Lower Bounds for Kinetic Sorting

Mohammad Ali Abam* Mark de Berg†

Abstract

Let S be a set of n points moving on the real line. The kinetic sorting problem is to maintain a data structure on the set S that makes it possible to quickly generate a sorted list of the points in S, at any given time. We prove tight lower bounds for this problem, which show the following: with a subquadratic maintenance cost one cannot obtain any significant speed-up on the time needed to generate the sorted list (compared to the trivial $O(n \log n)$ time), even for linear motions.

1 Introduction

Background. In many areas of computer science one has to store, analyze and manipulate geometric data. More and more often this involves objects in motion. Hence, the study of geometric data structures for moving objects has recently attracted a lot of attention in computational geometry, especially since Basch *et al.* [6] introduced the *kinetic-data-structure* (KDS, for short) framework [3, 4, 6, 8, 9].

A KDS is a structure that maintains a certain attribute of a set of continuously moving objects—the convex hull of moving objects, for instance, or the closest distance among moving objects. It consists of two parts: a combinatorial description of the attribute and a set of certificates with the property that as long as the outcomes of the certificates do not change, the attribute does not change. It is assumed that each object follows a known trajectory so that one can compute the failure time of each certificate. Whenever a certificate fails—we call this an *event*—the KDS must be updated. The KDS remains valid until the next event. See the excellent survey by Guibas [8] for more background on KDSs and their analysis.

Computing the convex hull of a set of points in the plane is a classic problem in computational geometry. It is therefore not surprising that the kinetic maintenance of the convex hull of a set of n moving points in the plane was already studied by Basch *et al.* [6]. They designed a KDS that needs to be updated $O(n^{2+\epsilon})$ times and each time takes $O(\log^2 n)$ time.

*Department of Computing Science, TU Eindhoven, m.a.abam@tue.nl

†Department of Computing Science, TU Eindhoven, mdberg@win.tue.nl

In some applications it may be necessary to maintain the attribute of interest explicitly. If one uses a KDS for collision detection, for instance, any external event—a collision in this case— must be reported. In such cases, the number of changes to the attribute is a lower bound on the number of events to be processed. Since the convex hull of n linearly moving points can change $\Omega(n^2)$ times [5], this means that any KDS that maintains an explicit representatie of the convex hull must process $\Omega(n^2)$ events in the worst case. Hence, the convex-hull KDS of Basch *et al.* [6], which indeed maintains the convex hull explicitly, is close to optimal in the worst case.

In other applications, however, explicitly maintaining the attribute at all times may not be necessary; the attribute is only needed at certain times. This is for instance the case when a KDS is used as an auxiliary structure in another KDS. The auxiliary KDS is then used to update the main KDS efficiently when a certificate of the main KDS fails. In this case, even though the main KDS may have to be maintained explicitly, the attribute maintained by the auxiliary KDS only needs to be available at certain times. This leads us to view a KDS as a query structure: we want to maintain a set S of moving objects in such a way that we can reconstruct the attribute of interest efficiently whenever this is called for. This makes it possible to reduce the maintenance cost, as it is no longer necessary to update the KDS whenever the attribute changes. On the other hand, a reduction in maintenance cost will have an impact on the query time, that is, the time needed to reconstruct the attribute. Thus there is a trade-off between maintenance cost and query time. Our main goal is to study such trade-offs for kinetic convex hulls.

Our results. As stated above, our main interest lies in trade-offs between the maintenance cost of a kinetic convex-hull structure and the time to reconstruct the convex hull at any given time. In this short abstract, however, we restrict our attention to the simpler *kinetic sorting problem*: maintain a KDS on a set of n points moving on the real line such that at any time we can quickly reconstruct a sorted list of the points. We prove in Section 2 that already for the kinetic sorting problem one cannot get good trade-offs: even for linear motions, the worst-case maintenance cost is $\Omega(n^2)$ if one wants to be able to do the reconstruction in $o(n)$ time. Note that with $\Omega(n^2)$ maintenance cost,

we can explicitly maintain the sorted list at all times, so that the reconstruction cost is zero. Thus interesting trade-offs are only possible in a very limited range of the spectrum, namely for reconstruction costs between $\Omega(n)$ and $O(n \log n)$. For this range we also prove lower bounds: we show that one needs $\Omega(n^2/m)$ maintenance cost if one wants to achieve $o(n \log m)$ reconstruction cost, for any m with $2 \leq m \leq n$. (See Section 2.1 for a definition of our lower-bound model.) We also give a matching upper bound.

Related work. Some existing KDSs—the kinetic variants of various range-searching data structures [2, 3, 4, 9], for instance—do not maintain a uniquely defined attribute such as the convex hull, but they maintain a query data structure. In this setting the KDS is, of course, a query structure as well. Our setting is different because we are studying the maintenance of a single, uniquely defined, attribute such as the convex hull.

One of the main results of our paper is a lower bound on the trade-offs between reconstruction time and maintenance cost for the kinetic sorting problem. Lower bounds for trade-offs between query time and maintenance cost were also given by De Berg [7], but he studied the kinetic dictionary problem, where one wants to maintain a dictionary on a set S of n points moving on the real line. He showed that any kinetic dictionary with worst-case query time $O(Q)$ must have a worst-case total maintenance cost of $\Omega(n^2/Q^2)$, even if the points move linearly.

2 The kinetic sorting problem

Let $S = \{x_1, \cdots, x_n\}$ be a set of n point objects[1] moving continuously on the real line. In other words, the value of x_i is a continuous function of time, which we denote by $x_i(t)$. We define $S(t) = \{x_1(t), \cdots, x_n(t)\}$. For simplicity, we write S and x_i instead of $S(t)$ and $x_i(t)$, respectively, provided that no confusion arises. The kinetic sorting problem asks to maintain a structure on S such that at any given time t we can quickly generate a sorted list for $S(t)$. We call such a structure a *sorting KDS*.

We focus on trade-offs between the sorting cost and the maintenance cost: what is the worst-case maintenance cost if we want to guarantee a sorting cost of $O(Q)$, where Q is some parameter, under the assumption that the point objects follow trajectories that can be described by bounded-degree polynomials.

2.1 The lower-bound model

We shall prove our lower bounds for the kinetic sorting problem in the comparison-graph model introduced by De Berg [7], which is defined as follows. A *comparison graph* for a set S of numbers is defined as a directed graph $\mathcal{G}(S, A)$ such that if $(x_i, x_j) \in A$, then $x_i < x_j$. The reverse is not true: the fact that $x_i < x_j$ does not mean there must be an arc in $\mathcal{G}$. The idea is that the comparison graph represents the ordering information encoded in a sorting KDS on the set S: if $(x_i, x_j) \in A$, then the fact that $x_i < x_j$ can be derived from the information stored in the KDS, without doing any additional comparisons.

Maintenance cost. The operations we allow on the comparison graph are insertions and deletions of arcs. For the maintenance cost, we only charge the algorithm for insertions of arcs; deletions are free. Following De Berg [7], we therefore define the maintenance cost as the total number of such arcs ever inserted into the comparison graph, either at initialization or during maintenance operations.

We say that the arc $(x_i, x_j) \in A$ *fails* at time t if $x_i(t) = x_j(t)$.The arcs in the comparison graph essentially act as certificates, and their failures trigger events at which the KDS needs to be updated.

Query cost. A query at time t asks to construct a sorted list on the points in the current set S (that is, $S(t)$). We shall consider two different measures for the query cost.

The comparison-graph sorting model. The first measure is in a very weak model, where we only charge for the minimum number of comparisons needed to obtain a sorted list, assuming we have an oracle at our disposal telling us exactly which comparisons to do. This is similar to the query cost used by De Berg when he proved lower bounds for the kinetic dictionary. For the sorting problem this simply means that the query cost is equal to the number of pairs $x_i, x_j \in S$ that are adjacent in the ordering and for which there is no arc in the comparison graph.

The algebraic decision-tree model. In this model we also count the number of comparisons needed to sort the set S, but this time we not have an oracle telling us which comparisons to do. We shall use the following basic fact: Suppose the number of different orderings of S that are compatible with the comparison graph at some given time is N. Then the cost to sort S in the algebraic decision-tree model is at least $\log N$.

2.2 A lower bound in the comparison-graph sorting model

The point objects in our lower-bound instance will move with constant (but different) velocities on the real line. Hence, if we view the line on which the point objects move as the x-axis and time as the t-axis, then the trajectories of the point objects are straight lines in the tx-plane. We use ξ_i to denote the line in the

[1] We use the term "point objects" for the points in S to distinguish them from other points that play a role in our proofs.

tx-plane that is the trajectory of x_i. It is somewhat easier to describe the lower-bound instance in the dual plane. We shall call the two axes in the dual plane the u-axis and the v-axis. We use the standard duality transform, where a line $\xi : x = at + b$ in the tx-plane is mapped to the point $\xi^* : (a, -b)$ in the dual plane, and a point $p : (a, b)$ in the primal plane is mapped to the line $p^* : v = au - b$ in the dual plane.

Now let $p_1, \ldots, p_n$ be the vertices of a regular n-gon in the dual plane that is oriented such that the diagonal $p_{l-1}p_{l+1}$ connecting the two neighbors of the leftmost vertex p_l is almost parallel to the v-axis and has negative slope. The trajectories $\xi_1, \cdots, \xi_n$ in our lower-bound instance are the primals of the vertices p_i, that is, $\xi_i^* = p_i$. In the remainder of this section we will prove a lower bound on the maintenance cost of any comparison graph for this instance whose sorting cost (in the comparison-graph sorting model) is bounded by Q, where Q is a parameter with $0 \leq Q < n$.

For any pair of vertices p_i, p_j, let ℓ_{ij} denote the line passing through p_i and p_j. Since the p_i are the vertices of a regular n-gon, the lines ℓ_{ij} have only n distinct slopes. Note that ℓ_{ij} corresponds to the intersection of ξ_i and ξ_j in the tx-plane, with the slope of ℓ_{ij} being equal to t-coordinate of the intersection. This implies that the intersection points of the trajectories in the tx-plane have only n distinct t-values. Let $t_1, \cdots, t_n$ be the sorted sequence of these t-values. The times $t_1, \ldots, t_n$ define $n + 1$ open time intervals $(-\infty, t_1), (t_1, t_2), \cdots, (t_n, +\infty)$. Since no two trajectories intersect inside any of these intervals, the order of the point objects is the same throughout any interval. We say that x_i is *directly below* x_j in such an interval if $x_i(t) < x_j(t)$ for times t in the interval and there is no other point object x_k in between them in that interval. Furthermore, we call a vertex p_i a *lower vertex* if it lies on the lower part of the boundary of the n-gon, and we call p_i an *upper vertex* if it lies on the upper part of the boundary of the n-gon; the leftmost and rightmost vertices are neither upper nor lower vertices.

Lemma 1 *(i) If p_i is a lower vertex or the leftmost vertex, then the object x_i is below any other object x_j in exactly one time interval. If n is odd, this also holds for the rightmost vertex.*
(ii) If p_i is an upper vertex, then x_i is directly below any other point object x_j in at least one interval. If n is even, this also holds for the rightmost vertex.

We can now prove the lower bound. Suppose that we have a comparison graph on the point objects whose sorting cost is Q during each of the time intervals defined above. This implies that during each such time interval, there must be at least $n - Q - 1$ arcs (x_i, x_j) in the comparison graph such that x_i is directly below x_j. In total, $(n + 1)(n - Q - 1)$ arcs are needed over all $n + 1$ time interval. Some arcs, however, can be used in more than one interval. For x_i, let k_i be the number of arcs of the form (x_i, x_j) that are used. For any of the $\lceil n/2 \rceil$ objects x_i for which case (i) of Lemma 1 applies, all these arcs are distinct. For the remaining $\lfloor n/2 \rfloor$ objects case (ii) applies and so at least $k_i - 2$ arcs are distinct. Hence, the total number of arcs inserted over time is at least $(n+1)(n-Q-1) - 2\lfloor n/2 \rfloor \geq n(n-Q-2)$. We get the following theorem.

Theorem 2 *There is an instance of n point objects moving with constant velocities on the real line, such that any comparison graph whose worst-case sorting cost in the comparison-graph cost model is Q, must have maintenance cost at least $n(n - Q - 2)$, for any parameter Q with $0 \leq Q < n$.*

2.3 A lower bound in the algebraic decision-tree model

In the previous section we gave a lower bound for the maintenance cost for a given sorting cost Q in comparison-graph sorting model. Obviously, this is also a lower bound for the algebraic decision-tree model. Hence, the results of the previous section imply that for any sorting cost $Q = o(n)$ in the algebraic decision-tree model, the worst-case maintenance cost is $\Omega(n^2)$. Since with $O(n^2)$ maintenance cost we can process all swaps—assuming the trajectories are bounded-degree algebraic, so that any pair swaps at most $O(1)$ times—this bound is tight: with $O(n^2)$ maintenance cost we can achieve sorting cost zero. What remains is to investigate the range where the sorting cost is $o(n \log m)$, where $1 < m \leq n$.

Lemma 3 *There is a constant c such that if the sorting cost of a comparison graph is at most $cn \log m$, then there is a path in the comparison graph whose length is at least $n/m^{1/3}$.*

Next we describe the lower bound construction. As before, it will be convenient to describe the construction in the dual plane. To this end, let $G_a := \{0, 1, \cdots, a-1\}^2$ be the $a \times a$ grid. The trajectories of the point objects in our lower-bound instance will be straight lines in the tx-plane, such that the duals of these lines are the grid points of $G_{\sqrt{n}}$. (We assume for simplicity that n is a square number.) Before we proceed, we need the following lemma.

Lemma 4 *Let $p = (p_x, p_y)$ be a grid point of $G_{\sqrt{n}}$ and $p_x, p_y \leq a$, where $a \leq \sqrt{n}/2$. Let ℓ_p be the line through the origin and p. The number of different lines passing through at least one point of $G_{\sqrt{n}}$ and being parallel to ℓ_p is at most $4a\sqrt{n}$.*

Theorem 5 *There is an instance of n point objects moving with constant velocities on the real line such that, for any m with $1 < m \leq n$, any comparison graph whose worst-case sorting cost in the algebraic decision-tree model is $Q = o(n \log m)$, must have maintenance cost $\Omega(n^2/m)$.*

Proof. Let $a := \sqrt{n}/(8m^{1/3})$. Consider the comparison graph at some time $s + \varepsilon$ with $s = p_x/p_y$, where $p_x, p_y \leq a$ and $\varepsilon > 0$ is sufficiently small. Suppose the sorting cost at time s is $o(n \log m)$. Then the sorting cost will be at most $cn \log m$ for any constant c, so by Lemma 3 there must be a path in the comparison graph of length at least $n/m^{1/3}$. We claim (and will prove below) that at least half of the arcs in this path are between point objects x_i, x_j such that ξ_i^* and ξ_j^* lie on a common line of slope s. The number of distinct values for s is equal to the number of pairs (p_x, p_y) where p_x and p_y are integer numbers between 0 and $a - 1$ (including 0 and $a - 1$) and $GCD(p_x, p_y) = 1$. Because of symmetry, we count the number of pairs (p_x, p_y) with the property $p_x \leq p_y$. For a nonnegative integer i, let $\varphi(i)$ be the number of nonnegative integers that are less than i and relatively prime to i. Then the number of pairs (p_x, p_y) with the desired properties is $\sum_{i=1}^{a-1} \varphi(i)$. It is known [10] that this summation is $\Theta(a^2)$. Then, the total number of arcs needed over all times of the form $p_x/p_y + \varepsilon$ with $p_x, p_y \leq a$ is at least $n/(2m^{1/3}) \cdot \Theta(a^2) = \Omega(n^2/m)$, which proves the theorem.

It remains to prove the claim that at least half of the arcs in the path are between point objects x_i, x_j such that ξ_i^* and ξ_j^* lie on a common line of slope s. Note that the sorted order of the point objects $x_i(s + \varepsilon)$ corresponds to the sorted order of the orthogonal projections of the points ξ_i^* onto a line with slope $-1/(s + \varepsilon)$. If $\varepsilon > 0$ is sufficiently small, then the projections of all the points lying on a common line of slope s will be adjacent in this order. Let's group the point objects x_i into subsets such that any two point objects x_i, x_j for which ξ_i^* and ξ_j^* lie on a common line of slope s are in the same subset. Then, at time $s+\varepsilon$, any path in the comparison graph can enter and leave a subset at most once. By Lemma 4 the number of subsets is at most $4a\sqrt{n}$. Hence, the number of arcs connecting point objects in the same subset is at least

$$n/m^{1/3} - 4a\sqrt{n} = n/(2m^{1/3}),$$

as claimed. □

2.4 Upper bounds for the kinetic sorting problem

It is straightforward to obtain a KDS that matches the lower bounds of the previous section: Partition the set S into m subsets of size at most n/m in an arbitrary manner, and maintain each subset in a sorted array. This gives the following result.

Theorem 6 *Let S be a set of n point objects moving on the line, where any pair of points swaps $O(1)$ times. For any m with $1 < m \leq n$, there is a data structure with maintenance cost $O(n^2/m)$ such that at any time a sorted list of the points in S can be constructed in $O(n \log m)$ time.*

3 Conclusions

We have studied trade-offs for the kinetic sorting problem, which is to maintain a KDS on a set of points moving on the real line such that one can quickly generate a sorted list of the points, at any given time. We have proved a lower bound for this problem showing the following: with a subquadratic maintenance cost one cannot obtain any significant speed-up on the time needed to generate the sorted list (compared to the trivial $O(n \log n)$ time), even for linear motions.

This negative result gives a strong indication that good trade-offs are not possible for a large number of geometric problems—Voronoi diagrams and Delaunay triangulations, for example, or convex hulls—as the sorting problem can often be reduced to such problems. In [1], we show that the convex hull can be maintained more efficiently if it has only few vertices.

References

[1] M.A Abam and Mark de Berg. Kinetic sorting and kientic convex hulls. *Submited to ACM Sympos. on Computational Geometry*, 2005.

[2] P. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *Proc. ACM Sympos. Principles Database Syst.*, pages 175–186, 2000.

[3] P. Agarwal, L. Arge, J. Erickson, and H. Yu. Efficient tradeoff schemes in data structures for querying moving objects. In *Proc. 12th European Sympos. on Algorithms*, pages 4–15, 2004.

[4] P. Agarwal, J. Gao, and L. Guibas. Kinetic medians and kd-trees. In *Proc. 10th European Sympos. on Algorithms*, pages 5–16, 2002.

[5] P. Agarwal, L. Guibas, J. Hershberger, and E. Veach. Maintaining the extent of a moving point set. In *Proc. 5th Workshop Algorithms and Data Structurs*, pages 31–44, 1997.

[6] J. Bash, L. Guibas, and J. Hershberger. Data structure for mobile data. *J. Algorithms*, 31:1–28, 1999.

[7] M. de Berg. Kinetic dictionaries: How to shoot a moving target. In *Proc. 11th European Sympos. on Algorithms*, pages 172–183, 2003.

[8] L. Guibas. Kinetic data structure: a state of art report. In *Proc. 3rd Workshop Algorithmic Found. Robot.*, pages 191–209, 1998.

[9] J. Hershberger and S. Suri. Kinetic connectivity of rectangles. In *Proc. 15th ACM Sympos. Comput. Geom.*, pages 237–246, 1999.

[10] E. Weinstein. *CRC Concise Encyclopedia of Mathematics*. CRC Press, 1999.