq i \leq j} M_i = M' \geq d_k$.

It remains to show how to implement this algorithm efficiently, i.e. how to compute the values l_i, α_i, $1 \leq i \leq p$. First we show how to compute α_i. In other words we need to compute the cardinalities of A_i and B_i, $1 \leq i \leq p$. Recall that each pair (A_i, B_i) in WSPD is represented by two nodes v_i, u_i of the split tree T. The cardinality of $A_i(B_i)$ equals to the number of leaves in the subtree of T rooted at $v_i(u_i)$. Thus, by postorder traversal of T we are able to compute all the required cardinalities. Bespamyatnikh and Segal [9] showed how to compute the values m_i, M_i, $1 \leq i \leq p$ exactly using Voronoi diagrams [6] and Bentley's [5] logarithmic method. By assuming that the singleton set of each pair (A_i, B_i) in WSPD is $A_i = \{a_i\}$ they reduce the original problem of computing m_i and M_i values to the problem of computing for each a_i, $1 \leq i \leq p$ the nearest and the farthest neighbor in corresponding B_i. Since the computing of all Voronoi Diagrams may lead to undesired $O(n^2)$ runtime factor, they maintain dynamically Voronoi Diagrams while traversing a split tree T in a bottom-up fashion. Let S_v be a subset of S associated with a node v in T. By traversing a split tree T in a postorder fashion starting from leaves they use a partition R_v of S_v into disjoint sets $S_v^1, \ldots, S_v^q$ and maintain the Voronoi Diagram VD with corresponding point location data structure PL for each set S_v^j, $1 \leq j \leq q$ in R_v. The sizes of the sets in R_v are different and restricted to be the powers of two. As a consequence the number of such sets is at most $\log n$, i.e. $q \leq \log n$. It can be shown that the total time needed to spend for all described operations is $O(n \log^3 n)$.

3.2 Dynamic updates

The main drawback of the above scheme is the fact that during processing of T the Voronoi Diagram data structures are destroyed, so that at the end of the process we know only the Voronoi Diagram for the entire set S. Suppose that now we insert or delete some leaf from T. It may have influence on a number of other internal nodes. How we can determine now the new values of m_i and M_i? Basically, we have two major problems. The first one is how to store the Voronoi Diagram in each one of the internal nodes of T and the second one is how to update it quickly when T changes its structure by insertion of a new point or deletion

of an existing point from T.

Computing minimal and maximal values

In order to solve the first problem we will use fully persistent binary search tree described by Driscoll et al. [18]. A fully persistent structure supports any sequence of operations in which each operation can be applied to any previously existing version. The result of the update is an entirely new version, distinct from all others. Unfortunately we cannot represent a Voronoi Diagram as a collection of a sublinear number of binary search trees and therefore, we need to find a way of computing the values m_i and M_i using another strategy. In fact we are interested in computing the values l_i. Let us first consider the L_∞ metric. The points defining M_i should lie on the boundary of the smallest axis-parallel bounding box of set $A_i \cup B_i$. Recall that A_i and B_i are well separated and, thus, the L_∞ diameter of $A_i \cup B_i$ is defined by a pair (p, q) such that $p \in A_i$ and $q \in B_i$.

The computation of $m_i, 1 \leq i \leq p$ can be done similarly to the approach described in [8]. Suppose we use a WSPD with $p = O(n \log n)$ and assume $A_i = \{a_i\}$, $1 \leq i \leq p$. For each point a_i we need to find the closest neighbor in corresponding B_i. Consider, for example, the planar case. Let l_1 be a line whose slope is 45° passing through the a_i and l_2 be a line whose slope is 135° passing through the a_i. These lines define four wedges: $Q_{top}, Q_{bottom}, Q_{left}, Q_{right}$. For any point p lying in $Q_{left} \cup Q_{right} (Q_{bottom} \cup Q_{top})$ the L_∞-distance to a_i is defined by the x-distance (y-distance, resp.) to a_i. We perform four range queries, using orthogonal range tree [6] data structure (in coordinate system defined by lines l_1, l_2), each of them corresponding to the appropriate wedge. For each node in a secondary data structure we keep four values $x_{min}, x_{max}, y_{min}, y_{max}$ (computed in the initial coordinate system) of points in corresponding range. Consider, for example, the wedge Q_{right}. Our query corresponding to Q_{right} marks $O(\log^2 n)$ nodes. The minimum of x_{min} values stored in these nodes define the closest neighbor point to a_i lying in Q_{right}. We proceed similarly with the other wedges. We maintain orthogonal range tree data structures dynamically in a bottom-up fashion while traversing split tree T. In order to merge two data structures we simply insert all the points stored in the smaller range tree into the larger one. However, we are interested in the values of m_i computed for the Euclidean metric. We will use the following two results in order to accomplish our task. The first result has been proposed by Kapoor and Smid [26] that finds, for a given query point $p \in \mathbb{R}^d$ a $(1+\gamma)$-approximate L_2-neighbor of p in a given set of n points in $O(\log^{d-1} n/\gamma^{d-1})$ time using a data structure of $O(n \log^{d-2} n)$ space. They [26] store a set S in a constant number of a range trees, where each range tree stores the points according to its own coordinate system using the

construction of Yao [32]. Then, for a given p, they use all the range trees to compute L_∞ nearest neighbors of p in all coordinate systems. One of these L_∞ neighbors is $(1+\gamma)$-approximate L_2 nearest neighbor of p. But we still need to compute the values of M_i. The second result is due to Indyk [23] that shows how to compute $(1-\gamma)$-approximate farthest neighbor of a given point p by performing a constant number of $(1+\gamma)$-approximate nearest neighbor queries. The idea is to construct a set of a constant number of concentric disks (balls) around the origin. Each point is rounded to the nearest circle (sphere). For each disk (ball) we build a $(1+\gamma)$-approximate nearest neighbor data structure for the set of points on corresponding circle (sphere). Next, for each point $p \in S$ and each disk (ball) B_i, the "antipode" p^i of p with respect to B_i is defined as follows. Let p_1 and p_2 be the two points of the intersection of the circle (sphere) of B_i with the line passing through p and origin. Let h_p denote the hyperplane through the origin that is perpendicular to the line through p and origin. The point p^i is one of the points p_1, p_2 which lies on the side of h_p different from the side containing p. In order to find the farthest neighbor of q, we issue $(1+\gamma)$-approximate nearest neighbor query with the point q^i in the data structure for the points on each one of the circles (spheres). Among the points found, we return the one farthest from q. Preprocessing time is $O(d^{O(1)}n)$ plus the cost of initiating a constant number of data structures for $(1+\gamma)$-approximate nearest neighbor queries. The query time is bounded by the the query time for the $(1+\gamma)$-approximate nearest neighbor query.

Making it all persistent

The good thing in the described algorithms is the fact that all of them can be implemented using orthogonal range search trees, or in other words, binary search trees. This will allow us to make all of them fully persistent using Driscoll et al. [18] algorithm, thus solving our task of storing the appropriate data structure for each of the nodes of T without being destroyed. Generally speaking, ordinary data structures are ephemeral in the sense that making a change to the structure destroys the old version, leaving only the new one. In a fully persistent data structure, past versions of the data structure are remembered and can be queried and updated. In [18] a method termed *node copying with displaced storage of changes* was developed that could make red-black tree data structure to become fully persistent, in worst-case time per operation of $O(\log n)$ and worst-case space cost of $O(1)$ per insertion or deletion. Instead of indicating a change to an ephemeral node x by storing the change in the corresponding persistent node x', Driscoll et al. [18] stores information about the change in some possibly different node that lies on the access path to x' in the new version. Thus the record of the change is in general displaced from the node to which the change applies.

The path from the node containing the change information to the affected node is called the *displacement path*. By copying nodes judiciously, Driscoll et al. [18] were able to keep the displacement paths sufficiently disjoint to guarantee an $O(1)$ worst-case space bound per insertion or deletion while having $O(\log n)$ worst-case time bound per access, insertion or deletion.

While traversing a tree T, we maintain all the described data structures for computing $(1+\gamma)$-approximate nearest neighbor and $(1-\gamma)$-approximate farthest neighbor. We use again Bentley's [5] logarithmic method as described before. Notice, that each point in S can be inserted at most $O(\log n)$ times into the data structures while traversing T in a bottom-up fashion. Each insertion takes $O(\log^3 n)$ time. To give access to the persistent structure, the access pointers to the roots of the various versions must be stored in a balanced search tree, ordered by index. The total time for maintaining the range trees and computing l_i, $1 \leq i \leq p$ is $O(n \log^4 n)$, since $p = O(n \log n)$, each query takes $O(\log^2 n)$ time and each node contains a logarithmic number of the related data structures. The above computation can also be generalized to d-dimensional space, $d > 2$. Thus, we have

Theorem 2 *Given a set S of n points in $\mathbb{R}^d$, a number k, $1 \leq k \leq \binom{n}{2}$, $\varepsilon > 0$, $\gamma > 0$ a pair of points realizing $(1-\gamma, 1+\varepsilon)$ $((1-\varepsilon, 1+\gamma))$-approximation of d_k can be determined in $O(n \log n/\varepsilon^d + n \log^{d+2} n/\gamma^{d-1})$ time.*

Remark 3. Notice that we can obtain better running time (by logarithmic factor) using orthogonal range trees with the fractional cascading technique [16]. However, in order to allow persistence for the future dynamic updates we use orthogonal range trees avoiding this technique.

Remark 4. We can use a simpler strategy in order to compute the M_i values. We maintain the bounding boxes for sets of points corresponding to the nodes of T. The new bounding box can be computed in $O(1)$ time using the information from the previous steps. It results in a very fast algorithm with $(1-\frac{1}{\sqrt{2}}, 1+\varepsilon)$-approximation of k^{th} distance which can be made dynamic fairly easy.

Remark 5. The runtime of the algorithm presented in [9] and the approximation factor achieved by that algorithm is better than in Theorem 2 for $d = 2$. Moreover, we should note that there is a more efficient algorithm even for $d > 2$. Instead of using Kapoor and Smid data structure [26] for querying approximate nearest neighbor, we can use either Kleinberg [28] or Indyk and Motwani [24] or Kushilevitz et al. [30] or Chan's [15] data structures for the same purpose. For example, using the result by Chan [15] that gave an ALTAS-LOG algorithm of order $(\frac{d-1}{2}, \frac{d-1}{2})$ that achieves $(1+\varepsilon)$-approximation for nearest neighbor query instead of Kapoor and Smid [26] ALTAS-LOG algorithm of order $(d-1, d-1)$ we obtain a better

runtime of the entire algorithm. Unfortunately, the algorithm in [9] and also [15, 24, 28, 30] data structures cannot be made dynamic with a polylogarithmic update time. As we will see later, the result in Theorem 2 can be extended to deal with the dynamic point sets.

Following Remark 2 we also can conclude

Theorem 3 *Given a set S of n points in $\mathbb{R}^d$, a number k, $1 \leq k \leq \binom{n}{2}$, $\varepsilon > 0$, a pair of points realizing $(1-\varepsilon, 1+\varepsilon)$-approximation of d_k can be determined in $O(n \log n/\varepsilon^d)$ time.*

Dynamization

It remains to check what happens with the tree T when a new point is inserted or some existing point is deleted. By $\sigma(v)$, $v \in T$ we denote the subset of points associated with v at some instance in the sequence of updates. If v has two children w_1 and w_2 then $\sigma(v) = \sigma(w_1) \cup \sigma(w_2)$. If v is a leaf, then $|\sigma(v)| = 1$. Since the fair split property depends on the value of $l_{max}(\sigma(v))$, each time we insert a new point, this may increase the value of $l_{max}(\sigma(v))$ for all its ancestors in T, and the fair split property may be violated. Deletion of a point will not increase the value of $l_{max}(\sigma(v))$ for any of its ancestors, and hence can be performed on any fair split tree without restructuring. Callahan [10] shows that we can deal with the updates by maintaining a labeled binary tree T in which each node satisfies the following invariants:

1. For all internal nodes v with children w_1 and w_2, there is a fair cut that partitions $R(v)$ into two rectangles R_1 and R_2, such that $\sigma(w_1) = \sigma(v) \cap R_1$, $\sigma(w_2) = \sigma(v) \cap R_2$, $R(w_1)$ can be constructed from R_1 by applying a sequence of fair splits and $R(w_2)$ can be constructed from R_2 by applying a sequence of fair splits.

2. For all leaves v, $\sigma(v) = \{p\}$, and $R(v) = p$.

To insert a point p into this structure, we first retrieve the deepest internal node v in T such that $p \in R(v)$, ignoring the case in which p lies outside the rectangle at the root node. Let R_1 and w_1 have the same meaning as in the first invariant. Assume w.l.o.g. that $p \in R_1$. The way we chose v guarantees that $p \notin R(w_1)$. Now we introduce a new internal node u, which replaces w_1 as a child of v. We insert w_1 along with its subtree as a child of u, and insert a new leaf u' as the other child of u, where $\sigma(u') = \{p\}$. Finally we construct a rectangle $R(u)$ satisfying the first invariant. To delete the point p, we simply find the leaf v such that $\sigma(v) = \{p\}$, delete v, and compress the internal node $p(v)$. Callahan [10] proves that once we have determined where to insert a point p, we may perform such an insertion

in constant time, while preserving the invariants of the tree. Using the directed topology tree of Frederickson [21], Callahan has been able to maintain T in $O(\log n)$ time, where n is the current size of the point set. Generally speaking only $O(\log n)$ nodes of T can be affected during insertion or deletion of a point and therefore we can maintain the persistent structures associated with these nodes at sublinear cost.

Another problem that we have to deal with is the fact that introduction of a single new point can require the creation of many new pairs. Callahan [10] proposed an idea to predict all but a constant number of the new pairs ahead of time. The way to do it is to introduce *dummy points* where appropriate. Let $\bar{S}$ be a set of dummy points. Such points will not be counted in $\sigma(v)$ for any $v \in T$, but the tree T will have the same structure and rectangle labels as a fair split tree of $S \cup \bar{S}$. For efficiency we introduce only a constant number of dummy points for each well-separated pair $\{v, w\}$, such that $\sigma(v)$ and $\sigma(w)$ are not-empty. Since the number of new pairs is constant we can compute and maintain the relevant persistent structures efficiently.

Query

The only missing thing is how to perform a query, i.e. how, for a given value of k, we can find the approximate k^{th} distance? We maintain a balanced binary search tree T' for distances d'_i as defined before. Suppose that we build a binary tree T' with the leaves corresponding to $d'_1, \ldots, d'_p$. Each internal node $v \in T_r$ will keep three values: $\Sigma_{i=q_1}^{q_2} \alpha_i$, $\Sigma_{i=q_2+1}^{q_3} \alpha_i$, where $\alpha_{q_1}, \ldots, \alpha_{q_2}$ $(\alpha_{q_2+1}, \ldots, \alpha_{q_3})$ are the values that correspond to the leaves of the left subtree (resp. right subtree) of a tree rooted by v, and the third value $L_v = \max_{i=q_1}^{q_3} l_i$ (or $R_v = \min_{i=q_1}^{q_3} r_i$, $(1+\gamma)m_i \geq r_i \geq m_i$). Clearly, the construction of this tree T' with the augmented values can be computed in $O(p)$ time. We associate with each node $v \in T'$ an index j_v, such that d'_{j_v} corresponds to the rightmost leaf in the subtree rooted at v. Given a value k, we traverse T' starting from the root towards its children. We need to find a node u, with the smallest j_u such that $\Sigma_{i=1}^{j_u} \alpha_i \geq k$. It can be done in $O(\log n)$ time, by simply keeping the total number of nodes to the left of the current searching path. At each node where the path goes right, we collect the value $L_v(R_v)$ stored in the left subtree. At the end, we report the maximal (minimal) of the collected $L_v(R_v)$ values. If T' is implemented as a balanced binary search tree then the update of the values r_i and l_i can be done in logarithmic time. Moreover, while updating T new pairs may appear (and the previous pairs may disappear). Thus, we need to update the corresponding d'_i values in T' together with L_i, R_i, α_i values. The whole process can be accomplished in $O(\log^4 n)$ time since we have a logarithmic number of affected nodes in T, each query/update takes $O(\log^2 n)$ time

and each node contains at most logarithmic number of associated data structures.

Therefore we can conclude the following.

Theorem 4 *Given a set S of n points in $\mathbb{R}^d$, $\varepsilon > 0$, $\gamma > 0$ we can construct a data structure in time $O(n \log n/\varepsilon^d + n \log^{d+2} n/\gamma^{d-1})$ and $O(n \log n/\varepsilon^d)$ space with $O(\log^{d+2} n/\gamma^{d-1})$ update time for insertions/deletions of points such that given a number k, $1 \leq k \leq \binom{n}{2}$, a pair of points realizing $(1-\gamma, 1+\varepsilon)$ $((1-\varepsilon, 1+\gamma))$-approximation of d_k can be determined in $O(\log n)$ time.*

Theorem 5 *Given a set S of n points in $\mathbb{R}^d$, $\varepsilon > 0$, we can construct a data structure in time $O(n \log n/\varepsilon^d)$ and $O(n/\varepsilon^d)$ space such that given a number k, $1 \leq k \leq \binom{n}{2}$, a pair of points realizing $(1-\varepsilon, 1+\varepsilon)$-approximation of d_k can be determined in $O(\log n)$ time under insertions and deletions of points.*

4 Conclusions

We studied the dynamic problem for computing k-th Euclidean interdistance between n points in $\mathbb{R}^d$. The dynamization makes the problem more complicated. We are not aware of any other algorithms for exact or approximate solutions. We designed two efficient algorithms for maintaining a set of points and answering distance queries. The algorithms are based on the well-separated pair decomposition by Callahan and Kosaraju [11] and persistent data structures for approximate nearest/farthest neighbor. Both algorithms answer the queries in $O(\log n)$ time. The first algorithm provides $(1-\varepsilon, 1+\varepsilon)$ approximation and the second one provides a two-parameter approximation $(1-\varepsilon, 1+\gamma)$ (or $(1-\gamma, 1+\varepsilon)$). It would be intersting to reduce the dependence of runtime and space of our algorithms on ε and γ.

References

[1] P. Agarwal, B. Aronov, M. Sharir, S. Suri, "Selecting distances in the plane", *Algorithmica*, 9, pp. 495–514, 1993.

[2] P. Agarwal, M. Sharir, E. Welzl "The discrete 2-center problem", *Proc. 13th ACM Symp. on Computational Geometry*, pp. 147–155, 1997.

[3] P.K. Agarwal and C.M. Procopiuc, "Exact and Approximation Algorithms for Clustering", in *Proc. SODA'98*, pp. 658–667, 1998.

[4] S. Arya and D. Mount, "Approximate range searching", in *Proc.* 11^{th} *ACM Symp. on Comp. Geom.*, pp. 172–181, 1995.

[5] J. Bentley, "Decomposable searching problems", *Inform. Process. Lett.*, 8, pp. 244–251, 1979.

[6] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, "Computational Geometry: Algorithms and Applications", Springer-Verlag, 1997.

[7] S. Bespamyatnikh, "An Optimal Algorithm for Closest-Pair Maintenance", *Discrete Comput. Geom.*, 19, pp. 175–195, 1998.

[8] S. Bespamyatnikh, K. Kedem, M. Segal and A. Tamir "Optimal Facility Location under Various Distance Function", in *Workshop on Algorithms and Data Structures'99*, pp. 318–329, 1999.

[9] S. Bespamyatnikh and M. Segal "Fast algorithm for approximating distances", *Algorithmica*, 33(2), pp. 263–269, 2002.

[10] P. Callahan "Dealing with higher dimensions: the well-separated pair decomposition and its applications", Ph.D thesis, Johns Hopkins University, USA, 1995.

[11] P. Callahan and R. Kosaraju "A decomposition of multidimensional point sets with applications to k-nearest neighbors and n-body potential fields", *Journal of ACM*, 42(1), pp. 67–90, 1995.

[12] P. Callahan and R. Kosaraju "Faster Algorithms for Some Geometric Graph Problems in Higher Dimensions", in *Proc. SODA'93*, pp. 291–300, 1993.

[13] T. Chan "On enumerating and selecting distances", *International Journal of Computational Geometry and Applications*, 11, pp. 291–304, 2001.

[14] T. Chan "Semi-online maintenance of geometric optima and measures", in *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms*, pp. 474–483, 2002.

[15] T. Chan "Approximate nearest neighbor queries revised", *Discrete and Computational Geometry*, 20, pp. 359–373, 1998.

[16] B. Chazelle and L. Guibas, "Fractional Cascading: I. A data structuring technique, II. Applications", *Algorithmica*, 1, pp. 133–162, 163–192, 1986.

[17] S. Govindarajan, T. Lukovzski, A. Maheshwari and N. Zeh, "I/O Efficient Well-Separated Pair Decomposition and its Applications", In *Proc. of the 8th Annual European Symposium on Algorithms*, pp. 220–231 , 2000.

[18] J. Driscoll, N. Sarnak, D. Sleator and R. Tarjan "Making data structures persistent", *Journal of Computer and System Sciences*, 38, pp. 86–124, 1989.

[19] D. Eppstein, "Dynamic Euclidean minimum spanning trees and extrema of binary functions", *Discrete and Computational Geometry*, 13, pp. 111–122, 1995.

[20] D. Eppstein, "Average case analysis of dynamic geometric optimization", in *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pp. 77–86, 1994.

[21] G. Frederickson "A data structure for dynamically maintaining rooted trees", in *Proc. 4th Annu. Symp. on Disc. Alg.*, pp. 175–184, 1993.

[22] M. Goodrich, "Geometric partitioning made easier, even in parallel", *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pp. 73-82, 1993.

[23] P. Indyk, "High-dimensional computational geometry", Ph.D. thesis, Stanford University, pp. 68–70, 2000.

[24] P. Indyk and R. Motwani "Approximate nearest neighbors: towards removing the curse of dimensionality", in *Proc. 30th ACM Symp. Theory of Comput.*, 1998.

[25] R. Janardan, "On maintaining the width and diameter of a planar point-set online", *Int. J. Comput. Geom. Appls.*, 3, pp. 331-344, 1993.

[26] S. Kapoor and M. Smid, "New techniques for exact and approximate dynamic closest-point problems", *SIAM J. Comput.*, 25, pp. 775–796, 1996.

[27] M. Katz and M. Sharir, "An expander-based approach to geometric optimization", *SIAM J. Comput.*, 26(5), pp. 1384–1408, 1997.

[28] J. Kleinberg "Two algorithms for nearest-neighbor search in high dimensions", in *Proc. 29th ACM Symp. Theory of Comput.*, pp. 599-608, 1997.

[29] D. Krznaric "Progress in hierarchical clustering and minimum weight triangulation", *Ph. D. thesis*, Lund University, 1997.

[30] E. Kushelevitz, R. Ostrovsky and Y. Rabani "Efficient search for approximate nearest neighbor in high dimensional spaces", in *Proc. 30th ACM Symp. Theory of Comput.*, 1998.

[31] J. Salowe, "L_∞ interdistance selection by parametric searching", *Inf. Process. Lett.*, 30, pp. 9–14, 1989.

[32] A. C. Yao "On constructing minimum spanning trees in k-dimensional spaces and related problems", in *SIAM Journal on Computing*, 11, pp. 721–736, 1982.