

Real-Time Kinetic Algorithms

*Patchrawat “Patch” Uthaisombut**

Abstract

The *real-time kinetic (RK)* model for studying kinetic problems is introduced. We considered algorithms that maintain, in real time, the attributes of interests of systems that evolve continuously and unpredictably over time. This works differ from *kinetic data structures (KDS)* introduced by Guibas 1998 in two ways. First, the algorithm learns the position of moving objects in a completely online fashion. Instead we assume that the speed of any object at any time is no greater than a certain maximum speed S_{\max} . Second, the algorithms that maintains the attributes of interest of the system runs in real time, that is, the algorithms run while the objects are moving at the same time.

We consider the problem of maintaining the approximate sorted order of moving points on the line. We give an algorithm that has a maximum error of $O(S_{\max}n)$ where n is the number of objects. Note that the naive method of running a static sorting algorithm repeatedly over time has a maximum error of $\Omega(S_{\max}n \log n)$. We give a definition to quantify how much objects congregate together. We give an algorithm that has a maximum error of $O(S_{\max}k)$ where k , $0 \leq k \leq n$ is the degree of congregation of the objects.

*Computer Science Department, University of Pittsburgh, Pittsburgh PA 15260 USA. utp@cs.pitt.edu.

1 Introduction

1.1 Kinetic Problems

In a *kinetic* problem, there is a system that contains real-valued parameters and a function on these parameters. The combinatorial description of the value of the function at any time is called the *configuration* of the system. The *value* of the configuration is the value of the function that the configuration describes. The parameters of the system change continuously over time, As the parameters change, the configuration changes. The goal of the problem is to maintain the configuration over time. The challenge is to find the most efficient way to do this by exploiting fact that the parameters change in a continuous manner.

Kinetic problems are a natural extension of many classical static problems. As an example, in the static version of the Euclidean minimum spanning tree problem (EMST), there are n points in the plane, and the goal is to compute a tree spanning all points with the minimum total edge cost. In the kinetic version, these n points move continuously, and the goal is to maintain a minimum spanning tree of these n points over time. In this problem, the *configuration* is the set of edges (or point pairs) forming a minimum spanning tree, and the *value* is the sum of the distances between the two points in each point pairs in the configuration. The static graph version can be extended to a kinetic version by allow the edge weights to change over time.

1.2 Real-Time Kinetic (RK) Model

The *real-time kinetic (RK)* model for studying kinetic problems is introduced. This model has the following features. The speed of any object at any time is no greater than a certain maximum speed S_{\max} . An object can move on an arbitrary trajectory. Furthermore, the algorithm learns the position of the objects in an online fashion, that is the algorithm does not know and cannot predict with certainty the position of an object in the future. The algorithm must maintain the configuration in real time, that is, it performs computation at the same time that the parameters of the system are changing. Generally the optimal configuration cannot be maintained at all time under reasonable assumptions. Thus, we settle for an approximate configuration. In such cases, we are interested in the quality of the *value* of the configuration, *i.e.* how close it is to the value of the optimal configuration.

1.3 Related Work by Others

Kinetic problems have been studied under different assumptions. If the complete trajectory of each object is known in advance, it is possible to efficiently compute the combinatorially distinct configurations that the objects will be in. A framework for such an analysis is given by Atallah [4]. However, in many practical situations, the trajectory is not known in advance, and the algorithm has to compute the configuration in an online fashion. In contrast, our model does not assume that the trajectories of objects are known in advance.

Kinetic data structures (KDS) was introduced by Basch, Guibas, and Hershberger [8]. In the KDS model, it is still assumed that trajectories of objects are known. However, trajectories can be updated over time in an online fashion, that is the algorithm does not know when trajectories will change. In the KDS model, two of the goals are to minimize the *amount of computation per event* to maintain the certificates (*responsiveness*) and the ratio of the number of internal and external events (*efficiency*).

Several problems have been studied under the KDS model. This includes median [2, 1], facility location [9], convex hull [7], smallest bounding box [3], covering points by unit cubes and balls [13, 10], closest pair [7, 5], collision detection [15], minimum spanning tree [5], shortest path [11], line segments intersection [6], connectivity of rectangles [14], and approximate extent measures [3]. A survey on KDS can be found in [12].

The RK model differ from the KDS model in 3 ways. Firstly, in the RK model, the algorithm learns the trajectories of the objects in a completely online fashion. Secondly, the algorithms that maintains the attributes of interest of the system runs in real time, that is, the algorithms run while the objects are moving at the same time. Finally, the goal in the RK model is to minimize the *maximum absolute error* over time.

1.4 Our contributions

- We introduce a new model to study kinetic problems.
- We study the problem of maintaining sorted order of objects to illustrate various features of the model and give a very simple algorithm for maintaining an approximate sorted order of moving points on the line. The maximum error is $O(S_{\max}n)$. For comparison, the naive method has maximum error of $O(S_{\max}n \log n)$.
- We introduce a data-driven paradigm for real-time kinetic model. We give a definition that quantify how much objects congregate together. We give an algorithm the maximum error of $O(S_{\max}k)$ where $k < n$ is the degree of congregation.
- We apply the algorithms mentioned above to the problem of maintaining orthogonal bounding box.

This work should be seen as an initial investigation into a research area where there appear to be many interesting problems at the intersection between the discrete and the continuous. Many static problems can be naturally extended to kinetic ones and studied under the RK model, and this is wide open.

2 Maintaining sorted order of moving points on the line

Basic Definitions

Object p is said to be ε -far above object q if $p \geq q + \varepsilon$. Object p is ε -far below q if $p \leq q - \varepsilon$. Object p is ε -far from q if either p is ε -far above q , or p is ε -far below q . Object p is ε -close to q if p is not ε -far from q , that is, if $p < q + \varepsilon$ and $p > q - \varepsilon$.

An object has *rank* k if there are $k - 1$ objects strictly below it. Note that if there are m objects located at the same position, and there are $k - 1$ objects below them, then these m objects have the same rank k , and the object(s) immediately above them has rank $k + m$. An object with the smallest value has rank 1, the lowest rank. We define the *value of rank* k , denoted v^k , as follows. For $k = 1, \dots, n$, if there is an object of rank k , then define v^k to be the position of an object of rank k . If there is no object of rank k , then define $v^k = v^{k-1}$.

Problem Description

In this problem, there are n points on the real line. We assume that the line orients vertically. Each point moves at a speed at most S_{\max} . The location of object i at time t is denoted by $v_i(t)$. The goal is to support the following queries in real time.

- [search by rank]: Given a rank k , the operation should return the identity i of an object of rank k . If there is no such object, it should return an object of the highest rank that is less than k . If j is the object returned by the search, the *error* of the search operation is $|v_j - v_i|$.
- [nearest neighbor]: Given a real number v , the search operation should return the identity i of an object such that $|v - v_i|$ is smallest. If j is the object returned by the search, the *error* of the search operation is $||v - v_j| - |v - v_i||$.

Although not explicitly stated, the general goal of this problem is to maintain an (approximate) ordering of the objects over time.

How to describe a solution to a real-time kinetic problem

A solution should consist of the following pieces.

1. A data structure to store information about the configuration of the input.
2. A maintenance algorithm that runs in real time to maintains the data structure.
3. Query-handling algorithms.

We will refer to these pieces collectively as an *algorithm* to emphasize the importance of the maintenance algorithm. An algorithm ε -*quasi-optimal* if the result of any query is ε -quasi-optimal. Intuitively, an ε -quasi-optimal query will return an exact object if relevant objects are far apart. The query returns an approximate object only when relevant objects are close together. In some sense, the value of ε indicates the resolution power of the algorithm.

3 Results on Poll-Based Algorithms

3.1 How to measure the quality of a real-time kinetic algorithm

A natural quality measure of a real-time kinetic algorithm is the worst-case error of any query at any time for any input instance. We defines some error measures for algorithms that follow a simple strategy describe below.

Suppose I is an input instance of the problem. Let I_t denote the snapshot of the input variables at time t . The maintenance algorithm loops forever and executes in each iteration a procedure that modifies the data structure. Suppose an iteration starts at time t . The snapshot I_t is used in combination with the data structure created from the previous iteration. Suppose the worst-case running time of each iteration is $M(n)$. Then the current iteration will finish at time $t + M(n)$ and the new data structure will be ready. Let D_t be the data structure created from snapshot I_t . Data structure D_t will be used for any query from time $t + M(n)$ to time $t + 2M(n)$, when the next data structure will be ready. A query could arrive just before time $t + 2M(n)$. Data structure D_t would be used to perform the query. Suppose $Q(n)$ is the worst-case running time of the query. Then the result of the query will be returned at time $t + 2M(n) + Q(n)$.

There are 4 critical times for this kind of algorithms, namely t , $t + M(n)$, $t + 2M(n)$, and $t + 2M(n) + Q(n)$. So we define errors based on these 4 critical times.

- $E^{\text{snap}}(t)$: the error of any query on D_t compared to the correct value at time t (when the *snapshot* I_t is taken).
- $E^{\text{rdy}}(t)$: the error of any query on D_t compared to the correct value at time $t + M(n)$ (when the data structure D_t is *ready*).
- $E^{\text{exp}}(t)$ and $E^{\text{q.int}}(t)$: the error of any query on D_t compared to the correct value at time $t + 2M(n)$. This is the time the data structure D_t *expires*. This is also the latest time a query on D_t could be *initiated*.

- $E^{\text{q.cmp}}(t)$: the error of any query on D_t compared to the correct value at time $t+2M(n)+Q(n)$ (the latest time by which a query on D_t must be *completed*).

We assume that the query algorithm and the maintenance algorithm run on separate machines, and that no new queries arrive while a query is still being processed. Note that $E_{\text{snp}} \leq E_{\text{rdy}} \leq E_{\text{exp}} \leq E_{\text{q.cmp}}$.

3.2 Lower Bound for Exact Algorithm

A simple algorithm for this problem is to repeatedly run a static sorting algorithm such as Merge-Sort. Clearly, $E_{\text{snp}} = 0$. However, algorithms like this have large errors under different measures as summarized in the following theorem.

Theorem 1. *Let A be any algorithm where $E_{\text{snp}} = 0$. Then $E_{\text{q.cmp}} \geq E_{\text{exp}} \geq E_{\text{rdy}} = \Omega(S_{\text{max}}n \log n)$.*

Proof. Since $E_{\text{snp}} = 0$, then the algorithm must correctly sort the objects in each iteration. Let $T(n)$ be the worst-case running time of an iteration. Since any exact sorting algorithm takes time $\Omega(n \log n)$, then $T(n) = \Omega(n \log n)$.

Consider the situation where n objects are initially located at distinct locations very close to position 0. Suppose an iteration of the algorithm starts at time t . Let l and h be the lowest and the highest objects at time t , respectively. This iteration of the algorithm will finish at time $t+T(n)$. At time t , object l starts moving up at speed S_{max} while other objects stay put. Consider a query, that arrives just before time $t+T(n)$, for the value of highest object (rank n). At this time, the current data structure has h as the highest object. Thus, the algorithm returns $v_h(t) \approx 0$. However, the correct value at time $t+T(n)$ is $v_l(t+T(n)) \approx S_{\text{max}}T(n) = \Omega(S_{\text{max}}n \log n)$. Thus, $E_{\text{rdy}} = \Omega(S_{\text{max}}n \log n)$. \square

Note that there are many approximate sorting algorithms with running time better than $O(n \log n)$. However, they give *combinatorial* approximation guarantee rather than *geometric* guarantee. Thus, they do not apply to our problem here.

The result suggests that to make E_{rdy} small, we have to balance between E_{snp} and $T(n)$.

3.3 Algorithm that Maintains a Doubly Linked List of Clusters

Here we describe a simple scheme that is better than running a static algorithm over and over. In this data structure, objects that are close together are grouped into clusters called *d-clusters*. Each cluster can be stored as a linked list. Clusters are kept in sorted order in a doubly linked list. The following invariants, collectively called *ordered clusters invariants*, will be maintained over time.

- [*partition*] Objects are partitioned into clusters, *i.e.* an object belongs to exactly 1 cluster.
- [*representative*] Each cluster has an object that is called the *representative*. The representative of a cluster never changes throughout the life time of the cluster. We refer to a cluster and its representative interchangeably. In particular, the position of a cluster is defined to be the position of its representative.
- [*compactness*] The distance of an object to the representative of the cluster is at most d .
- [*separation*] The distance between the representatives of any two clusters is more than d .
- [*ordering*] The clusters in the doubly linked list are in increasing order.

Maintenance Algorithm $M1(\varepsilon)$: The algorithm loops forever executing Parts R, A, and O in each iteration.

- **Part R:** For the representative r in each cluster C_i starting from the lowest, compare its position with the one below it (C_{i-1}). **Case R1:** If C_i is ε -far above C_{i-1} , then do nothing. **Case R2:** If C_i is ε -close to C_{i-1} , then destroy C_i , let r join C_{i-1} , and let associates of C_i , if any, become orphans.
- **Part A:** For each associate a of each cluster C , compare its position with the representative r of C .
Case A1: If a is ε -close to r , then do nothing.
Case A2: If a is ε -far from r , then let a leave C and become an orphan.
- **Part O:** For each orphan o , compare its position with nearby clusters. In particular, if o was in cluster C_i when the algorithm starts, compare its position with two clusters below and two clusters above C_i .
Case O1: If o is ε -close to a cluster C , then let o join C .
Case O2: If o is ε -far from any cluster, then let o form a new cluster with itself as the representative, and insert the cluster at the appropriate place in the doubly linked list.

Note that all representatives are processed before any associate in each iteration. Each cluster also keeps track of the number of its members. When members leave or join a cluster, this number is updated in a straightforward manner. This is omitted from the algorithm description.

Query-by-Rank Algorithm $QR1(k)$: Suppose the current linked list is created from snapshot I_t . The algorithm simply scans the linked list and keeps count of the number of objects in the clusters seen so far. It stops at the cluster that contains the k 'th object. Suppose i is the identity of the representative of this cluster. The algorithm returns $v_i(t)$.

The running time of $QR1$ is clearly $O(n)$. Due to space limitation, we omit discussion on nearest neighbor search.

3.4 Analysis

Theorem 2. [Run time] *The maintenance algorithm $M1$ takes time $O(n)$ in each iteration.*

Proof. At the beginning of each iteration, each object is either a representative or an associate. An associate may become an orphan temporarily in the middle of an iteration. The cost of making an object orphan (case R2) is charged to the orphan. Other than that, when an object is processed in parts R, A, or O, the cost is charged to that object. \square

Theorem 3. [Invariants] *Suppose D is a data structure that satisfies the ordered clusters invariants for snapshot S_t . Suppose $T(n)$ is the worst-case running time of one iteration of $M1$. Let $t' = t + T(n)$. Suppose D' is the data structure created in an iteration of $M1$ from snapshot S_t by modifying D . Then D' satisfies the ordered clusters invariants for snapshot $S_{t'}$.*

The theorem says that algorithm $M1$ maintains the *ordered clusters invariants* over time.

Proof. Part R ensures the *separation* invariant among the clusters. Part A ensures the *compactness* invariant of each cluster. In case O2, when a cluster is created, a representative is assigned to it. Thus, every cluster has a *representative* and this never changes until the cluster is destroyed in case R2 in a later iteration.

When a cluster is destroyed in case R2, its representative joins another cluster, and its associates temporarily become orphans. Orphans eventually join new clusters (case O1) or form new ones (case O2). Thus, the *partition* invariant is preserved.

To show that the *ordering* invariant is preserved, first consider any two consecutive clusters C_{i-1} and C_i (where C_i is ε -far above C_{i-1} in D with respect to S_t) that already exist when the algorithm starts. If C_i remains ε -far above C_{i-1} with respect to $S_{t'}$ then this falls into case R1, and the ordering between the two clusters remains the same. We claim that C_i cannot become ε -far below C_{i-1} with respect to $S_{t'}$. At time t , C_i is more than ε unit above C_{i-1} . By time t' , C_i can move down by at most $S_{\max}T(n) = \varepsilon$ unit and C_{i-1} can move up by at most $S_{\max}T(n) = \varepsilon$ unit. Thus, by time t' , C_i is no more than ε unit below C_{i-1} . Thus, C_i is ε -close to C_{i-1} at time t' . This would fall in case R2 where C_i is destroyed.

Now consider any new cluster, which is created in case O2 one at a time. When a new cluster is created, it is inserted at the appropriate position in the doubly linked list. Thus, the ordering invariant is preserved. \square

Lemma 4. [*Degradation over time*] Suppose there are n objects that are moving at speed at most S_{\max} on the real line. Then for any rank k and time t and t' such that $t < t'$, it is the case that $|v^k(t') - v^k(t)| \leq S_{\max}(t' - t)$.

Proof sketch. For clarity we assume that objects have distinct ranks. Let $K_i(t)$ be the rank of object i at time t . Fix k . Consider the case where $v^k(t') > v^k(t)$. Suppose object j has rank k at time t and object i has rank k at time t' . Firstly, consider the case that $K_j(t) > K_i(t)$. Since i can move up at most $S_{\max}(t' - t)$ unit between time t and t' , and j is above i at time t , then $v^k(t') - v^k(t) = v_i(t') - v_j(t) \leq S_{\max}(t' - t)$. Secondly, consider the case that $K_i(t) > K_j(t)$. Thus, i 's rank decreases by exactly $K_i(t) - K_i(t')$ between time t and t' . For this to happen, there must be $K_i(t) - K_i(t')$ objects that move up pass i between time t and t' . One of these objects must have rank at most k . Call it l . Since $v_l(t) \leq v_j(t)$, $v_l(t') > v_i(t')$, and l can move up at most $S_{\max}(t' - t)$ unit between time t and t' , then $v^k(t') - v^k(t) = v_i(t') - v_j(t) \leq v_l(t') - v_l(t) \leq S_{\max}(t' - t)$.

The case where $v^k(t') < v^k(t)$ can be proved similarly. The argument can be generalized to the case where objects do not necessarily have distinct rank. \square

Theorem 5. [*Max error*] Suppose $M(n)$ is the worst-case running time of one iteration of algorithm M1, and $Q(n)$ is the worst-case running time of the query algorithm QR1. If $\varepsilon = S_{\max} \cdot M(n)$, then E_{snp} , E_{rdy} , E_{exp} , and $E_{\text{q.cmp}}$ is $O(S_{\max}n)$.

Proof. From Lemma 3, algorithm M1 with parameter ε maintains the ordered clusters invariants, and each cluster has radius $\varepsilon = S_{\max}M(n)$. Thus, $E_{\text{snp}} \leq S_{\max}M(n)$. From Lemma 4, $E_{\text{rdy}} \leq E_{\text{snp}} + S_{\max}M(n)$, $E_{\text{exp}} \leq E_{\text{snp}} + 2S_{\max}M(n)$, and $E_{\text{c.cmp}} \leq E_{\text{snp}} + 2S_{\max}M(n) + S_{\max}Q(n)$. From Lemma 2, $M(n) = O(n)$, and the results follow. \square

It is possible to improve the running time of query by rank to $O(1)$ by augmenting the data structure. The technique is part of the discussion in the next section. However, if a similar augmentation is made with the data structure in this section, the error of the algorithm remains $O(S_{\max}n)$.

4 Results on Data-Driven Algorithms

4.1 Overview of Data-Driven Model

The main idea in the data-driven paradigm is to run the maintenance algorithm only when it is needed, that is, when the motion of the objects cause the underlying configuration to change. The task of maintaining the configuration can be divided into *reading or monitoring the positions*

of the objects and manipulating the data structure. In many situations, the tasks of monitoring objects positions can be done by a separate computing entity or entities other than the “central” algorithm. In many cases, some computation can be performed right at the “data source”. This is true if moving objects are laptop computers, PDA, or cell phones. This is also true if data is generated from a sensor network. An application where this assumption is not true is the tracking of the positions of airplanes by radar.

In this section, we assume that the data source can perform some computation, and can be programmed with simple instructions like “**when you move X units away from your current position, notify the central algorithm**” or “**when the distance between you and node Y is greater than the distance between you and node Z , notify the central algorithm**”. We call such instructions *triggers*. When a trigger goes off, we say that it generates an *event*. When an event occurs, the central algorithm is activated and performs any necessary computation to update the configuration.

4.2 Quantifying the Degree of Congregation

In many real situations, objects do not congregate together very much. In this section we first give a natural definition to quantify how much the objects congregate together. When the objects do not congregate together too much, we can improve the performance of maintaining sorted order.

Definition 6. *Suppose S is a set of stationary points on the line. A set D is an ε -dominating set for S if for any point x in S , either x is in D or x is ε -close to a point in D . A minimum ε -dominating set for S , is an ε -dominating set with the smallest cardinality. The cardinality of such sets is denoted by $\lambda_\varepsilon(S)$. The degree of ε -congregation of S , denoted $\gamma_\varepsilon(S)$, is defined to be $n - \lambda_\varepsilon(S)$.*

The degree of ε -congregation indicates how much the objects congregate together. The value of $\gamma_\varepsilon(S)$ is between 0 and $|S| - 1$ inclusive. For $\gamma_\varepsilon(S) = 0$, it means no two objects are within ε unit distance of one another. For $\gamma_\varepsilon(S) = |S| - 1$, it means there is an object x such that all objects are within ε unit distance of x . In other words, there is an interval of length 2ε that contains all objects. Also, if $\varepsilon \leq \delta$, then $\gamma_\varepsilon(S) \geq \gamma_\delta(S)$.

Lemma 7. *Let D be any ε -clustering satisfying the ordered clusters invariants for a set S of stationary objects.*

- (1) *The number of representatives (clusters) in D is between $\lambda_\varepsilon(S)$ and $\min\{|S|, 2\lambda_\varepsilon(S)\}$ inclusive.*
- (2) *The number of associates in D is between $\max\{0, \gamma_\varepsilon(S) - \lambda_\varepsilon(S)\}$ and $\gamma_\varepsilon(S)$ inclusive.*
- (3) *If $\gamma_\varepsilon(S) \leq k$, then the number of representatives in D is between $|S| - k$ and $|S|$ inclusive, (4)*
- If $\gamma_\varepsilon(S) \leq k$, the the number of associates in D is between 0 and k inclusive.*

Proof. Let D^* be a minimum ε -clustering. Let $|D^*|$ denote the number of clusters in D^* . Thus, $|D^*| = \lambda_\varepsilon(S)$. First we prove statement (1). By definition of D^* , $|D| \geq |D^*|$. Since D can have at most $|S|$ clusters, then $|D| \leq |S|$. To show that $|D| \leq 2|D^*|$, assume to the contrary that $|D| \geq 2|D^*| + 1$. Thus, there exists a cluster C in D^* that contains the representatives of 3 or more clusters in D . Since adjacent clusters must be more than ε units apart, then the distance between the two distant representatives of the three is more than 2ε units. However, By the definition of clusters, the maximum distance between any pair of elements in C is at most 2ε . A contradiction.

Statement (2) follows from statement (1) because the number of representatives and the number of associates sums to $|S|$. Statements (3) and (4) follow from statements (1) and (2). \square

4.3 Synchronized Data-Driven Algorithm

We make an additional assumption that the triggers are equipped with a globally synchronized clock. We call this model the *synchronized data-driven model*. In this model, time is divided into periods of certain length. Each trigger does not check its status continuously. Instead, each trigger tests its status at the beginning of each period. It is possible that the triggers may miss some events. However, this does not effect the maximum error of the algorithm.

Our algorithm utilizes 2 kinds of triggers. A type-R trigger associates with a representative of a cluster. Suppose r is a representative of a cluster, and q is a representative of a cluster immediately below it. Then the *type-R trigger for r against s* has the condition $v_r > v_q + \varepsilon$. If $v_r \leq v_q + \varepsilon$ at the beginning of a period, the trigger will go off. A type-A trigger associates with an associate of a cluster. Suppose a is an associate of a cluster with r as the representative. Then the *type-A trigger for a against r* has the condition $v_a \leq v_r + \varepsilon$ and $v_a \geq v_r - \varepsilon$. If $v_a > v_r + \varepsilon$ or $v_a < v_r - \varepsilon$ at the beginning of a period, the trigger will go off.

We assume that there is a *queue* that serializes all events. This allows the maintenance algorithm to handle these events one at a time. We also assume that the value of *all* n input variables are updated by an external process at the beginning of each synchronization period and these values are available to the maintenance and query algorithms.

Initialization Algorithm $I2(\varepsilon)$

- Construct an initial data structure.
- **Type-R triggers:** For each object r which is the representative of some cluster, set a trigger for r against s (with condition $v_r > v_q + \varepsilon$) where q is the representative of the cluster immediately below it.
- **Type-A triggers:** For each object a which is an associate of a cluster with r as the representative, set a trigger for a against r (with condition $v_j \leq v_r + \varepsilon$ and $v_a \geq v_r - \varepsilon$).

Maintenance Algorithm $M2(\varepsilon)$ The algorithm is activated once every period. It gets a triggered event from the queue and do the following.

- If the trigger is of type-R and is associated with object r from cluster C , then *destroy* C , let r *join* the cluster C' immediately below it. and force associates of C , if any, to *leave* C and become orphans. Remove triggers associated with members of C . Let C'' be the cluster immediately above C . Remove triggers associated with C'' . Set a new type-A trigger for r against C' . Set a new type-R trigger for C'' against C' . Then handle the orphans as described below.
- If the trigger is of type A and is associated with object a from cluster C . then let object a *leave* C and become an orphan. Remove the trigger associated with a . Then handle the orphan as described below.
- For each orphan o , compare its position with nearby clusters. In particular, if o was in cluster C when the algorithm starts, compare its position with two clusters below and two clusters above C
 - Case O1:** If o is ε -close to a cluster C' , then let o *join* C . Then set a new type-A trigger for o against C' .
 - Case O2:** If o is ε -far from any cluster, then *create* a new cluster C' with o as the representative, and insert the cluster into the data structure. Suppose the new cluster is inserted above

C'' below C''' , then remove type-R trigger associated with C''' . Set a new type-R trigger for C' against C'' . Set a new type-R trigger for C''' against C' .

For each object that is processed by $M2$, it goes through one of the 3 possible processes: *destroy-join*, *leave-join*, or *leave-create*. A *destroy-join* applies to a representative whose type-R trigger goes off. In this case, its cluster is destroyed and it joins another cluster. A *leave-join* applies to an associate that becomes an orphan and then later joins another cluster. A *leave-create* applies to an associate that becomes an orphan and then later forms a new cluster.

4.4 Analysis

Theorem 8. *Suppose D is a data structure that satisfies the ordered clusters invariants for snapshot S_t . Suppose $\gamma_\varepsilon(S) \leq k$ at all time for some ε . Let t' be any time after t at which the triggers are synchronized and the maintenance algorithm $M2$ is activated. Then it is the case that algorithm $M2$ processes at most $2k$ objects, and its running time is $O(k)$.*

The theorem states that if the degree of congregation is low, no matter how long the time has passed since the last data structure update, when the maintenance algorithm is activated, it only takes $O(k)$. This theorem does not claim that the resulting data structure satisfies the *ordered clusters invariants*. This claim is made in the Theorem 11.

Proof. First, we show that at most $2k$ objects are processed in each period. Assume to reach a contradiction that at least $2k + 1$ objects are processed. Thus, either there are at least $k + 1$ *leave-joins* and *destroy-joins*, or there are at least $k + 1$ *leave-creates*. First consider the case that there are at least $k + 1$ *leave-joins* and *destroy-joins*. All objects processed by these operations join some clusters. Therefore, in the end, there are at least $k + 1$ associates. However, from Lemma 7 and the fact that $\gamma_\varepsilon \leq k$, there can be at most k associates at any time. A contradiction.

Now consider the case that there are at least $k + 1$ *leave-creates*. Before the objects involved are processed, they must be associates in some clusters because the only operation involving a representative is *destroy-join*. Thus, there are at least $k + 1$ associates before the objects are processed. However, this contradicts the fact that $\gamma_\varepsilon \leq k$.

With a similar argument to Theorem 2, the cost charged to each object processed is a constant. Since at most $2k$ objects are processed, then the running time is $O(k)$. \square

Definition 9. *Let $T(k)$ be the worst-case running time of $M2$ to update the data structure where $\gamma_\varepsilon(S) \leq k$ at all time.*

Note that $T(k)$ does not depend on ε , that is, if $\gamma_x(S1) \leq k$ and $\gamma_y(S2) \leq k$ at all time, then $M2$ takes time at most $T(k)$ in both cases. Furthermore, from Theorem 8, $T(k) = O(k)$.

Definition 10. *For a given set S of moving objects, define $\hat{k}(S)$ to be the smallest integer k such that $\gamma_{ST(k)}(S) \leq k$ at all time.*

Note that $\hat{k}(S)$ is between 0 and $|S|$ inclusive. Small value means objects do not congregate together too much.

Theorem 11. *[Invariants] Suppose D is a data structure that satisfies the ordered clusters invariants for snapshot S_t . Suppose $\hat{k}(S) = k$ and $\varepsilon = S_{\max}T(k)$. Let $t' = t + T(k)$. Suppose D' is the data structure created by $M2$ from snapshot $S_{t'}$ by modifying D . Then D' satisfies the ordered clusters invariants for snapshot $S_{t'}$.*

Proof. The 5 invariants can be shown using arguments similar to Theorem 3. We make some observations about the *ordering* invariant. All clusters have radius $\varepsilon = S_{\max}T(k)$ and are more than ε unit distance apart. The running time of $M2$ is $T(k)$. Between two consecutive activations of $M2$, objects can move by at most $S_{\max}T(k)$ unit distance. If we consider two adjacent clusters, they do not have enough time to switch their ordering cross in time $T(k)$. Thus, after time $T(k)$, they either remain as two separate clusters in the same ordering or one cluster is destroyed because it is too close to the other one. \square

Theorem 12. [Max error] Suppose $M(n)$ is the worst-case running time of one iteration of algorithm $M1$, and $Q(n)$ is the worst-case running time of the query algorithm $QR1$. If $\hat{k}(S) = k$ and $\varepsilon = S_{\max}T(k)$. then E_{snp} , E_{rdy} , and E_{exp} is $O(S_{\max}k)$.

Proof. From Lemma 11, algorithm $M1$ with parameter ε maintains the *ordered clusters invariants*, and each cluster has radius $\varepsilon = S_{\max}M(k)$. Thus, $E_{\text{snp}} \leq S_{\max}M(k)$. From Lemma 4, $E_{\text{rdy}} \leq E_{\text{snp}} + S_{\max}M(k)$ and $E_{\text{exp}} \leq E_{\text{snp}} + 2S_{\max}M(k)$. From Lemma 8, $M(k) = O(k)$, and the results follow. \square

4.5 Additional Results

Performing a search by rank or a nearest neighbor search on the data structure in the previous section, takes time $O(n)$ (by scanning through the linked list). Thus, $E_{\text{c.cmp}} \leq E_{\text{snp}} + 2S_{\max}M(k) + S_{\max}Q(n) = O(S_{\max}n)$, which is the same as the result using poll-based algorithm. It is possible to augment the data structure with an array so that a search by rank takes $O(1)$ time (by direct accessing the array) and nearest neighbor search takes $O(\log n)$ time (by binary search on the array).

The algorithms for maintaining sorted order of moving points on the line can also be used to maintain the orthogonal bounding box in d dimensions.

5 Closing Remarks

This work should be seen as an initial investigation into a research area where there appear to be many interesting problems at the intersection between the discrete and the continuous. Another interesting feature of the model is that an algorithm should not spend too much time computing, because the objects are moving at the same time. The more time it uses, the more its result will degrade. This is in contrast to most computational models where the more time an algorithm spends doing computation, the better the solution.

The real-time kinetic (RK) model can be investigated in many directions.

- Many static problems can be naturally extended to kinetic ones and studied under the RK model, and this is wide open.
- Suppose we want to keep track of only the highest object over time, it is unclear if the maximum error can be lower than $O(S_{\max}n)$. Consider the case where all n objects are initially locate at the same position, and then suddenly an object or a set of objects breaks away.
- Our algorithms need to know the maximum speed of the objects *a priori*. Is there an algorithm that adaptively varies its error rate according to the current speed of the objects?
- In many applications, partial knowledge of objects trajectories is known. How to incorporate this partial knowledge? How to combine the KDS and the RK models?

References

- [1] Pankaj K. Agarwal, Mark de Berg, Jie Gao, Leonidas J. Guibas, and Sariel Har-Peled. Staying in the middle: Exact and approximate medians in r_1 and r_2 for moving points. Manuscript.
- [2] Pankaj K. Agarwal, Jie Gao, and Leonidas J. Guibas. Kinetic medians and kd-trees. In *ESA 2002*, pages 5–16, 2002.
- [3] Pankaj K. Agarwal and Sariel Hal-Peled. Maintaining approximate extent measures of moving points. In *Symposium on Discrete Algorithms*, pages 148–157, 2001.
- [4] M.J. Atallah. Some dynamic computational geometry problems. *Computers and Mathematics with Applications*, 11:1171–1181, 1985.
- [5] J. Basch, L. Guibas, and L. Zhang. Proximity problems on moving points. In *Proceedings of the 13th Symposium of Computational Geometry*, 1997.
- [6] J. Basch, L. J. Guibas, and G. D. Ramkumar. Reporting red-blue intersections between two sets of connected line segments. *Algorithmica*, 35:120, 2003.
- [7] Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1997.
- [8] Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1–28, 1999.
- [9] S. Bespamyatnikh, B. Bhattacharya, D. Kirkpatrick, and M. Segal. Mobile facility location. In *Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 46–53, 2000.
- [10] Jie Gao, Leonidas J. Guibas, John Hershberger, Li Zhang, and An Zhu. Discrete mobile centers. *Discrete and Computational Geometry*, 30(1), 2003.
- [11] Camillo Gentile, Jerome Haerri, and Robert E. VanDyck. Kinetic minimum power routing and clustering in mobile ad hoc networks. In *Proc. VTC 2002*, 2002.
- [12] L. Guibas. Kinetic data structures: A state of the art report. In *Proc. 3rd Workshop on Algorithmic Foundations of Robotics*, 1998.
- [13] John Hershberger. Smooth kinetic maintenance of clusters. In *Proceedings of the nineteenth annual symposium on Computational geometry*, pages 48–57, 2003.
- [14] John Hershberger and Subhash Suri. Kinetic connectivity of rectangles. In *Proceedings of the fifteenth annual symposium on Computational geometry*, pages 237 – 246, 1999.
- [15] Bettina Speckmann. *Kinetic Data Structures for Collision Detection*. PhD thesis, The University of British Columbia, 2001. citeseer.ist.psu.edu/speckmann01kinetic.html.