Efficient Tradeoff Schemes in Data Structures for Querying Moving Objects

Pankaj K. Agarwal[1], Lars Arge[1], Jeff Erickson[2], and Hai Yu[1]

[1] Department of Computer Science, Duke University,
Durham, NC 27708-0129, USA
{pankaj,large,fishhai}@cs.duke.edu

[2] Department of Computer Science, University of Illinois,
Urbana, IL 61801-2302, USA
jeffe@cs.uiuc.edu

Abstract. The ability to represent and query continuously moving objects is important in many applications of spatio-temporal database systems. In this paper we develop data structures for answering various queries on moving objects and study tradeoffs between various performance measures — query time, size, and accuracy of the result.

1 Introduction

With the rapid advances in geographical positioning technologies, it now becomes feasible for spatial database systems to keep track of continuously moving objects accurately. In many applications, people would like to store these moving objects in a way such that certain queries on them can be answered quickly. For example, the mobile phone service provider may wish to know how many users are currently present in a specified area. In such kinetic scenarios, most of the traditional techniques fail to work because they usually assume that data do not change over time. The problem of efficiently representing and querying moving objects has been investigated by many researchers from database and computational geometry communities, and several new techniques have been developed recently [1, 7, 9, 16, 23].

The kinetic data structure framework (KDS) proposed by Basch *et al.* [9] has been successfully applied to a variety of geometric problems to efficiently maintain data structures for moving objects. The key observation of KDS is that, though the actual structure of the moving objects may change continuously over time, its combinatorical description changes only at discrete time instances. Although this framework has proven to be very useful for maintaining geometric structures, it is not clear whether KDS is the right framework if we simply want to query moving objects: on the one hand, there is no need to maintain the underlying structure explicitly, and on the other hand the KDS maintains a geometric structure on the current configuration of points while one may wish to answer queries on the past as well as the future configurations of points.

For example, one can use the KDS by Basch *et al.* [9] to maintain the convex hull of a set of linearly moving points. It processes roughly $O(n^2)$ events, each of which requires $O(\log^2 n)$ time. Using this structure, one can easily determine whether a query point lies in the convex hull of the current configuration of points. But suppose one is interested in the queries of the following form: "Does a point p lie in the convex hull of the point set at time t_q?" In order to answer this convex-hull query, there is no need to maintain the convex hull explictly, and we need a data structure that maintains convex hull at all times implicitly. The focus of this paper is to develop data structures for answering these types of queries and to study tradeoffs between various performance measures — query time, size, and accuracy of the result.

1.1 Problem statement

Let $S = \{p_1, p_2, \cdots, p_n\}$ be a set of moving points in $\mathbb{R}^1$ or $\mathbb{R}^2$. For any time t, let $p_i(t)$ be the position of p_i at time t, and let $S(t) = \{p_1(t), p_2(t), \cdots, p_n(t)\}$ be the configuration of S at time t. We assume that each p_i moves along a straight line at some fixed speed. We are interested in answering the following queries.

Q1. Given a y-range $R = [y_1, y_2] \subseteq \mathbb{R}^1$ and a time stamp t_q, report $S(t_q) \cap R$, i.e., all points of S that lie inside R at time t_q.

Q2. Given an axis-aligned rectangle $R \subseteq \mathbb{R}^2$ and a time stamp t_q, report $S(t_q) \cap R$, i.e., all points of S that lie inside R at time t_q.

Q3. Given a query point $p \in \mathbb{R}^2$ and a time stamp t_q, report an δ-approximate nearest neighbor $p' \in S$ of p at time t_q, such that $d(p, p'(t_q)) \leq (1 + \delta) \cdot \min_{q \in S} d(p, q(t_q))$.

Q4. Given a query point $p \in \mathbb{R}^2$, a time stamp t_q and a value δ, report an δ-approximate farthest neighbor $p' \in S$ of p at time t_q, such that $d(p, p'(t_q)) \geq (1 - \delta) \cdot \max_{q \in S} d(p, q(t_q))$.

Q5. Given a query point $p \in \mathbb{R}^2$ and a time stamp t_q, a *convex-hull query* asks whether p lies inside the convex hull of $S(t_q)$.

1.2 Previous results

Range trees and kd-trees are two widely used data structures for orthogonal range queries. People have developed kinetic algorithms for both structures: Two-dimensional kinetic range trees were proposed in [1] with $O(n \log n / \log \log n)$ space and $O(\log n + k)$ query time[3]; Agarwal *et al.* [4] developed kinetic data structures for two variants of the standard kd-tree, pseudo kd-tree and overlapping kd-tree, that can answer queries in $O(n^{1/2+\varepsilon} + k)$ time. Based on partition trees [17], Kollios *et al.* [16] proposed a linear-size data structure for answering one-dimensional range queries of type Q1 in $O(n^{1/2+\varepsilon} + k)$ time. This result was

[3] Some of the previous work described in this section was actually done in the standard two-level I/O model aiming to minimize the number of I/Os. Here we interpret and state their results in the standard internal-memory setting.

later extended to $\mathbb{R}^2$ by Agarwal *et al.* [1] with the same space and query time bounds.

Agarwal *et al.* [1] were the first to propose *time-responsive* data structures. The time-responsive data structure optimizes the structure for the time close to the current time, and only stores a rough approximation for the time far away in the future. It has the nice property that queries on recent future configurations can be answered much more quickly than on distant future ones. Agarwal *et al.* [1] developed a time-responsive data structure for Q1 and Q2 queries whose query time is a monotonically increasing function of the difference between the query time stamp t_q and the current time, and is bounded by $O(n^{1/2+\varepsilon} + k)$ in the worst case. However, no exact bounds on the query time were known except for a few special cases. Time-responsive data structures with provable query time bounds were developed later on in [2], based on a fairly complicated technique. The size of the structure is $O(n \log n)$, and the query time is $O(\varphi(t_q)/n + \log n + k)$ and $O(\sqrt{\varphi(t_q)} + \sqrt{n} \log n / \sqrt{\lceil \varphi(t_q)/n \rceil} + k)$ for Q1 and Q2 queries respectively, where t_q is the query time stamp and $\varphi(t_q) \le O(n^2)$ is the number of kinetic events between t_q and the current time.

Kollios *et al.* [15] described a data structure for answering one-dimensional nearest-neighbor queries, but did not give any bound on the query time. Later, an efficient structure was given by Agarwal *et al.* [1] to answer δ-approximate nearest-neighbor queries in $O(n^{1/2+\varepsilon}/\sqrt{\delta})$ time using $O(n/\sqrt{\delta})$ space. Their data structure can also be used to answer δ-approximate farthest neighbor queries, for some fixed δ. More practical algorithms to compute exact and approximate k-nearest-neighbors were proposed by Procopiuc *et al.* [19] based on a data structure called STAR-tree.

Basch *et al.* [9] described how to maintain the convex hull of a set of moving points in $\mathbb{R}^2$ under the KDS framework. They showed that each kinetic event can be handled in logarithmic time, and the total number of kinetic events is $O(n^2)$ if points move linearly. With the kinetic convex hull at hand, one can easily answer a convex-hull query in $O(\log n)$ time. However, as noted earlier, the kinetic convex hull can only answer queries on current configurations.

1.3 Our results

In this paper we present data structures for answering queries of type Q1–Q5. A main theme of these data structures is to obtain various tradeoffs: time-responsive data structures in which query time dependes on the value of t_q; tradeoff between query time and the accuracy of the result; and tradeoff between query time and the size of the data structure.

In Section 2, we describe a new time-responsive data structure for orthogonal range queries in $\mathbb{R}^1$ and $\mathbb{R}^2$. It uses $O(n^{1+\varepsilon})$ space and can answer Q1 or Q2 queries in $O(\lceil \varphi(t_q)/n \rceil^{1/2+\varepsilon} \cdot \text{polylog } n + k)$ time. To appreciate how this new query time improves over those of previously known data structures [2] (see Section 1.2 for previous bounds), let us examine two extreme cases. For near future queries (i.e., $\varphi(t_q) = O(n)$), previous structures answer Q1 and Q2 queries in $O(\log n)$ and $O(\sqrt{n} \log n)$ time respectively, while our structures answer both queries in $O(\text{polylog } n)$ time. For distant future queries (i.e., $\varphi(t_q) = \Omega(n^2)$), previous structures answer both Q1 and Q2 queries in $O(n)$ time, which is no better than the naïve method, while our structures provide a query time bounded by roughly $O(\sqrt{n})$. Using this result, we also obtain the first time-responsive structure for approximate nearest-neighbor queries.

In Section 3, we present a data structure for answering δ-approximate farthest-neighbor queries, which provides a tradeoff between the query time and the accuracy of the result. In particular, for any fixed parameter $m > 1$, we build a data structure of size $O(m)$ in $O(n + m^{7/4})$ time so that a δ-approximate farthest-neighbor query can be answered in $O(1/\delta^{9/4+\varepsilon})$ time, given a parameter $\delta > 0$ as a part of the query.

In Section 4, we present data structures for answering convex-hull queries. We obtain data structures that produce a continuous tradeoff between space and query time. In particular, we obtain two data structures: one with linear size, and the other with logarithmic query time. We then present a data structure that provides a tradeoff between efficiency and accuracy for approximate convex-hull queries, following the general scheme discussed in Section 3.

2 Time-Responsive Data Structures

In this section, we describe our time-responsive data structures for orthogonal range reporting in $\mathbb{R}^1$ and $\mathbb{R}^2$.

2.1 One-dimensional data structure

Preliminaries. Let L be a set of n lines in $\mathbb{R}^2$. A $(1/r)$-*cutting* for L within a simplex Δ is a triangulation of Δ such that each triangle of the triangulation intersects at most n/r lines in L. Chazelle [12] described a deterministic algorithm for computing a $(1/r)$-cutting of optimal size $O(r^2)$ in $O(nr)$ time. The basic idea of his algorithm is to refine cuttings by composition. Consequently, the whole computation process of the algorithm can be naturally represented as a tree $\mathcal{T}$, referred to as the *cutting tree*. Each node v of $\mathcal{T}$ is associated with a triangle Δ_v, and the root of $\mathcal{T}$ is associated with Δ. For any internal node v, the set of triangles associated with its child nodes is a triangulation of Δ_v, which forms a $(1/r_0)$-cutting for the partial arrangement of L lying inside Δ_v, where r_0 is a sufficiently large constant. The height of $\mathcal{T}$ is $O(\log_{r_0} r) = O(\log r)$, and the set of triangles associated with the leaves of the tree is the $(1/r)$-cutting for L within Δ. We refer to this cutting as a *hierarchical cutting*. An important fact about the cutting, which was implicitly proved in [12], is that its size depends on the complexity of the arrangment of L within Δ.

Lemma 1 ([12]). *Let Δ be a triangle in $\mathbb{R}^2$, L be a set of n lines intersecting Δ, and $\varepsilon > 0$ be a constant. A hierarchical $(1/r)$-cutting of size $O(r^{1+\varepsilon} + \kappa r^2/n^2)$ for L inside Δ can be computed in $O(nr^\varepsilon + \kappa r/n)$ time, where κ is the number of intersections of L inside Δ.*

Overall structure. We interpret time t as an additional dimension and consider the trajectories of points of S in the ty-plane. Since the velocities of the points are fixed, each of them traces out a line in the ty-plane. Let $L = \{\ell_1, \ell_2, \cdots, \ell_n\}$ be the resulting set of lines, and $\mathcal{A}(L)$ the arrangement of L. Now a Q1 query is equivalent to a vertical-segment stabbing query, i.e., reporting all lines in L that intersect a query vertical segment $\{t_q\} \times [y_1, y_2]$.

Each vertex in $\mathcal{A}(L)$ corresponds to a change of y-ordering for a pair of points in S, which we refer to as a *kinetic event*. Similar to the approach by Agarwal *et al.* [2], we divide the ty-plane into $O(\log n)$ vertical slabs as follows. Suppose $t = 0$ is the current time. Let us consider the partial arrangement of $\mathcal{A}(L)$ that lie in the halfplane $t \geq 0$ (the other side of the arrangement can be handled symmetrically). Let τ_i be the x-coordinate of the $(2^i \cdot n)$-th leftmost vertex in that partial arrangement, $1 \leq i \leq \lceil \log_2 n \rceil$. Using an optimal slope selection algorithm [14], $\tau_1, \tau_2, \cdots$ can be computed in $O(n \log^2 n)$ time. We then define the first slab $\mathcal{W}_1$ to be $[0, \tau_1] \times \mathbb{R}$, and the i-th slab $\mathcal{W}_i$ to be $[\tau_{i-1}, \tau_i] \times \mathbb{R}$, for $i \geq 2$. Note that the number of kinetic events inside the first i slabs is bounded by $O(2^i \cdot n)$. For each $\mathcal{W}_i$, we construct a *window data structure* $\mathcal{WI}_i$ described below, for answering vertical-segment stabbing queries within $\mathcal{W}_i$. Our overall one-dimensional data structure consists of these $O(\log n)$ window data structures.

To answer a Q1 query, we first let $\xi = \{t_q\} \times [y_1, y_2]$, and locate the slab $\mathcal{W}_i$ that contains ξ. We then use the corresponding window data structure $\mathcal{WI}_i$ to report all lines in L that intersect ξ.

Window data structure. Before we describe the window data structure $\mathcal{WI}_i$ for answering general vertical-segment stabbing queries in $\mathcal{W}_i$, we first show how to construct a data structure $\mathcal{WT}_i$ to handle the case in which the query segment is of the form $\{t_q\} \times (-\infty, y_2]$.

Let $r = n/2^i$. We first construct a $(1/r)$-cutting Ξ inside $\mathcal{W}_i$ using Lemma 1. Let us denote by $\mathcal{T}_\Xi$ the cutting tree of Ξ; $\mathcal{T}_\Xi$ forms the top part of the tree $\mathcal{WT}_i$ (see Figure 1). For any node u in $\mathcal{T}_\Xi$, we denote by L_u the set of lines in L that intersect Δ_u (recall that Δ_u is the triangle associated with the node u). For each leaf node v of $\mathcal{T}_\Xi$, we build a partition tree [17], denoted by $\mathcal{T}_v$, for the set of points dual to the lines in L_v. These partition trees form the bottom part of $\mathcal{WT}_i$. Finally, let $p(u)$ denote the parent node of a node u in $\mathcal{T}_\Xi$. At each node u of $\mathcal{T}_\Xi$ (except for the root), we store a set $J_u \subseteq L_{p(u)}$ of lines that lie below Δ_u. This completes the construction of $\mathcal{WT}_i$.

The cutting tree $\mathcal{T}_\Xi$ requires $O(r^{1+\varepsilon_1} + \kappa r^2/n^2)$ space, where $\kappa = O(2^i \cdot n)$ is the complexity of $\mathcal{A}(L)$ within $\mathcal{W}_i$. There are $O(r^{1+\varepsilon_1} + \kappa r^2/n^2)$ partition trees in $\mathcal{WT}_i$, each of size $O(n/r)$. Therefore the total size of the partition trees is bounded by $O(r^{1+\varepsilon_1} + \kappa r^2/n^2) \cdot O(n/r) = O(nr^{\varepsilon_1} + \kappa r/n)$. Since $J_u \subseteq L_{p(u)}$, the size of J_u for a node u at the j-th level of $\mathcal{T}_\Xi$ is bounded by $O(n/r_0^{j-1})$ (recall that r_0 is the sufficiently large constant in the construction of the cutting). The

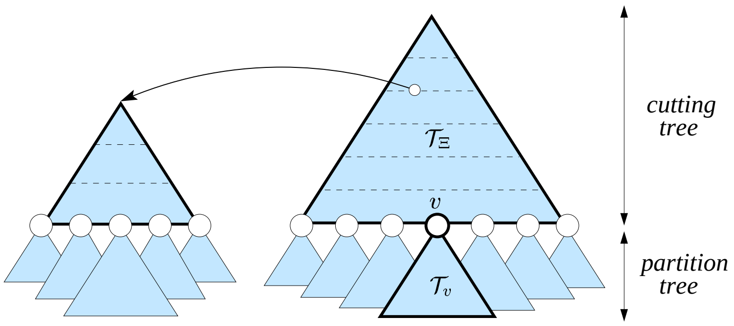

Fig. 1. The window data structure for one slab in the one-dimensional time-responsive data structure.

total size of J_u's associated with the nodes of $\mathcal{T}_\Xi$ is thus bounded by

$$\sum_{j=1}^{\log_{r_0} r} O\left(r_0^{j(1+\varepsilon_1)} + \kappa r_0^{2j}/n^2\right) \cdot (n/r_0^{j-1}) = O(nr^{\varepsilon_1} + \kappa r/n).$$

Summing up the above three quantities and substituting the values of r and κ, we conclude that the size of $\mathcal{WT}_i$ is $O(n^{1+\varepsilon_1}/2^{i\varepsilon_1})$. The time for constructing $\mathcal{WT}_i$ is $O(n^{1+\varepsilon_1}\log n/2^{i\varepsilon_1})$, which can be computed similarly.

Given a query segment $\xi = \{t_q\} \times (-\infty, y_2]$ in $\mathcal{W}_i$, we answer the stabbing query using $\mathcal{WT}_i$ as follows. We search down the tree $\mathcal{WT}_i$ in a natural manner to locate the leaf node v of $\mathcal{T}_\Xi$ whose associated triangle Δ_v contains the point (t_q, y_2). Whenever we reach a node u of $\mathcal{T}_\Xi$, we report all lines in J_u. When we reach the leaf node v, we can use the partition tree $\mathcal{T}_v$ to report all lines in L_v that lie below (t_q, y_2), as it is just a two-dimensional halfspace range query in the dual (the reason we use partition trees for answering halfspace range query is to accommodate the general vertical-segment stabbing query described below). It is straightforward to verify that the set of reported lines are indeed the set of lines in L intersected by ξ.

We spend $O(\log r) = O(\log n)$ time to locate the leaf node v, and then $O((n/r)^{1/2+\varepsilon_2}) = O(2^{i(1/2+\varepsilon_2)})$ time for querying the parition tree $\mathcal{T}_v$. Taking into account the time for reporting, the total time spent in answering a query is bounded by $O(\log n + 2^{i(1/2+\varepsilon_2)} + k)$, where k is the number of reported lines.

Lemma 2. *The structure $\mathcal{WT}_i$ can answer the Q1 query with a given y-range $R = (-\infty, y_2]$ in $O(\log n + 2^{i(1/2+\varepsilon_2)} + k)$ time, where k is the number of reported points. It uses $O(n^{1+\varepsilon_1}/2^{i\varepsilon_1})$ space and can be constructed in $O(n^{1+\varepsilon_1}\log n/2^{i\varepsilon_1})$ time.*

For each node u of $\mathcal{T}_\Xi$ (except for the root), let $\overline{J}_u \subseteq L_{p(u)}$ be the set of lines that lie *above* Δ_u. Notice that if we associate $\overline{J}_u$ instead of J_u to each node u, we then have a data structure $\overline{\mathcal{WT}}_i$ that can answer vertical-segment stabbing queries of the form $\{t_q\} \times [y_1, +\infty]$.

We are now ready to describe the window data structure $\mathcal{WI}_i$. It is constructed by modifying $\mathcal{WT}_i$ as follows. Instead of associating J_u directly to a node u of $\mathcal{T}_\Xi$, we build a secondary data structure for J_u and associate it to u. This secondary structure is similar to $\overline{\mathcal{WT}}_i$, except that it is constructed within the triangle $\Delta_{p(u)}$, and the cutting used in the structure is a $(2^i/|J_u|)$-cutting. The query on $\mathcal{WI}_i$ can be performed in a standard manner.

Using the general framework for multi-level structures, see, e.g. [3], we can prove that the size of $\mathcal{WI}_i$ is $O(n^{1+\varepsilon_1}\log n/2^{i\varepsilon_1})$, its preprocessing time is $O(n^{1+\varepsilon_1}\log^2 n/2^{i\varepsilon_1})$, and the query time is $O(\log^2 n + 2^{i(1/2+\varepsilon_2)}\log n + k)$.

Returning to the overall one-dimensional data structure, recall that it consists of $O(\log n)$ independent window structures. Therefore the overall structure has size $\sum_i O(n^{1+\varepsilon_1}\log n/2^{i\varepsilon_1}) = O(n^{1+\varepsilon_1}\log n)$, and can be constructed in total time $O(n\log^2 n)+\sum_i O(n^{1+\varepsilon_1}\log^2 n/2^{i\varepsilon_1}) = O(n^{1+\varepsilon_1}\log^2 n)$. The query performance of the whole data structure deteriorates as time passes by. Therefore, we update the data structure periodically in order to restore its performance. More precisely, we rebuild the entire structure after every $O(n)$ kinetic events. The amortized cost for each event is bounded by $O(n^{\varepsilon_1}\log^2 n)$.

Let $\varphi(t_q)$ be the number of kinetic events (i.e., the number of swaps in y-ordering of the points) between t_q and the current time. Putting everything together, choosing a parameter $\varepsilon > \max\{\varepsilon_1, \varepsilon_2\}$, and noticing that $2^i = O(\lceil\varphi(t_q)/n\rceil)$ for $t_q \in [\tau_{i-1}, \tau_i]$, we obtain the following theorem.

Theorem 1. *Let S be a set of n linearly moving points in $\mathbb{R}^1$. A data structure of size $O(n^{1+\varepsilon})$ can be built in $O(n^{1+\varepsilon})$ time and maintained in amortized $O(n^\varepsilon)$ time per kinetic event such that a Q1 query on S can be answered in $O(\log^2 n + \lceil\varphi(t_q)/n\rceil^{1/2+\varepsilon}\log n + k)$ time, where k is the number of reported points, t_q is the time stamp of the query, and $\varphi(t_q)$ is the number of kinetic events between t_q and the current time.*

2.2 Two-dimensional data structure

Let $L = \{\ell_1, \ell_2, \cdots, \ell_n\}$ be the set of lines traced out by points of S in the txy-space. Given a time stamp t_q and a rectangle $R = R_x \times R_y$, where $R_x = [x_1, x_2]$ and $R_y = [y_1, y_2]$, the corresponding Q2 query is equivalent to reporting all lines in L that intersect a horizontal rectangle $\{t_q\} \times R$ in the txy-space. Let ℓ_i^x and ℓ_i^y be the projections of ℓ_i onto the tx-plane and ty-plane respectively, for $1 \le i \le n$. We then let $L_x = \{\ell_1^x, \ell_2^x, \cdots, \ell_n^x\}$ and $L_y = \{\ell_1^y, \ell_2^y, \cdots \ell_n^y\}$.

In the two-dimensional case, each vertex of $\mathcal{A}(L_x)$ or $\mathcal{A}(L_y)$ is regarded as a kinetic event. Similar to the one-dimensional structure as described in the previous section, we divide the txy-space into $O(\log n)$ slices along the t-axis such that the number of kinetic events inside the i-th slice is $O(2^i \cdot n)$, and then build a *slice data structure* for each of these slices.

To build one slice data structure, observe that for any $\ell_i \in L$, ℓ_i intersects $\{t_q\} \times R$ if and only if ℓ_i^x intersects $\{t_q\} \times R_x$ and ℓ_i^y intersects $\{t_q\} \times R_y$. Thus we can build a four-level data structure similarly to the way we build the window structure $\mathcal{WI}_i$ in previous section. More specifically, we first build a two-level window data structure for L_x as described in Section 2.1, with one minor modification: all partition trees in the structure are replaced by multilevel partition trees as discussed in [1]. For each node u of the cutting tree in the secondary data structure, define $J'_u = \{\ell_i^y \mid \ell_i^x \in J_u\}$ (recall that J_u is the set of lines that intersect $\Delta_{p(u)}$ and lie above Δ_u). We then build another two-level window data structure for J'_u and associate it to the node u.

Theorem 2. *Let S be a set of n linearly moving points in $\mathbb{R}^2$. A data structure of size $O(n^{1+\varepsilon})$ can be built in $O(n^{1+\varepsilon})$ time and maintained in amortized $O(n^\varepsilon)$ time per kinetic event such that a Q2 query on S can be answered in $O(\lceil\varphi(t_q)/n\rceil^{1/2+\varepsilon} \cdot \operatorname{polylog} n + k)$ time, where k is the number of reported points, t_q is the time stamp of the query, and $\varphi(t_q)$ is the number of kinetic events between t_q and the current time.*

Combining the above result and the techniques developed in [1], we can construct a time-responsive data structure for answering δ-approximate nearest-neighbor queries[4]. Omitting further details, we conclude with the following theorem.

Theorem 3. *Let S be a set of n linearly moving points in $\mathbb{R}^2$. A data structure of size $O(n^{1+\varepsilon}/\sqrt{\delta})$ can be built in $O(n^{1+\varepsilon}/\sqrt{\delta})$ time and maintained in amortized $O(n^\varepsilon/\sqrt{\delta})$ time per kinetic event so that a Q3 query on S can be answered in $O(\lceil\varphi(t_q)/n\rceil^{1/2+\varepsilon}\operatorname{polylog}(n)/\sqrt{\delta})$ time, where t_q is the time stamp of the query and $\varphi(t_q)$ is the number of kinetic events between t_q and the current time.*

3 Approximate Farthest-Neighbor Queries

In this section, we describe a data structure for answering Q4 queries that achieves a tradeoff between efficiency and accuracy. The query time only depends on how much accuracy is required for a particular query. The general scheme is very simple, but it crucially relies on the notion of core-sets introduced by Agarwal *et al.* [6]. We first prove a lemma about farthest neighbors.

Lemma 3. *Let $S = \{p_1, \cdots, p_n\}$ be a set of n points in $\mathbb{R}^d$. For any $0 < \delta < 1$, a subset $Q \subseteq S$ of size $O(1/\delta^{(d+1)/2})$ can be computed in $O(n + 1/\delta^{d+1})$ time so that given any point $q \in \mathbb{R}^d$, we have*

$$(1-\delta) \cdot \max_{p \in S} d(p,q) \le \max_{p \in Q} d(p,q) \le \max_{p \in S} d(p,q).$$

Proof. We define $f_i : \mathbb{R}^d \to \mathbb{R}$ to be $f_i(q) = (d(p_i,q))^2$, and let $f'_i(q) = -f_i(q)$. Let $\mathcal{F} = \{f_1, \cdots, f_n\} \bigcup \{f'_1, \cdots, f'_n\}$. Notice that $\mathcal{F}$ is a family of d-variate polynomials that admits a linearization of dimension $d+1$. One can compute in

[4] Of course, this result also holds for approximate farthest neighbor queries. However, it will be immediately improved by Lemma 4 of the next section.

$O(n + 1/\delta^{d+1})$ time a subset $\mathcal{G} \subseteq \mathcal{F}$ of size $O(1/\delta^{(d+1)/2})$ that δ-approximates the *extent* of $\mathcal{F}$ [6, 11], i.e., for any $q \in \mathbb{R}^d$,

$$(1-\delta) \cdot \left(\max_{f \in \mathcal{F}} f(q) - \min_{f \in \mathcal{F}} f(q) \right) \le \max_{f \in \mathcal{G}} f(q) - \min_{f \in \mathcal{G}} f(q).$$

Let $Q = \{p_i \mid f_i \in \mathcal{G} \text{ or } f'_i \in \mathcal{G}\}$. Then for any $q \in \mathbb{R}^d$, we have

$$\begin{aligned}
\max_{p \in Q} d(p,q) &\ge \left((\max_{f \in \mathcal{G}} f(q) - \min_{f \in \mathcal{G}} f(q))/2 \right)^{1/2} \\
&\ge \left((1-\delta) \cdot (\max_{f \in \mathcal{F}} f(q) - \min_{f \in \mathcal{F}} f(q))/2 \right)^{1/2} \\
&\ge (1-\delta) \cdot \left(\max_{f \in \mathcal{F}} f(q) \right)^{1/2} \\
&= (1-\delta) \cdot \max_{p \in S} d(p,q).
\end{aligned}$$

Therefore, Q satisfies the desired property, which completes the proof. □

The subset Q in the above lemma is referred to as a δ-*kernel* for δ-approximate farthest neighbors (under L_2 metric). Similarly, one can prove that such a core-set also exists under L_1 or L_∞ metric. The size of the core-set becomes $O(1/\delta^{d/2})$ under L_1 metric and $O(1/\delta^{1/2})$ under L_∞ metric in $\mathbb{R}^d$. We can further prove the following result for a set of moving points. We omit the proof since it is almost identical to the one given above.

Lemma 4. *Let $S = \{p_1, \cdots, p_n\}$ be a set of n points moving linearly in $\mathbb{R}^d$. For any $0 < \delta < 1$, a subset $Q \subseteq S$ of size $O(1/\delta^{d+3/2})$ can be computed in $O(n + 1/\delta^{2d+3})$ time so that given any time $t \in \mathbb{R}$ and any point $q \in \mathbb{R}^d$, we have*

$$(1-\delta) \cdot \max_{p \in S} d(p(t), q) \le \max_{p \in Q} d(p(t), q) \le \max_{p \in S} d(p(t), q).$$

As we mentioned in Section 1.2, for a set of n linearly moving points in $\mathbb{R}^2$, one can build in $O(n \log(n)/\sqrt{\delta})$ time a data structure of size $O(n/\sqrt{\delta})$ that can answer δ-approximate farthest-neighbor queries in $O(n^{1/2+\varepsilon}/\sqrt{\delta})$ time, for any fixed δ. To achieve tradeoff between efficiency and accuracy, we do as follows. Let $m > 1$ be a fixed parameter, and let $\delta_i = (2^i/m)^{1/4}$, for $0 \le i < \log m$. Let S_i be a δ_i-kernel of S. A straightforward application of Lemma 4 with $d = 2$ shows that all S_i's for $0 \le i < \log m$ can be computed in $O(n \log m + m^{7/4})$ total time. One can reduce the time complexity to $O(n + m^{7/4})$ by first computing a δ_0-kernel S_0 of S and then computing a $(\delta_i - \delta_{i-1})$-kernel S_i of S_{i-1} for $1 \le i < \log m$. For each S_i, we then build a data structure for answering δ_i-approximate farthest-neighbor queries on S_i. The size of one such data structure is $O(|S_i|/\sqrt{\delta_i}) = O(m/2^i)$; as such, the total size of these data structures is $O(m)$. Given a δ-approximate farthest-neighbor query, where $(1/m)^{1/4} \le \delta < 1$, we first find an index i such that $2\delta_i \le \delta < 2\delta_{i+1}$, and then query the corresponding data structure for S_i. The returned point is a $1 - (1-\delta_i)^2 < 2\delta_i \le \delta$ approximate farthest neighbor of the query point. The query time is $O(|S_i|^{1/2+\varepsilon}/\sqrt{\delta_i}) = O((1/\delta)^{9/4+\varepsilon})$.

Theorem 4. *Let S be a set of n linearly moving points in $\mathbb{R}^2$. For a parameter $m > 1$, a data structure of size $O(m)$ can be built in $O(n + m^{7/4})$ time so that a δ-approximate farthest-neighbor query on S can be answered in $O((1/\delta)^{9/4+\varepsilon})$ time, for any $(1/m)^{1/4} \leq \delta < 1$.*

4 Convex-Hull Queries

In this section, we describe data structures for convex-hull queries amid moving points, i.e., Q5 queries defined in Section 1.1, which provide various tradeoffs. We begin with data structures that produce a continuous tradeoff betwee space and query time. In particular, we present two data structures for this problem — one with linear size, and the other with logarithmic query time. We then show how to achieve a tradeoff between efficiency and accuracy, following the general spirit of Section 3.

Tradeoff between space and query time. In the txy-space, the set of linearly moving points of S traces out a set of n oriented lines $L = \{\ell_1, \ell_2, \cdots, \ell_n\}$, whose orientations are naturally defined to be along the positive t-axis. Given a time stamp t_q and a query point $p \in \mathbb{R}^2$, let $\overline{p} = (p, t_q) \in \mathbb{R}^3$ and $\gamma \subset \mathbb{R}^3$ be the plane $t = t_q$. To determine whether $p \in \mathrm{conv}(S(t_q))$, we need to decide whether $\overline{p} \in \mathrm{conv}(\gamma \cap L)$. Observe that $\overline{p} \notin \mathrm{conv}(\gamma \cap L)$ if and only if there exists a line ℓ lying in the plane γ that passes through $\overline{p}$ and lies outside $\mathrm{conv}(\gamma \cap L)$. If we assign an arbitrary orientation to ℓ, then it can be verified that ℓ has the same relative orientation[5] with respect to all lines in L (see Figure 2).

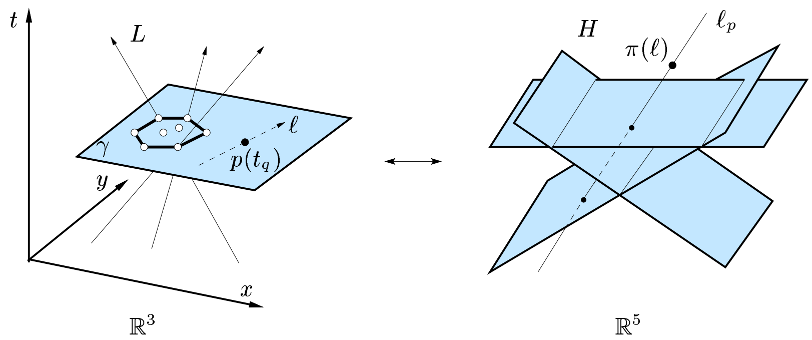

Fig. 2. Plücker coordinates and convex-hull queries on moving points in $\mathbb{R}^2$.

[5] Let ℓ_1 be an oriented line in $\mathbb{R}^3$ whose orientation is defined by an ordered pair of points a and b on ℓ_1, and let ℓ_2 be another oriented line whose orientation is defined by two points c and d on ℓ_2. The *relative orientation* of ℓ_1 and ℓ_2 is defined by the sign of the determinant of the 4-by-4 matrix $\begin{pmatrix} a & b & c & d \\ 1 & 1 & 1 & 1 \end{pmatrix}$.

Plücker coordinates [13, 22] maps an oriented line ℓ in $\mathbb{R}^3$ to either a point $\pi(\ell)$ in $\mathbb{R}^5$ (referred to as the *Plücker point* of ℓ) or a hyperplane $\varpi(\ell)$ in $\mathbb{R}^5$ (referred to as the *Plücker hyperplane* of ℓ). Furthermore, for any two oriented lines ℓ_1 and ℓ_2, ℓ_1 has a positive orientation with respect to ℓ_2 if and only if $\pi(\ell_1)$ lies above $\varpi(\ell_2)$.

Lemma 5. *The union of the set of Plücker points of all oriented lines lying in γ and passing through $\overline{p}$ is a line in $\mathbb{R}^5$, denoted by ℓ_p.*

Proof. Let $\overline{p} = (t_q, p_x, p_y)$. A simple calculation shows that the Plücker point of an oriented line lying in γ and passing through $\overline{p}$ can be written as

$$(\alpha, 0, p_x\alpha - p_y, t_q, -t_q\alpha),$$

for $\alpha \in \mathbb{R}$. The union of these Plücker points is a line passing through two points $(0, 0, -p_y, t_q, 0)$ and $(1, 0, p_x, 0, -t_q)$ in $\mathbb{R}^5$. □

Let $H = \{\varpi(\ell_i) \mid \ell_i \in L\}$ be the set of Plücker hyperplanes in $\mathbb{R}^5$. A line ℓ has the same relative orientation with respect to all lines in L if and only if $\pi(\ell)$ lies above or below all hyperplanes of H. Hence, determining whether $p \in \mathrm{conv}(S(t_q))$ is equivalent to determining whether there is a point on ℓ_p that lies above or below all hyperplanes in H. This can be formulated as a linear-programming query problem, and a linear-size data structure with query time $O(n^{1-1/\lfloor d/2 \rfloor}\mathrm{polylog}\ n)$ was developed by Matoušek [18] in $\mathbb{R}^d$ for $d \geq 4$. In fact, he obtained a continuous tradeoff between space and query time for this problem. Invoking this result[6] with $d = 5$, we obtain the following result.

Theorem 5. *Let S be a set of n linearly moving points in $\mathbb{R}^2$. For a parameter $n \leq m \leq n^2$, a data structure of size $O(m^{1+\varepsilon})$ can be built in $O(m^{1+\varepsilon})$ time so that a Q5 query on S can be answered in $O(n \cdot \mathrm{polylog}(n)/\sqrt{m})$ time.*

Remark. Being a little careful, one can show that it is enough to build a linear-programming query structure in $\mathbb{R}^4$ instead of $\mathbb{R}^5$. This slightly decreases the polylog factor in the query time. Using the same idea, one can build a data structure for answering two-dimensional halfspace range queries amid moving points.

A data structure with logarithmic query time. The polylog factor in the query time of Theorem 5 is prohibitively large. We now describe a simple data structure for convex-hull queries that avoids the use of the complicated linear-programming query structure. The query time is reduced to $O(\log n)$. Althought the size of the data structure is $O(n^2)$ in the worst case, a tradeoff between efficiency and accuracy described in the next section alleviates this problem easily.

Let us denote by $\ell_i(t)$ the dual line of point $p_i(t)$ at time t. As time t varies, each $\ell_i(t)$ traces out an algebraic surface, denoted by f_i, in the dual-time space.

[6] The data structures of Chan [10] or Ramos [20] may also be used for this purpuse.

Let $\mathcal{F} = \{f_1, \cdots, f_n\}$ and $\mathcal{F}(t) = \{\ell_1(t), \cdots, \ell_n(t)\}$. It is clear that each vertex on the upper or lower envelope of $\mathcal{F}$ corresponds to a kinetic event, indicating a combinatorial change on the convex hull of $S(t)$. We can compute the upper and lower envelopes of $\mathcal{F}$ using an $O(n^{2+\varepsilon})$ algorithm described in [8], and then store each combinatorial change in a standard persistent data structure [21].

Given a point p and a time stamp t_q, deciding whether p lies outside the convex hull of $S(t_q)$ is equivalent to deciding whether there exists a nonvertical line passing through p such that the set $S(t_q)$ lies above or below it. If such a nonvertical line exists, observe that in the dual, it would be mapped to a point on $\ell(p)$ — the dual line of p — that lies either above the upper envelope or below the lower envelope of $\mathcal{F}(t_q)$, in which case $\ell(p)$ must therefore intersect the envelopes of $\mathcal{F}(t_q)$. We can detect whether they indeed intersect by two standard binary searches on the upper and lower envelopes of $\mathcal{F}(t_q)$ respectively, which can be performed in the persistent structure for any $t_q \in \mathbb{R}$.

Denote by κ the number of combinatorial changes to the convex hull of S over time. It is known that $\kappa = O(n^2)$ for a set of n linearly moving points in the plane [5]. Therefore the size of the persistent structure is bounded by $O(n+\kappa) = O(n^2)$, and the query time is bounded by $O(\log\kappa + \log n) = O(\log n)$.

Theorem 6. *Let S be a set of n linearly moving points in $\mathbb{R}^2$, and let $\kappa = O(n^2)$ be the total number of combinatorial changes to the convex hull of S over time. A data structure of size $O(n + \kappa)$ can be built in $O(n^{2+\varepsilon})$ time so that a Q5 query on S can be answered in $O(\log n)$ time.*

Tradeoff between efficiency and accuracy. We can further build a data structure for *approximate convex-hull queries.* Given any unit vector $u \in \mathbb{S}^2$, the *directional width* of a point set P is defined to be $w_u(P) = \max_{p\in P}\langle u, p\rangle - \min_{p\in P}\langle u, p\rangle$. A subset $C \subseteq S$ is a *δ-kernel* of a set S of moving points if for any time t and any unit vector u, $w_u(C(t)) \geq (1-\delta)\cdot w_u(S(t))$. Given any parameter δ, one can compute in $O(n+1/\delta^3)$ time a δ-kernel of S of size $O(1/\delta^{3/2})$, if points of S move linearly [6, 11]. To achieve tradeoff between efficiency and accuracy for convex-hull queries, we do as follows. Let $m > 1$ be a fixed parameter, and let $\delta_i = (2^i/m)^{1/3}$, for $0 \leq i < \log m$. Let S_i be a δ_i-kernel of S, for $1 \leq i < \log m$. We can compute all S_i's in $O(n + m)$ time by first computing a δ_0-kernel S_0 of S and then computing a $(\delta_i - \delta_{i-1})$-kernel S_i of S_{i-1} for $1 \leq i < \log m$. We then build a data structure for each S_i using Theorem 6. The size of one such data structure is $O(|S_i|^2) = O(m/2^i)$; as such, the total size of these data structures is $O(m)$. Given a δ-approximate convex-hull query, where $(1/m)^{1/3} \leq \delta < 1$, we first find an index i such that $\delta_i \leq \delta < \delta_{i+1}$, and then query the corresponding data structure for S_i. The query time is $O(\log |S_i|) = O(\log(1/\delta))$.

Theorem 7. *Let S be a set of n linearly moving points in $\mathbb{R}^2$. For a parameter $m > 1$, a data structure of size $O(m)$ can be built in $O(n + m^{2+\varepsilon})$ time so that a δ-approximate convex-hull query on S can be answered in $O(\log(1/\delta))$ time, for any $(1/m)^{1/3} \leq \delta < 1$.*

References

1. P. K. Agarwal, L. Arge, and J. Erickson, Indexing moving points, *Proc. Annu. ACM Sympos. Principles Database Syst.*, 2000, pp. 175–186.
2. P. K. Agarwal, L. Arge, and J. Vahrenhold, Time responsive external data structures for moving points, *Proc. 7th Workshop on Algorithms and Data Structures*, 2001.
3. P. K. Agarwal and J. Erickson, Geometric range searching and its relatives, *Contemporary Mathematics*, 223 (1999), 1–56.
4. P. K. Agarwal, J. Gao, and L. Guibas, Kinetic medians and *kd*-trees, *Proc. 10th European Symposium on Algorithms*, 2002, pp. 5–16.
5. P. K. Agarwal, L. Guibas, J. Hershberg, and E. Veach, Maintaining the extent of a moving point set, *Discrete Comput. Geom.*, 26 (2001), 253–274.
6. P. K. Agarwal, S. Har-Peled, and K. Varadarajan. Approximating extent measure of points, 2004. to appear in J. ACM.
7. P. K. Agarwal and C. M. Procopiuc, Advances in indexing for moving objects, *IEEE Bulletin of Data Engineering*, 25 (2002), 25–34.
8. P. K. Agarwal and M. Sharir, Arrangements and their applications, *Handbook of Computational Geometry (J.-R. Sack and J. Urrutia, eds.)*, (2000), 49–119.
9. J. Basch, L. Guibas, and J. Hershberger, Data structures for mobile data, *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, 1997, pp. 747–756.
10. T. M. Chan, Fixed-dimensional linear programming queries made easy, *Proc. 12th Sympos. Comput. Geom.*, 1996, pp. 284–290.
11. T. M. Chan, Faster core-set constructions and data stream algorithms in fixed dimensions, *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, to appear.
12. B. Chazelle, Cutting hyperplanes for divide-and-conquer, *Discrete Comput. Geom.*, 9 (1993), 145–158.
13. B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir, and J. Stolfi, Lines in space: combinatorics and algorithms, *Algorithmica*, 15 (1996), 428–447.
14. R. Cole, J. Salowe, W. Steiger, and E. Szemerédi, Optimal slope selection, *SIAM J. Computing*, 18 (1989), 792–810.
15. G. Kollios, D. Gunopulos, and V. Tsotras, Nearest neighbor queries in a mobile environment, *Spatiotemporal Database Management*, (1999), 119–134.
16. G. Kollios, D. Gunopulos, and V. Tsotras, On indexing mobile objects, *Proc. ACM Sympos. Principles Database Syst.*, 1999, pp. 261–272.
17. J. Matoušek, Efficient partition trees, *Discrete Comput. Geom.*, 8 (1992), 315–334.
18. J. Matoušek, Linear optimization queries, *J. Algorithms*, 14 (1993), 432–448.
19. C. M. Procopiuc, P. K. Agarwal, and S. Har-Peled, STAR-tree: an efficient self-adjusting index for moving points, *Proc 4th Workshop on Algorithms Engineering and Experiments*, 2002.
20. E. Ramos, Linear programming queries revisited, *Proc. 16th Sympos. Comput. Geom.*, 2000, pp. 176–181.
21. N. Sarnak and R. Tarjan, Planar point location using persistent search trees, *Communications of the ACM*, (1986), 669–679.
22. J. Stolfi, *Oriented Projective Geometry*, Academic Press, Boston, 1991.
23. S. Šaltenis, C. Jensen, S. Leutenegger, and M. López, Indexing the positions of continuously moving objects, *Proc. SIGMOD Intl. Conf. Management of Data*, 2000, pp. 331–342.