Kinetic Data Structures for the Semi-Yao Graph and All Nearest Neighbors in $\mathbb{R}^d$

Zahed Rahmati* Mohammad Ali Abam† Valerie King* Sue Whitesides*

Abstract

This paper presents kinetic data structures (KDS's) for maintaining the Semi-Yao graph, all the nearest neighbors, and all the $(1+\epsilon)$-nearest neighbors of a set of moving points in $\mathbb{R}^d$.

Our technique provides the first KDS for the Semi-Yao graph in $\mathbb{R}^d$. It generalizes and improves on the previous work on maintaining the Semi-Yao graph in $\mathbb{R}^2$.

Our KDS for all nearest neighbors is *deterministic*. The best previous KDS for all nearest neighbors in $\mathbb{R}^d$ is *randomized*. Our structure and analysis are simpler and improves on the previous work.

Finally, we provide a KDS for all the $(1+\epsilon)$-nearest neighbors, which in fact gives better performance than the exact KDS's for all nearest neighbors.

1 Introduction

Let P be a set of n points in R^d. Finding the nearest neighbor to a query point, which was called the *post office problem* by Donald Knuth in 1973, is fundamental in computational geometry. The *all nearest neighbors* problem, a variant of the post office problem, is to find the nearest neighbor to each point $p \in P$. Given any $\epsilon > 0$, the *all* $(1+\epsilon)$*-nearest neighbors* problem is to find some $\hat{q} \in P$ for each point $p \in P$, such that the Euclidean distance $|p\hat{q}|$ between p and $\hat{q}$ is within a factor of $(1+\epsilon)$ of the Euclidean distance between p and its nearest neighbor. The *nearest neighbor graph* is constructed by connecting each point $p \in P$ to its nearest neighbor. The *closest pair* problem is to find the endpoints of the edge in the nearest neighbor graph with minimum length. The *Semi-Yao graph* (or *theta-graph*) is a well-studied sparse proximity graph [12, 15]. This graph can be constructed by partitioning the space around each point $p \in P$ into c cones $C_l(p)$, $1 \le l \le c$, with p the shared apex of the cones, and then connecting the point p to a point q inside each of these cones, such that the point $q \in P \cap C_l(p)$ has the minimum length projection on the vector in the direction of the axis of $C_l(p)$. By treating the number c of cones at each point as a parameter of the Semi-Yao graph, one obtains an important class of sparse graphs, *t-spanners*, with different stretch factors t [6, 9, 10, 11].

The maintenance of attributes (e.g., the closest pair) of sets of moving points has been studied extensively over the past 15 years [2, 3, 5, 7, 14, 18, 19, 20, 21]. A basic framework for this study is that of a kinetic data structure (KDS), which is in fact a set of data structures and algorithms to track the attributes of moving points. The problem of maintaining all the nearest neighbors, the closest pair, and the Semi-Yao graph on moving points are called the *kinetic all nearest neighbors* problem, the *kinetic closest pair* problem, and the *kinetic Semi-Yao graph*, respectively.

The kinetic maintenance of attributes is generally considered in two models: the *standard KDS model* [7] and the *black-box KDS model* [13]. In the black-box model, the locations of the objects are received at regular time steps. In the standard model, each object has a flight plan known in advance, and during the motion the object can change its flight plan at some times which are not known in advance. This paper considers the all nearest neighbors problem and the Semi-Yao graph in $\mathbb{R}^d$ for the standard KDS model and improves previous results. In addition, it offers results on all $(1+\epsilon)$-nearest neighbors in $\mathbb{R}^d$.

Standard KDS framework. Basch, Guibas, and Hershberger [7] introduced a *kinetic data structure framework* to maintain the attributes of moving points. The d coordinates of the trajectory of each point p in $\mathbb{R}^d$, which determine the position of p as a function of time, are d algebraic functions of at most constant degree s. The correctness of an attribute over time is determined based on correctness of a set of *certificates*. A certificate is a boolean function of time, and its *failure time* is the next time after the current time at which the certificate will become invalid. When a certificate fails, we say that an *event* occurs. Using a *priority queue* of the failure times of the certificates, we can know the next time after the current time that an event occurs. When the failure time of the certificate with highest priority in the priority queue is equal to the current time we invoke the update mechanism to reorganize the data structures and replace the invalid certificates with new valid ones.

*This work was partially supported by a British Columbia Graduate Student Fellowship and by NSERC discovery grants.
Department of Computer Science, University of Victoria, Canada, {rahmati,val,sue}@uvic.ca

†Department of Computer Engineering, Sharif University of Technology, Iran, abam@sharif.edu

To analyse the performance of a KDS there are four standard criteria. A KDS distinguishes between two types of events: *external events* and *internal events*. An event that changes the desired attribute itself is called an external event, and those events that cause only some internal changes in the data structures are called internal events. If the ratio between the total number of events and the number of external events is $O(\text{polylog}(n))$, the KDS is *efficient*. If the response time of the update mechanism to an event is $O(\text{polylog}(n))$, the KDS is *responsive*. The compactness of a KDS refers to size of the priority queue: if the KDS uses $O(n.\text{polylog}(n))$ certificates, it is *compact*. The KDS is *local* if the number of events associated with any point in the KDS is $O(\text{polylog}(n))$. The locality of a KDS is an important criterion; if a KDS satisfies locality, the KDS can be updated quickly when a point changes its trajectory.

Related work. Basch, Guibas, and Hershberger (SODA'97) [7] provided a KDS for maintenance of the closest pair in $\mathbb{R}^2$. Their KDS uses linear space and processes $O(n^2\beta_{2s+2}(n)\log n)$ events, each in time $O(\log^2 n)$. Here, $\beta_s(n) = \frac{\lambda_s(n)}{n}$ is an extremely slow-growing function and $\lambda_s(n)$ is the maximum length of Davenport-Schinzel sequences of order s on n symbols.

A common way to maintain attributes of moving points in $\mathbb{R}^d$ is to use *kinetic multidimensional range trees* [8, 4, 1]. Basch *et al.* [8] and Agarwal *et al.* [4] use dynamic balanced trees to implement a kinetic range tree. Using rebalancing operations, they handle events to maintain a range tree. In particular, in their approaches, when an event between two points p and q occurs, we must delete p and q and reinsert them into the range tree. The range tree can be maintained over time using a dynamic range tree. One of the approaches to update the range trees is to carry out local and global rebuilding after a few operations, which gives $O(\log^d n)$ amortized time per operation [16]. Another approach, which uses merge and split operations, gives worst-case time $O(\log^d n)$ per operation [23]. To avoid rebalancing the range tree after each operation, Abam and de Berg [1] introduced a variant of the range trees, a *rank-based range tree* (RBRT), which gives worst-case time $O(\log^d n)$ per operation.

Basch, Guibas, and Zhang (SoCG'97) [8] used multidimensional range trees to maintain the closest pair. For a fixed dimension d, their KDS uses $O(n\log^{d-1} n)$ space and processes $O(n^2\beta_{2s+2}(n)\log n)$ events, each in worst-case time $O(\log^d n)$. Their KDS is responsive, efficient, compact, and local.

Using multidimensional range trees, Agarwal, Kaplan, and Sharir (TALG'08) [4] gave KDS's for both maintenance of the closest pair and all the nearest neighbors in $\mathbb{R}^d$. The closest pair KDS by Agarwal *et al.*, which supports insertions and deletions of points, uses $O(n\log^{d-1} n)$ space and processes $O(n^2\beta_{2s+2}(n)\log n)$ events, each in amortized time $O(\log^d n)$; this KDS is efficient, responsive (in an amortized sense), local, and compact. Agarwal *et al.* gave the first efficient KDS to maintain all the nearest neighbors in $\mathbb{R}^d$. For the efficiency of their KDS, they implemented range trees by using randomized search trees (treaps). Their *randomized* kinetic approach uses $O(n\log^d n)$ space and processes $O(n^2\beta^2_{2s+2}(n)\log^{d+1} n)$ events; the expected time to process all events is $O(n^2\beta^2_{2s+2}(n)\log^{d+2} n)$. Their all nearest neighbors KDS is efficient, responsive (in an amortized sense), compact, but in general is not local.

Rahmati, King, and Whitesides (SoCG'13) [17] gave the first KDS for maintenance of the Semi-Yao graph in $\mathbb{R}^2$. They use a constant number of kinetic Delaunay triangulations to maintain the Semi-Yao graph. Their Semi-Yao graph KDS uses linear space and processes $O(n^2\beta_{2s+2}(n))$ events with total processing time $O(n^2\beta_{2s+2}(n)\log n)$. Using the kinetic Semi-Yao graph, they improved the previous KDS by Agarwal *et al.* to maintain all the nearest neighbors in $\mathbb{R}^2$. In particular, their *deterministic* kinetic algorithm, which is also arguably simpler than the randomized kinetic algorithm by Agarwal *et al.*, uses $O(n)$ space and processes $O(n^2\beta^2_{2s+2}(n)\log n)$ events with total processing time $O(n^2\beta^2_{2s+2}(n)\log^2 n)$. With the same complexity as their KDS for maintenance of all the nearest neighbors, they maintain the closest pair over time. Their KDS's for maintenance of the Semi-Yao graph, all the nearest neighbors, and the closest pair are efficient, responsive (in an amortized sense), compact, but in general are not local.

Our technique, results, and improvements. For a set of n moving points in fixed dimension d, where the trajectory of each point is an algebraic function of at most constant degree s, we provide a simple, deterministic KDS for maintenance of all the nearest neighbors. Our kinetic approach is based on maintaining the edges of the Semi-Yao graph, a sparse graph whose edge set includes the pairs of nearest neighbors as a subset. We use a constant number of range trees to apply necessary changes to the Semi-Yao graph over time.

Our Semi-Yao graph KDS uses $O(n\log^d n)$ space and processes $O(n^2)$ events with total processing time $O(n^2\beta_{2s+2}(n)\log^{d+1} n)$. The KDS is compact, efficient, responsive (in an amortized sense), and it is local. Our KDS generalizes the previous KDS for the Semi-Yao graph by Rahmati *et al.* [17] that only works in $\mathbb{R}^2$. Also, our kinetic approach yields improvements of the KDS for maintenance of the Semi-Yao graph by Rahmati *et al.* [17]: Our KDS is local, but their KDS is not. In particular, each point in our KDS participates in $O(1)$ events, but in their KDS each point partici-

pates in $O(n)$ events. Our KDS handles $O(n^2)$ events, but their KDS handles $O(n^2\beta_{2s+2}(n))$ events in $\mathbb{R}^2$.

Our KDS for maintenance of all the nearest neighbors uses $O(n\log^d n)$ space and processes $O(n^2\beta^2_{2s+2}(n)\log n))$ events; the total processing time to handle all the events is $O(n^2\beta_{2s+2}(n)\log^{d+1} n)$. Our KDS is compact, efficient, responsive (in an amortized sense), but it is not local in general. For each point $p \in P$ in the Semi-Yao graph we construct a tournament tree to maintain the edge with minimum length among the edges incident to the point p. Summing over elements of all the tournament trees in our KDS is linear in n, which leads to a total number of events $O(n^2\beta^2_{2s+2}(n)\log n)$, which is *independent* of d. This improves with simpler structure and analysis the previous *randomized* kinetic algorithm by Agarwal *et al.* [4]: The expected total size of the tournament trees in their KDS is $O(n\log^d n)$; thus their KDS processes $O(n^2\beta^2_{2s+2}(n)\log^{d+1} n)$ events, which depends on d. Also, we improve their KDS by a factor of $\log n$ in the total cost. Furthermore, on average, each point in our KDS participates in $O(1)$ events, but in their KDS each point participates in $O(\log^d n)$ events.

For maintaining all the nearest neighbors, neither our KDS nor the KDS by Agarwal *et al.* is local in the *worst-case*, and furthermore, each event in our KDS and in their KDS is handled in a polylogarithmic *amortized* time. To satisfy the locality criterion and to get a worst-case processing time for handling events, we provide a KDS for all the $(1+\epsilon)$-nearest neighbors. In particular, for each point p we maintain some point $\hat{q}$ such that $|p\hat{q}| < (1+\epsilon).|pq|$, where q is the nearest neighbor of p and $|pq|$ is the Euclidean distance between p and q. This KDS uses $O(n\log^d n)$ space, and handles $O(n^2\log^d n)$ events, each in worst-case time $O(\log^d n \log\log n)$; it is compact, efficient, responsive, and local.

Paper organization. Section 2 describes the construction of the Semi-Yao graph and gives a solution to the all nearest neighbors problem in $\mathbb{R}^d$. In Section 3, we show how the Semi-Yao graph can be maintained. Using the kinetic Semi-Yao graph, we give a KDS for maintenance of all the nearest neighbors in Section 4. Section 5 shows how to maintain all the $(1+\epsilon)$-nearest neighbors.

2 The Construction

In the following we describe the construction of the Semi-Yao graph and construction of all the nearest neighbors, which will aid in understanding how our kinetic approach works.

Let $\overrightarrow{v}$ be a unit vector in $\mathbb{R}^d$ with apex at the origin o, and let θ be a constant. We define the *infinite right circular cone* with respect to $\overrightarrow{v}$ and θ to be the set of points $x \in \mathbb{R}^d$ such that the angle between $\overrightarrow{ox}$ and $\overrightarrow{v}$ is at most $\theta/2$. A *polyhedral cone* inscribed in this infinite right circular cone is formed by intersection of d distinct half-spaces such that all the half-spaces contain o. The angle between any two rays inside the polyhedral cone emanating from o is at most θ. The d-dimensional space around o can be covered by a collection of disjoint polyhedral cones $C_1, ..., C_c$, where $c = O(1/\theta^{d-1})$ [4, 1]. Denote by x_l the vector in the direction of the unit vector $\overrightarrow{v}$ of C_l, $1 \le l \le c$, with the origin at o. Let $C_l(p)$ denote a translated copy of C_l with apex at p (see Figure 1(a)). From now on, we assume d is arbitrary but fixed, so c is constant.

Given a point set P in $\mathbb{R}^d$, the Semi-Yao graph is constructed by connecting each point $p \in P$ to the point in $P \cap C_l(p)$, $1 \le l \le c$, whose x_l-coordinate is minimum. Figure 1(b) depicts some edges incident to the point p in the Semi-Yao graph in $\mathbb{R}^2$ where $\theta = \pi/3$; here $x_1 = -x_4$, $x_2 = -x_5$, and $x_3 = -x_6$.

The following lemma is used in [7, 4, 17] to maintain the closest pair and all the nearest neighbors for a set P of moving points (see Figure 1(c)).

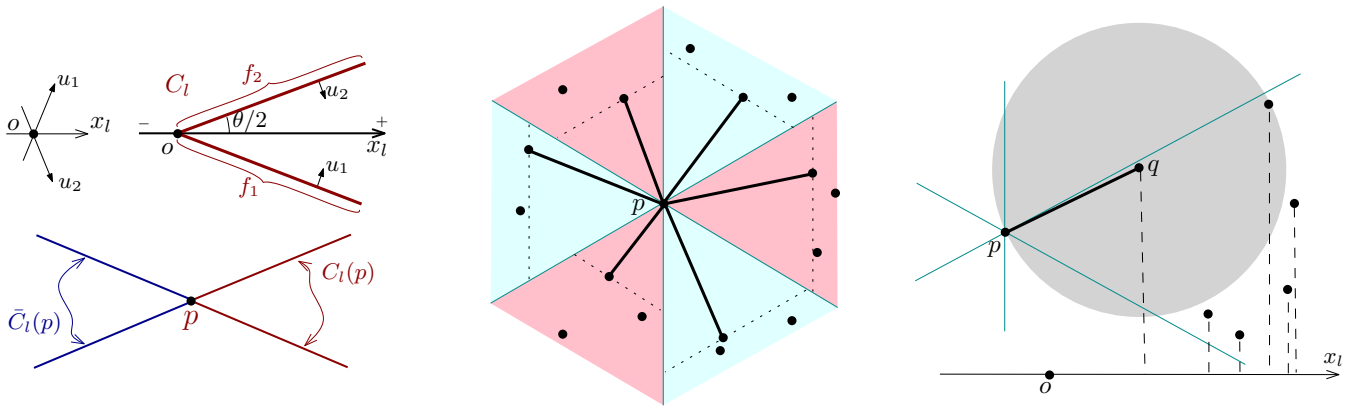
Lemma 1 (`Lemma 8.1. of [4]`) *Let p be the nearest point to q and let $C_l(p)$ be a cone of opening angle $\theta \le \pi/3$ that contains q. Then q has the minimum x_l-coordinate among the points in $P \cap C_l(p)$.*

For a set of points in the plane, Rahmati *et al.* [17] used Lemma 1 to show that the Semi-Yao graph is a super-graph of the nearest neighbor graph. The following lemma gives the same result for a set of points in higher dimensions.

Lemma 2 *The Semi-Yao graph of a set of points in $\mathbb{R}^d$ is a super-graph of the nearest neighbor graph.*

Proof. Let (p, q) be an edge in the nearest neighbor graph such that p is the nearest neighbor to q. The point q is in some cone $C_l(p)$. The restriction $\theta \le \pi/3$ of the cone $C_l(p)$ together with Lemma 1 imply that the point q has minimum length projection on x_l among the points in $P \cap C_l(p)$; this implies that (p, q) is an edge of the Semi-Yao graph. □

For a fixed dimension d, there is a constant number of cones C_l. Denote by $f_1, ..., f_d$ the bounding half-spaces of the cone C_l and let u_i be the coordinate axis orthogonal to f_i, $1 \le i \le d$; Figure 1(a) depicts u_1 and u_2 of the half-spaces f_1 and f_2 of the cone C_l. Corresponding to each cone C_l, we construct a ranked-based range tree (RBRT) T_l [1], which is described below, and for each point p in $\mathcal{T}_l$, we find the point in $P \cap C_l(p)$ whose x_l-coordinate is minimum; this gives a construction for the Semi-Yao graph.



(a) (b) (c)

Figure 1: (a) The cone C_l and its translated copy with apex at p. (b) The point p is connected to the point in $P \cap C_l(p)$ that has minimum x_l-coordinate. The dotted lines are orthogonal to the cone axes. (c) The point p is the nearest neighbor to q and, so q has the minimum x_l-coordinate among the points in $P \cap C_l(p)$.

Ranked-based range tree. The RBRT T_l is a variant of the range trees and has the following property. When two points exchange their order along an axis u_i, the RBRT can be updated without rebalancing the subtrees. The points at the level i of the RBRT T_l are sorted at the leaves in ascending order according to their u_i-coordinates. The skeleton of an RBRT is independent of the position of the points in $\mathbb{R}^d$ and depends on the ranks of the points in each of the u_i-coordinates. The *rank* of a point in a tree at level i of the RBRT is its position in the sorted list of all the points ordered by their u_i-coordinates. Any tree at any level of the RBRT is a balanced binary tree, and no matter how many points are in the tree, it is a tree on n ranks [1].

Let v be an internal node at level d of T_l. Denote by $R(v)$ the set of points at the leaves of the subtree rooted at v. The set $P \cap C_l(p)$ is the union of $O(\log^d n)$ sets $R(.)$; all these $O(\log^d n)$ sets can be reported in time $O(\log^d n + k)$, where k is the cardinality of $P \cap C_l(p)$.

For each node v at level d of T_l we define another set $B(v)$. Denote by $\mathcal{P}_p$ the path from the parent of p to the root of a tree at level d of T_l. A point p belongs to $B(v)$ if v is the right child of some node $\bar{v} \in \mathcal{P}_p$; a point p is in $B(v)$ if $R(v)$ is one of the $O(\log^d n)$ sets encountered while reporting the points of $P \cap C_l(p)$.

Let $\bar{C}_l(p) = -C_l(p)$ be the reflection of $C_l(p)$ through p; $\bar{C}_l(p)$ is formed by following the lines through p in the half-spaces of $C_l(p)$; see Figure 1(a). Similar to the way that we report the points of P inside a query cone $C_l(q)$, we can also report the points of P inside a query cone $\bar{C}_l(q)$. The set $P \cap \bar{C}_l(q) = \cup_v B(v)$, where the nodes v are on the paths $\mathcal{P}_q$.

The set of all the pairs $(B(v), R(v))$, for all the internal nodes v at level d of $\mathcal{T}_l$, is called a *cone separated pair decomposition* (CSPD) for P with respect to C_l; denote this set by $\Psi_{C_l} = \{(B_1, R_1), ..., (B_m, R_m)\}$. The CSPD Ψ_{C_l} has the following properties [1]:

- For the two points $p \in P$ and $q \in P$, where $q \in C_l(p)$, there exists a unique pair $(B_i, R_i) \in \Psi_{C_l}$ such that $p \in B_i$ and $q \in R_i$.
- For the pair $(B_i, R_i) \in \Psi_{C_l}$, if $p \in B_i$ and $q \in R_i$, then $q \in C_l(p)$ and $p \in \bar{C}_l(q)$.

Reporting all the nearest neighbors. Let $r(v)$ be the point with minimum x_l-coordinate among the points in $R(v)$. Denote by $lc(v)$ and $rc(v)$ the left and the right child of the node v, respectively. For each node v, the value of $r(v)$ is generated from the values of it children, $r(lc(v))$ and $r(rc(v))$, namely from the one which stores the point with minimum x_l-coordinate. Thus for all internal nodes v at level d of the RBRT T_l, we can find all the $r(v)$ in $O(n \log^{d-1} n)$ time. Since for each point $p \in P$ the point with minimum x_l-coordinate in $P \cap C_l(p)$ is chosen among $O(\log^d n)$ points $r(.)$, the following lemma results.

Lemma 3 *The Semi-Yao graph of a set of n points in $\mathbb{R}^d$ can be constructed in time $O(n \log^d n)$.*

Vaidya [22] gave an $O(n \log n)$ time algorithm to solve the all nearest neighbors problem. Given the Semi-Yao graph in $\mathbb{R}^d$, by examining the edges incident to any point, we can find the nearest neighbor to the point. Since the Semi-Yao graph has $O(n)$ edges, we obtain the following.

Lemma 4 *Given the Semi-Yao graph, the all nearest neighbors problem in $\mathbb{R}^d$ can be solved in $O(n)$ time.*

3 Kinetic Semi-Yao Graphs

Fix a cone C_l and the corresponding coordinate axes $u_1, ..., u_d$, and x_l. When the points are moving, the Semi-Yao graph remains unchanged as long as the order of the points in each of the coordinates $u_1, ..., u_d$, and x_l remains unchanged. To maintain the Semi-Yao graph over time, we distinguish between two types of events: (i) *u-swap event:* Such an event occurs if two points exchange their order in the u_i-coordinate. This event can change the structure of the range tree. (ii) *x-swap event:* This event occurs if two points exchange their x_l-order. The range tree structure remains unchanged when this event occurs.

To track the above changes, we maintain sorted lists $L(u_1), ..., L(u_d)$, and $L(x_l)$ of the points in each of the coordinates $u_1, ..., u_d$, and x_l, respectively. For each two consecutive points in each sorted list $L(u_i)$ we define a certificate that certifies the order of the two points in the u_i-coordinate. To track the closest time to the current time we put the failure times of all the certificates in a priority queue; the element with the highest priority in the queue gives the closest time.

Our Semi-Yao graph KDS is based on maintenance of the RBRT $\mathcal{T}_l$. Abam and de Berg describe how to maintain an RBRT. Their approach uses $O(n \log^d n)$ space, and a u-swap event can be handled in time $O(\log^d n)$ without rebalancing the subtrees of the RBRT [1].

For the point $w \in P$, the set $P \cap C_l(w) = \bigcup_{j=1}^{j=k} R(v_j)$, where the nodes v_j are the right child nodes of the nodes on the paths $\mathcal{P}_w$. Denote by $\ddot{w}_l$ the point in $P \cap C_l(w)$ with minimum x_l-coordinate. To maintain the Semi-Yao graph, for each point $w \in P$ we must track $\ddot{w}_l$, which in fact is the point in $\{r(v_1), ..., r(v_k)\}$ whose x_l-coordinate is minimum. To apply changes to the $\ddot{w}_l$, for all $w \in P$, in addition $r(v)$, we need to maintain more information at each internal node v at level d of the RBRT $\mathcal{T}_l$. We describe the extra information in the next paragraph.

Allocate a *label* to each point in P. Let $B'(v) = \{(w, \ddot{w}_l) |\ w \in B(v)\}$ and let $L(B'(v))$ be a sorted list of the pairs of $B'(v)$ according to the labels of the second components $\ddot{w}$ of the pairs $(w, \ddot{w}_l)$. This sorted list is used to answer the following query while processing x-swap events: Given a query point p, find all the points $w \in B(v)$ such that $\ddot{w}_l = p$. Since we perform insertions/deletions into the sorted lists $L(B'(.))$ over time, we implement them using a dynamic binary search tree (*e.g.*, a *red-black tree*); each insertion/deletion operation is performed in worst-case time $O(\log n)$. Furthermore, we maintain a set of links between each point $w \in P$ and the pair $(w, \ddot{w}_l)$ in the sorted lists $L(B'(.))$ where $w \in B(.)$; denote this set by $Link(w)$. Since the point w is in at most $O(\log^d n)$ sets $B(.)$, the cardinality of the set $Link(w)$ is $O(\log^d n)$.

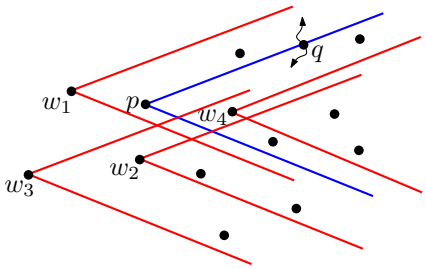


Figure 2: A u-swap between p and q does not change the points in other cones $C_l(w_i)$.

In the preprocessing step before the motion, for any internal node v at level d of the RBRT and for any point $w \in P$, we find $r(v)$ and $\ddot{w}_l$, and then we construct $L(B'(v))$ and $Link(w)$.

Handling the events. Now let the points move. The following shows how to maintain and reorganize $Link(.)$, $L(B'(.))$ and $r(.)$ when a u-swap event or an x-swap event occurs. Note that maintenance of the sets $Link(w)$, for all $w \in P$, gives a kinetic maintenance of the Semi-Yao graph.

u-swap event. When two points p and q exchange their order in the u_i-coordinate, we swap them in the sorted list $L(u_i)$ and update the invalid certificates with new valid ones; applying $O(1)$ changes to the priority queue takes $O(\log n)$ time. Then we delete p and q and reinsert them into the RBRT with their new ranks [1]. Next we update the values $r(v)$ where the nodes v are ancestors of p and q. A change to some $r(v)$ can only change $r(v_{par})$, where v_{par} is the parent of v. These updates can easily be done in $O(\log^d n)$ time.

Let $q \in C_l(p)$ (resp. $q \notin C_l(p)$) before the event. After the event, q moves outside (resp. inside) the cone $C_l(p)$; see Figure 2. Note that this event does not change the points in cones $C_l(w)$ of other points $w \in P$. Therefore, the only change that can happen to the Semi-Yao graph is deletion of an edge incident to p inside the cone $C_l(p)$ and addition of a new one.

We perform the following steps when such an event occurs. We first delete the pairs $(p, \ddot{p}_l)$ of the sorted lists $L(B'(.))$ where $p \in B(.)$; by using the links in $Link(p)$, this can be done in time $O(\log^{d+1})$. Then we delete the members of $Link(p)$. Next we find the point $\ddot{p}_l$ in $P \cap C_l(p)$ whose x_l-coordinate is minimum. Recall that v_j, $j = 1, ..., O(\log^d n)$, are the right child nodes of the nodes on the paths $\mathcal{P}_p$. Since we might get a new value for $\ddot{p}_l$ among all the $r(v_j)$, we must add the new pair $(p, \ddot{p}_l)$, according to the label of the new value of $\ddot{p}_l$, into all the sorted lists $L(B'(v_j))$ where $p \in B(v_j)$. Finally we construct $Link(p)$ of the new links between p and

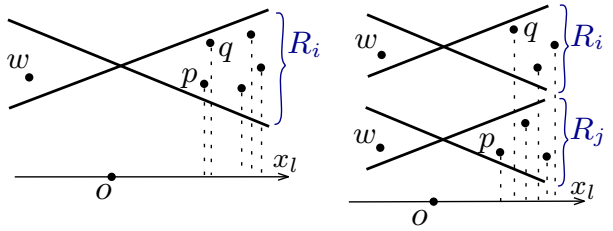


Figure 3: Two cases when an x-swap between p and q occurs.

the pair $(p, \ddot{p}_l)$ of the sorted lists $L(B'(v_j))$, which takes $O(\log^{d+1} n)$ time.

Since the number of swaps between the points in $L(u_i)$, $1 \leq i \leq d$, is $O(n^2)$, the following results.

Lemma 5 *For maintenance of the Semi-Yao graph, our KDS handles $O(n^2)$ u-swap events, each in worst-case time $O(\log^{d+1} n)$.*

x-swap event. Let p and q be two consecutive points with p preceding q in the sorted list $L(x_l)$: $x_l(p) < x_l(q)$ before the event. When p and q exchange their order, we swap them in $L(x_l)$ and update the invalid certificates with new valid ones, which takes $O(\log n)$ time. This event does not change the structure of T_l; but it might change the second components of the pairs in some sorted lists $L(B'(.))$ and if so, we must apply the necessary changes to the Semi-Yao graph.

The number of all changes to the Semi-Yao graph depends on how many points $w \in P$ have both p and q in their cones $C_l(w)$. While reporting the points in $P \cap C_l(w)$, note that w can have both p and q in the same set R_i (see Figure 3(a)) or in two different sets R_i and R_j (Figure 3(b)). To find such points w, (i) we seek internal nodes v_{pq} at level d of $\mathcal{T}_l$ where $\{p, q\} \subseteq R(v_{pq})$, which means that the nodes v_{pq} are common ancestors of p and q, (ii) we look for internal nodes v_p and v_q where $p \in R(v_p)$ and $q \in R(v_q)$. In the first case, it is obvious that we must find any point $w \in B(v_{pq})$ such that p is the point with minimum x_l-coordinate in the cone $C_l(w)$, meaning that $\ddot{w}_l = p$. Then we replace p by q ($\ddot{w}_l = q$), which means we replace the edge wp of the Semi-Yao graph with wq.

Note that in the second case there is no point $w' \in B(v_p)$ such that $\ddot{w'}_l = q$, because $x_l(p) < x_l(q)$. Also note that if there is a point $w'' \in B(v_p)$ such that $\ddot{w''}_l = p$, we change the value of $\ddot{w''}_l$ to q if $q \in C_l(w'')$. Since we can find w'' in $B(v_q)$, we do not need to check whether such points w'' are in $B(v_p)$ or not. Therefore, for the second case, we only need to check whether there is a point $w \in B(v_q)$ such that $\ddot{w}_l = p$; if so, we change the value of $\ddot{w}_l$ to q ($\ddot{w}_l = q$).

From the above discussion, the following steps summarize the update mechanism of our KDS to maintain the Semi-Yao graph when an x-swap event occurs.

1. Find all the internal nodes v at level d of T_l such that $\{p, q\} \subseteq R(v)$ and $r(v) = p$ (see Figure 3(a)). Also, find all the internal nodes v where $r(v) = q$ (see Figure 3(b)).

2. For each node v (from Step 1), find all the pairs $(w, \ddot{w}_l)$ in the sorted list $L(B'(v))$ where $\ddot{w}_l = p$.

3. For each w (from Step 2), using the links in $Link(w)$, find all the corresponding sorted lists $L(B'(.))$, delete the pair $(w, \ddot{w}_l)$ from them, change the value of the second component $\ddot{w}_l$ to q, and add $(w, \ddot{w}_l)$ into the sorted lists according to the label of q.

The number of edges incident to a point p in the Semi-Yao graph is $O(n)$. Thus when an x-swap event between p and some point q occurs, it might cause $O(n)$ changes to the Semi-Yao graph. The following shows that an x-swap event can be handled in polylogarithmic amortized time.

Lemma 6 *For maintenance of the Semi-Yao graph, our KDS handles $O(n^2)$ x-swap events with total processing time $O(n^2 \beta_{2s+2}(n) \log^{d+1} n)$.*

Proof. All the internal nodes v at Step 1 can be found in $O(\log^d n)$ time.

For each internal node v of Step 2, the update mechanism spends $O(\log n + k_v)$ time where k_v is the number of all the pairs $(w, \ddot{w}_l) \in B'(v)$ such that $\ddot{w}_l = p$. For all the internal nodes v, the second step takes $O(\log^{d+1} n + \sum_v k_v)$ time. Note that $\sum_v k_v$ is equal to the number of exact changes to the Semi-Yao graph. Since, the number of changes to the Semi-Yao graph of a set of n moving points in a fixed dimension d is $O(n^2 \beta_{2s+2}(n))$ [17], the total processing time of Step 2 for all the $O(n^2)$ x-swap events is $O(n^2 \log^{d+1} n + n^2 \beta_{2s+2}(n)) = O(n^2 \log^{d+1} n)$.

The processing time to apply changes to the KDS for each w of Step 3, which in fact is a change to the Semi-Yao graph, is $O(\log^{d+1} n)$. Thus the update mechanism spends $O(n^2 \beta_{2s+2}(n) \log^{d+1} n)$ time to handle all the $O(n^2)$ events.

Therefore, the total cost to handle all the $O(n^2)$ x-swap events in our KDS is $O(n^2 \beta_{2s+2}(n) \log^{d+1} n)$. □

The following gives the complexity of our Semi-Yao graph KDS.

Theorem 7 *Our KDS for maintenance of the Semi-Yao graph of a set of n moving points in $\mathbb{R}^d$, where the trajectory of each point is an algebraic function of at most constant degree s, uses $O(n \log^d n)$*

space and handles $O(n^2)$ events with a total cost of $O(n^2\beta_{2s+2}(n)\log^{d+1} n)$. The KDS is compact, efficient, responsive (in an amortized sense), and local.

Proof. The total cost to process all the $O(n^2)$ events is $O(n^2\beta_{2s+2}(n)\log^{d+1} n)$ (by Lemmas 5 and 6); this means that the KDS is responsive in an amortized sense.

Since $\sum_v |B(v)| = O(n\log^d n)$ and the number of the certificates is $O(n)$, the KDS uses $O(n\log^d n)$ space, and the KDS is compact.

A particular point in a sorted list $L(u_i)$ participates in two certificates, one certificate is created with the previous point and one with the next point. Therefore, the number of events associated to a particular point at any time is $O(1)$; this implies that the KDS is local.

Since the number of the external events is $O(n^2\beta_{2s+2}(n))$ and the number of the events that the KDS processes is $O(n^2)$, the KDS is efficient. □

4 Kinetic All Nearest Neighbors

Given the kinetic Semi-Yao graph, a super-graph of the nearest neighbor graph over time, from Section 3, we can easily maintain the nearest neighbor to each point $p \in P$. Using a *dynamic and kinetic tournament tree* (DKTT) [4], we can maintain the edge with minimum length among the edges in the Semi-Yao graph incident to p; this gives the nearest neighbor to p over time.

Now we can get the following lemma.

Lemma 8 *Given a KDS for maintenance of the Semi-Yao graph, all the nearest neighbors can be maintained by using a kinetic algorithm that generates $O(n^2\beta^2_{2s+2}(n)\log n)$ tournament events, for a total cost of $O(n^2\beta^2_{2s+2}(n)\log^2 n)$. Each event can be handled in the worst-case time $O(\log^2 n)$. The number of events associated to a particular point is constant on average.*

Proof. Let $Inc(p)$ be the set of edges in the Semi-Yao graph incident to the point p. Denote by $\mathcal{TT}_p$ the dynamic and kinetic tournament tree (DKTT) corresponding to the point p. The elements of $\mathcal{TT}_p$ are the edges in $Inc(p)$. The root of the $\mathcal{TT}_p$ maintains the edge with minimum length between the edges in $Inc(p)$. Let m_p be the number of all insertions/deletions into the set $Inc(p)$ over time.

For a sequence of m_p insertions and deletions into the $\mathcal{TT}_p$, whose maximum size $\mathcal{TT}_p$ at any time is n, the $\mathcal{TT}_p$ generates at most $O(m_p\beta_{2s+2}(n)\log n)$ tournament events, for a total cost of $O(m_p\beta_{2s+2}(n)\log^2 n)$; each update can be handled in the worst-case time $O(\log^2 n)$ (from Theorem 3.1. of [4]). Therefore, the number of all events for maintenance of all the nearest neighbors is equal to $O(\sum_{p\in P} m_p\beta_{2s+2}(n)\log n) = O(\beta_{2s+2}(n)\log n\sum_{p\in P} m_p)$. Inserting (resp. deleting) an edge pq in the Semi-Yao graph makes one insertion (resp. deletion) in $\mathcal{TT}_p$ and one in $\mathcal{TT}_q$. Thus $\sum_{p\in P} m_p$ is in order of the number of exact changes to the Semi-Yao graph. Since $\sum_p |Ins(p)| = \sum_p |\mathcal{TT}_p| = O(n)$, and since the number of all changes (edge insertions and edge deletions) to the Semi-Yao graph is equal to $O(n^2\beta_{2s+2}(n))$ [17], the proof obtains.

□

The following theorem summarizes the KDS for maintenance of all the nearest neighbors. The proof follows from Theorem 7 and Lemma 8.

Theorem 9 *Our kinetic data structure for all the nearest neighbors of a set of n moving points in $\mathbb{R}^d$, where the trajectory of each point is an algebraic function of at most constant degree s, has the following properties.*

1. *The KDS uses $O(n\log^d n)$ space.*
2. *It processes $O(n^2)$ u-swap events, each in the worst-case time $O(\log^{d+1} n)$.*
3. *It processes $O(n^2)$ x-swap events, for a total cost of $O(n^2\beta_{2s+2}(n)\log^{d+1} n)$.*
4. *The KDS processes $O(n^2\beta^2_{2s+2}(n)\log n)$ tournament events, and processing all the events takes $O(n^2\beta^2_{2s+2}(n)\log^2 n)$ time.*
5. *The KDS is compact, efficient, responsive in an amortized sense, and local on average, meaning that each point participates in a constant number of certificates on average.*

5 Kinetic All $(1+\epsilon)$-Nearest Neighbors

Let q be the nearest neighbor of p and let $\hat{q}$ be some point such that $|p\hat{q}| < (1+\epsilon).|pq|$. We call $\hat{q}$ the $(1+\epsilon)$-*nearest neighbor* of p. In this section, we maintain some $(1+\epsilon)$-nearest neighbor for any point $p \in P$.

Consider a cone C_l of opening angle θ, which is bounded by d half-spaces. Let x_l be a vector inside the cone C_l that passes through the apex of C_l. Recall a CSPD $\Psi_{C_l} = \{(B_1, R_1), ..., (B_m, R_m)\}$ for P with respect to the cone C_l. Figure 4 depicts the cone C_l and a pair $(B_i, R_i) \in \Psi_{C_l}$.

Let b_i (resp. r_i) be the point with the maximum (resp. minimum) x_l-coordinate among the points in B_i (resp. R_i). Let $E_l = \{(b_i, r_i) |\ i = 1, ..., m\}$. We call the graph $G(P, E_l)$ the *relative nearest neighbor graph* (or RNN_l graph for short) with respect to C_l. Call the graph $G(P, \cup_l E_l)$ the *RNN graph*. The RNN graph has the following interesting properties: (i) It can be constructed in $O(n\log^d n)$ time by using a d-dimensional RBRT, (ii) it has $O(n\log^{d-1} n)$ edges, and (iii) the degree of each point is $O(\log^d n)$. Lemma 10 below shows another property of the RNN graph which leads us to find some $(1+\epsilon)$-nearest neighbor for any point $p \in P$.

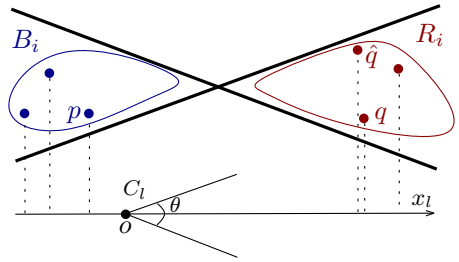


Figure 4: A pair $(B_i, R_i) \in \Psi_{C_l}$.

Lemma 10 *Between all the edges incident to a point p in the RNN graph, there exists an edge $(p, \hat{q})$ such that $\hat{q}$ is some $(1+\epsilon)$-nearest neighbor to p.*

Proof. Let q be the nearest neighbor to p and let $q \in C_l(p)$. From the definition of a CSPD with respect to C_l, for p and q there exists a unique pair $(B_i, R_i) \in \Psi_{C_l}$ such that $p \in B_i$ and $q \in R_i$. From Lemma 1, p has the maximum x_l-coordinate among the points in B_i.

Let $\hat{q}$ be the point with the minimum x_l-coordinate among the points in R_i. For any $\epsilon > 0$, there exist an appropriate angle θ and a vector x_l such that $|p\hat{q}| + (1+\epsilon).|q\hat{q}| \leq (1+\epsilon).|pq|$ [1]; this satisfies that $|p\hat{q}| \leq (1+\epsilon).|pq|$.

Therefore, the edge $(p, \hat{q})$ which is an edge of the RNN graph gives some $(1+\epsilon)$-nearest neighbor. □

Consider the set E_l of the edges of the RNN$_l$ graph. Let $N_l(p) = \{r_i |\ (b_i, r_i) \in E_l \text{ and } b_i = p\}$. Denote by $n_l(p)$ the point in $N_l(p)$ whose x_l-coordinate is minimum. Let $L(N_l(p))$ be a sorted list of the points in $N_l(p)$ in ascending order according to their x_l-coordinates; the first point in $L(N_l(p))$ gives $n_l(p)$.

From Lemma 10, if the nearest neighbor of p is in some set R_i, then r_i gives some $(1+\epsilon)$-nearest neighbor to p. Note that we do not know which cone $C_l(p)$, $1 \leq l \leq c$, of p contains the nearest neighbor of p, but it is obvious that the nearest point to p among these c points $n_1(p), ..., n_c(p)$ gives some $(1+\epsilon)$-nearest neighbor of p. Thus for all $l = 1, ..., c$, we track the distances of all the $n_l(p)$ to p over time. A kinetic sorted list (or a tournament tree) of size c with $O(1)$ certificates can be used to maintain the nearest point to p.

Similar to Section 3 we handle two types of events, *u-swap events* and *x-swap events*. Note that we do not need to define a certificate for each two consecutive points in $L(N_l(.))$. The following shows how to apply changes (*e.g.*, insertion, deletion, and exchanging the order between two consecutive points) to the sorted lists $L(N_l(.))$ when an event occurs.

Each event can make $O(\log^d n)$ updates to the edges of E_l. Consider an updated pair (b_i, r_i) that the value of r_i (resp. b_i) changes from p to q. For this update, we must delete p (resp. r_i) form the sorted list $L(N_l(b_i))$ (resp. $L(N_l(p))$) and insert q (resp. r_i) into $L(N_l(b_i))$ (resp. $L(N_l(q))$). If the event is an x-swap event, we must find all the subscripts i where $r_i = q$ and check whether $n_l(b_i) = p$ or not; if so, p and q are in the same set $N_l(.)$ and we need to exchange their order in the corresponding sorted list $L(N_l(.))$.

Now the following theorem gives the main result of this section.

Theorem 11 *Our KDS for maintenance of all the $(1+\epsilon)$-nearest neighbors of a set of n moving points in $\mathbb{R}^d$, where the trajectory of each one is an algebraic function of constant degree s, uses $O(n \log^d n)$ space and handles $O(n^2 \log^d n)$ events, each in the worst-case time $O(\log^d n \log\log n)$. The KDS is compact, efficient, responsive, and local.*

Proof. The proof of the preprocessing time and space follows from the properties of an RNN graph. Each event can make $O(\log^d n)$ changes to the edges of the RNN graph. Each update to a sorted list $L(N_l(.))$ can be done in $O(\log\log n)$. Thus an event can be handled in worst-case time $O(\log^d n \log\log n)$.

Since each event makes $O(\log^d n)$ changes to the values of $n_l(.)$, and since the size of each kinetic sorted list is constant, the number of all events to maintain all the $(1+\epsilon)$-nearest neighbors is $O(n^2 \log^d n)$.

Each point participates in a constant number of certificates in the kinetic sorted lists corresponding to the coordinate axes u_i and x_l. Since the degree of each point in the RNN graph is $O(\log^d n)$, a change to the trajectory of a point may causes $O(\log^d n)$ changes in the certificates of the kinetic sorted lists corresponding to $n_1(.), ..., n_c(.)$. Therefore, each point participates in $O(\log^d n)$ certificates. □

6 Discussion

We have provided exact KDS's for maintenance of both the Semi-Yao graph and all the nearest neighbors. These KDS's are responsive in an amortized sense. A future direction is to give exact KDS's for the Semi-Yao graph and all the nearest neighbors such that each event can be handled in a polylogarithmic worst-case time. Another open direction is to design an exact KDS for maintenance of all the nearest neighbors that is local in the worst-case.

References

[1] M. A. Abam and M. de Berg. Kinetic spanners in $\mathbb{R}^d$. *Discrete & Computational Geometry*, 45(4):723–736, 2011.

[2] M. A. Abam, Z. Rahmati, and A. Zarei. Kinetic pie delaunay graph and its applications. In *Proceedings of the 13th Scandinavian Symposium and*

Workshops on Algorithm Theory (SWAT '12), volume 7357 of *LNCS*, pages 48–58, 2012.

[3] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. *Journal of Computer and System Sciences*, 66:207–243, 2003.

[4] P. K. Agarwal, H. Kaplan, and M. Sharir. Kinetic and dynamic data structures for closest pair and all nearest neighbors. *ACM Transactions on Algorithms*, 5:4:1–37, 2008.

[5] G. Alexandron, H. Kaplan, and M. Sharir. Kinetic and dynamic data structures for convex hulls and upper envelopes. *Computational Geometry: Theory and Applications*, 36(2):144–158, 2007.

[6] L. Barba, P. Bose, J.-L. De Carufel, A. van Renssen, and S. Verdonschot. On the stretch factor of the theta-4 graph. In *Proceedings of the 13th International Conference on Algorithms and Data Structures (WADS '13)*, volume 8037 of *LNCS*, pages 109–120, 2013.

[7] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '97)*, pages 747–756,1997.

[8] J. Basch, L. J. Guibas, and L. Zhang. Proximity problems on moving points. In *Proceedings of the 13th Annual Symposium on Computational Geometry (SoCG '97)*, pages 344–351, 1997.

[9] N. Bonichon, C. Gavoille, N. Hanusse, and D. Ilcinkas. Connections between theta-graphs, delaunay triangulations, and orthogonal surfaces. In *Proceedings of the 36th International Conference on Graph-theoretic Concepts in Computer Science (WG'10)*, volume 6410 of *LNCS*, pages 266–278, 2010.

[10] P. Bose, P. Morin, A. van Renssen, and S. Verdonschot. The θ_5-graph is a spanner. In *Proceedings of the 39th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '13)*, volume 8165 of *LNCS*, pages 100–114, 2013.

[11] P. Bose, A. van Renssen, and S. Verdonschot. On the spanning ratio of theta-graphs. In *Proceedings of the 13th Workshop on Algorithms and Data Structures (WADS '13)*, volume 8037 of *LNCS*, pages 182–194, 2013.

[12] K. Clarkson. Approximation algorithms for shortest path motion planning. In *Proceedings of the 19th Aannual ACM Symposium on Theory of Computing (STOC '87)*, pages 56–65, 1987.

[13] M. de Berg, M. Roeloffzen, and B. Speckmann. Kinetic convex hulls and delaunay triangulations in the black-box model. In *Proceedings of the 27th Annual ACM Symposium on Computational Geometry (SoCG '11)*, pages 244–253, 2011.

[14] M. I. Karavelas and L. J. Guibas. Static and kinetic geometric spanners with applications. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '01)*, pages 168–176, 2001.

[15] J. M. Keil. Approximating the complete euclidean graph. In *Proceedings of the 1st Scandinavian Workshop on Algorithm Theory (SWAT '88)*, pages 208–213, 1988.

[16] K. Mehlhorn. *Data structures and algorithms 3: multi-dimensional searching and computational Geometry*. Springer-Verlag New York, Inc., New York, NY, USA, 1984.

[17] Z. Rahmati, V. King, and S. Whitesides. Kinetic data structures for all nearest neighbors and closest pair in the plane. In *Proceedings of the 29th Symposium on Computational Geometry (SoCG '13)*, pages 137–144, 2013.

[18] Z. Rahmati, S. Whitesides, and V. King. Kinetic and stationary point-set embeddability for plane graphs. In *Proceedings of the 20th International Symposium on Graph Drawing (GD '12)*, volume 7704 of *LNCS*, pages 279–290, 2013.

[19] Z. Rahmati and A. Zarei. Kinetic Euclidean minimum spanning tree in the plane. *Journal of Discrete Algorithms*, 16(0):2–11, 2012.

[20] N. Rubin. On topological changes in the delaunay triangulation of moving points. In *Proceedings of the 28th Symposium on Computational Geometry (SoCG '12)*, pages 1–10, 2012.

[21] N. Rubin. On kinetic delaunay triangulations: A near quadratic bound for unit speed motions. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS '13)*, volume 0, pages 519–528, 2013.

[22] P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete & Computational Geometry*, 4(2):101–115, 1989.

[23] D. E. Willard and G. S. Lueker. Adding range restriction capability to dynamic data structures. *Journal of the ACM*, 32(3):597–617, 1985.