Approximating the Minimum Closest Pair Distance and Nearest Neighbor Distances of Linearly Moving Points

Timothy M. Chan* Zahed Rahmati*

Abstract

Given a set of n moving points in $\mathbb{R}^d$, where each point moves along a linear trajectory at arbitrary but constant velocity, we present an $\tilde{O}(n^{5/3})$-time algorithm[1] to compute a $(1+\epsilon)$-factor approximation to the *minimum closest pair distance* over time, for any constant $\epsilon > 0$ and any constant dimension d. This addresses an open problem posed by Gupta, Janardan, and Smid [12].

More generally, we consider a data structure version of the problem: for any linearly moving query point q, we want a $(1+\epsilon)$-factor approximation to the *minimum nearest neighbor distance* to q over time. We present a data structure that requires $\tilde{O}(n^{5/3})$ space and $\tilde{O}(n^{2/3})$ query time, $\tilde{O}(n^5)$ space and polylogarithmic query time, or $\tilde{O}(n)$ space and $\tilde{O}(n^{4/5})$ query time, for any constant $\epsilon > 0$ and any constant dimension d.

1 Introduction

In the last two decades, there has been a lot of research on problems involving objects in motion in different computer science communities (*e.g.*, robotics and computer graphics). In computational geometry, maintaining attributes (*e.g.*, closest pair) of moving objects has been studied extensively, and efficient kinetic data structures are built for this purpose (see [17] and references therein). In this paper, we pursue a different track: instead of maintaining an attribute over time, we are interested in finding a time value for which the attribute is minimized or maximized.

Let P be a set of moving points in $\mathbb{R}^d$, and denote by $p(t)$ the position (trajectory) of $p \in P$ at time t. Let $d(p(t), q(t))$ denote the Euclidean distance between $p(t)$ and $q(t)$. The following gives the formal statements of the two kinetic problems we address in this paper, generalizing two well-known standard problems for stationary points, *closest pair* and *nearest neighbor search*:

- *Kinetic minimum closest pair distance*: find a pair (p, q) of points in P and a time instant t, such that $d(p(t), q(t))$ is minimized.
- *Kinetic minimum nearest neighbor distance*: build a data structure so that given a moving query point q, we can find a point $p \in P$ and a time instant t such that $d(p(t), q(t))$ is minimized.

Related work. The *collision detection problem*, *i.e.*, detecting whether points ever collide [10], has attracted a lot of interest in the context. This problem can trivially be solved in quadratic time by brute force. For a set P of n linearly moving points in $\mathbb{R}^2$, Gupta, Janardan, and Smid [12] provided an algorithm, which detects a collision in P in $O(n^{5/3}\log^{6/5} n)$ time.

Gupta *et al.* also considered the minimum diameter of linearly moving points in $\mathbb{R}^2$, where the velocities of the moving points are constant. They provided an $O(n\log^3 n)$-time algorithm to compute the minimum diameter over time; the running time was improved to $O(n\log n)$ using randomization [7, 9]. Agarwal *et al.* [2] used the notion of ϵ-kernel to maintain an approximation of the diameter over time. For an arbitrarily small constant $\delta > 0$, their kinetic data structure in $\mathbb{R}^2$ uses $O(1/\epsilon^2)$ space, $O(n+1/\epsilon^{3s+3/2})$ preprocessing time, and processes $O(1/\epsilon^{4+\delta})$ events, each in $O(\log(1/\epsilon))$ time, where s is the maximum degree of the polynomials of the trajectories; this approach works for higher dimensions.

For a set of n stationary points in $\mathbb{R}^d$, the closest pair can be computed in $O(n\log n)$ time [5]. Gupta *et al.* [12] considered the kinetic minimum closest pair distance problem. Their solution is for the $\mathbb{R}^2$ case, and works only for a limited type of motion, where the points move with the same constant velocity along one of the two orthogonal directions. For this special case their algorithm runs in $O(n\log n)$ time. Their work raises the following open problem: Is there an efficient algorithm for the kinetic minimum closest pair distance problem in the more general case where points move with constant but possibly different velocities and different moving directions?

For a set of stationary points in $\mathbb{R}^d$, there are data structures for approximate nearest neighbor search with linear space and logarithmic query time [4]. We are not aware of any prior work on the kinetic minimum nearest neighbor distance problem. Linearly moving points in $\mathbb{R}^d$ can be mapped to lines in $\mathbb{R}^{d+1}$ by viewing time as an extra dimension. There have been previous papers

*Cheriton School of Computer Science, University of Waterloo, {tmchan, zrahmati}@uwaterloo.ca

[1] The notation $\tilde{O}$ is used to hide polylogarithmic factors. That is, $\tilde{O}(f(n)) = O(f(n)\log^c n)$, where c is a constant.

on approximate nearest neighbor search in the setting when the data objects are lines and the query objects are points [13], or when the data objects are points and the query objects are lines [16] (in particular, the latter paper contains some results similar to ours for any constant dimension d). However, in our problem, both data and query objects are mapped to lines; moreover, our distance function does not correspond to Euclidean distances between lines in $\mathbb{R}^{d+1}$.

An approach to solve the kinetic minimum closest pair distance and nearest neighbor distance problems would be to track the closest pair and nearest neighbor over time using known kinetic data structures [3, 18, 19]. The chief drawback of this approach is that the closest pair can change $\Omega(n^2)$ times in the worst case, and the nearest neighbor to a query point can change $\Omega(n)$ times (even if approximation is allowed). The challenge is to solve the kinetic minimum closest pair distance problem in $o(n^2)$ time, and obtain a query time $o(n)$ for the kinetic minimum nearest neighbor distance problem. To this end, we will allow approximate solutions.

Our contributions. We focus on the setting where each point in P (and each query point) has an arbitrary, constant velocity, and moves along an arbitrary direction.

We present an algorithm to compute a $(1+\epsilon)$-factor approximation to the minimum closest pair distance in $\tilde{O}(n^{5/3})$ time for any constant $\epsilon > 0$. More generally, we present a data structure for the kinetic minimum nearest neighbor distance problem with approximation factor $1+\epsilon$ with $\tilde{O}(m)$ preprocessing time and space, and $\tilde{O}(n/m^{1/5})$ query time for any m between n and n^5. For example, setting m appropriately, we obtain a data structure with $\tilde{O}(n^{5/3})$ space and $\tilde{O}(n^{2/3})$ query time, $\tilde{O}(n^5)$ space and $\tilde{O}(1)$ query time, or $\tilde{O}(n)$ space and $\tilde{O}(n^{4/5})$ query time. The results hold in any constant dimension d. Our solution uses techniques from range searching (including multi-level data structures and parametric search).

Perhaps the most notable feature of our results is that the exponents do not grow as a function of the dimension. (In contrast, for the exact kinetic minimum closest pair problem, it is possible to obtain subquadratic-time algorithms by range searching techniques, but with much worse exponents that converge to 2 as d increases.)

2 Kinetic Minimum Nearest Neighbor Distance

Let $p(t) = \mathbf{p}' + t\mathbf{p}''$ denote the linear trajectory of a point $p \in P$, where $\mathbf{p}' \in \mathbb{R}^d$ is the initial position vector of p, and $\mathbf{p}'' \in \mathbb{R}^d$ is the velocity vector of p. For any moving query point q with a linear trajectory $q(t) = \mathbf{q}' + t\mathbf{q}''$, we want to approximate the minimum nearest neighbor distance to q over time.

We first consider the following decision problem:

Decision Problem 1 *Given a point q and a real parameter r, determine whether there exists $p \in P$ with*

$$\min_{t\in\mathbb{R}} d(p(t), q(t)) \le r. \tag{1}$$

Afterwards we use the parametric search technique to find the minimum nearest neighbor distance of q in P.

Approximating Decision Problem 1. Let $\mathbf{w}$ be a vector in $\mathbb{R}^d$. The Euclidean norm $\|\mathbf{w}\|$ of $\mathbf{w}$ can be approximated as follows [6]. Assume $\theta = \arccos(1/(1+\epsilon))$, for a small $\epsilon > 0$. The d-dimensional space around the origin can be covered by a set of $b = O(1/\theta^{d-1}) = O(1/\epsilon^{(d-1)/2})$ cones of opening angle θ [20]. *i.e.*, there exists a set $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_b\}$ of unit vectors in $\mathbb{R}^d$ that satisfies the following property: for any $\mathbf{w} \in \mathbb{R}^d$ there is a unit vector $\mathbf{v}_i \in V$ such that $\angle(\mathbf{v}_i, \mathbf{w}) \le \theta$. Note that $\angle(\mathbf{v}_i, \mathbf{w}) = \arccos(\mathbf{v}_i \cdot \mathbf{w}/\|\mathbf{w}\|)$, where $\mathbf{v}_i \cdot \mathbf{w}$ denotes the inner product of the unit vector $\mathbf{v}_i$ and $\mathbf{w}$. Therefore,

$$\|\mathbf{w}\|/(1+\epsilon) \le \max_{i\in B} \mathbf{v}_i \cdot \mathbf{w} \le \|\mathbf{w}\|, \tag{2}$$

where $B = \{1, \ldots, b\}$.

From (2), we can use the following as an approximation of $d(p(t), q(t))$:

$$\max_{i\in B} \mathbf{v}_i \cdot (\mathbf{p}' - \mathbf{q}' + t(\mathbf{p}'' - \mathbf{q}'')).$$

Let $p'_i = \mathbf{v}_i \cdot \mathbf{p}'$, $p''_i = \mathbf{v}_i \cdot \mathbf{p}''$, $q'_i = \mathbf{v}_i \cdot \mathbf{q}'$, and $q''_i = \mathbf{v}_i \cdot \mathbf{q}''$. From the above discussion, a solution to Decision Problem 1 can be approximated by deciding the following.

Decision Problem 2 *Given a point q and a real parameter r, test whether there exists $p \in P$ with*

$$\min_{t\in\mathbb{R}} \max_{i\in B} (p'_i - q'_i) + t(p''_i - q''_i) \le r. \tag{3}$$

Solving Decision Problem 2. Consider the inequality in (3). Minimizing the maximum of $\gamma_i(t) = (p'_i - q'_i) + t(p''_i - q''_i)$, over $i \in B$, is equivalent to finding the lowest point on the upper envelope of the linear functions $\gamma_i(t)$ in the $t\gamma$-plane; see Figure 1(a). Thus (3) is equivalent to checking whether the lowest point of the upper envelope is on or below the line $\gamma = r$.

Let $t_i = (r - p'_i + q'_i)/(p''_i - q''_i)$ denote the time that $\gamma_i(t)$ intersects with $\gamma = r$, *i.e.*, the root for $\gamma_i(t) = r$. Let $m_i = p''_i - q''_i$ denote the slope of the linear function $\gamma_i(t)$.

Deciding the following is equivalent to deciding whether the lowest point on the upper envelope of $\gamma_i(t)$ is on or below the line $\gamma = r$.

- The maximum root of the linear functions $\gamma_i(t) = r$ with *negative* slope is less than or equal to the

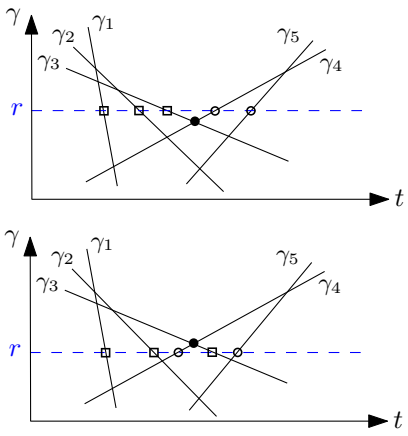

Figure 1: The intersections of γ_i with $\gamma = r$ are shown by empty circles (○) if the slope m_i of γ_i is positive, or empty squares (□) if $m_i < 0$. (a) The lowest point (bullet point ●) on the upper envelope is below $\gamma = r$. (b) The lowest point on the upper envelope is above $\gamma = r$, $m_3 < 0$, $m_4 > 0$, and $t_3 > t_4$.

minimum root of the linear functions with *positive* slope. In other words,

$$\max_{i:m_i<0} t_i \le \min_{j:m_j>0} t_j. \tag{4}$$

Note that `if` the lowest point of the upper envelope is *above* the line $\gamma = r$, then there exists a pair (i, j) of indices such that the clause $(m_i > 0) \vee (m_j < 0) \vee (t_i < t_j)$ is false (see Figure 1(b)); `otherwise`, the conjunction of all clauses (for all $i, j \in B$) is true. Therefore, we can obtain the following.

Lemma 1 *The inequality of (3) is satisfied iff the following conjunction is true:*

$$\bigwedge_{i,j\in B} ((p_i'' > q_i'') \vee (p_j'' < q_j'') \vee (t_i < t_j)),$$

where $t_i = (r - p_i' + q_i')/(p_i'' - q_i'')$.

Each condition in the clauses in Lemma 1 may be represented as a half-space in the following manner.

Consider the inequality $t_i < t_j$, *i.e.*,

$$\frac{r - p_i' + q_i'}{p_i'' - q_i''} < \frac{r - p_j' + q_j'}{p_j'' - q_j''}.$$

Assuming $(p_i'' < q_i'') \wedge (p_j'' > q_j'')$, $t_i < t_j$ is equivalent to

$$(r - p_i' + q_i')(p_j'' - q_j'') - (r - p_j' + q_j')(p_i'' - q_i'') > 0,$$

which can be expanded as

$$rp_j'' - rq_j'' - p_i'p_j'' + p_i'q_j'' + q_i'p_j'' - q_i'q_j'' - rp_i'' + rq_i'' + p_j'p_i'' - p_j'q_i'' - p_i''q_j' + q_i''q_j' > 0.$$

By factoring some terms in the above inequality, we obtain

$$p_i'(q_j'') + p_i''(-r - q_j') + p_j'(-q_i'') + p_j''(r + q_i') + p_i''p_j' - p_i'p_j'' + rq_i'' - rq_j'' - q_i'q_j'' + q_i''q_j' > 0,$$

which can be expressed in the form

$$A_1X_1 + A_2X_2 + A_3X_3 + A_4X_4 + X_5 > A_5,$$

where $X_1 = p_i'$, $X_2 = p_i''$, $X_3 = p_j'$, $X_4 = p_j''$, $X_5 = p_i''p_j' - p_i'p_j''$, $A_1 = q_j''$, $A_2 = -r - q_j'$, $A_3 = -q_i''$, $A_4 = r + q_i'$, and $A_5 = -rq_i'' + rq_j'' + q_i'q_j'' - q_i''q_j'$.

Lemma 2 *For each pair (i, j) of indices, the clause $(p_i'' > q_i'') \vee (p_j'' < q_j'') \vee (t_i < t_j)$ in Lemma 1 can be represented as*

$$(X_2 > -A_3) \vee (X_4 < A_1) \vee \tag{5}$$
$$(A_1X_1 + A_2X_2 + A_3X_3 + A_4X_4 + X_5 > A_5). \tag{6}$$

From Lemmas 1 and 2, we have reduced Decision Problem 2 to a searching problem $\mathcal{S}$, which is the conjunction of $O(b^2)$ simplex range searching problems $\mathcal{S}_l$, $l = 1, \ldots, O(b^2)$. Each $\mathcal{S}_l$ is a 5-dimensional simplex range searching problem on a set of points, each with coordinates $(X_1, X_2, X_3, X_4, X_5)$ that is associated with a point $p \in P$. The polyhedral range (5–6) for $\mathcal{S}_l$, which can be decomposed into a constant number of simplicial ranges, is given at query time, where $A_1, \ldots, A_5$ can be computed from the query point q and the parameter r.

Data structure for the searching problem $\mathcal{S}$. *Multi-level data structures* can be used to solve complex range searching problems [1] involving a conjunction of multiple constraints. In our application, we build a multi-level data structure $\mathcal{D}$ to solve the searching problem $\mathcal{S}$ consisting of $O(b^2)$ levels. To build a data structure for a set at level l, we form a collection of canonical subsets for the 5-dimensional simplex range searching problem $\mathcal{S}_l$, and build a data structure for each canonical subset at level $l + 1$. The answer to a query is expressed as a union of canonical subsets. For a query for a set at level l, we pick out the canonical subsets corresponding to all points in the set satisfying the l-th clause by 5-dimensional simplex range searching in $\mathcal{S}_l$, and then answer the query for each such canonical subset at level $l + 1$.

A multi-level data structure increases the complexity by a polylogarithmic factor (see Theorem 10 of [1] or the papers [8, 14]). In particular, if $S(n)$ and $Q(n)$ denote the space and query time of 5-dimensional simplex range searching, our multi-level data structure $\mathcal{D}$ requires $O(S(n)\log^{O(b^2)} n)$ space and $O(Q(n)\log^{O(b^2)} n)$ query time.

Assume $n \le m \le n^5$. A 5-dimensional simplex range searching problem can be solved in $\tilde{O}(\frac{n}{m^{1/5}})$ query time

with $\tilde{O}(m)$ preprocessing time and space [8, 14]. We conclude:

Lemma 3 *Let $n \le m \le n^5$. A data structure $\mathcal{D}$ for Decision Problem 2 can be built that uses $O(m \log^{O(b^2)} n)$ preprocessing time and space and can answer queries in $O(\frac{n}{m^{1/5}} \log^{O(b^2)} n)$ time.*

Solving the optimization problem. Consider Decision Problem 2, and denote by r^* the smallest r satisfying (3). We use Megiddo's parametric search technique [15] to find r^*. This technique uses an efficient *parallel* algorithm for the decision problem to provide an efficient *serial* algorithm for the optimization problem (computing r^*); the running time typically increases by logarithmic factors. Suppose that the decision problem can be solved in T time sequentially, or in τ parallel steps using π processors. Then the running time to solve the optimization problem would be $O(\tau \cdot \pi + T \cdot \tau \cdot \log \pi)$.

In our case, $T = \pi = O(\frac{n}{m^{1/5}} \log^{O(b^2)} n)$ (by Lemma 3) and $\tau = O(\log^{O(b^2)} n)$, where $b^2 = O(1/\epsilon^{d-1})$. Therefore, we obtain the main result of this section:

Theorem 4 *Let $n \le m \le n^5$. For a set P of n linearly moving points in $\mathbb{R}^d$ for any constant d, there exists a data structure with $O(m \log^{O(1/\epsilon^{d-1})} n)$ preprocessing time and space that can compute a $(1+\epsilon)$-factor approximation of the minimum nearest neighbor distance to any linearly moving query point over time in $O(\frac{n}{m^{1/5}} \log^{O(1/\epsilon^{d-1})} n)$ time.*

Remark 1 Our approach can be modified to compute the minimum distance over all time values inside any query interval $[t_0, t_f]$. The conjunction in Lemma 1 becomes $\bigwedge_{i,j \in B}((p_i'' > q_i'') \vee (p_j'' < q_j'') \vee (t_i < t_j) \vee (t_i > t_f) \vee (t_j < t_0))$. The condition $t_i > t_f$ is equivalent to $r - p_i' + q_i > t_f(p_i'' - q_i'')$, which can be expressed in the form $B_1 Y_1 + Y_2 < B_2$, where $Y_1 = p_i''$, $Y_2 = p_i'$, $B_1 = t_f$, and $B_2 = r + q_i + t_f q_i''$. This corresponds to a 2-dimensional halfplane range searching problem. The condition $t_j < t_0$ can be handled similarly. We can expand the entire expression into a disjunction of $5^{O(b^2)}$ subexpressions, where each subexpression is a conjunction of $O(b^2)$ conditions and can then be handled by a multi-level data structure similar to $\mathcal{D}$.

Remark 2 Our approach can be used to compute the *exact* minimum nearest neighbor distance in the L_∞ metric to any moving query point. Let $\mathbf{v}_j$ and $\mathbf{v}_{d+j}$ be the unit vectors of the negative x_j-axis and positive x_j-axis, respectively, in the d dimensional Cartesian coordinate system, where $1 \le j \le d$. We define $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_b\}$ with $b = 2d$, and solve the problem as before.

3 Kinetic Minimum Closest Pair Distance

To approximate the kinetic minimum closest pair distance, we can simply preprocess P into the data structure of Theorem 4, and for each point $p \in P$, approximate the minimum nearest neighbor distance to p. The total time is $O((m + \frac{n^2}{m^{1/5}}) \log^{O(1/\epsilon^{d-1})} n)$ time. Setting $m = 5/3$ gives the main result:

Theorem 5 *For a set of n linearly moving points in $\mathbb{R}^d$ for any constant d, there exists an algorithm to compute a $(1+\epsilon)$-factor approximation of the minimum closest pair distance over time in $O(n^{5/3} \log^{O(1/\epsilon^{d-1})} n)$ time.*

Remark 3 By Remark 2, we can compute the *exact* minimum closest pair distance in the L_∞ metric, of a set of n linearly moving points in $\mathbb{R}^d$, in $O(n^{5/3} \log^{O(d^2)} n)$ time.

4 Discussion

For a set P of linearly moving points in $\mathbb{R}^d$, we have given efficient algorithms and data structures to approximate the minimum value of two fundamental attributes: the closest pair distance and distances to nearest neighbors. We mention some interesting related open problems along the same direction:

- The Euclidean minimum spanning tree (EMST) on a set P of n moving points in $\mathbb{R}^2$ can be maintained by handling nearly cubic events [19], each in polylogarithmic time. Can we compute the minimu weight of the EMST on P, for linearly moving points, in subcubic time?
- For a set of n moving unit disks, there exist kinetic data structures [11] that can efficiently answer queries in the form "Are disks D_1 and D_2 in the same connected component?". This kinetic data structure handles nearly quadratic events, each in polylogarithmic time. Can we find the first time when all the disks are in the same connected component in subquadratic time?

References

[1] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. *Contemporary Mathematics*, 223:1–56, 1999.

[2] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *Journal of the ACM*, 51(4):606–635, 2004.

[3] 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.

[4] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[5] J. L. Bentley and M. I. Shamos. Divide-and-conquer in multidimensional space. In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing (STOC)*, pages 220–230, ACM, 1976.

[6] T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. *International Journal of Computational Geometry & Applications*, 12(1-2):67–85, 2002.

[7] T. M. Chan. An optimal randomized algorithm for maximum Tukey depth. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 430–436, SIAM, 2004.

[8] T. M. Chan. Optimal partition trees. *Discrete & Computational Geometry*, 47(4):661–690, 2012.

[9] K. L. Clarkson. Algorithms for the minimum diameter of moving points and for the discrete 1-center problem, 1997. http://kenclarkson.org/moving_diam/p.pdf.

[10] K. Fujimura. *Motion Planning in Dynamic Environments*. Springer-Verlag, Secaucus, NJ, USA, 1992.

[11] L. Guibas, J. Hershberger, S. Suri, and L. Zhang. Kinetic connectivity for unit disks. *Discrete & Computational Geometry*, 25(4):591–610, 2001.

[12] P. Gupta, R. Janardan, and M. Smid. Fast algorithms for collision and proximity problems involving moving geometric objects. *Computational Geometry*, 6(6):371–391, 1996.

[13] S. Mahabadi. Approximate nearest line search in high dimensions. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 337–354, SIAM, 2015.

[14] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete & Computational Geometry*, 10(1):157–182, 1993.

[15] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.

[16] W. Mulzer, H. L. Nguyen, P. Seiferth, and Y. Stein. Approximate k-flat nearest neighbor search. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, 2015.

[17] Z. Rahmati. *Simple, Faster Kinetic Data Structures*. PhD thesis, University of Victoria, 2014.

[18] Z. Rahmati, M. A. Abam, V. King, and S. Whitesides. Kinetic k-semi-Yao graph and its applications. *Computational Geometry* (to appear).

[19] Z. Rahmati, M. A. Abam, V. King, S. Whitesides, and A. Zarei. A simple, faster method for kinetic proximity problems. *Computational Geometry*, 48(4):342–359, 2015.

[20] A. C.-C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.