Out-of-Order Event Processing in Kinetic Data Structures*

Mohammad Ali Abam† Pankaj K. Agarwal‡ Mark de Berg§ Hai Yu¶

Abstract

We study the problem of designing kinetic data structures (KDS's for short) when event times cannot be computed exactly and events may be processed in a wrong order. In traditional KDS's this can lead to major inconsistencies from which the KDS cannot recover. We present more robust KDS's for the maintenance of two fundamental structures, kinetic sorting and tournament trees, which overcome the difficulty by employing a refined event scheduling and processing technique. We prove that the new event scheduling mechanism leads to a KDS that is correct except for finitely many short time intervals. We analyze the maximum delay of events and the maximum error in the structure, and we experimentally compare our approach to the standard event scheduling mechanism.

1 Introduction

The recent advances in sensing and tracking technology have led researchers to investigate the problem of maintaining various geometric attributes of a set of moving objects, as evident from a large body of literature on kinetic geometric algorithms. Basch *et al.* [6] introduced the *kinetic data structure (KDS)* framework for designing and analyzing algorithms for continuously moving objects. The KDS framework consists of two parts: a combinatorial description of the attribute, and a set of *certificates*, each of which is a predicate, with the property that as long as the certificates remain valid, the maintained attribute remains valid. 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. This involves updating the attribute and the set of certificates. The KDS then remains valid until the next event has to be processed, and so on.

To be able to process each event at the right time, a global event queue Q is maintained to process the events in the right (chronological) order. This is a priority queue on the events, with the priority of an event being its failure time. Unfortunately, the event scheduling is not as easy as it seems. Suppose that a new certificate arises due to some event. When the failure time of the certificate lies in the past we should not schedule it, and when it lies in the future we should. But what if the event time is equal to the current time t_{curr}? In such a degenerate situation one has to be very careful to avoid an infinite loop. A more serious problem arises when the event times are not computed exactly. This will indeed be the case if the trajectories are polynomials of high degree or more complex curves. As a result, events may be processed in a wrong order, or we may fail to schedule an event because we think it has already taken place. This in turn may not only lead to serious errors in the geometric attribute the KDS is maintaining but also cause the algorithm to crash.

*M.A. was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 612.065.307. M.d.B. was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301. P. A. and H. Y. were supported by NSF under grants CCR-00-86013, EIA-01-31905, CCR-02-04118, and DEB-04-25465, by ARO grants W911NF-04-1-0278 and DAAD19-03-1-0352, and by a grant from the U.S.–Israel Binational Science Foundation.

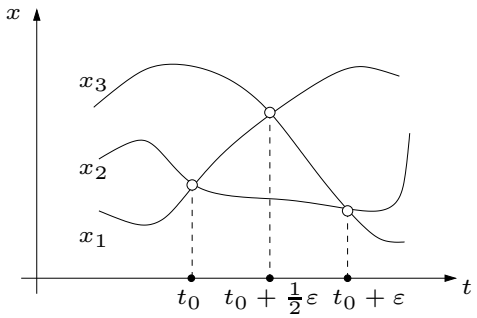
†Department of Computer Science, TU Eindhoven, Eindhoven, The Netherlands. m.a.abam@tue.nl.

‡Department of Computer Science, Duke University, Durham, NC 27708, USA. pankaj@cs.duke.edu.

§Department of Computer Science, TU Eindhoven, Eindhoven, The Netherlands. mdberg@win.tue.nl.

¶Department of Computer Science, Duke University, Durham, NC 27708, USA. fishhai@cs.duke.edu.

As a concrete example, consider the kinetic sorting problem: maintain the sorted order of a set S of points moving on the real line. We store S in a sorted array $A[1..n]$. For each $1 \leq i < n$ there is a certificate $\big[A[i] < A[i+1]\big]$. Whenever $A[j] = A[j+1]$ for some j, we have a certificate failure. At such an event we swap $A[j]$ and $A[j+1]$. Furthermore, at most three new certificates arise: $\big[A[j-1] < A[j]\big]$, $\big[A[j] < A[j+1]\big]$, and $\big[A[j+1] < A[j+2]\big]$. We compute the failure time of each of them, based on our knowledge of their current motions, and insert the failure times that are not in the past into the event queue Q. Some certificates may also disappear because the two points involved are no longer neighbors; they have to be deleted from Q. Now suppose that due to errors in the computed failure times the difference between the exact and the computed failure time of each certificate can be as large as ε, for some $\varepsilon > 0$. Consider three moving points x_1, x_2 and x_3 whose trajectories in the tx-plane are depicted in Figure 1. Table (i) shows what happens when we can compute the exact failure times. Table (ii) shows what happens when the computed failure times of the certificates $\big[x_1 < x_2\big]$, $\big[x_1 < x_3\big]$, and $\big[x_2 < x_3\big]$ are $t_0 + \varepsilon$, $t_0 + \frac{3}{2}\varepsilon$, and t_0 respectively: the KDS is not just temporarily incorrect, but gets into an incorrect state from which it never recovers.



t_{curr}	List	Certificates	Failure Time
0	x_1, x_2, x_3	$[x_1 < x_2]$ $[x_2 < x_3]$	t_0 $t_0 + \varepsilon$
t_0	x_2, x_1, x_3	$[x_1 < x_3]$	$t_0 + \frac{1}{2}\varepsilon$
$t_0 + \frac{1}{2}\varepsilon$	x_2, x_3, x_1	$[x_2 < x_3]$	$t_0 + \varepsilon$
$t_0 + \varepsilon$	x_3, x_2, x_1		

(i) without error

t_{curr}	List	Certificates	Failure Time
0	x_1, x_2, x_3	$[x_1 < x_2]$ $[x_2 < x_3]$	$t_0 + \varepsilon$ t_0
t_0	x_1, x_3, x_2	$[x_1 < x_3]$	$t_0 + \frac{3}{2}\varepsilon$
$t_0 + \frac{3}{2}\varepsilon$	x_3, x_1, x_2		

(ii) with error

Figure 1. An example that numerical errors in the event times may cause fatal errors in the KDS. Left: the trajectories of the points. Right: the status of the KDS at various times of execution.

This is a serious problem for the applicability of the KDS framework in practice. The goal of our paper is to address this issue: is it possible to do the event scheduling and processing in such a way that the KDS is more robust under errors in the computation of event times? The KDS may process the events in a wrong order and thus may maintain a wrong geometric attribute from time to time, but we would like the KDS to detect these errors and fix them quickly.

Related work. The KDS framework [6] is a widely used algorithmic method for modeling motion. It has been applied to maintain a variety of geometric attributes of moving objects, including convex hull [5, 6], closest pair [6, 7], range searching structures [1, 2], extent measures [3, 4], and much more. See the survey by Guibas [13] and the references therein. There has been much work on modeling motion in many other fields as well, including computer graphics, spatial databases, robotics, and sensor networks.

There is a large body of work on robust computations in geometric algorithms [11, 21, 22], including geometric software libraries [8, 10]. The goal there is to implement various geometric primitives in a robust manner, including *predicates*, which test the sign of an arithmetic expression (e.g., ORIENTATION and INCIRCLE predicates), and *constructions*, which compute the value of an arithmetic expression (e.g., computing the intersection of two lines). There are two broad paradigms. The first approach, exact computation, performs computation with enough precision to ensure predicates can be evaluated correctly. This has been the main paradigm in computational geometry. Many methods have been proposed to remove degeneracies (e.g., simulation of simplicity) and to speedup the computation by adaptively changing the precision (e.g., floating point

filters). The second approach focuses on performing computation with finite precision and computing an output as close to the correct one as possible.

Despite much work on robust geometric computation, little has been done on addressing robustness issues in KDS's. One could use exact computation but, as noticed by several researchers [14, 16], in practice a significant portion of the running time of a KDS is spent on computing certificate failure times. Expensive exact root comparisons will only make this worse and, hence, may lead to unacceptable performance in practice. See [15] for a comparison of various exact root computation techniques in the context of kinetic data structures. Guibas and Karavelas [14] described a method to speedup exact root comparisons by grouping the roots into intervals that are refined adaptively. However, like other exact methods, the performance of the algorithm deteriorates when many events are very close to each other.

An alternative is to apply *controlled perturbation* [17] to the KDS. In this method, we perturb the initial positions of the moving objects by some amount δ so that with high probability the roots of all pertinent functions are at least Δ far away from each other. This means one can compare any two roots exactly as long as every root is computed within a precision of $\Delta/2$. While controlled perturbation has been successful on a number of static problems [12, 17, 19], it does not seem to work well on kinetic data structures because the large number of events in the KDS makes the required perturbation bound δ fairly large.

Recently, Milenkovic and Sacks [20] studied the computation of arrangements of x-monotone curves in the plane using a plane sweep algorithm, under the assumption that intersection points of curves cannot be computed exactly. For infinite curves this boils down to the kinetic sorting problem, because one has to maintain the sorted order of the curves along the sweep line. In fact, our KDS for the kinetic sorting problem is very similar to their algorithm. The difference is in the subroutine to compute intersection points of curves which we assume to have available; this subroutine is stronger than the subroutine they assume—see Section 2 for details. This allows us to ensure that we never process more events than the number of actual crossings, whereas Milenkovic and Sacks may process a quadratic number of events in the worst case even when there is only a linear number of crossings. The main difference between our and their papers, however, lies in the different view on the problem: since we are looking at the problem from a KDS perspective, we are especially interested in the delay of events and the error in the output for each snapshot of the motion, something that was not studied in [20]. Moreover, we study other KDS problems as well.

Our results. The main problem we face when event times are not computed exactly is that events may be processed in a wrong order. We present KDS's that are robust against this out-of-order processing, including kinetic sorting and kinetic tournaments. Our algorithms are *quasi-robust* in the sense that the maintained attribute of the moving objects will be correct for most of the time, and when it is incorrect, it will not be far from the correct attribute. For the kinetic sorting problem, we obtain the following results:

- We prove that the KDS can only be incorrect when the current time is close to an event.
- We prove that an event may be processed too late, but not by more than $O(n\varepsilon)$ time. This bound is tight in the worst case.
- We prove bounds on the geometric error of the structure—the maximum distance between the i-th point in the maintained list and the i-th point in the correct list—that depend on the velocities of the points.

We obtain similar results for kinetic tournaments and kinetic range trees. As a by-product of our approach, degeneracy problems (how to deal with multiple events occurring simultaneously) arising in traditional KDS algorithms naturally disappear, because our KDS no longer cares about in which order these simultaneous events are processed.

We have implemented the robust sorting and tournament KDS algorithms and tested them on a number of inputs, including highly degenerate ones. Our sorting algorithm works very well on these inputs: of course it does not get stuck and the final list is always correct (after all, this is what we proved), but the maximum delay

of an event is usually much less than the worst-case bound suggests (namely $O(\varepsilon)$ instead of $\Theta(n\varepsilon)$). This is in contrast to the classical KDS, which either falls into an infinite loop or misses many kinetic events along the way and maintains a list that deviates far from the true sorted list both geometrically and combinatorially. Our kinetic tournament algorithm is also robust and reduces the geometric error by orders of magnitude.

The paper is organized as follows. We begin by describing our model in Section 2. We then study the robust KDS for kinetic sorting in Section 3, for kinetic tournaments in Section 4, and for kinetic range trees in section 5 . The experimental results are reported in Section 6. We conclude in Section 7.

2 Our Model

In this section we describe our model for computing the event times of certificates. In a standard KDS, each certificate c is a predicate, and there is a characteristic function $\chi_c : \mathbb{R} \to \{1, 0, -1\}$ associated with c so that $\chi_c(t) = 1$ if c is true at time t, -1 if c is false at time t. The values of t at which χ_c is switching from 1 to -1 or vice vera are the event times of c, and $\chi_c(t) = 0$ at these event times. In our applications, $\chi_c(t)$ can be derived from the sign of some continuous function $\varphi_c(t)$. For example, if $x(t)$ and $y(t)$ are two points, each moving in $\mathbb{R}^1$, then for the certificate $c := [x < y]$ we have $\chi_c(t) = 1$ if and only if $\mathrm{sign}(\varphi_c, t) > 0$ for $\varphi_c(t) = y(t) - x(t)$. For simplicity, we assume that $\mathrm{sign}(\varphi_c, t) = 0$ for a finite number, s, of values of t.

We assume that the trajectory of each object is explicitly described by a function of time, which means in our applications that the function φ_c is also explicitly described, and that event times can be computed by computing the roots of the function φ_c. These are standard assumptions in traditional KDS's. In order to model the inaccuracy in computing event times, we fix a parameter $\varepsilon > 0$, which will determine the accuracy of the root computation. We assume there is a subroutine, denoted by $\textsc{crop}\big(f(t)\big)$, to compute the roots of a function $f(t)$, whose output is as follows:

(A1) a set of disjoint, open intervals $U_1, \ldots, U_m$, where $|U_i| \leq \varepsilon$ for each i, that cover all roots of $f(t)$.
(A2) the sign of $f(t)$ between any two consecutive intervals;

For polynomial functions, Descartes' sign rule [9] and Sturm sequences [18] are standard approaches to implement CROP. We also assume that

(A3) CROP is deterministic: it always returns the same result when run on the same function.

Among the intervals returned by $\textsc{crop}\big(f(t)\big)$, we call an interval whose two endpoints have the same sign a *turbulent interval*, and an interval whose two endpoints have different signs an *event interval*; see Figure 2. Let $\mathcal{R}_f$ denote the union of all the turbulent and event intervals. In our applications, we can ignore turbulent intervals (intuitively, we can pretend that the sign of $f(t)$ does not change during a turbulent interval). We will use $\mathcal{I} = \langle I_1, \ldots I_k \rangle$ to denote the set of event intervals, and assume that

(A4) CROP only outputs the set $\mathcal{I}$ of event intervals.

Let λ_j (resp. ρ_j) denote the left (resp. right) endpoint of I_j, i.e., $I_j = (\lambda_j, \rho_j)$. As we will see below, we will always schedule events at the right endpoints of event intervals (intuitively, we can pretend that the sign of $f(t)$ within an event interval is the same as at its left endpoint and that it changes at its right endpoint). Observe that if $f(t)$ does not have any roots, then $\textsc{crop}\big(f(t)\big)$ does not return any intervals and no events will be scheduled. This is where our subroutine is more powerful than the subroutine of Milenkovic and Sacks [20], and this is why we can ensure that we only handle events if there is a real crossing of trajectories.

We use t_{curr} to denote the current time of the KDS, which is the maximum computed event time over all processed events. We assume that tests as to whether t_{curr} lies inside an event interval computed by CROP are exact. In the actual implementation, this can be achieved by enforcing all interval endpoints (and consequently,

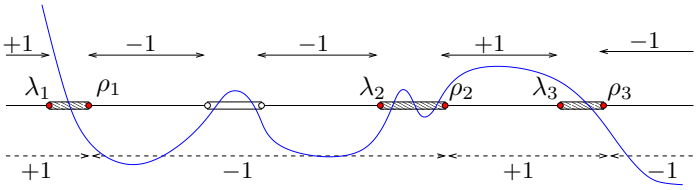


Figure 2. A function and the intervals computed by the CROP procedure. Intervals with filled (hollow) endpoints are event (turbulent) intervals; solid arrow lines denote the intervals where the sign of the function is known, and dashed arrow lines denote the signs pretended by the KDS.

t_{curr}) to be rationals and using exact arithmetic to compare between rationals. The pseudo-code for computing the failure time of a certificate c at time t_{curr} is given below.

Algorithm EVENTTIME (c)
1. $\mathcal{I} := \langle I_1 = (\lambda_1, \rho_1), \ldots, I_k = (\lambda_k, \rho_k)\rangle \leftarrow$ CROP (φ_c)
2. $\rho_0 \leftarrow -\infty$; $\rho_{k+1} \leftarrow +\infty$
3. $\mathsf{last} \leftarrow$# intervals in $\mathcal{I}$ to the left of t_{curr}
4. **if** $\chi_c(\rho_{\mathsf{last}}) = -1$
5. **then** return ρ_{last}
6. **else** return $\rho_{\mathsf{last}+1}$

Note that if $\chi_c(\rho_{\mathsf{last}}) = -1$, then the event time returned by EVENTTIME(c) (i.e., ρ_{last}) is in the past. As we will see in the next section, when we handle an event in the past, we do not reset t_{curr}: the time t_{curr} will always be the maximum of the computed event times over all processed events. Finally, the above procedure has the following properties: If it returns a finite value ρ_i, then

(I1) ρ_i is the right endpoint of an event interval;

(I2) the certificate c is valid at λ_i and invalid at ρ_i, i.e., $\chi_c(\lambda_i) = 1$ and $\chi_c(\rho_i) = -1$. In fact, c is valid at all times in $[\rho_{i-1}, \rho_i] \setminus \mathcal{R}_{\chi_c}$, and is invalid at all times in $[\rho_i, \rho_{i+1}] \setminus \mathcal{R}_{\chi_c}$.

Lemma 1 *Suppose* EVENTTIME(c) *returns a finite value* ρ_i. *For any* $t \in \mathbb{R}$, *if (i)* $t \in [\rho_{i-1}, \rho_i]$ *and* c *is invalid at time* t, *or (ii)* $t \in [\rho_i, \rho_{i+1}]$ *and* c *is valid at time* t, *then* $\chi_c(\gamma) = 0$ *for some* $\gamma \in (t - \varepsilon, t)$.

Proof: We only prove case (i) as case (ii) is similar. By (A1) and (I2), it is clear that $t \in \mathcal{R}_{\chi_c}$. As such, a turbulent or event interval (λ, ρ) of c contains t. Note that $\lambda \in [\rho_{i-1}, \rho_i]$, and therefore c is valid at time λ by (I2). However, c is invalid at time t by our assumption. This implies that there exists a value $\gamma \in (\lambda, t)$ such that $\chi_c(\gamma) = 0$. Finally, observe that $(\lambda, t) \subseteq (t - \varepsilon, t)$. □

3 Kinetic Sorting

Let S be a set of n points moving continuously on the real line. The value of a point $x \in S$ is a continuous function of time t, which we denote by $x(t)$. Let $S(t) = \{x(t) : x \in S\}$ denote the configuration of S at time t. For simplicity, we write S and x instead of $S(t)$ and $x(t)$, respectively, provided that no confusion arises. In the kinetic sorting problem, we want to maintain the sorted order of S during the motion.

The algorithm. As in the standard algorithm, we maintain an array A that stores the points in S. The events are stored in a priority queue Q, called global event queue. The certificates are standard as well: the certificate $c := \big[x < y\big]$ belongs to the current certificate set of the KDS if $x = A[k]$ and $y = A[k+1]$ for some $1 \le k \le n-1$. We call these $n-1$ certificates *active*. We need the following notation regarding the failure times.

$t_{\mathsf{cp}}(x,y)$: the computed failure time[1] of certificate $\big[x < y\big]$
$t_{\mathsf{pr}}(x,y)$: the time at which the failure of $\big[x < y\big]$ is actually processed
$t_{\mathsf{ex}}(x,y)$: the exact time at which the certificate $\big[x < y\big]$ fails

For the exact failure time, more formally,

$$t_{\mathsf{ex}}(x,y) := \arg \max_{t < t_{\mathsf{cp}}(x,y)} x(t) = y(t). \tag{1}$$

Note that $t_{\mathsf{ex}}(x,y) < t_{\mathsf{cp}}(x,y) \le t_{\mathsf{pr}}(x,y)$. Furthermore, we know by (I1) that $t_{\mathsf{cp}}(x,y)$ is the right endpoint of an event interval of c, and $t_{\mathsf{ex}}(x,y)$ lies inside that event interval by (1). As such, $t_{\mathsf{cp}}(x,y) < t_{\mathsf{ex}}(x,y) + \varepsilon$.

The new kinetic sorting algorithm is described below. The major difference with the standard algorithm is that we use the algorithm EVENTTIME to compute the failure time of a certificate.

Algorithm KINETICSORTING
1. $t_{\mathsf{curr}} \leftarrow -\infty$; Initialize A and Q.
2. **while** $Q \neq \emptyset$
3. **do** $c : \big[x < y\big] \leftarrow$ DELETEMIN(Q)
4. $t_{\mathsf{curr}} \leftarrow \max\{t_{\mathsf{curr}}, t_{\mathsf{cp}}(x,y)\}$
5. Swap x and y (which are adjacent in A).
6. Remove from Q all certificates that become inactive.
7. $\mathcal{C} \leftarrow$ set of new certificates that become active.
8. **for** each $c : \big[a < b\big] \in \mathcal{C}$
9. **do** $t_{\mathsf{cp}}(a,b) \leftarrow$ EVENTTIME (c)
10. **if** $t_{\mathsf{cp}}(a,b) \neq \infty$
11. **then** Insert $\big[a < b\big]$ into Q, with $t_{\mathsf{cp}}(a,b)$ as failure time.

Note that in lines 10–11, even in the case $t_{\mathsf{cp}}(a,b) < t_{\mathsf{curr}}$ for some certificate $\big[a < b\big] \in \mathcal{C}$ (i.e., the event lies in the past), we still insert this event into the queue because the certificate $\big[a < b\big]$ is not valid at t_{curr} and thus the combinatorial structure of the KDS is not correct. Apparently we missed an event, which we must still handle. As such, unlike the standard algorithm, our algorithm may process events in the past. Note that t_{curr} is not affected when this happens (see line 4).

Basic properties. The status of the KDS *at time* t is defined as the status of the KDS after all events whose processing times are at most t have been processed. In the kinetic sorting problem, the status refers to the maintained array A. We say that a point x *precedes* a point y in the maintained array A if $x = A[k]$ and $y = A[l]$ for some $k < l$. If $k = l - 1$, then x *immediately precedes* y.

Since events may be processed in a wrong order, the above KDS could perhaps get into an infinite loop. However, if a certificate c is processed by the algorithm (line 5) at time t_0 and c becomes active again at time t_0, then EVENTTIME ensures that the failure time of c is in the future. This implies that the algorithm does not get into an infinite loop. We next show the KDS almost always maintains a correctly sorted list in A.

[1]This is a slight abuse of notation, because points can swap more than once, so the same certificates can fail multiple times. It will be convenient to treat these certificates as different. Formally we should write $t_{\mathsf{cp}}((x,y), t_{\mathsf{curr}})$ for the failure time of the certificate $\big[x < y\big]$ computed by the KDS at time t_{curr}. Since this is always clear from the context we omit the time parameter.

Lemma 2 *If x immediately precedes y in A at time t_{curr}, then either (i) $x(t_{\mathsf{curr}}) \leq y(t_{\mathsf{curr}})$, which means the order is correct, or (ii) $x(\gamma) = y(\gamma)$ for some $\gamma \in (t_{\mathsf{curr}} - \varepsilon, t_{\mathsf{curr}})$.*

Proof: Let t^* be the last time less than or equal to t_{curr} at which x becomes a neighbor of y such that x is immediately preceding y. (Note that t^* may be equal to $-\infty$, referring to the time of initialization of the KDS; see lines 1 of KINETICSORTING.) Let $c = \big[x < y\big]$, and let $t_{\mathsf{cp}}(x, y)$ be the time returned by EVENTTIME(c) at time t^*. Since x and y are always adjacent between time t^* and t_{curr}, either $t_{\mathsf{cp}}(x, y) = \infty$, in which case the certificate failure is not scheduled (line 10), or $t_{\mathsf{curr}} < t_{\mathsf{cp}}(x, y) < \infty$, in which case the certificate failure is scheduled but not yet handled by the KDS. In either case, $t_{\mathsf{cp}}(x, y) > t_{\mathsf{curr}}$. Now assume case (i) is not true, i.e., the certificate c is invalid at time t_{curr}. By Lemma 1 (i), there exists a value $\gamma \in (t_{\mathsf{curr}} - \varepsilon, t_{\mathsf{curr}})$ such that $x(\gamma) - y(\gamma) = 0$, which is case (ii), as desired. □

The following theorem shows when the ordering maintained by the kinetic sorting algorithm is correct.

Theorem 1 ***(Correctness).*** *The ordering maintained by the kinetic sorting algorithm is correct except during at most μ time intervals of length at most ε, where μ is the number of collisions of points in S over the entire motion.*

Proof: Let $t \in \mathbb{R}$ be a time such that no two points of S collide within time $(t-\varepsilon, t)$. We claim that the ordering maintained by the KDS at time t must be correct. The theorem then follows since there are only μ collisions of points in S.

Suppose at time t there exist two points $x, y \in S$ that are adjacent in A but in incorrect order. By Lemma 2 applied to x and y at time t, we have $x(\gamma) = y(\gamma)$ for some $\gamma \in (t - \varepsilon, t)$. But this contradicts with our assumption that no two points collide within the time interval $(t - \varepsilon, t)$. Therefore all adjacent pairs of points in the maintained list A are in correct order, implying that the list A itself must also be correct. □

Delay of events. Theorem 1 shows that the ordering may be incorrect only near collision times, but many collisions may "cascade" and thus an event may not be processed for a long time, thereby resulting in a wrong ordering in the KDS for a long time. Specifically, when the failure of a certificate $\big[x < y\big]$ is handled by the KDS, we define its *delay* by $t_{\mathsf{pr}}(x, y) - t_{\mathsf{ex}}(x, y)$. Next we bound the maximum delay of an event. The bound holds when every pair of points swaps at most s times for some parameter $s > 0$.

Lemma 3 *Let $c = \big[x < y\big]$ be a certificate that fails at the exact time $t_{\mathsf{ex}}(x, y)$ and is handled by the KDS at time $t_{\mathsf{pr}}(x, y)$. Let τ be such that $t_{\mathsf{ex}}(x, y) \leq \tau < t_{\mathsf{pr}}(x, y) - \varepsilon$. Then there is a point $p \in S \setminus \{x\}$ such that $x(t) = p(t)$ for some $t \in (\tau, \tau + \varepsilon]$.*

Proof: Suppose to the contrary that $x(t) \neq p(t)$ for all $p \in S \setminus \{x\}$ during the interval $(\tau, \tau + \varepsilon]$. We first claim that $y(t) < x(t)$ during this interval. Indeed, otherwise we have $y(t) > x(t)$ and hence the certificate c is always valid during the interval $(\tau, \tau + \varepsilon]$. However, by applying Lemma 1 (ii) with $t = \tau + \varepsilon$, we know that $\chi_c(\gamma) = 0$ for some $\gamma \in (\tau, \tau + \varepsilon)$, a contradiction. (Note that $t = \tau + \varepsilon$ satisfies the condition of Lemma 1 (ii) because $t_{\mathsf{cp}}(x, y) < t < t_{\mathsf{pr}}(x, y)$.)

Next, let $A[\tau + \varepsilon]$, the list maintained by the algorithm at time $\tau + \varepsilon$, be $\langle \ldots, x = z_0, z_1, \ldots, z_m = y, \ldots \rangle$. Let $\mathcal{B} \subseteq \{z_1, \ldots, z_m\}$ be the subset of points that are smaller than x during the interval $(\tau, \tau + \varepsilon]$. Since no point collides with x during this interval, $\mathcal{B}$ remains fixed during $(\tau, \tau + \varepsilon]$. Note that $z_m \in \mathcal{B}$. Let $1 \leq i \leq m$ be the smallest index such that $z_i \in \mathcal{B}$. Then $z_i(t) < x(t) \leq z_{i-1}(t)$, for all $t \in (\tau, \tau + \varepsilon]$, and z_{i-1} immediately precedes z_i in $A[\tau + \varepsilon]$, which contradicts Lemma 2. This completes the proof of the lemma. □

Theorem 2 ***(Delay).*** *Suppose that the trajectories of every pair of points in S intersect at most s times. Then an event can be delayed by at most $ns \cdot \varepsilon$ time.*

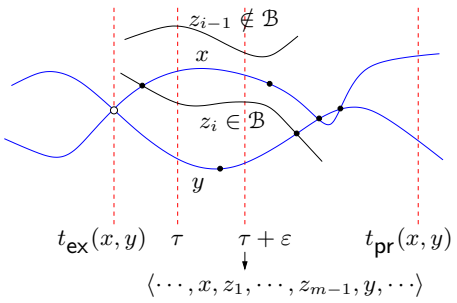


Figure 3. Illustration for the proof of Theorem 2.

Proof: Consider a certificate $c = \left[x < y\right]$ that fails at the exact time $t_{\mathsf{ex}}(x,y)$ and is handled by the KDS at time $t_{\mathsf{pr}}(x,y)$. Let t be a time such that $t_{\mathsf{ex}}(x,y) \le t$ and $t + \varepsilon < t_{\mathsf{pr}}(x,y)$. By Lemma 3, there is a point $p \in S \setminus \{x\}$ whose trajectory intersects the trajectory of x during $(t, t+\varepsilon]$. Let k be an integer such that $t_{\mathsf{pr}}(x,y) - t_{\mathsf{ex}}(x,y) = k\varepsilon + \delta$ where $\delta < \varepsilon$. We split the interval $[t_{\mathsf{ex}}(x,y), t_{\mathsf{pr}}(x,y)]$ into k intervals, each of width ε, and one interval (the last one) of width δ. Now we can charge each of the first k intervals to an intersection point of the trajectory of x and the trajectory of a point $p \in S$. Since any two trajectories intersect at most s times, k is at most $(n-1)s$, implying that $t_{\mathsf{pr}}(x,y) - t_{\mathsf{ex}}(x,y) \le (n-1)s \cdot \varepsilon + \delta < ns \cdot \varepsilon$. □

The following theorem shows the above bound on the delay is almost tight in the worst case.

Theorem 3 *For any n, there is a set S of n points such that the trajectories of any two points intersect at most s times and $t_{\mathsf{pr}}(x,y) - t_{\mathsf{ex}}(x,y) \ge (n-2)s \cdot \varepsilon$ for some pair $x, y \in S$.*

Proof: We first describe a lower bound example for linear motions. Let $S = \{x, y, x_1, \ldots, x_{n-2}\}$. The trajectories of x and y in the tz-plane are set to be $z = 1$ and $z = at + 1$ for a sufficiently small positive number a. The trajectory of the x_i's in tz-plane is parallel lines such that $i\delta < t_{\mathsf{ex}}(x, x_i) < t_{\mathsf{ex}}(y, x_i) < i\mu$ where δ and μ are two numbers satisfying the following inequalities:

$$\frac{n-1}{n}\varepsilon < \delta < \mu < \left(\frac{n-1}{n} + \frac{1}{n^2}\right)\varepsilon.$$

Now, assume CROP $(x(t) - y(t)) = ((\mu-\varepsilon)/2, (\mu+\varepsilon)/2)$, CROP $(x(t) - x_i(t)) = (i\delta, i\delta+\varepsilon)$ and CROP $(y(t) - x_i(t)) = (i\mu - \varepsilon, i\mu)$ for any $1 \le i \le n-2$ (see Figure 4(a)).

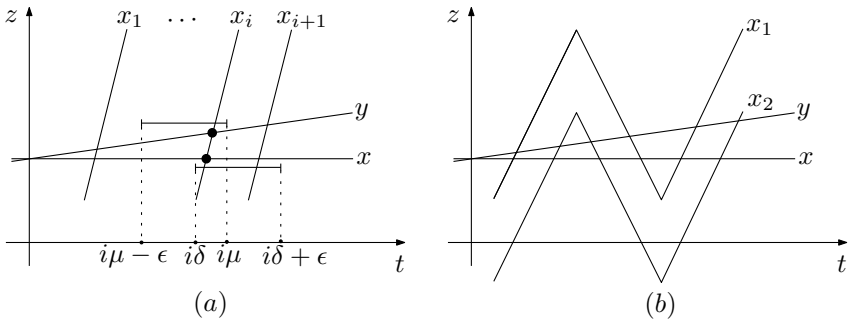


Figure 4. The lower-bound example.

Using induction, we prove that the status of the maintained list at time $i\mu$ ($i = 2, \ldots, n-2$) is

$$\langle x_{n-2}, \ldots, x_{i+1}, y, x_i, x_{i-1}, x, x_{i-2}, \ldots x_1 \rangle.$$

Since the maintained list at $-\infty$ is $\langle x_{n-2}, \ldots, x_1, y, x \rangle$ and the right endpoints of all returned intervals by CROP are greater than zero, the maintained list at time 0 is $\langle x_{n-2}, \ldots, x_1, y, x \rangle$. At time 0, the only certificate failures in the event queue are $t_{\mathsf{cp}}(x, y) = (\mu + \varepsilon)/2$ and $t_{\mathsf{cp}}(y, x_1) = \mu$. Since $\mu < (\mu + \varepsilon)/2$, the status of the KDS at time μ is $\langle x_{n-2}, \ldots, x_2, y, x_1, x \rangle$ and the certificate failures in the event queue are $t_{\mathsf{cp}}(x, x_1) = \delta + \varepsilon$, $t_{\mathsf{cp}}(y, x_2) = 2\mu$. Since $2\mu < \delta + \varepsilon$ (later we will show $(i+1)\mu < i\delta + \varepsilon$), the status of the KDS at time 2μ is $\langle x_{n-2}, \ldots, x_3, y, x_2, x_1, x \rangle$ which means the case $i = 2$ is true. Now assume the maintained list at time $i\mu$ is $\langle x_{n-2}, \ldots, x_{i+1}, y, x_i, x_{i-1}, x, x_{i-2}, \ldots x_1 \rangle$. We have to show that the maintained list at time $(i+1)\mu$ is $\langle x_{n-2}, \ldots, x_{i+2}, y, x_{i+1}, x_i, x, x_{i-1}, \ldots x_1 \rangle$.

The computed failure times of current certificates in the event queue are $t_{\mathsf{cp}}(x, x_{i-1}) = (i-1)\delta + \varepsilon$, $t_{\mathsf{cp}}(y, x_{i+1}) = (i+1)\mu$. Since $(i-1)\delta + \varepsilon < (i+1)\mu$, the point x_{i-1} swaps with the point x at time $(i-1)\delta + \varepsilon$ and at the same time $t_{\mathsf{cp}}(x, x_{i-1})$ is removed from the event queue and $t_{\mathsf{cp}}(x, x_i) = i\delta + \varepsilon$ is inserted into the event queue. We know that

$$i\delta + \varepsilon > \frac{n-1}{n} \cdot i\varepsilon + \varepsilon > \left(\frac{n-1}{n} + \frac{1}{n^2} \right) (i+1)\varepsilon > (i+1)\mu.$$

This implies at time $(i+1)\mu$ the points x_{i+1} and y swap and $t_{\mathsf{cp}}(y, x_{i+1})$ is removed from the event queue and instead $t_{\mathsf{cp}}(y, x_{i+2}) = (i+2)\mu$ which is greater than $(i+1)\mu$ is inserted into the event queue. Therefore, the status of the KDS at time $(i+1)\mu$ is

$$\langle x_{n-2}, \ldots, x_{i+2}, y, x_{i+1}, x_i, x, x_{i-1}, \ldots x_1 \rangle.$$

Now consider the time $(n-2)\mu$ at which the maintained list is

$$\langle y, x_{n-2}, x_{n-3}, x, x_{n-4}, \ldots, x_1 \rangle.$$

The only certificate failure scheduled in the KDS is for $[x_{n-3} < x]$, with failure time $(n-3)\delta + \varepsilon$. After processing this certificate failure, the only certificate failure in the event queue is $t_{\mathsf{cp}}(x, x_{n-2}) = (n-2)\delta + \varepsilon$. After processing $[x_{n-2} < x]$, we realize that the certificate $[y < x]$ which fails in the past must be processed. Therefore,

$$t_{\mathsf{pr}}(x, y) - t_{\mathsf{ex}}(x, y) = (n-2)\delta + \varepsilon > (n-2)\varepsilon.$$

We use the above construction as a base component to construct a lower-bound example for the general case where any two points can swap s times. To this end, we glue s base components together such that the slopes of lines alternate between being positive and negative, i.e., the slopes of lines in the first component is positive, in the second component is negative, and so on as depicted in Fig. 4(b). Note that in the odd components, certificates $[x_i < y]$ are roughly processed at the right time and certificates $[x_i < x]$ are roughly processed with a delay of ε, but in the even components, certificates $[x < x_i]$ are roughly processed at the right time and certificates $[y < x_i]$ are roughly processed with a delay of ε (indeed we can imagine that x and y are exchanged). The main condition that we need is $(i+1)\mu < i\delta + \varepsilon$ for any $i = 1, \ldots, s(n-2)$. We can satisfy this condition by choosing δ and μ such that

$$\frac{sn-1}{sn}\varepsilon < \delta < \mu < \left(\frac{sn-1}{sn} + \frac{1}{s^2 n^2} \right) \varepsilon.$$

Next we discuss what happens to the maintained list when two components are glued together. Because of symmetry, we just consider the status of the KDS around the time at which the first and the second component are glued together. Consider time $(n-2)\mu$ in which the KDS is

$$\langle y, x_{n-2}, x_{n-3}, x, x_{n-4}, \ldots, x_1 \rangle.$$

As we explained above, at time $(n-3)\delta+\varepsilon$, the certificate $[x_{n-3} < x]$ is processed and x and x_{n-2} become adjacent, which means $[x_{n-2} < x]$ must be scheduled. Because two intersections of x and x_{n-2} are at most ε far away from each other, we replace the previous assumption $t_{\mathsf{cp}}(x, x_{n-2}) = (n-2)\delta+\varepsilon$ with the assumption that the turbulent interval $((n-2)\delta, (n-2)\delta+\varepsilon)$ contains both intersections. Since CROP ignores turbulent intervals, the order of x and x_{n-2} does not change. Moreover, since $t_{\mathsf{cp}}(x_{n-2}, y) = (n-1)\delta+\varepsilon$ (recall that in the even components $[y < x_i]$ is processed with a delay of ε), x_{n-2} and y do not swap before time $(n-1)\delta+\varepsilon$. This implies x and y cannot get adjacent before $(n-1)\delta+\varepsilon$. On the other hand, x_{n-3} and x must swap before this time—note that $t_{\mathsf{cp}}(x_{n-3}, x) = n\mu$. After time $(n-1)\delta+\varepsilon$, the same scenario as the first component happens. Putting everything together we conclude that $t_{\mathsf{pr}}(x, y) - t_{\mathsf{ex}}(x, y) \geq (n-2)s \cdot \varepsilon$ in the above construction. □

Error bounds. We turn our attention to the "error" in the array A. Combinatorially, Lemma 2 implies that if there are k event intervals containing t_{curr}, then the array A at time t_{curr} can be decomposed into at most $k+1$ (contiguous) subarrays, each of which is in sorted order. Next we discuss how far the maintained order can be from the correct order geometrically. In particular, we present a bound on the maximum distance between two points that are in the wrong order in the array and on how far away the k-th point in the maintained order—that is, the point $A[k]$—can be from the true point of rank k.

Theorem 4 ***(Geometric error).*** *Let $\langle y_1, \ldots, y_n\rangle$ and $\langle z_1, \ldots, z_n\rangle$ be the sequence maintained by the algorithm and the correctly sorted sequence at some given time t_{curr}, respectively. Let V_{max} be the maximum velocity of any point in S over the time interval $[t_{\mathsf{curr}} - \varepsilon, t_{\mathsf{curr}}]$. Then for any $1 \leq i < j \leq n$,*

(i) $y_i(t_{\mathsf{curr}}) - y_j(t_{\mathsf{curr}}) \leq (j-i+1)\varepsilon \cdot V_{\mathsf{max}}$, *and*

(ii) $|y_i(t_{\mathsf{curr}}) - z_i(t_{\mathsf{curr}})| \leq n\varepsilon \cdot V_{\mathsf{max}}$.

Proof:

(i) For simplicity we write $t = t_{\mathsf{curr}}$. For any $1 \leq k < n$, if y_k and y_{k+1} are in the correct order in the maintained list, then $y_k(t) \leq y_{k+1}(t)$. If they are in the incorrect order, then by Lemma 2 (ii), there exists a time $\gamma \in (t-\varepsilon, t)$ such that $y_k(\gamma) = y_{k+1}(\gamma)$. Hence,

$$y_k(t) - y_{k+1}(t) = (y_k(t) - y_k(\gamma)) + (y_k(\gamma) - y_{k+1}(\gamma)) + (y_{k+1}(\gamma) - y_{k+1}(t)) \leq 2\varepsilon V_{\mathsf{max}}.$$

Therefore we always have $y_k(t) - y_{k+1}(t) \leq 2\varepsilon V_{\mathsf{max}}$, which immediately implies that

$$y_i(t) - y_j(t) = \sum_{\ell=i}^{j-1} (y_\ell(t) - y_{\ell+1}(t)) \leq 2(j-i)\varepsilon \cdot V_{\mathsf{max}}$$

for any $1 \leq i < j \leq n$. To further prove the promised upper bound, let us consider bounding $y_k(t) - y_{k+2}(t)$. If either $y_k(t) \leq y_{k+1}(t)$ or $y_{k+1}(t) \leq y_{k+2}(t)$, then we immediately have

$$y_k(t) - y_{k+2}(t) = (y_k(t) - y_{k+1}(t)) + (y_{k+1}(t) - y_{k+2}(t)) \leq 2\varepsilon V_{\mathsf{max}}.$$

Now assume $y_k(t) > y_{k+1}(t) > y_{k+2}(t)$, which means that the relative order of y_k and y_{k+1}, as well as the relative order of y_{k+1} and y_{k+2} are incorrect in the maintained list. As such, there exist $\gamma_1, \gamma_2 \in (t-\varepsilon, t)$ such that $y_k(\gamma_1) = y_{k+1}(\gamma_1)$ and $y_{k+1}(\gamma_2) = y_{k+2}(\gamma_2)$. It follows that

$$\begin{aligned} y_k(t) - y_{k+2}(t) &= (y_k(t) - y_k(\gamma_1)) + (y_k(\gamma_1) - y_{k+1}(\gamma_1)) + (y_{k+1}(\gamma_1) - y_{k+1}(\gamma_2)) \\ &\quad + (y_{k+1}(\gamma_2) - y_{k+2}(\gamma_2)) + (y_{k+2}(\gamma_2) - y_{k+2}(t)) \\ &\leq |t - \gamma_1| \cdot V_{\mathsf{max}} + 0 + |\gamma_1 - \gamma_2| \cdot V_{\mathsf{max}} + 0 + |t - \gamma_2| \cdot V_{\mathsf{max}} \\ &\leq 2\varepsilon V_{\mathsf{max}}. \end{aligned}$$

Hence we always have $y_k(t) - y_{k+2}(t) \leq 2\varepsilon V_{\mathsf{max}}$. Now, for any $1 \leq i < j \leq n$, one can prove $y_i(t) - y_j(t) \leq (j-i+1)\varepsilon V_{\mathsf{max}}$ by a simple induction on $j-i$ (the base case $j-i=1$ has been proved above):

$$\begin{aligned} y_i(t) - y_j(t) &= (y_i(t) - y_{i+2}(t)) + (y_{i+2}(t) - y_j(t)) \\ &\leq 2\varepsilon V_{\mathsf{max}} + (j-(i+2)+1)\varepsilon V_{\mathsf{max}} \\ &\leq (j-i+1)\varepsilon V_{\mathsf{max}}. \end{aligned}$$

(ii) We consier the case $z_i \neq y_i$; otherwise the claim is trivially true. Suppose $z_i = y_j$ for some $j > i$; the other case $j < i$ is symmetric. Also suppose $y_i = z_k$ for some $1 \leq k \leq n$. We have two cases. If $k > i$, then since $y_j(t_{\mathsf{curr}}) = z_i(t_{\mathsf{curr}}) \leq z_k(t_{\mathsf{curr}}) = y_i(t_{\mathsf{curr}})$, we can write

$$|z_i(t_{\mathsf{curr}}) - y_i(t_{\mathsf{curr}})| = y_i(t_{\mathsf{curr}}) - y_j(t_{\mathsf{curr}}) \leq n\varepsilon \cdot V_{\mathsf{max}},$$

by (i). Otherwise if $k < i$, there must exist r and ℓ with $r < i < \ell$, such that $z_\ell = y_r$. Then

$$\begin{aligned} |z_i(t_{\mathsf{curr}}) - y_i(t_{\mathsf{curr}})| &= z_i(t_{\mathsf{curr}}) - z_k(t_{\mathsf{curr}}) \leq z_\ell(t_{\mathsf{curr}}) - z_k(t_{\mathsf{curr}}) \\ &= y_r(t_{\mathsf{curr}}) - y_i(t_{\mathsf{curr}}) \leq n\varepsilon \cdot V_{\mathsf{max}}, \end{aligned}$$

by (i), thus proving the theorem.

□

4 Kinetic Tournaments

A kinetic tournament [6] is a KDS that maintains the maximum of a set S of moving points in $\mathbb{R}$ by maintaining a tournament tree $\mathcal{T}$ over S. Each interior node u of $\mathcal{T}$ has a certificate of the form $\big[x < y\big]$, where $x, y \in S$ are the two points stored at the children of u, and y is also currently stored at u. To handle events, we need a subroutine that compares two points at time t_{curr} in a way that is consistent with EVENTTIME.

Algorithm COMPUTEMAX(x, y)
1. $\mathcal{I} := \langle I_1 = (\lambda_1, \rho_1), \ldots, I_k = (\lambda_k, \rho_k)\rangle \leftarrow$ CROP $(x(t) - y(t))$
2. $\rho_0 \leftarrow -\infty$
3. last $\leftarrow$number of intervals in $\mathcal{I}$ to the left of t_{curr}
4. **if** $\mathrm{sign}(x(\rho_{\mathsf{last}}) - y(\rho_{\mathsf{last}})) = 1$
5. **then** return x
6. **else** return y

In the algorithm below, the point stored at a node $u \in \mathcal{T}$ is denoted by p_u, and we assume $\mathrm{parent}(root) =$ **nil**.

Algorithm KINETICTOURNAMENT
1. $t_{\mathsf{curr}} \leftarrow -\infty$; Initialize $\mathcal{T}$ and Q.
2. **while** $Q \neq \emptyset$
3. **do** $c : \big[x < y\big] \leftarrow$ DELETEMIN(Q)
4. $t_{\mathsf{curr}} \leftarrow t_{\mathsf{cp}}(x, y)$
5. $u \leftarrow$ the node at which the certificate c fails.
6. **while** $u \neq$ **nil**
7. **do** Let z_1 and z_2 be the points stored at u's children.
8. $p_u \leftarrow$COMPUTEMAX(z_1, z_2); $u \leftarrow \mathrm{parent}(u)$

9. Remove from Q all certificates that become inactive.
10. $\mathcal{C} \leftarrow$ set of new certificates that become active.
11. **for** each $c : \left[a < b\right] \in \mathcal{C}$
12. **do** $t_{\mathsf{cp}}(a, b) \leftarrow$EVENTTIME (c)
13. **if** $t_{\mathsf{cp}}(a, b) \neq \infty$
14. **then** Insert $\left[a < b\right]$ into Q, with $t_{\mathsf{cp}}(a, b)$ as failure time.

The set $\mathcal{C}$ in line 10 consists of certificates that correspond the nodes along the path from the node where the event occurs to the root. In lines 5–8, the algorithm has used COMPUTEMAX to make sure that each certificate $c \in \mathcal{C}$ is valid at the right endpoint of the last event interval of c before time t_{curr}. Since COMPUTEMAX (line 8) and EVENTTIME (line 12) base their decisions on the order at the same time, we obtain the following lemma.

Lemma 4 *In line 12, the computed event time $t_{\mathsf{cp}}(a, b)$ is always in the future (i.e., $t_{\mathsf{cp}}(a, b) > t_{\mathsf{curr}}$).*

The lemma implies that we never schedule an event in the past and, in fact, never schedule an event at the current time either. Hence, the algorithm does not get into an infinite loop.

Lemma 5 *After an event has been processed at time t_{curr}, the point p_u stored at any internal node u of the tournament is always one of the points stored at its children. Moreover, either p_u is the correct current maximum of the two children, or the trajectories of points stored at the two children intersect during the period $(t_{\mathsf{curr}} - \varepsilon, t_{\mathsf{curr}})$.*

Proof: It is obvious that the first part of the lemma is true. The proof of the second part is similar to Lemma 2. Assume there is a node u with children u_1 and u_2, and assume without loss of generality that $p_u = p_{u_1}$ while in fact $p_{u_2}(t_{\mathsf{curr}}) > p_{u_1}(t_{\mathsf{curr}})$. Let t^* be the last time at which p_{u_1} and p_{u_2} were compared. Thus COMPUTEMAX(p_{u_1}, p_{u_2}) executed at time t^* returns p_{u_1}. But then, since $p_{u_2}(t_{\mathsf{curr}}) > p_{u_1}(t_{\mathsf{curr}})$, an event must have been scheduled for the certificate $c = [p_{u_2} < p_{u_1}]$, and the failure time t' of this certificate must have satisfied $t' > t^*$ by Lemma 4. We cannot have $t' < t_{\mathsf{curr}}$, because that contradicts the definition of t^*. Hence $t' \geq t_{\mathsf{curr}}$. Since c is invalid at time t_{curr}, by Lemma 1 (i), it follows that that the trajectories of p_{u_1} and p_{u_2} must intersect during the period $(t_{\mathsf{curr}} - \varepsilon, t_{\mathsf{curr}})$. □

Following standard KDS terminology, we call an event *external* if the attribute to be maintained changes due to the event; for a kinetic tournament this means an event where the maximum of S changes. Other events are *internal*.

Lemma 6 *If there is no external event during the period $(t_{\mathsf{curr}} - \varepsilon, t_{\mathsf{curr}})$, then the maximum maintained by the algorithm is correct at time t_{curr}.*

Proof: By assumption, the true maximum of S during $(t_{\mathsf{curr}} - \varepsilon, t_{\mathsf{curr}})$ is a unique point, x. In particular, x does not cross any other point in S during this time period. Suppose for the sake of contradiction that x is not the maximum maintained by the algorithm at time t_{curr}. Then at time t_{curr}, the algorithm stores x at an internal node v of the tournament tree, and stores another point $y \in S$ in the sibling and the parent u of v. Applying Lemma 5 to the node u, we obtain that the trajectories of x and y intersect at some time in $(t_{\mathsf{curr}} - \varepsilon, t_{\mathsf{curr}})$, a contradiction. □

The following two results are immediate consequences of Lemma 6.

Theorem 5 ***(Correctness).*** *The maximum maintained by the kinetic tournament is correct except during at most μ time intervals of length at most ε, where μ is the number of external events.*

Theorem 6 ***(Delay).*** *If a point $x \in S$ becomes the true maximum at time t (i.e., an external event at time t), then either x becomes the maintained maximum by time $t + \varepsilon$ (i.e., the external event is delayed by at most ε), or another external event occurs before time $t + \varepsilon$ (i.e., the old external event becomes obsolete).*

We now turn our attention to the geometric error of our KDS—the difference in value between the point stored in the root of the kinetic tournament tree and the true maximum—as a function of the maximum velocity. Interestingly, the geometric error is much smaller than in the sorting KDS, because it now depends on the depth of the tournament tree, which is $\lceil \log n \rceil$. The following theorem makes this precise.

Theorem 7 ***(Geometric error).*** *Let x denote the point stored in the root of the kinetic tournament tree at some time t_{curr}, and let y denote the point with the maximum value at time t_{curr}. Then $x(t_{\mathsf{curr}}) \geq y(t_{\mathsf{curr}}) - (\lceil \log n \rceil + 1)\varepsilon \cdot V_{\mathsf{max}}$, where V_{max} is the maximum velocity of any point in S over the time interval $[t_{\mathsf{curr}} - \varepsilon, t_{\mathsf{curr}}]$.*

Proof: Consider a node v (other than the root) and its parent u. We claim that

$$p_v(t_{\mathsf{curr}}) - p_u(t_{\mathsf{curr}}) \leq 2\varepsilon V_{\mathsf{max}}. \tag{2}$$

If $p_v(t_{\mathsf{curr}}) \leq p_u(t_{\mathsf{curr}})$, (2) is trivially true. Otherwise, by Lemma 5, the trajectories of p_v and p_u intersect at some time in $(t_{\mathsf{curr}} - \varepsilon, t_{\mathsf{curr}})$. Arguing as in Theorem 4 (i), we can then obtain (2). Summing up (2) for all consecutive nodes along the path from the node storing the true maximum y to the root, we obtain $y(t_{\mathsf{curr}}) - x(t_{\mathsf{curr}}) \leq 2h\varepsilon \cdot V_{\mathsf{max}}$, where $h \leq \lceil \log n \rceil$ is the length of the path. The inequality can be further improved to $y(t_{\mathsf{curr}}) - x(t_{\mathsf{curr}}) \leq (h+1)\varepsilon \cdot V_{\mathsf{max}}$ by using the same argument as in Theorem 4 (i), thus completing the proof. □

5 Kinetic Range Trees

Our robust kinetic sorting algorithm can be applied directly to maintaining the standard kinetic range trees [7] of a set S of moving points in $\mathbb{R}^d$ for orthogonal range searching. By the properties of the robust kinetic sorting algorithm, we immediately know that the robust kinetic range tree is correct except for at most E time intervals of length at most ε, where E is the total number of swaps of the input points along each axis, and that the delay of each event is at most $O(n\varepsilon)$.

We can also prove bounds on the geometric error. For a d-dimensional (axis-aligned) box $R = \Pi_{i=1}^{d}[a_i, b_i]$ and a parameter $\Delta > 0$, let $R_\Delta^- = \Pi_{i=1}^{d}[a_i + \Delta, b_i - \Delta]$ and $R_\Delta^+ = \Pi_{i=1}^{d}[a_i - \Delta, b_i + \Delta]$. We call a subset $Q \subseteq S$ a *Δ-approximation* to $S \cap R$ if

$$S \cap R_\Delta^- \subseteq Q \subseteq S \cap R_\Delta^+.$$

In other words, points at L_∞-distance at most Δ to the boundary of R may or may not be included in Q, but other points are in Q if and only if they are in R. The next theorem shows that the kinetic range tree, when using our robust kinetic sorting algorithm, always returns a Δ-approximation to the true answer of an orthogonal range query, for an appropriate value of Δ. This follows more or less from Theorem 4. (The fact that the maintained tree is not necessarily a correct search tree does not impose any difficulty upon performing a standard binary search on the tree.)

Theorem 8 *For any time t and any d-dimensional (axis-aligned) box $R \subseteq \mathbb{R}^d$, the subset $Q(t) \subseteq S(t)$ returned by querying R on the maintained kinetic range tree at time t is a Δ-approximation to $S(t) \cap R$, where $\Delta = n\varepsilon V_{\mathsf{max}}$ and V_{max} is the maximum speed of a point in S over the time interval $[t - \varepsilon, t]$.*

Proof: We proceed by induction on d. Let us first consider the one-dimensional case, where a range tree of S is simply a binary search tree of S. Let $\langle y_1(t), y_2(t), \ldots, y_n(t)\rangle$ be the sequence maintained by the algorithm; also let $y_0 = -\infty$ and $y_{n+1} = +\infty$. Suppose for a query range $R = [a, b]$ the maintained tree returns $Q(t) = \langle y_i, y_{i+1}, \ldots, y_j\rangle$. Observe that although the maintained binary search tree is not necessarily correct, we still have $y_{i-1} < a \le y_i$ and $y_j \le b < y_{j+1}$. By Theorem 4, for each $i \le \ell \le j$, $y_\ell \ge y_i - \Delta \ge a - \Delta$ and $y_\ell \le y_j + \Delta \le b + \Delta$. Thus $Q(t) \subseteq S(t) \cap R^+_\Delta$. On the other hand, for each $\ell < i$, $y_\ell \le y_{i-1} + \Delta < a + \Delta$, and for each $\ell > j$, $y_\ell \ge y_{j+1} - \Delta > b - \Delta$. This implies $S(t) \cap R^-_\Delta \subseteq Q(t)$. Hence $Q(t)$ is a Δ-approximation to $S(t) \cap [a, b]$.

In $\mathbb{R}^d$, to perform a query $R = \Pi_{i=1}^d [a_i, b_i]$ on the maintained d-dimensional range tree, one first performs the query $[a_1, b_1]$ on the primary range tree, and then performs the query $\Pi_{i=2}^d [a_i, b_i]$ recursively into appropriate secondary range trees. Let $S' \subseteq S$ be the subset of points stored in those queried secondary trees. It follows from the above analysis that

$$S(t) \cap \Big([a_1 + \Delta, b_1 - \Delta] \times \mathbb{R}^{d-1}\Big) \subseteq S'(t) \subseteq S(t) \cap \Big([a_1 - \Delta, b_1 + \Delta] \times \mathbb{R}^{d-1}\Big). \tag{3}$$

Furthermore, by the induction hypothesis,

$$S'(t) \cap \Big(\mathbb{R} \times \prod_{i=2}^{d} [a_i + \Delta, b_i - \Delta]\Big) \subseteq Q(t) \subseteq S'(t) \cap \Big(\mathbb{R} \times \prod_{i=2}^{d} [a_i - \Delta, b_i + \Delta]\Big). \tag{4}$$

Putting (3) and (4) together, we obtain $S(t) \cap R^-_\Delta \subseteq Q(t) \subseteq S(t) \cap R^+_\Delta$, as desired. □

6 Experiments

We have implemented our robust kinetic sorting and kinetic tournament algorithms to test the effectiveness of our technique for handling out-of-order event processing. The programs are written in C++ and run in the Linux 2.4.20 environment. We also implemented these two algorithms using the traditional KDS event-scheduling approach and compared them with their robust counterparts by testing the errors in the output.

Input data. We used the following synthetic datasets in our experiments, as illustrated in Figure 5. The inputs are low-degree motions because we have not yet implemented a full-fledged CROP procedure, and it becomes easier for us to compute delays of the events. Nonetheless, these inputs already cause trouble to traditional KDS's and are sufficient to illustrate the effectiveness of our algorithms.

- GRIDS: a set of linear trajectories whose dual points form a uniform grid;
- PARABOLA: a set of congruent parabolic trajectories with apexes sitting on a grid in tx-plane;
- RANDDC: a set of linear trajectories whose dual points are randomly distributed in a disk;
- RANDCR: a set of linear trajectories whose dual points are randomly distributed on a circle.

Kinetic sorting. We tested the kinetic sorting algorithms on the first three types of input data. All experiments were run on inputs of size 900. We measure the error of a sorting KDS at time t by

$$\mathrm{err}(t) = \max_i |y_i(t) - z_i(t)|,$$

where $\langle y_1, \ldots, y_n\rangle$ and $\langle z_1, \ldots, z_n\rangle$ are the sequence maintained by the KDS and the correctly sorted sequence at time t respectively. In Figures 6-8 we plot $\mathrm{err}(t)$ as t varies, by measuring $\mathrm{err}(t)$ every other 10^{-7} seconds.

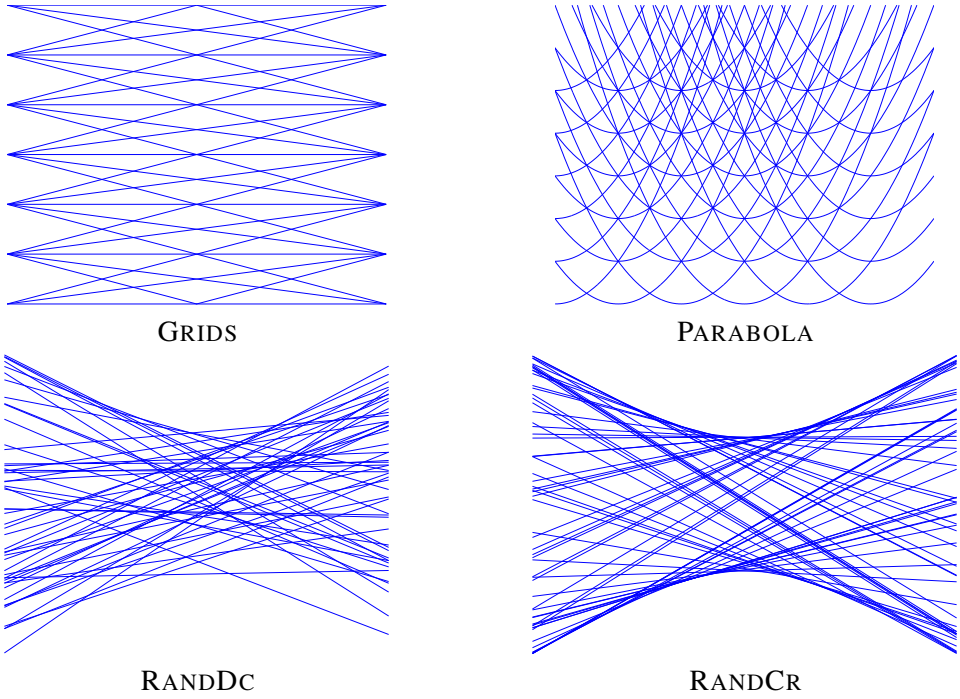


Figure 5. Datasets used in the experiments. The figures depict trajectories of the moving points in tx-plane, after an appropriate scaling.

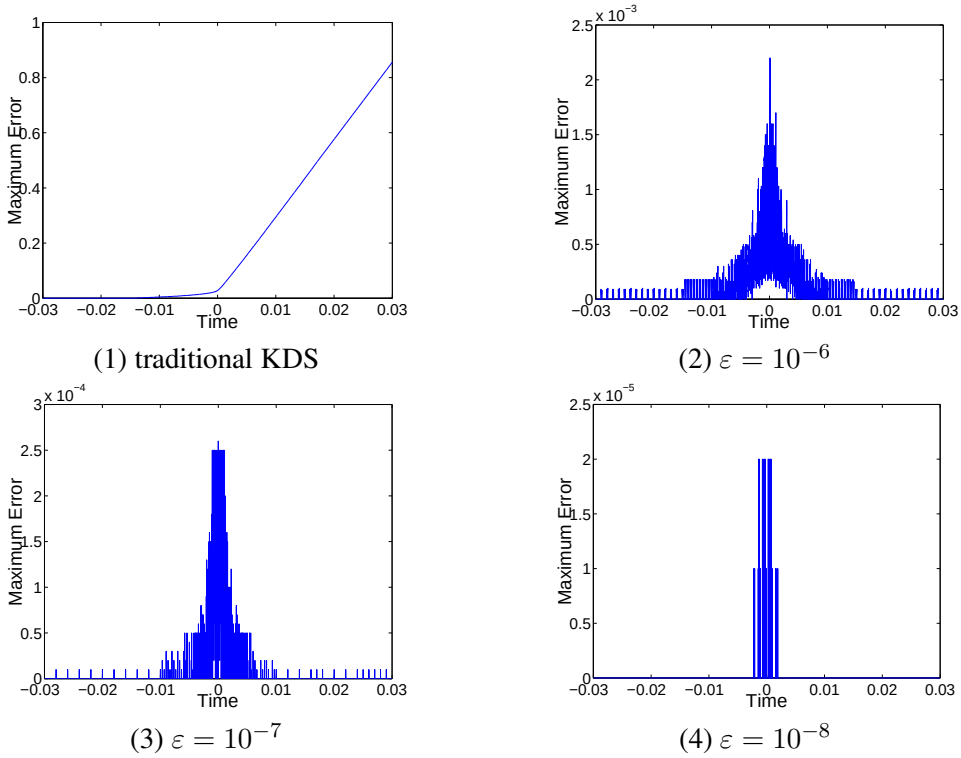
(The correctly sorted sequence at time t is computed by sorting their values at time t using floating point arithmetic.) Note the different scales on the vertical axis in these figures.

We first discuss the behavior of the traditional kinetic sorting algorithm, which uses floating point arithmetic. In a few instances, the algorithm went into an infinite loop because of simultaneous events. Although this problem could be fixed in general, a more careful implementation of the traditional KDS is required. As for the geometric error in the maintained structures, the traditional KDS was very fragile: it quickly ran into noticeable errors and was unable to recover from these errors (see Figures 6 (1), 7 (1), and 8 (1)). The reason is that some events that should have been scheduled into the global queue were discarded by the KDS because their computed event times happened to lie in the past because of numerical errors.

We now turn our attention to the geometric error in the structures maintained by our robust kinetic sorting algorithm, under different precisions ε in the CROP procedure. As can be seen, while the traditional KDS quickly ran into serious errors and was never able to recover, our robust KDS maintained a rather small error all the time. Observe that the error of the robust KDS reduces as the precision of the CROP procedure increases. We also tested the algorithm on a number of larger inputs, and the error remained roughly the same.

We also studied how long an event could be delayed before it is eventually processed in the robust kinetic sorting algorithm—see Table 1. It can be seen that the Root Mean Square (RMS, for short) of the delays are always very small for all inputs. As for the maximum delay, we only observed one instance in the first two types of inputs in which some events are delayed by about 2ε; in all other cases, the maximum delay never exceeds ε, which is far below the rather contrived worst-case bound in Theorem 3.

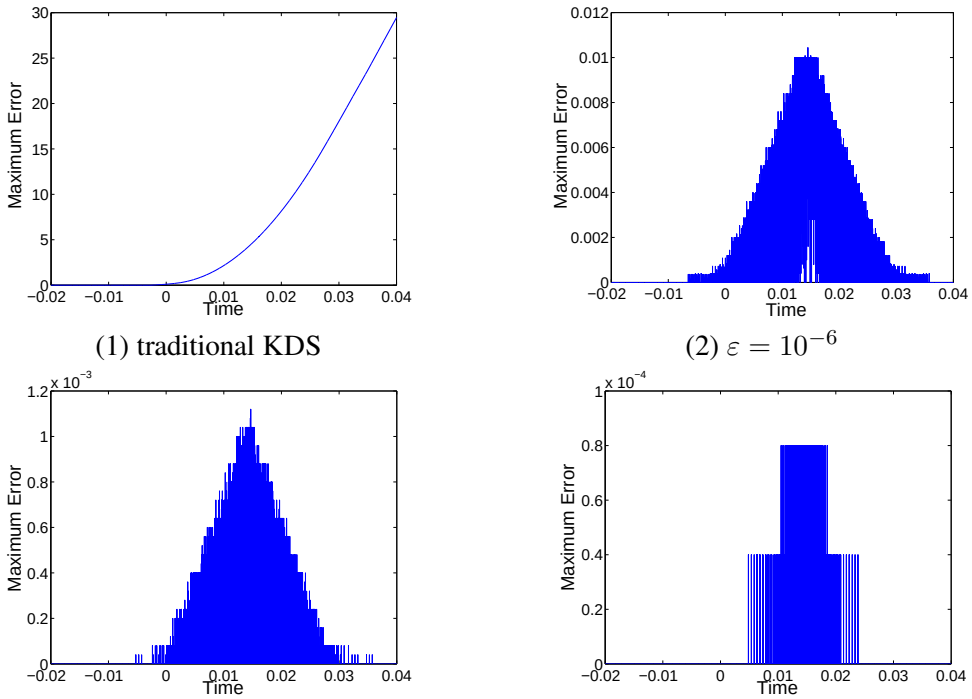
Kinetic tournament. We tested the kinetic tournament algorithms on the RANDCR data as this input tends to have a large number of external events. The geometric error is measured by $\mathrm{err}(t) = z(t) - y(t)$, where y and z are the maximum maintained by the KDS and the true maximum at time t respectively. Since kinetic



(1) traditional KDS (2) $\varepsilon = 10^{-6}$

(3) $\varepsilon = 10^{-7}$ (4) $\varepsilon = 10^{-8}$

Figure 6. Maximum error of kinetic sorting on a GRID input of size 900; scales on the vertical axis are different.



(1) traditional KDS (2) $\varepsilon = 10^{-6}$

(3) $\varepsilon = 10^{-7}$ (4) $\varepsilon = 10^{-8}$

Figure 7. Maximum error of kinetic sorting on a PARABOLA input of size 900; scales on the vertical axis are different.

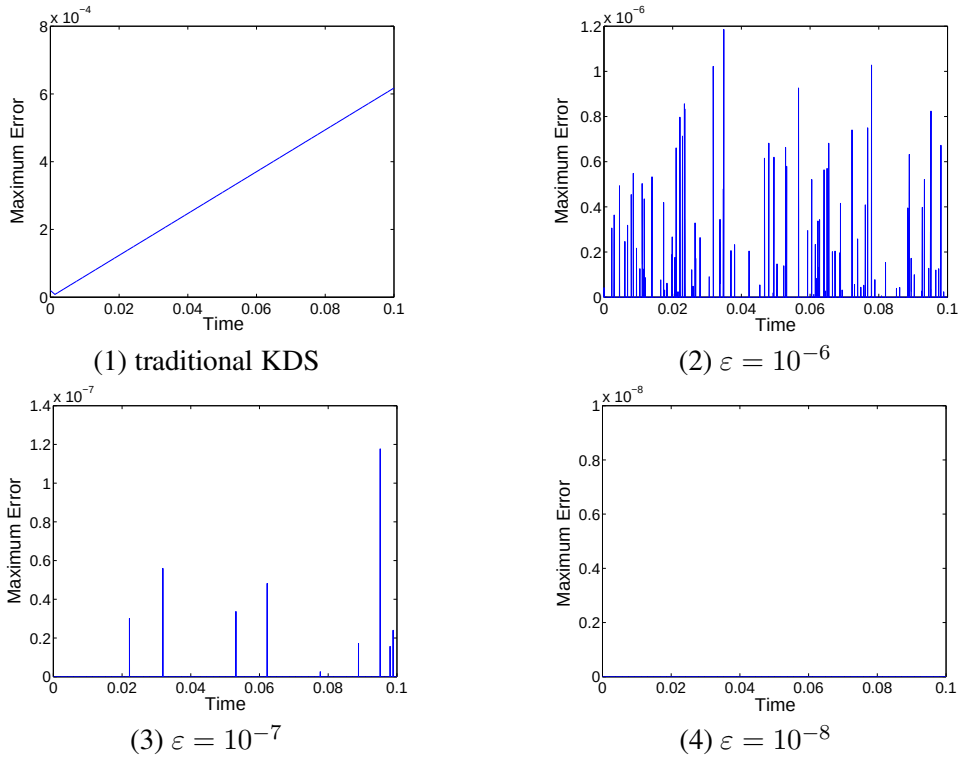


Figure 8. Maximum error of kinetic sorting on a RANDDC input of size 900; scales on the vertical axis are different.

Precision of CROP	GRIDS		PARABOLA		RANDDC	
	RMS	Max	RMS	Max	RMS	Max
$\varepsilon = 10^{-6}$	$0.48 \times \varepsilon$	$2.00 \times \varepsilon$	$0.37 \times \varepsilon$	$1.00 \times \varepsilon$	$0.42 \times \varepsilon$	$1.00 \times \varepsilon$
$\varepsilon = 10^{-7}$	$0.43 \times \varepsilon$	$1.00 \times \varepsilon$	$0.37 \times \varepsilon$	$1.00 \times \varepsilon$	$0.42 \times \varepsilon$	$1.00 \times \varepsilon$
$\varepsilon = 10^{-8}$	$0.42 \times \varepsilon$	$1.00 \times \varepsilon$	$0.39 \times \varepsilon$	$1.00 \times \varepsilon$	$0.41 \times \varepsilon$	$1.00 \times \varepsilon$

Table 1. Delay of events in kinetic sorting.

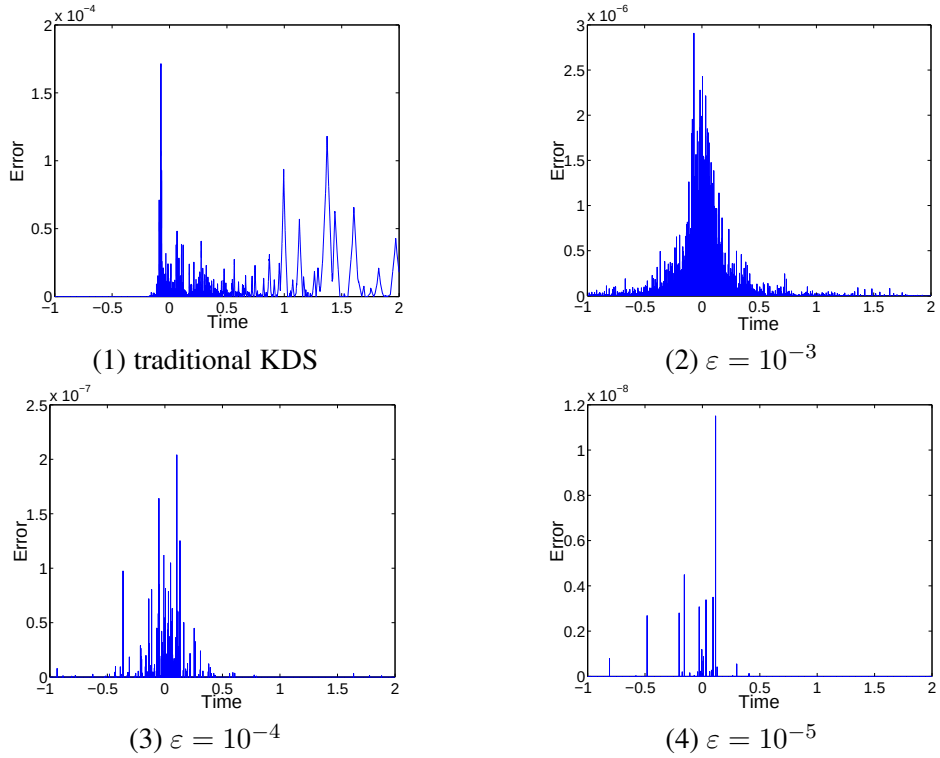


Figure 9. Geometric error of the kinetic tournament on a RANDCR input of size 10000; scales on the vertical axis are different.

tournaments are less sensitive to simultaneous events than kinetic sorting, we artificially lowered the precision in computing the event times so as to cause noticeable geometric errors in the tested algorithms. Specifically, in the traditional KDS we round the event times to the precision of 10^{-5}, and in the robust KDS we vary the precision ε in CROP from 10^{-3} to 10^{-5}.

We first noticed that the traditional kinetic tournament algorithm did not go into an infinite loop; this is because events are always "pushed" up in the tournament tree. However, as for the geometric error, one can see from Figure 9 (1) that the KDS maintains a rather inaccurate maximum over time. In contrast, the geometric errors in our robust KDS are smaller by orders of magnitudes, even though the event time computation is less precise than in the traditional KDS.

7 Conclusions

In this paper we studied the problem of designing kinetic data structures that are robust against out-of-order event processing due to numerical errors in computing event times. We showed that the proposed robust kinetic sorting and kinetic tournament algorithms have several nice properties, including guaranteed correctness for all but a finite number of small time intervals, short delays in event processing, and small geometric errors over time. Combining the resulting kinetic range tree and kinetic tournament, we can also maintain the closest-pair of a set of moving points robustly [7]. It is interesting to see whether similar results can be obtained for other more complex kinetic data structures as well. In particular, so far we have been unable to extend our techniques to kinetic Delaunay triangulations. The main difficulty lies in proposing a strategy that reconstructs the Delaunay triangulation when the current triangulation is no longer a planar embedding. We leave it as an interesting open question for future research.

References

[1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. *J. Comput. Syst. Sci.*, 66:207–243, 2003.

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

[3] P. K. Agarwal, L. J. Guibas, J. Hershberger, and E. Veach. Maintaining the extent of a moving point set. *Discrete Comput. Geom.*, 26:353–374, 2001.

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

[5] G. Alexandron, H. Kaplan, and M. Sharir. Kinetic and dynamic data structures for convex hulls and upper envelopes. *Comput. Geom. Theory Appl.*, 36:144–158, 2006.

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

[7] J. Basch, L. J. Guibas, and L. Zhang. Proximity problems on moving points. In *Proc. 13th Annu. Sympos. Comput. Geom.*, pages 344–351, 1997.

[8] The CGAL Library. `http://www.cgal.org/`.

[9] G. Collins and A. Akritas. Polynomial real root isolation using descarte's rule of signs. In *Proc. 3rd ACM Sympos. Symbol. Algebra. Comput.*, pages 272–275, 1976.

[10] The Core Library. `http://www.cs.nyu.edu/exact/`.

[11] S. Fortune. Progress in computational geometry. In R. Martin, editor, *Directions in Geometric Computing*, pages 81–128. Information Geometers Ltd., 1993.

[12] S. Funke, C. Klein, K. Mehlhorn, and S. Schmitt. Controlled perturbation for Delaunay triangulations. In *Proc. 16th ACM-SIAM Sympos. Discrete Algorithms*, pages 1047–1056, 2005.

[13] L. J. Guibas. Algorithms for tracking moving objects. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, 2nd edition, pp. 1117–1134, 2004.

[14] L. J. Guibas and M. Karavelas. Interval methods for kinetic simulation. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 255–264, 1999.

[15] L. J. Guibas, M. Karavelas, and D. Russel. A computational framework for handling motion. In *Proc. 6th Workshop on Algorithm Engineering and Experiments*, pages 129–141, 2004.

[16] L. J. Guibas and D. Russel. An empirical comparison of techniques for updating Delaunay triangulations. In *Proc. 20th Annu. Sympos. Comput. Geom.*, pages 170–179, 2004.

[17] D. Halperin and C. Shelton. A perturbation scheme for spherical arrangements with application to molecular modeling. *Comput. Geom. Theory Appl.*, 10:273–287, 1998.

[18] N. Jacobson. *Basic Algebra I*. W. H. Freeman, New York, 2nd edition, 1985.

[19] K. Mehlhorn, R. Osbild, and M. Sagraloff. Reliable and efficient computational geometry via controlled perturbation. In *Proc. 33rd Internat. Colloq. Automat. Langu. Program.*, pages 299–310, 2006.

[20] V. Milenkovic and E. Sacks. An approximate arrangement algorithm for semi-algebraic curves. *Internat. J. Comput. Geom. Appl.*, 17:175–198, 2007.

[21] S. Schirra. Robustness and precision issues in geometric computation. In J. R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 597–632. Elsevier Science, 2000.

[22] C. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, 2nd edition, pp. 927–952, 2004.