Proximity Problems on Moving Points

Julien Basch Leonidas J. Guibas* Li Zhang

Computer Science Department
Stanford University
Stanford, CA 94305

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.

For the case of points in $\mathcal{R}^d$, both the closest pair and the MST problems have been extensively studied in computational geometry. 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 [BGH97a].

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 certain 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

* Support was provided in part by National Science Foundation grant CCR–9623851 and by US Army MURI grant 5–23542–A.

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. 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 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, and (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 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 [BGH97a].

The results of this paper are based on kinetizing (i.e., maintaining under continuous point motion) a certain type of *multidimensional range searching tree* (MDRS tree) [Meh84] used to compute a *decomposable function* [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 the n moving points. Though the structures we propose are fully on-line and work 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. Section 2 presents our kinetic MDRS tree and shows that it is efficient, compact, and local, a result which is of independent interest. Section 3 gives the first kinetic algorithm for closest pair in $\mathcal{R}^d$ under the Euclidean metric — via a structure which is also efficient, compact, and local. Our method is based on a certain packing lemma, and extends to moving balls only under the assumption that the radii of the balls involved have bounded ratios (an assumption which holds in practice). 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 any polyhedral metric (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 algorithms for the the MST are compact and local, but they are not as efficient as we might like.

In [BGH97a] an efficient, compact, and local kinetic data structure is given for the closest pair in $\mathcal{R}^2$. Although in principle the data structure proposed there may be generalizable to $\mathcal{R}^d$, the new closest pair structure proposed here is conceptually significantly simpler for two reasons: (1) it is based on a single global range searching tree and, (2) its kinetization depends only on maintaining the sorted order of the points along various directions. For a practical evaluation of the performance of the earlier structure, see [BGSZ]; the current structure has not yet been implemented. Another event-based algorithm for detecting the collision of moving balls was proposed in [KGS]. For high dimensions the performance of the algorithms given in this paper is good 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.

2 Kinetized Multidimensional Range Search Tree

In this section, we will describe our variant of a multidimensional range search tree and show how to kinetize it.

2.1 Multidimensional range searching

This is a standard topic in computational geometry and it is treated, for example, in [Meh84, Mul94]. The precise structure we will use is as follows. We are given a finite set $S = \{s_1, s_2, \ldots, s_n\}$ of n points in $\mathcal{R}^d$, and for each point in S we have a value or *rank*. We want to build a data structure on S which can be used to efficiently determine the *cost* of any subset of S specified by a *conical* range $\mathcal{C}$. Our ranks, costs, and ranges are defined in terms of projections of the the points of S along a fixed set of directions in $\mathcal{R}^d$.

For a direction δ (a point on S^{d-1} within $\mathcal{R}^d$), we define the rank of a point $s_i \in S$ according to δ to be the rank of s_i in the sorted order $\preceq_\delta$ of all the points of S when projected to a line with direction δ. The cost of a point will just be equal to its rank in this ordering, and the cost of a subset of S is defined to be smallest cost of any of its elements. For certain applications, however, we will define the cost of a given subset of S as the vector of the costs of the t points of smallest rank in the subset (for a constant t). We will denote by f the function mapping a subset of S to its (scalar or vector) cost. It is not hard to verify that such f are decomposable functions in the sense of [BS80].

Similarly, a direction δ and a point $p \in \mathcal{R}^d$ define two halfspaces in $\mathcal{R}^d$: $H_p^- = \{x \in \mathcal{R}^d | x \preceq_\delta p\}$ (negative halfspace) and $H_p^+ = \{x \in \mathcal{R}^d | x \succeq_\delta p\}$ (positive halfspace). We will use H_p to denote either of those two halfspaces. In addition to the direction specifying the rank, we assume we are given another set of k (again k is a constant) directions implying k linear orderings ($\preceq_1, \preceq_2$,..., $\preceq_k$) over S. A conical range, or *cone* for short, based at point p is a range of the form $C(p) = H_p^1 \cap H_p^2 \cap \cdots \cap H_p^k$, where the superscripts denote a (positive or negative) halfspace chosen according to the indicated order. Note that our cones are closed convex sets. We call the above k orderings the *range orderings* of S.

For a static set S, repeated median partitioning along the various linear orders 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 has size $O(n \log^{k-1} n)$ and depth $O(k + \log n)$.
- Each node i in $\mathcal{T}$, stores $f(T(i))$, where $T(i)$ is the set of all the elements in the subtree rooted at node i.
- Each element appears at most $O(\log^{k-1} n)$ times in the tree. For each conical range $\mathcal{C}$, there exist $m = O(\log^k n)$ nodes in $\mathcal{T}$, such that $\mathcal{C} = T(i_1) \cup T(i_2) \cup \cdots \cup T(i_m)$, and the $T(i_j)$ are disjoint.

Therefore, for any query range $C(p)$, $f(C(p))$ can be computed in two steps: first find the $m = O(\log^k n)$ nodes of which the subtrees constitute $C(p)$ and then compute

$$f(C(p)) = \min(f(T(i_1)), \cdots, f(T(i_m)))$$

in $O(m)$ time (for vector-valued f this means merge the lists and extract the t smallest elements of the result). This gives us an $O(\log^k n)$ time algorithm because of decomposability of f.

In the dynamic setting, we can allow the following operations:

- deletion of an element,
- insertion of an element, and

- update of $f(\{s\})$ (which can be viewed as deleting an element and then reinserting it with a new value)

By relaxing the equal-sized partitioning in the recursive definition of $\mathcal{T}$ to just a balanced partitioning, the above structure can be maintained within $O(\log^k n)$ time per operation. Another way to dynamize $\mathcal{T}$ is to do some local and global rebuilding after a few operations. This gives us an amortized bound of $O(\log^k n)$ per operation ([Meh84, Ove83]).

2.2 Kinetizing the tree $\mathcal{T}$

A kinetic version of the MDRS tree $\mathcal{T}$ is easy to obtain, because the 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 $k+1$ sorted lists, one for each of the k linear orders over S used in defining the cones, and one for the ordering defining the ranks.
- A global event queue $\mathcal{Q}$, to be used for processing the kinetic 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 of rank 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 $\mathcal{T}$. We omit details in this abstract. In addition, we update the sorted lists by exchanging the two elements and rescheduling the events for the newly adjacent pairs in the list, so there are at most $2(k+1)$ 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 rank orderings can only change when two adjacent points swap order. Suppose all the points in S follow algebraic trajectories of fixed degree. By using the time bounds for dynamic multidimensional search tree, we have:

Theorem 1 *The kinetized multidimensional range search tree is local, compact, and efficient. It can be maintained within amortized time $O(\log^k n)$ per event and, if the points follow algebraic trajectories of fixed degree, the total number of events will be $O(n^2)$.*

3 Maintaining the Closest Pair of Moving Points

We now show how an instance of the kinetic MDRS tree can be used to maintain the closest pair of the moving points in S (in the L_2 sense). We will use for range orderings the d orderings corresponding to the d principal axes of some orthonormal coordinate system for $\mathcal{R}^d$. 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.

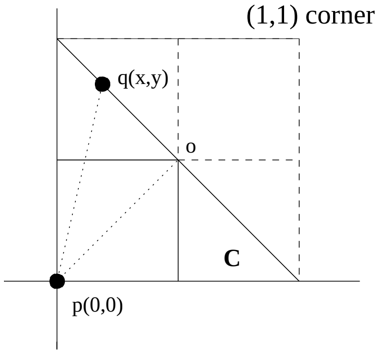

Figure 1: Partitioning of corner

3.1 A key 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 d dimensions, for each point p, the axis-aligned hyperplanes through p partition $\mathcal{R}^d$ into 2^d orthogonal parts ('orthants'). Formally, this can be formulated as follows:

For $p = (p_1, \ldots, p_d) \in \mathcal{R}^d$, define its σ-orthant, where $\sigma = (\varepsilon_1, \ldots, \varepsilon_d)$ with $\varepsilon_i \in \{-1, 1\}$, by:

$$Orth_\sigma(p) = \{(q_1, \ldots, q_d) \in \mathcal{R}^d \mid \forall i \; \varepsilon_i q_i \geq \varepsilon_i p_i\}$$

and its σ-ordinate by:

$$Ord_\sigma(p) = (\varepsilon_1 p_1 + \cdots + \varepsilon_d p_d)/d\,.$$

For each $p \in S$ and any orthant σ, sort the set $Orth_\sigma(p) \cap S$ by increasing σ-ordinate, and call the $2^d - 1$-smallest elements thus obtained the *σ-candidates* of p. For two points p and q, we say that q is a *candidate point* of p if q is *σ-candidate* of p for some σ. The pair (p, q) is *candidate pair* if p is a *candidate* of q and vice versa.

The key observation on the connection between the closest pair and candidate pairs is that:

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

Proof. Suppose that (p, q) is the closest pair of S and, w.l.o.g., place p at the origin and let q be in the $(1, \ldots, 1)$-orthant of p. See figure 1,

Let o be the points all of whose coordinates are equal to $Ord_{(1,\ldots,1)}(q)$. Through o, divide the region C enclosed by the axis-aligned hyperplanes and the hyperplane $x_1 + \cdots + x_d = d\,Ord_{(1,\ldots,1)}(q)$ into $2^d - 1$ cells. If q is not a candidate point of p in the $(1, \ldots, 1)$-orthant, there must be at least 2^d points (including point p itself) lying in C. Therefore, there are two of them lying within the same cell, which implies that their distance is less than $d(p, o)$ 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 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 and thus, each point can be in at most 2^{2d} candidate pairs. $\square$

In the following section, we will show how to maintain these candidate pairs with a kinetized MDRS tree. With the addition of a kinetic tournament [BGH97a] on the distances defined by these pairs, we will have a KDS for the closest pair.

3.2 Kinetic data structure for the closest pair

For every possible orthant type σ we proceed as follows. We build an MDRS tree $\mathcal{T}$ for S based on the d principal directions as the range directions and ranked according to the σ-ordinate. To maintain candidate pairs according to σ, we maintain a list of the smallest $2^d - 1$ points for each internal node in the MDRS tree. Thus our cost function f_σ for a subset $U \subseteq S$ is the sorted list of the $2^d - 1$ smallest elements in U w.r.t. Ord_σ. It is easy to see that f_σ is a decomposable function.

Moreover, point p is a candidate point of q iff $p \in f_\sigma(Orth_\sigma(q))$ where σ is the direction such that $p \in Orth_\sigma(q)$. The pair (p, q) is a candidate pair iff both $p \in f_\sigma(Orth_\sigma(q))$ and $q \in f_{-\sigma}(Orth_{-\sigma}(q))$. Thus, by the σ MDRS tree, the σ-candidates for each point p can be found within time $O(2^d \log^d n)$, by computing $f_\sigma(Orth_\sigma(p))$. So we immediately get a static algorithm to compute the closest pair in time $O(n \log^d n)$ by constructing the multidimensional trees at first and then searching for the candidate points of each point in each orthant. The pair with shortest distance among all such candidate pairs is the closest pair by Lemma 2. Furthermore, by the result in Section 2.1, we can maintain this search tree for moving points. We now show how to maintain candidate pairs by using these trees. Note that there are two types of events:

- exchange of the order of two points p and q along some coordinate direction, and
- exchange of the ranks of two points p and q along some rank direction.

Fortunately, in both cases, the following lemma holds:

Lemma 3 *When an order exchange event happens for points p and q, only those candidate pairs involving p and q may change.*

Proof. The easy proof is omitted in this abstract. □

Therefore, in addition to maintaining the MDRS trees, we can proceed as follows to maintain the set of candidate pairs. Again, let p and q be the points involved in an event.

- An axis coordinate exchange event can affect the structure of the candidate pairs in that p may leave/enter some orthant of q and vice versa. We can recompute $f_\sigma(Orth_\sigma(q))$ where σ is the direction such that p is originally in $Orth_\sigma(q)$. Then, for each point r in $f_\sigma(Orth_\sigma(q)))$, we identify whether q is also the candidate point of r within $O(2^d \log^d n)$ time. We do this for each of the 2^d orthants involved and thus, within time $O(2^{2d} \log^d n)$, the candidate pairs can be updated properly.

- For the second type event, we can compute $F = f_{-\sigma}(p) \cap f_{-\sigma}(q)$, where σ is the direction along which p, q exchange their ranks. Then we can update candidate pairs by, for each $r \in F$, deciding whether p, q are in $f_\sigma(r)$. In total, this can be done within $O(2^{2d} \log^d n)$ time as well.

Now, by the above algorithm, we can maintain all the candidate pairs. Then, we just need to build a kinetic tournament on the distances between each candidate pair of points as in [BGH97a]. We state the result without proof here:

Lemma 4 *The total number of events in the required tournament tree are $O(n\lambda_{2s+2}(n) \log n)$.*

Here s is a Davenport-Schinzel parameter capturing the number of times two distances can interchange values during the entire motion. This roughly quadratic bound shows that our structure is efficient, since the closest pair can itself change a quadratic number of times. Putting everything together, we have:

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

As a final remark, note that a point q can be a σ-candidate for many points p, but it can be in a candidate pair with only 2^{2d} other points p. This is what makes the above distance tournament a local KDS in terms of the original data.

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 collide, or when a ball hits the boundary and put those events into a global queue. Once a collision happens, then 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 events:

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

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 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 $\frac{r_{max}}{r_{min}} \leq \beta$, where r_{max}, r_{min} are the maximum and minimum radius of any ball, respectively, then the MDRS tree technique can still be made to work, by keeping $(2\beta)^d$ nearest candidates for each center in each orthant. A similar packing lemma holds and we can show:

Lemma 6 *The closest pair of balls must be one of such $2^{2d}\beta^d$ candidate pairs.*

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

4 Maintaining the MST of Moving Points

The *Euclidean minimum spanning tree* (EMST or L_2-MST) of a set $S = \{s_1, s_2, \ldots, s_n\}$ of n points in $\mathcal{R}^d$ is a spanning tree connecting all the points and having the minimum total edge cost, where the edge cost 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 other general polyhedral metrics as well. For a constant $\epsilon > 0$, we define an $(1+\epsilon)$-EMST to be a spanning tree with total edge cost within a factor of $(1+\epsilon)$ of the cost of the exact EMST. In this section, we will give kinetic data structures maintaining exact polyhedral MSTs and $(1+\epsilon)$-EMSTs. We first focus on the structural results needed to approximate the EMST using a MDRS tree auxiliary structure.

4.1 Approximating the EMST by a polyhedral MST

To approximate the L_2 distance between the 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 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.

Lemma 7 *For any $\epsilon > 0$, there exists a set of d dimensional unit vectors, $\mathcal{V}_\epsilon^d$ with $|\mathcal{V}_\epsilon^d| = O(\epsilon^{-(d-1)/2})$ such that for any vector $\vec{p} \in \mathcal{R}^d$, there exists $\vec{v} \in \mathcal{V}_\epsilon^d$ such that $\|\vec{p}\| \leq (1+\epsilon)\vec{p}\vec{v}$, where $\|\vec{p}\|$ is the L_2 norm of $\vec{p}$.*

Proof. For any $\alpha > 0$, we can always find $O((1/\alpha)^{d-1})$ vectors $\mathcal{V}_\alpha^d$, such that for any $\vec{p} \in \mathcal{R}^d$, there is $\vec{v} \in \mathcal{V}_\alpha^d$, so that the angle between $\vec{p}, \vec{v}$ is less than $(d-1)\alpha$. Therefore, if we set $\alpha = \epsilon^{1/2}/(d-1)$, since $\cos\alpha \approx 1 - \alpha^2/2$ for small α, we have $\vec{p}\vec{v} \geq \cos((d-1)\alpha)\|\vec{p}\| \geq (1-\epsilon/2)\|\vec{p}\|$, i.e., $\|\vec{p}\| \leq (1+\epsilon)\vec{p}\vec{v}$.

Now take $\mathcal{V}_\epsilon^d = \mathcal{V}_\alpha^d$ for $\alpha = \epsilon^{1/2}/(d-1)$, proving the lemma. □

We now show that approximating distances actually lets us approximate the length of the MST. Let $d_\epsilon = |\mathcal{V}_\epsilon^d| = O(\epsilon^{-(d-1)/2})$. We raise each point p in $\mathcal{R}^d$ into d_ϵ-dimensional space via the mapping

$$R(\vec{p}) = (\vec{p}\vec{v}_1, \vec{p}\vec{v}_2, \cdots \vec{p}\vec{v}_{d_\epsilon}) \text{ for each } \vec{v}_i \in \mathcal{V}_\epsilon^d\,.$$

By Lemma 7, we have

$$d_\infty(R(p_1), R(p_2)) \leq d_2(p_1, p_2) \leq (1+\epsilon) d_\infty(R(p_1), R(p_2)),$$

where $d_\infty(x, y) = \max_{1 \leq i \leq d} |x_i - y_i|$ is the L_∞ distance.

Define $R(S) = \{R(p) \mid p \in S\} \subset \mathcal{R}^{d_\epsilon}$. Let T_1, T_2 be the EMST of S and L_∞-MST of $R(S)$, respectively. Define also $c_2(T)$, $c_\infty(T)$ to be the total edge costs of T under the L_2 metric in $\mathcal{R}^d$ and the L_∞ metric in $\mathcal{R}^{d_\epsilon}$ respectively. Then $c_\infty(T_2) \leq c_\infty(T_1)$ because T_2 is L_∞-MST of S. Combining with the facts that $c_\infty(T) \leq c_2(T) \leq (1+\epsilon)c_\infty(T)$, we get

$$c_2(T_2) \leq (1+\epsilon)c_\infty(T_2) \leq (1+\epsilon)c_\infty(T_1) \leq (1+\epsilon)c_2(T_1)\,.$$

That is:

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

By Theorem 8, the $(1+\epsilon)$-EMST problem in $\mathcal{R}^d$ can be converted to a MST problem under L_∞ metric in $\mathcal{E}^{d_\epsilon}$ where $d_\epsilon = O(\epsilon^{-\frac{d-1}{2}})$. By Yao's nearest neighbor graph (NNG) construction [Yao82], we can compute an L_∞-MST by range search in time $O(n \operatorname{polylog}(n))$. But here, the exponent of the log function is a large constant depending on ϵ. However, note that our high-dimensional points actually live inside a d-dimensional subspace. After a more careful analysis, we can show that the Yao construction can be done within time $O(\epsilon^{-(d-1)/2} n \log^d n)$. By techniques similar to those in [Yao82], we create a *frame* of cones $\mathcal{F} = \{F_i | 1 \leq i \leq d_\epsilon\}$ satisfying:

Property 1 $\mathcal{F}$ consists of disjoint cones which partition the entire space $\mathcal{R}^d$. For each $F \in \mathcal{F}$, F is enclosed by constant number of hyperplanes, with the constant depending on d only.

Property 2 For any vector $\vec{p} \in F_i$, $\vec{p}\vec{v_i} = \max_{\vec{v} \in \mathcal{V}_\epsilon^d} |\vec{p}\vec{v}|$.

Property 3 For any $\vec{p}, \vec{q} \in F_i$, let $\vec{r} = \vec{p} - \vec{q}$, then $\max_{\vec{v} \in \mathcal{V}_\epsilon^d} (\vec{r}\vec{v}) \leq \max(\vec{p}\vec{v_i}, \vec{q}\vec{v_i})$

More concretely, the cones are constructed by creating, for each vector $\vec{v_i}$ in $\mathcal{V}_\epsilon^d$, bisector hyperplanes between $\vec{v_i}$ and every other vector in $\mathcal{V}_\epsilon^d$. Each hyperplane partitions the space into two halfspaces. If we call the part containing $\vec{v_i}$ *positive*, then F_i is the intersection of all the positive halfspaces. Among the $O(\epsilon^{-(d-1)/2} n)$ halfspaces, we just need to consider those created by vectors whose angle to $\vec{v_i}$ is $\leq 2(d-1)\alpha$ because, by Lemma 7, each $\vec{p} \in F_i$ has an angle to $\vec{v_i}$ which is less than or equal to $(d-1)\alpha$. Thus we have, for each vector $\vec{v}$, the number of vectors which have angle to $\vec{v}$ less than or equal to $2(d-1)\alpha$ is $O((2d)^d)$.

Let $\vec{p} + F_i = \{\vec{p} + \vec{v} | \vec{v} \in F_i\}$. Note that for any $q \in p + F_i$, by Property 2, $d_\infty(R(\vec{p}), R(\vec{q})) = \vec{v_i}(\vec{q} - \vec{p})$. We now create a nearest neighbor graph $NNG = (S, E)$ where S is the set of points and E is obtained as follows. For each point p and cone F_i, project every point in $p + F_i$ along direction v_i, and let the nearest one to p be q; then connect p and q. With the same argument as in [Yao82], we have

Theorem 9 *L_∞MST of $R(S)$ is a subgraph of the NNG.*

By performing multidimensional search for each subcone, the graph NNG can be constructed within time $O(\epsilon^{-(d-1)/2} n \log^d n)$. Using a standard graph MST algorithm for the *NNG*, we can compute L_∞-MST of $R(S)$, and thus, a $(1+\epsilon)$-EMST of S within the same time bound.

4.2 MSTs for polyhedral metrics

Note that we can use the tools developed above to compute an exact MST under any polyhedral metric. It suffices to build an MDRS tree whose range directions correspond to to the normals of the hyperplanes defined by the $(d-2)$-faces of the unit polyhedral ball and the origin. Each cone thus defined will 'see' only a single facet of the distance function, and thus within each cone the nearest neighbor can be found by a linear ranking function.

Below we show how to maintain these MSTs based on nearest neighbor graphs under point motion. We focus the explanation on the $(1+\epsilon)$-EMST, but all the comments apply equally well to the exact polyhedral MSTs.

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

To maintain the $(1+\epsilon)$-EMST, two structures are needed. One is used to maintain the nearest neighbor graph *NNG*. 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 weights involved.

To maintain an *NNG*, we can use our kinetized multidimensional range search tree. Construct an MDRS tree for each ranking function defined by a vector in $\mathcal{V}_\epsilon^d$. These multidimensional search trees can be maintained kinetically. Moreover, by this data structure, the *NNG* can be maintained in the same way as maintaining candidate pairs in Section 3. To ensure locality, note that there are only $O(1)$ changes of the *NNG* if (p, q) is restricted further by putting (p, q) is NNG only if q is a nearest neighbor of p *and* p of q.

In maintaining the MST of the *NNG*, there are three types of events, because the MST of a graph is completely determined by the linear order of the edge costs:

- an insertion of an edge,
- a deletion of an edge, and
- a interchange in the order of two edges in the total ordering of all edges according to cost.

The former two can be maintained within time $O(\log n)$ by standard dynamic MST structures [ST83]. The interchange can be detected by maintaining a sorted list of the costs of all the edges. Suppose edges e_1, e_2 exchange their orders and e_1 was shorter than e_2 before the event happened. Then the MST changes if and only if all the following are true (1) e_1 is a MST edge, (2) e_2 is not a 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 as well (by the methods of [ST83]).

Thus we have a kinetic data structure to maintain the $(1+\epsilon)$-EMST. The performance of this data structure is analyzed 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 of in the weight order of two in *NNG*. The number of of events of the first 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 process 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 second type in each stage. The total is $O(\epsilon^{-(d-1)}n^3)$.

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

Locality The structure is also clearly local.

According to above analysis, we have

Theorem 10 *The above data structure maintains a $(1+\epsilon)$-EMST of moving points properly. There are $O(\epsilon^{-(d-1)}n^3)$ events and for each event, the structure can be updated within time $O(\log^d n)$. Further, it is compact and local.*

As mentioned in the introduction, we do not know if the above kinetic MST structure is efficient. For $d = 3$, 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 if 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. It is known that for linear motions the L_1- and L_∞-MSTs undergo at most $O(n^{5/2}\alpha(n))$ changes [KTI95]. Thus our structure is not as efficient as possible: it process about $\Theta(\sqrt{n})$ more events than it should.

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.

[BGH97a] J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. In *Proc. 7th SIAM Symp. on Discr. Algorithms*, to appear, 1997.

[BGSZ] Julien Basch, Leonidas J. Guibas, Craig D. Silverstein, and Li Zhang. A practical evaluation of kinetic data structures. Manuscript. 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.

[KGS] Dong-Jin Kim, Leonidas J. Guibas, and Sung-Yong Shin. Fast collision detection among multiple moving spheres. Manuscript. To appear (1997).

[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.

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