Proximity Problems on Moving Points

Julien Basch Leonidas J. Guibas* Li Zhang

Computer Science Department
Stanford University
Stanford, CA94305
{jbasch,guibas,lizhang}@cs.stanford.edu

Abstract

A kinetic data structure for the maintenance of a multidimensional range search tree is introduced. This structure is used as a building block to obtain kinetic data structures for two classical geometric proximity problems in arbitrary dimensions: the first structure maintains the closest pair of a set of continuously moving points, and is provably efficient. The second structure maintains a spanning tree of the moving points whose cost remains within some prescribed factor of the minimum spanning tree.

1 Introduction

The goal of this paper is to address a number of proximity problems for points and balls moving continuously in $\mathcal{R}^d$. Examples of such problems are the maintenance of the closest pair and the minimum spanning tree (MST) of the moving objects. In the dynamic simulation of virtual environments, collision detection between the moving objects is frequently important. Complex bodies can be approximated by bounding spheres and knowledge of the closest pair of these spheres at all times can be used to limit where a more elaborate collision-checking algorithm is invoked. In other applications, where the moving objects need to stay in constant communication with each other, the MST might be a desirable way of maintaining a set of point-to-point links among the objects. These are just some examples of the many proximity problems that can be posed for a system of moving bodies — problems which are interesting from both the theoretical and the practical points of view.

Both the closest pair and the MST problems have been extensively studied for sets of points in arbitrary dimensions. It would take us too far afield to survey the known results. What is surprising is that most of these are not directly applicable to the setting of objects moving continuously. Extant algorithms are either *static*, i.e. work for a fixed set of points, or *dynamic*, i.e. allow insertions and deletions to/from the set of points. It is not immediately obvious how to use these classical algorithms to maintain the closest pair and the MST as the points move continuously. In this paper we present algorithms which explicitly take advantage of the continuity or coherence in the motion of the points to gain efficiency. These new algorithms are *kinetic* — they are instances of *kinetic data structures* (KDSs for short), as introduced by Basch, Guibas, and Hershberger [BGH97].

In the kinetic setting, a set of objects is assumed to be continuously changing, or moving. Each object follows a posted *flight plan*, but a plan which can change at any moment through a *flight plan update*. A kinetic data structure maintains a *configuration function* of interest (in our case this will be the closest pair or the MST) by watching for critical events as the objects move. These events correspond to the times at which certain elementary conditions, called *certificates*, switch from being true to being false (an example might be 'points A and B are closer than points C and D'). At any one time, the conjunction of all the certificates being watched over by the kinetic data structure proves the combinatorial correctness of its output (for example, A and B are the closest pair). When a certificate fails, the proof structure needs to be modified and the combinatorial description of the configuration function may need to be updated (because now C and D may be the closest pair). A good kinetic data structure will take advantage of the continuity of the object motion to select certificate structures which are easy to update at these critical events; a structure satisfying this condition is called *responsive*. Other desiderata for a kinetic data structure are (1) that it be *efficient*, meaning that the number of events it processes is not much greater than the number of combinatorial changes in the configuration function itself, (2) that it be *compact*, so that the number of active certificates at any one time is roughly linear in the complexity of the moving objects, (3) that it be *local*, so that a flight plan update for any one object affects only a small number of certificates. A kinetic structure which is responsive, efficient, compact, and local is worst-case nearly optimal for the problem at hand. For precise definitions of these concepts the reader is referred to [BGH97].

The results of this paper are based on kinetizing (i.e., maintaining under continuous point motion) a certain type of *multidimensional range search tree* (MDRS tree) [Meh84] used to query the value of some specified function on subsets of the points [BS80]. We apply this MDRS tree to obtain kinetic data structures for the closest pair among n moving points in $\mathcal{R}^d$, the closest pair among a collection of n moving balls, and approximate and exact MSTs for n moving points. Though the structures we propose are fully on-line and work

*Supported in part by National Science Foundation grant CCR–9623851 and by US Army MURI grant DAAH04–96–1–0007.

To appear in the 13th Symposium of Computational Geometry, 1997.

for arbitrary motions, the bounds we give assume that the motions of the points are algebraic of fixed degree, or that they satisfy certain Davenport-Schinzel type conditions.

In [BGH97] an efficient, compact, responsive and local kinetic data structure is given for the closest pair in $\mathcal{R}^2$. Another event-based algorithm for detecting the collision of moving balls was proposed in [KGS]. For arbitrary dimensions the performance of the algorithms given in this paper is nearly optimal as a function of n (the number of points or balls), but not of d (the dimension — the 'hidden' constant in our bounds is 2^{2d}). A major open problem is to find kinetic algorithms for proximity problems which do not have this exponential dependence on the dimension.

Section 2 presents our kinetic MDRS tree and shows that it is efficient, compact, responsive, and local, a result which is of independent interest. Section 3 gives a kinetic algorithm for closest pair in $\mathcal{R}^d$ under the Euclidean metric — via a structure which is also efficient, compact, local, and responsive. Our method is based on a certain packing lemma, and extends to moving balls under the assumption that the ratio of the radii of any two balls involved is bounded by a constant. Our results for the MST are not as good. In Section 4 we give a kinetic data structure which maintains the exact MST of the moving points under certain polyhedral metrics (for example, L_1 or L_∞). For the L_2 metric we can only maintain a tree which is a $(1+\epsilon)$ approximation to the real MST. All of our KDSs for the the MST are compact, local, and responsive, but they are not efficient (in the strict KDS sense).

2 Kinetized Multidimensional Range Search Tree

In this section, we study in the kinetic setting the classic range searching problem when the ranges are determined by a fixed set of directions. This version of the range searching problem has received extensive treatment in the literature both in the static and in the dynamic case [Meh84, Mul94]. We focus here on a specific version of this problem that will be well suited for the applications to the closest pair and the MST.

In the generic range searching problem, a set S of n points of $\mathcal{R}^d$ is given, together with a (generally infinite) set of *ranges* which are certain restricted regions of $\mathcal{R}^d$; these ranges define subsets of S of interest. Each point of S is assigned a *weight* from a commutative semigroup, and the semigroup operation allows us to compute the weight of any subset of S induced by a range. The problem then is to compute a data structure $\mathcal{T}$ on S that allows us to find efficiently the weight of any query range.

If each point in the set is assigned a weight from a linearly ordered set, the semigroup setting is general enough to define, for instance, a "min-weight" function such that the min-weight of any range is the list of the ℓ elements of smallest weight in that range. In this case, the semigroup operation, which simply merges two sets of size at most ℓ each and keeps only the ℓ smallest elements, takes time $\Theta(\ell)$, and this factor multiplies all queries and update operations.

For the purpose of this paper, we now focus on a specific version of the range searching problem. It is easily seen that the kinetization described below can be applied to many variants on this kind of range search tree.

2.1 A Special Range Searching Problem

Our restricted range searching problem is as follows. We are given a model infinite polyhedral cone $\mathcal{C}_0$ delimited by a constant number k of hyperplanes passing through the origin. The set of ranges are the subsets of S induced by all possible translations of this cone. We denote by $\mathcal{C}_p$ the model cone with apex translated at p. For each hyperplane delimiting the model cone, we define an associated *range ordering* on S as the ordering of the projections of the points on the normal to this hyperplane.

For a static set S and a weight function f, repeated median partitioning along the range orderings can be used to recursively construct a multidimensional range search (MDRS) tree $\mathcal{T}$ storing the elements of S with the following properties [Meh84]:

- The tree $\mathcal{T}$ can be built in time $O(n \log^k n)$; it has size $O(n \log^{k-1} n)$ and depth $O(\log n)$.
- Each node ν in $\mathcal{T}$, stores $f(T_\nu)$, where T_ν, called the *span of* ν is the set of all the elements in the subtree rooted at node ν.
- Each element appears at most $O(\log^{k-1} n)$ times in the tree.
- For each conical range $\mathcal{C}$, there exists a set of $O(\log^k n)$ nodes $N_\mathcal{C}$ such that $S \cap \mathcal{C} = \bigcup_{\nu \in N_\mathcal{C}} T_\nu$, where the union is disjoint. Furthermore, this set of nodes can be found in time $O(\log^k n)$.

Therefore, for any query range $\mathcal{C}$, $f(\mathcal{C})$ can be computed in two steps: first find the set $N_\mathcal{C}$ of $O(\log^k n)$ nodes whose spans are disjoint and whose union is $S \cap \mathcal{C}$. Then compute

$$f(\mathcal{C}) = \sum_{\nu \in N_\mathcal{C}} f(T_\nu)\,,$$

where the Σ is to be understood as the generalized semigroup operation to an arbitrary number of values. The query time is thus $O(t_f \log^k n)$, where t_f is the time complexity of the semigroup operation associated with f.

In the dynamic setting, we allow the following operations:

- deletion of a point,
- insertion of a point, and
- change of a point weight (this can be viewed as deleting a point and then reinserting it with a new weight)

By relaxing the equal-sized partitioning in the recursive definition of $\mathcal{T}$ to a balanced partitioning and using a careful rotation scheme, the above structure can be maintained and queried within $O(t_f \log^k n)$ worst case time per operation [WL85]. Another way to dynamize $\mathcal{T}$ is to do some local and global rebuilding after every few operations, which gives an amortized bound of $O(t_f \log^k n)$ per operation [Meh84, Ove83], or to use randomized search trees [AS89] and obtain the same bounds in expectation.

2.2 Kinetizing the tree $\mathcal{T}$

We now assume that we are given a set S of points whose positions and weights are continuous functions of time; we wish to maintain their MDRS tree $\mathcal{T}$ as time goes on. We also assume that the semigroup operation depends only on the ordering of the weights. This will be the case in our applications, where the weight of a point will be taken to

be its inner product with a specific vector, and the semigroup operation will be the min, or some generalized min, as mentioned above.

A kinetic version of the MDRS tree $\mathcal{T}$ is easy to obtain, because the structure of $\mathcal{T}$ depends only on the relative order of the elements of S along the given directions. The kinetized structure consists of:

- A *dynamic* multidimensional search tree $\mathcal{T}$.
- A set of $k+1$ kinetic sorted lists, one for each of the k range orderings and one for the weight ordering. For each sorted list, the kinetic certificates are comparisons between every two adjacent elements in it.
- A global event queue $\mathcal{Q}$, to be used for storing the kinetic structure events; in our case these correspond simply to the times when two adjacent elements in a sorted list exchange their ordering.

There are two types of events which may cause the multidimensional tree to change: one is an exchange in some range ordering among two elements of S and the other is an exchange in their weight ordering. Both types of events can be easily handled by performing a deletion and a re-insertion of the points involved in the event, using the dynamic capabilities of $\mathcal{T}$. In addition, we update the kinetic sorted list involved by exchanging the two elements and rescheduling the events for the newly adjacent pairs in the list — so there are at most 3 events to reschedule. The correctness of this data structure is ensured by the continuity of motion because, for continuously moving points, the sorted lists and all our range and weight orderings can only change when two adjacent points swap order.

Theorem 1 *The kinetized multidimensional range search tree on n moving points with continuously changing weights is local, compact, and responsive. It can be maintained at a cost of $O(\log^k n)$ per event.*

If the points follow algebraic trajectories of fixed degree, and the weight function is also algebraic of fixed degree, the total number of events will be $O(n^2)$. It is not hard to see that the number of changes the MDRS tree will undergo is $\Theta(n^2)$ in the worst case. Hence, this kinetization is efficient. Note that all range events are external (they change the MDRS structure), but some weight events may not be.

3 Maintaining the Closest Pair of Moving Points

We now show how a combination of instances of the kinetic MDRS tree can be used to maintain the closest pair (in the L_2 sense) of a set S of moving points in $\mathcal{R}^d$. We take each of the 2^d orthants in $\mathcal{R}^d$as a model cone for which we maintain an MDRS tree on the set of points, with a weight for each point that is the projection on the 'diagonal' of this orthant (the line through the origin in the orthant making equal angles with the axes). Such 'quadrant-tilings' have also been previously used for dynamic closest-pair algorithms [Smi92]. What is surprising here is that our rank functions, which are all based on ordering the points along certain fixed directions, can be used to find an exact (not approximate) L_2 closest pair. As usual, what helps us is a certain packing lemma.

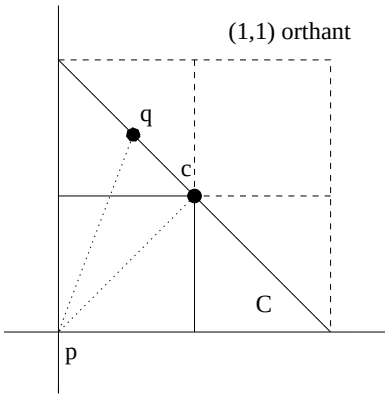

Figure 1: Partitioning of corner

3.1 A Packing Lemma

The key observation on the closest pair is that, if we project certain subsets of the points onto certain well-chosen lines and, for each point, we keep a certain number of its nearest neighbors along each of these lines, then the closest pair must be among the pairs thus considered.

In $\mathcal{R}^d$, each vector in an orthogonal basis has an associated normal hyperplane. These d hyperplanes divide the space into 2^d cones, called orthants. Formally, if $\sigma = \frac{1}{\sqrt{d}}(\varepsilon_1, \ldots, \varepsilon_d)$ with $\varepsilon_i \in \{-1, 1\}$ (a *diagonal vector*), we define the *σ-orthant* to be:

$$orth_\sigma = \{(q_1, \ldots, q_d) \in \mathcal{R}^d \mid \forall i \ \varepsilon_i q_i \geq 0\}.$$

We also define the σ-ordinate of a point p to be the inner product of p with σ.

For a point $p \in S$ and a diagonal vector σ, we say that the $2^d - 1$ points of $S \cap orth_\sigma(p)$ that have smallest σ-ordinates are the *σ-candidate points* of p, and we denote this set by $cand_\sigma(p)$. We say that a pair of points (p, q) is a *σ-candidate pair* if both $q \in cand_\sigma(p)$ and $p \in cand_{-\sigma}(q)$, and we say that that it is a *candidate pair* if it is a σ-candidate pair for some σ.

Lemma 2 *For any finite set $S \subset \mathcal{R}^d$, the closest pair in S must be a candidate pair. Furthermore, each point $p \in S$ can be part of at most 2^{2d} candidate pairs.*

Proof. Let (p, q) be the closest pair of S, and say that q is in the σ-orthant of p for some diagonal vector σ (Figure 1).

Let c be the projection of q on the line parallel to σ passing through p. The region C defined by the hyperplanes of the cone and the hyperplane orthogonal to σ passing through c can be covered by $2^d - 1$ interior-disjoint hypercubes. If q is not a candidate point of p in the σ-orthant, there must be at least 2^d points (including p itself) lying in C. Therefore, there must be two points lying within the same hypercube, which implies that their distance is less than $d(p, c)$ and therefore less than $d(p, q)$, contradicting the hypothesis that (p, q) is the closest pair. By the same argument, p must also be a candidate point of q. Thus (p, q) is a candidate pair.

Furthermore, for each point p, there are at most 2^d candidates in each orthant. Hence, there are at most 2^{2d} pairs in total involving p. □

3.2 Kinetic data structure for the closest pair

We now show how to maintain the candidate pairs with several kinetized MDRS trees. With the addition of a kinetic tournament [BGH97] on the distances defined by these pairs, we will have a KDS for the closest pair.

For each of the 2^d diagonal vectors σ, we build a kinetic MDRS tree $\mathcal{T}$ for S with the σ-orthant as the model cone and with a weight function f_σ that returns the $2^d - 1$ points of smallest σ-ordinate in the range (as in Section 2.1). With the help of this MDRS tree, the (at most) 2^d σ-candidate points of any point p can be found in time $O(2^d \log^d n)$, as they are simply $f_\sigma(S \cap orth_\sigma(p))$. Similarly, we can decide in the same amount of time whether a point q is a σ-candidate point of p or not.

Using these facts, we immediately obtain a static algorithm to compute the closest pair in time $O(n \log^d n)$ by first constructing the (static) multidimensional trees, then searching for all candidate points of each point in each orthant, and finally by finding the minimum distance amongst all candidate pairs. The two points realizing this distance are the closest pair by Lemma 2.

Furthermore, by the result of Section 2.1, we can maintain all search trees when the points are moving, as well as, for each point of S, the list of its candidate points along each diagonal vector. We will now have d range orderings (each one shared by several MDRS trees), and 2^d weight orderings (one for each diagonal vector).

If we were to throw into a tournament tree all pairs (p, q) such that q is a candidate point of p, the structure wouldn't be local, as a single point can be the candidate point of arbitrarily many points. This is the reason why the notion of candidate pair was introduced, as each point appears in at most 2^{2d} candidate pairs. Hence, in addition to the normal behavior of the MDRS trees, we need to be able to maintain the set of candidate pairs.

Lemma 3 *A pair (p, q) can become or cease to be a candidate pair only at times when the ordering of p or q changes in one of the $d + 2^d$ (range or weight) orderings.*

Proof. Omitted. □

Therefore, in addition to maintaining the MDRS trees, we can proceed as follows to maintain the set of candidate pairs. Let p and q be the points involved in an event; there are two cases.

- *The ordering of p, q along one of the axes changes.* In this case, let us assume that q moves from the σ_1-orthant of p to the σ_2-orthant (σ_1, σ_2 differ by only one coordinate), and oppositely for q. The difference between the set of σ_1-candidate points of p before and after the exchange can be computed by two queries on the MDRS tree for σ_1 immediately before and after updating it. If a point r disappears from the list of σ_1 candidate points, the pair (p, r) is removed from the list of candidate pairs (if it was in before). If r appears in this list, we can check whether p is a $-\sigma_1$ candidate point for r, and add (p, r) to the list of candidate pairs, if needed. The other three orthants where something happens can be handled similarly.

- *The ordering of p, q changes along a diagonal σ.* Let us assume that p passes above q along σ. In this case, we first update the MDRS tree. Then, for each point r such that (q, r) is a σ-candidate pair, we check whether q is still a $-\sigma$-candidate point of r. Finally, we check for all candidate points r of p whether p became a candidate point of r. In total, this can be done within $O(2^{2d} \log^d n)$ time. Note that a $-\sigma$ event happens at the same time that takes care of the $-\sigma$-orthant in the same way.

By the above algorithm, we have shown how to maintain the candidate pairs. We just need to build a kinetic tournament on the distances between each candidate pair of points [BGH97]. Note that the number of all the candidate pairs which can ever appear is at most $O(n^2)$. At any time, there is at most a linear number of candidate pairs. The following lemma follows from the results in [BGH97].

Lemma 4 *If each point follows a degree s algebraic trajectory in $\mathcal{R}^d$, the total number of events in the tournament tree is $O(n\lambda_{2s+2}(n) \log n)$, where $\lambda_{2s+2}(n)$ is the maximum length of a Davenport-Schinzel sequence with parameter $2s + 2$*[1].

This roughly quadratic bound shows that our structure is efficient, since the closest pair can itself change a quadratic number of times. Note also that a point q can be a σ-candidate for many points but it can be in a candidate pair with only 2^{2d} other points. This is what makes the above distance tournament a local KDS in terms of the original data (the MDRS trees are local as seen in the previous section).

Theorem 5 *The kinetic data structure for the closest pair problem is responsive, efficient, compact, and local.*

3.3 Dynamic simulation of moving balls

An interesting dynamic simulation problem is that of simulating the motion of rigid bodies under a physical model. Here, we will describe an application of the above data structure to simulating the motion of balls. In our model, there are moving balls in a bounding volume. Each ball moves in a fixed algebraic trajectory until it hits the boundary, or collides with another ball. Collisions are assumed to be perfectly elastic, or to obey some other standard physical law. Our task is to simulate the motion of balls for a given initial set of motions.

We can, of course, compute the time when a pair of balls may collide, or when a ball hits the boundary and put those events into a global queue. Once a collision happens, we can recompute all the events for the balls involved. However, this asks for $\Omega(n^2)$ event space and $\Omega(n)$ update time per event. But observe that just before two balls collide, they must be the closest pair — so it is enough to worry about collision among the closest pair of balls. For the balls with the same radius, the previous kinetic data structure for the closest pair solves the problem, when applied to the centers of the balls — the closest pair of the centers corresponds to the closest pair of balls. If we add the following two types of events

- the two closest balls collide, and
- a ball hits the boundary,

then we can use the previous structure to continue the simulation.

When a collision happens, we update all the certificates related to the two colliding balls (or the ball colliding with

[1] *The function $\lambda_{2s+2}(n)$* is nearly linear in n for any fixed s.

the wall). Because of the locality of the data structure, this can be done efficiently. In addition, the data structure takes roughly linear space.

Note that this approach does not work for balls with different radii because we can no longer just maintain the closest pair of centers. However, if the balls are of roughly similar size, say so that $r_{max}/r_{min} \leq \beta$, where r_{max}, r_{min} are the maximum and minimum radius of any ball, respectively, then the MDRS tree technique can be adapted, by keeping $(2\beta)^d$ candidates points for each center in each orthant. A similar packing lemma holds and we can show:

Lemma 6 *In a set of n balls where the maximum radius is β times the minimum radius, the closest pair is one of the $2^{2d}\beta^d n$ candidate pairs.*

Proof. Omitted. □

By using the same data structure, the moving balls with different but roughly similar radii can be simulated kinetically. More details will be given elsewhere.

4 Maintaining the MST of Moving Points

The *Euclidean minimum spanning tree* (EMST or L_2-MST) of a set S of n points in $\mathcal{R}^d$ is a spanning tree with minimum total edge length, where an edge length is defined by the Euclidean distance between its endpoints. We will also be interested in MSTs defined by the L_1 and L_∞ metrics, and more general polyhedral metrics as well. Given a distance function δ, we define the δ-MST to be the minimum spanning tree according to that distance.

For a constant $\epsilon > 0$, we say that a spanning tree is a $(1+\epsilon)$-EMST if its total edge length is within a factor of $(1+\epsilon)$ of the length of the true EMST. In this section, we will give kinetic data structures maintaining exact MSTs for certain polyhedral metrics and a $(1+\epsilon)$-EMST. We first focus on the structural results needed to approximate the EMST using several MDRS tree auxiliary structures.

4.1 Approximating the EMST by a polyhedral MST

To approximate the L_2 distance between two points we can project the points onto a line and measure the distance between their projections. If we choose sufficiently many and 'well-spaced' lines, we can make such an approximation arbitrarily good, as stated in the following lemma. This idea, which was also used by Vaidya [Vai84], is the basis of our method for approximating the Euclidean distance by a certain polyhedral metric and thus the EMST by a polyhedral MST. Similar constructions were employed in [AMS92] for the maximum spanning tree problem. However, the technical development given here is different.

In the following, for a point $p \in \mathcal{R}^d$, we denote $\vec{p}$ to be the vector from the origin to p. For any given α, $0 < \alpha < \pi$, we say that a set of unit vectors $\mathcal{V}$ is *α-well-spaced* if it satisfies the following properties:

Property 1 For any vector $\vec{p} \in \mathcal{R}^d$, there exists $\vec{v} \in \mathcal{V}$ such that the angle between $\vec{p}$ and $\vec{v}$ is no more than α, i.e. $\vec{v}\vec{p} \geq \|\vec{p}\| \cos\alpha$, where $\|\vec{p}\|$ is the L_2 norm of $\vec{p}$.

Property 2 The *Voronoi cone* $F_{\vec{v}}$ of $\vec{v} \in \mathcal{V}$, defined by $\{\vec{p} \,|\, \forall \vec{r} \in \mathcal{V},\ \vec{p}\vec{v} \geq \vec{p}\vec{r}\}$, is the intersection of a constant number (dependent on d) of halfspaces.

Property 3 For any $\vec{p}, \vec{q} \in F_{\vec{v}}$, let $\vec{r} = \vec{p} - \vec{q}$, then $\max_{\vec{v}\in\mathcal{V}}(\vec{r}\vec{v}) < \max(\vec{p}\vec{v}, \vec{q}\vec{v})$.

In section 4.2, we will show that in d dimensions, for any α, there exists a centrally symmetric set of α-well-spaced vectors that has cardinality $O(1/\alpha^{d-1})$. Such a set defines a polyhedral metric $\delta_\mathcal{V}$, with $\delta_\mathcal{V}(p,q) = \max_{\vec{v}\in\mathcal{V}}(\vec{p}-\vec{q})\vec{v}$. Intuitively, the well-spaced properties ensure that (1) the $\delta_\mathcal{V}$ metric is a good approximation of the L_2 norm, (2) the function $\delta_\mathcal{V}$ can be computed efficiently with a multidimensional range searching technique, and (3) the $\delta_\mathcal{V}$-MST can be built efficiently with a range searching technique as well.

Lemma 7 *For any $\epsilon > 0$, there exists a set $\mathcal{V}$ of d dimensional unit vectors, with $|\mathcal{V}| = O(\epsilon^{-(d-1)/2})$ such that, for all $p, q \in \mathcal{R}^d$,*

$$\delta_\mathcal{V}(p,q) \leq \|\vec{p}-\vec{q}\| \leq (1+\epsilon)\delta_\mathcal{V}(p,q),$$

where $d_\mathcal{V}$ is the polyhedral metric induced by $\mathcal{V}$.

Proof. Let $\alpha = \epsilon^{1/2}$ and let $\mathcal{V}$ be a corresponding set of α-well-spaced vectors, which has the required cardinality by Lemma 11. By Property 1 there exists $\vec{v} \in \mathcal{V}$ such that

$$\vec{v}(\vec{p}-\vec{q}) \geq \|\vec{p}-\vec{q}\| \cos\alpha$$

Since $\cos\alpha \approx 1 - \alpha^2/2$ for small α, we have

$$\begin{aligned}(\vec{p}-\vec{q})\vec{v} &\geq \|\vec{p}-\vec{q}\|\cos\alpha \\ &\geq (1-\epsilon/2)\|\vec{p}-\vec{q}\|\,.\end{aligned}$$

By the definition of $\delta_\mathcal{V}$, we have $\|\vec{p}-\vec{q}\| \leq (1+\epsilon)\delta_\mathcal{V}(p,q)$. In addition, as each vector in $\mathcal{V}$ is a unit vector, we have $\delta_\mathcal{V}(p,q) \leq \|\vec{p}-\vec{q}\|$. □

In the rest of this paper, we denote by $\mathcal{V}_\epsilon^d$ the set of well-spaced vectors of Lemma 7, and by δ_ϵ the distance function induced by this set.

Lemma 8 *For any set S in $\mathcal{R}^d$, the δ_ϵ-MST of S is a $(1+\epsilon)$-EMST of S.*

Proof. Let T_1 be the EMST of S and T_2 be its δ_ϵ-MST. Let $c(T)$, $c'(T)$ to be the total edge length of a tree T under the L_2 and δ_ϵ metric, respectively. Then, as T_2 is a δ_ϵ-MST, $c'(T_2) \leq c'(T_1)$. By Lemma 7, we have

$$c(T_2) \leq (1+\epsilon)c'(T_2) \leq (1+\epsilon)c'(T_1) \leq (1+\epsilon)c'(T_1)\,.$$

i.e., T_2 is within $(1+\epsilon)$ of the EMST in the L_2 norm. □

Hence, the $(1+\epsilon)$-EMST problem in $\mathcal{R}^d$ can be converted into the MST problem in the δ_ϵ metric. To compute the δ_ϵ-MST, as most efficient geometric MST algorithms do, we first construct a sparse graph on S which is guaranteed to contain all δ_ϵ-MST edges and then extract the MST from it by standard graph MST algorithms. Here, the sparse graph we are going to use is a variation on the *nearest geographic neighbor graph* [Yao82], also called the *local nearest neighbor graph (LNNG)* [KTI95].

As in the case of the closest pair, we say that, for a given $v \in \mathcal{V}_\epsilon^d$, point q is a *v-candidate point* of p if q is the point of smallest v-ordinate in the Voronoi cone of v translated to have its apex at p. Two points p, q form a candidate pair if they are candidate points of each other for some vectors in $\mathcal{V}_\epsilon^d$. We define our *LNNG* to be the set of candidate pairs. With an argument similar to that in [Yao82], we have:

Lemma 9 *The δ_ϵ-MST of S is a subgraph of the LNNG. The latter has $O(\epsilon^{-(d-1)/2}n)$ edges.*

Proof. (By contradiction) Suppose T is the δ_ϵ-MST of S and edge pq is in T but not in *LNNG*. Assume also that q is not a candidate point of p. As $\{F_{\vec{v}} \mid \vec{v} \in \mathcal{V}_\epsilon^d\}$ is a partition of the whole space, there exists $\vec{v} \in \mathcal{V}_\epsilon^d$ such that $q \in \vec{p} + F_{\vec{v}}$. By the assumption that q is not a candidate point of p, there is another point $r \in \vec{p} + F_{\vec{v}}$ such that $(\vec{r} - \vec{p})\vec{v} \leq (\vec{q} - \vec{p})\vec{v}$. By Property 3,

$$\max_{\vec{v} \in \mathcal{V}_\epsilon^d}((\vec{q} - \vec{r})\vec{v}) \leq \max\left((\vec{q} - \vec{p})\vec{v}, (\vec{r} - \vec{p})\vec{v}\right).$$

Because both q and r are in $\vec{p} + F_{\vec{v}}$, by the definition of δ_ϵ, we have

$$\delta_\epsilon(q, r) < \max\left(\delta_\epsilon(p, r), \delta_\epsilon(p, q)\right) = \delta_\epsilon(p, q).$$

Now, delete the edge pq from T, disconnecting T into two parts T_1, T_2. Suppose p is in T_1 and q in T_2. If r lies in T_1, then we connect T_1,T_2 by adding q, r. Otherwise, we connect by adding p, r. Either way, we will get another spanning tree of S, but with smaller total edge lengths, contradicting with that T is the δ_ϵ-MST of S. Thus, any edge of T must be in *LNNG*. The bound on the number of edges in *LNNG* follows immediately from the bound on the total number of cones in our construction. □

Hence, to construct the δ_ϵ-MST, we first build the *LNNG*, then find the MST of the *LNNG* by either a slightly superlinear deterministic algorithm ([GGST86]) or a linear randomized algorithm([KKT95]). Once again, the *LNNG* can be found by the multidimensional range search technique. In order to perform a d dimensional range search, we triangulate each Voronoi cone in constant time into a constant number of sub-cones (Property 2), bounded by d hyperplanes. All the *LNNG* edges can then be detected for each sub-cone in $F_{\vec{v}}$, by building one MDRS tree with $\vec{v}$ as the weight direction and querying each point in each of $O(\epsilon^{-(d-1)/2})$ MDRS trees.

Theorem 10 *The δ_ϵ-MST (which is also a $(1+\epsilon)$ EMST) can be found in time $O(\epsilon^{-(d-1)/2} n \log^{d-1} n)$.*

Proof. This follows from the above analysis and the complexity bounds for a static multidimensional range search tree. □

4.2 Construction of well spaced vectors

In this section, we construct a set of well-spaced vectors as defined in Section 4.1. We simply outline the construction in three dimensions and give indications for its extension to higher dimensions.

Consider a cube with side length $2\sqrt{3}/3$ (small enough so that it fits in the unit sphere) centered at the origin. Partition each of its faces, which are two-dimensional cubes, into a regular $n \times n$ mesh of patches. Define the *center vector* of a patch to be the vector directed from the origin to the center of the patch. We form a vector set $\mathcal{V}$ by normalizing to unit length the center vector of each patch.

It is easy to verify the following facts:

Lemma 11 *With the above construction,*

1. $|\mathcal{V}| = O(n^2)$.
2. *For any vector $\vec{v}$ whose projection on the cube lies in patch s, the angle between $\vec{v}$ and center vector of s is less than* $\arcsin(c/n)$ *for some constant c.*
3. *For any vector $\vec{v} \in \mathcal{V}$, the number of vectors in $\mathcal{V}$ whose angle with $\vec{v}$ is less than* $2\arcsin(c/n)$*, is bounded by a constant.*

For any given α, we claim that $\mathcal{V}$ is a well spaced vector set for some $n = O(1/\alpha)$. Indeed, Property 1 is ensured by facts 1 and 2 of Lemma 11, plus the fact that $\sin\alpha \approx \alpha$ for small α. Again by Lemma 11, we know that, for $\vec{v} \in \mathcal{V}$, any vector in its Voronoi cone has an angle with $\vec{v}$ that is less than $\arcsin(c/n)$. Thus, if $\vec{u} \in \mathcal{V}$ has an angle with $\vec{v}$ that is larger than $2\arcsin(c/n)$, the Voronoi cones of $\vec{u}$ and $\vec{v}$ can't share any plane. By fact 3 of Lemma 11, the complexity of each Voronoi cone is bounded by a constant. Therefore, Property 2 follows also. Property 3 is ensured by the fact that each cone is narrow enough, as argued in [Yao82].

Thus, for any given α, we can construct a well spaced vector set in three dimensions. The well spaced vector set in higher dimensions can be constructed in exactly the same way with a partitioning of the faces of the d dimensional hypercube (each one is a $d-1$ dimensional hypercube). The bounds still follow after some simple calculations, which are omitted here. Alas, the hidden constants are exponential in the dimension.

4.3 Maintaining a $(1+\epsilon)$-EMST of moving points

In this section, we kinetize the data structure of Theorem 10. By Lemma 9, in order to maintain the $(1+\epsilon)$-EMST, two structures are needed. One is used to maintain the local nearest neighbor graph *LNNG*. The other is used to maintain the MST of a graph under dynamic operations such as insertion and deletion of edges, as well as under continuous changes of the edge lengths involved.

To maintain the *LNNG*, we use our kinetized multidimensional range search tree. We construct an MDRS tree for each weight function and each sub-cone as in Section 2.1. These multidimensional search trees can be maintained kinetically as shown in Section 2.2. Moreover, with this data structure, the *LNNG* (i.e. the set of candidate pairs) can be maintained as in Section 3. Once again, it is the use of candidate pairs instead of candidate points which makes the structure local.

The MST of a graph is completely determined by the ordering of the edges by lengths. Thus, our kinetic data structure will maintain a list of edges ordered by length, and an event queue based on the associated kinetic certificates. There are three types of events that need to be processed in order to maintain the MST of the *LNNG*,

1. the insertion of an edge,
2. the deletion of an edge, and
3. the change of order between two *LNNG* edges in the length-ordered edge list.

The former two events are caused by the dynamic changes in the *LNNG*. We could handle them with the help of any fully dynamic MST algorithm, but such heavy machinery is unnecessary. As a range ordering event doesn't change the ordering of all pairwise distance, it cannot change the structure of the MST. Therefore, the deletion of an MST edge (p, q) can only happen in the following model situation: the point q was the v-candidate point of p for some v. Suddenly, the v-ordinate of q becomes greater than the v ordinate of a point r that also belongs to the Voronoi cone $\vec{p} + F_{\vec{v}}$. In this case, an argument similar to the proof of Lemma 9 shows that (p, r) is the new MST edge that replaces (p, q), and this

is also the only case where an edge becomes part of the MST as soon as it is inserted.

If we encounter an event of the third type, this is a signal that the MST may need to be changed. More precisely, if edge e_1 was shorter than edge e_2 just before they interchange order, this event changes the structure of the MST if and only if, just before this event, (1) e_1 is an MST edge, (2) e_2 is not an MST edge, and (3) e_1 is on the MST path connecting the two endpoints of e_2. This last condition can be detected and maintained within $O(\log n)$ time per operation using the link-cut tree data structure of [ST83].

Thus, we have a kinetic data structure to maintain the $(1+\epsilon)$-EMST. The performance of this data structure is analyzed and summarized as follows.

Number of internal events There are two types of events. One is caused by a change in (one of) the MDRS trees and the other is caused by the exchange in the length order of two edges in *LNNG*. The number of events of the first and second type is $O(\epsilon^{-(d-1)/2}n^2)$, because we have $O(\epsilon^{-(d-1)/2})$ linear orderings. To bound the number of events of the second type, as in [KTI95], we divide the MST history into $O(n)$ stages with each stage having $O(\epsilon^{-(d-1)/2}n)$ insertions and deletions of edges. Thus, within each stage, there are at most $O(\epsilon^{-(d-1)/2}n)$ edges which ever appear. In addition, for each pair of edges, they can exchange their orders by at most $O(1)$ times for constant degree algebraic point motions. Thus there are at most $O((\epsilon^{-(d-1)/2}n)^2) = O(\epsilon^{-(d-1)}n^2)$ events of third type in each stage. The total is $O(\epsilon^{-(d-1)}n^3)$.

Compactness Obviously, the space needed to store all the data structures is $O(\epsilon^{-(d-1)/2}n\log^{d-1} n)$. Therefore, our structure is compact.

Locality The structure is also clearly local.

Responsiveness Those events caused by maintaining the *LNNG* can be handled in $O(\log^d n)$ time. Those caused by maintaining the MST can be handled in $O(\log n)$ time.

According to above analysis, we have:

Theorem 12 *The above data structure maintains a $(1+\epsilon)$-EMST of moving points. If the points follow algebraic trajectories of fixed degree, there are $O(\epsilon^{-(d-1)}n^3)$ events. For each event, the structure can be updated within time $O(\log^d n)$. This KDS is compact, local and responsive.*

As mentioned in the introduction, we do not know if the above KDS for the MST is efficient. In any fixed dimension, it is known that the actual number of combinatorial changes to the real EMST for linearly moving points is $O(n^3 2^{\alpha(n)})$ [KTI95], but it is not known whether this bound is tight. Since the $(1+\epsilon)$-EMST is not uniquely defined, the number of combinatorial changes it undergoes has not been studied.

The kinetic data structure shown here can also be used to maintain exact MSTs under polyhedral metrics(for example, L_1 or L_∞). It is known that for linear motions the L_1- and L_∞-MSTs undergo $O(n^{5/2}\alpha(n))$ changes [KTI95]. Thus our structure is not as efficient as possible even in this case.

5 Conclusion and open problems

In this paper, we solved a number of classical proximity problems in the new kinetic setting of [BGH97]. We showed how to maintain a multidimensional range search tree, and how to use it to maintain the closest pair and an approximation of the minimum spanning tree of moving points.

In [BDIZ], an alternate *probabilistic* setting for judging the efficiency of kinetic data structures was proposed. In this setting, one computes the expected number of events processed when a kinetic data structure is run on points drawn independently at random from some prescribed distribution. It is easy to see that our KDS for the closest pair behaves very poorly according to this measure (it processes $\Theta(n^2)$ events in expectation). Can it be modified to process less events in expectation?

The results for the minimum spanning tree are very preliminary, especially as far as efficiency is concerned. A difficulty here is that tight bounds are still unknown for the number of changes of the MST itself for any type of metric. Even for a *fixed* (and even planar) graph with variable edge lengths the problem of kinetically maintaining an MST efficiently seems quite interesting. Finally, a kinetic data structure that can maintain the exact EMST in a responsive, compact, and local manner would also be highly desirable.

Acknowledgments

We wish to thank Aris Gionis and Tong Zhang for useful discussions, as well as an anonymous referee for helpful suggestions to simplify section 4.1.

References

[AMS92] Pankaj K. Agarwal, J. Matoušek, and S. Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Comput. Geom. Theory Appl.*, 1(4):189–201, 1992.

[AS89] C. Aragon and R. Seidel. Randomized search trees. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 540–545, 1989.

[BDIZ] J. Basch, H. Devarajan, P. Indyk, and L. Zhang. Probabilistic analysis for combinatorial functions of moving points. This volume.

[BGH97] J. Basch, L.J. Guibas, and J. Hershberger. Data structures for mobile data. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, page to appear, 1997.

[BS80] J. L. Bentley and J. B. Saxe. Decomposable searching problems I: Static-to-dynamic transformation. *J. Algorithms*, 1:301–358, 1980.

[GGST86] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6:109–122, 1986.

[KGS] Dong-Jin Kim, Leonidas J. Guibas, and Sung-Yong Shin. Fast collision detection among multiple moving spheres. This volume.

[KKT95] D. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42:321–328, 1995.

[KTI95] N. Katoh, T. Tokuyama, and K. Iwano. On minimum and maximum spanning trees of linearly moving points. *Discrete Comput. Geom.*, 13:161–176, 1995.

[Meh84] K. Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, volume 3 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1984.

[Mul94] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, Englewood Cliffs, NJ, 1994.

[Ove83] M. H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes in Computer Science.* Springer-Verlag, Heidelberg, West Germany, 1983.

[Smi92] M. Smid. Maintaining the minimal distance of a point set in polylogarithmic time. *Discrete Comput. Geom.*, 7:415–431, 1992.

[ST83] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–381, 1983.

[Vai84] P. M. Vaidya. A fast approximation for minimum spanning trees in k-dimensional space. In *Proc. 25th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 403–407, 1984.

[WL85] D. E. Willard and G. S. Lueker. Adding range restriction capability to dynamic data structures. *J. ACM*, 32:597–617, 1985.

[Yao82] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11:721–736, 1982.