A Computational Framework for Incremental Motion

David M. Mount* Nathan S. Netanyahu† Christine D. Piatko‡ Ruth Silverman§
Angela Y. Wu¶

ABSTRACT

We propose a generic computational framework for maintaining a discrete geometric structure defined by a collection of static and mobile objects. We assume that the mobile objects move incrementally, that is, in discrete time steps. We assume that the structure to be maintained is a function of the current locations of the mobile and static objects (independent of their prior motion). Unlike other models for kinetic computation, we place no restrictions on the motion nor on its predictability.

In order to handle unrestricted incremental motion, our framework is based on the coordination of two computational entities. The first is the *incremental motion algorithm*. It is responsible for maintaining the structure and a set of certificates, or conditions, that prove the structure's correctness. The other entity, called the *motion processor*, is responsible for handling all the low-level aspects of motion, including computing and/or tracking the motion of the mobile objects, answering queries about their current positions and velocities, and validating that the object motions satisfy simple motion estimates, which are generated by the incremental motion algorithm. Computational efficiency is measured in terms of the number of interactions between these two entities.

We present a simple online protocol, through which the incremental motion algorithm generates motion estimates. We show that, given a parameter $\epsilon > 0$, the number of motion estimates generated by this protocol has a competitive ratio of $O(1/\epsilon)$ relative to an optimal algorithm that has full knowledge of the future motion of the points but is required to maintain an additional ϵ-factor separation relative to the certificate failure regions.

Categories and Subject Descriptors: F.2.2 [Analysis of Algorithms and Problem Complexity]: Geometrical problems and computations

General Terms: Algorithms, Theory

Keywords: Incremental motion, kinetic data structures, competitive analysis

1 Introduction

An important emerging direction in computational geometry research is development of algorithms for mobile data. Problems of this variety arise in many areas of practice, including physical simulations, motion tracking, protein folding in computational biology, computer animation, collision detection, and geometric optimization. (See [1] for a survey.) Of particular relevance to computational geometry is the problem of maintaining discrete geometric structures for mobile objects.

In recent years there has been active research in algorithms for maintaining discrete geometric structures for objects moving continuously over time. For many years stochastic procedures, such as the Kalman filter [11, 19], have been used in control theory and dynamic motion tracking. Early work in computational geometry considered problems in an offline setting [3]. Lin and Canny [18] and Lin and Manocha [17] considered practical approaches for collision detection for moving objects. The area of kinetic data structures (KDS) has emerged as an important framework in which to model and analyze discrete problems involving moving objects. The maintenance of a wide variety of geometric structures has been successfully addressed within this framework, including convex hulls and closest pairs [5], binary space partitions [2], pseudo-triangulations for intersection detection [4, 16], and fixed-radius clustering [7, 8], to name a few.

Although KDS has proven to be a very powerful method for managing objects in motion, one of its principal shortcomings is the requirement that the future motion of each

*Department of Computer Science, University of Maryland, College Park, Maryland. Partially supported by grant CCR-0098151. Email: mount@cs.umd.edu.
†Department of Mathematics and Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel and Center for Automation Research, University of Maryland, College Park, Maryland. Email: nathan@macs.biu.ac.il.
‡The Johns Hopkins University Applied Physics Laboratory, Laurel, Maryland. The support of the Office of Naval Research under grant N00014-01-1-0964 is gratefully acknowledged. Email: christine.piatko@jhuapl.edu.
§Center for Automation Research, University of Maryland, College Park, Maryland. Email: ruth@cfar.umd.edu.
¶Department of Computer Science, American University, Washington, DC. Email: awu@american.edu.

SoCG'04, June 8–11, 2004, Brooklyn, New York, USA

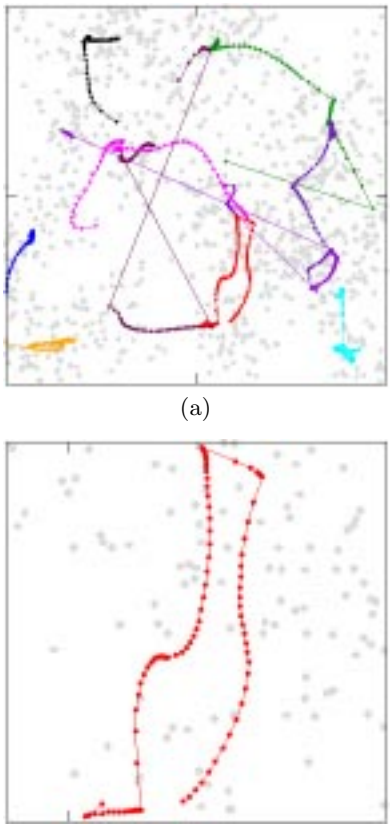
mobile object be specified. In KDS this is done by having each mobile object present a *flight plan*, which is a piecewise algebraic function of time that describes the future motion of the object. (We assume that the reader is familiar with the basic elements of KDS. See [5] for an introduction.) Although flight plans can change periodically, the KDS approach relies on the ability to accurately determine the times of future events. There are many applications of mobile data processing, however, where this assumption cannot be met. For example, this happens in motion tracking systems, in which a sensor tracks the locations of a set of moving agents. This is also the case in many physical simulation systems, where the motion is the result of the minimization of some real-valued energy or potential function, or results as the numerical solution of a system of differential equations. In these cases, it may be possible to predict only the very near-term motion of objects.

There has been relatively little theoretical work on unconstrained kinetic motion. Kahan [10] suggested such a model. It makes use of an *update function*, which can be queried to determine the exact locations of the objects. The objective is to minimize the number of queries to the update function. In order to obtain interesting bounds, Kahan had to assume that objects have specified upper bounds on their velocities. Another approach by Czumaj and Sohler [6] is to use randomized property testers to establish the approximate correctness of a data structure.

Our particular interest in this problems stems from an application in geometric optimization. The problem involves the simulation of a simple descent procedure that arises in a variant of Lloyd's k-means algorithm [12, 13, 14]. This is among the most widely used clustering methods in practice [9, 15]. We are given a large set of n data points and a smaller set of k cluster centers. In each iteration each of the data points computes its closest center, and each center moves toward the centroid of points associated with it. (Fig. 1(a) illustrates the process. Fig 1(b) shows the motion of a single center.)

Our objective is to develop a system, which like KDS, avoids unneeded computations whenever the results of these computations can be inferred due to the continuity of motion, but does not impose the strong restrictions that KDS does on the predictability of the future motion. In this paper we present a computational framework for maintaining discrete geometric structures under these assumptions. Rather than considering a specific application, we present a generic computational framework and a means for reasoning about the efficiency of algorithms running under this framework. In order to simplify the presentation, we assume henceforth that the mobile objects are modeled as moving points in real d-dimensional space $\mathbb{R}^d$, for some fixed constant d. For applications involving more complex objects, this can often be achieved by mapping the objects to points in an appropriate configuration space.

Our approach can be viewed as an extension to the general KDS concept, in the sense that the computation is organized around the maintenance of a set of boolean assertions, called *certificates*, which serve to prove the validity of the structure [5]. Our framework differs from KDS in that it involves the coordination of two computational units, the *incremental motion algorithm* (IM) and the *motion processor* (MP). The IM algorithm performs all the complex processing involved in maintaining certificates and the discrete structure. In contrast, the MP handles the low-level issues of tracking and/or computing point motions. The IM algorithm interacts with the MP by issuing queries as to the current locations and velocities of the moving points. Otherwise, all processing is performed in an event-driven manner. The IM algorithm specifies, or *registers*, estimates on the motions of the points to the MP and goes to sleep. The MP monitors the point motion and verifies that the points are indeed moving as expected, all without the involvement of the IM algorithm. Whenever a significant event occurs (e.g., the violation of a certificate or a point motion that deviates from the given motion estimate) the MP awakens the IM algorithm. The IM algorithm then ascertains the nature of the exception, updates the geometric structure and motion estimates as needed, and then goes back to sleep.



(a) (b)

Figure 1. An illustration of the motion of center points in k-means search.

Computational efficiency is determined by the number of interactions between the IM algorithm and the MP. Rather than present a particular application problem, we present a simple online protocol, through which the incremental motion algorithm generates motion estimates. We show that, given a parameter $\epsilon > 0$, the number of motion estimates generated by this protocol has a competitive ratio of $O(1/\epsilon)$ relative to an optimal algorithm that has full knowledge of the future motion of the points but is required to maintain an additional ϵ-factor separation relative to the certificate failure regions.

The paper is organized as follows. In the next section we present a more detailed description of our computational

framework. In Section 3 we present the simple online protocol for maintaining motion estimates, assuming single-point certificates. In Section 4 we present present the analysis of the competitive ratio of this online protocol. In Section 5 we discuss the generalization to multiple-point certificates.

2 The Computational Framework

In this section we present a detailed description of our computational framework. The motion M of a single point is a finite sequence of point positions in $\mathbb{R}^d$ sampled at discrete *time instances*:

$$M = \langle \mathbf{p}(t_0), \mathbf{p}(t_1), \ldots, \mathbf{p}(t_N) \rangle,$$

where $t_{i-1} < t_i$. The interval between two consecutive instances is a *time step*. The durations of the time steps need not be uniform, and in fact we place no lower bounds on their size. (Thus our framework could even be applied to *differential motion*.) A *time interval* is a closed interval bounded by time instances $T = [t_i, t_j]$ for $i \leq j$. The *duration* of an interval, $t_j - t_i$, is denoted by $|T|$.

We assume that the state of the geometric structure is purely a function of the current point positions and is independent of the prior motion. We will ignore a number of application-specific issues, such as how the sampling rate is determined and dealing with "missed events" due to finite time sampling. We also ignore issues related to real-time computation (in case the point locations are measured in real time).

As mentioned above, our framework involves the coordination between two computational units, the motion processor (MP) and the incremental motion algorithm (IM). The former is responsible for low-level details of motion and the latter for maintaining the structure. Before discussing their specific roles, we begin with some definitions regarding motion estimates. Consider a time step $[t_{i-1}, t_i]$, and let $s = t_i - t_{i-1}$ denote its duration. Let $\mathbf{v}$ denote an estimate of the point's current velocity. The estimated displacement of the object over this step is $s\mathbf{v}$, and its actual displacement is given by the vector $\mathbf{u} = \mathbf{p}(t_i) - \mathbf{p}(t_{i-1})$. Let $|\mathbf{u}|$ denote the Euclidean length of vector $\mathbf{u}$. We define the *drift*[1] of this point at time t_i to be the relative error between the actual and estimated displacements,

$$\frac{|\mathbf{u} - s\mathbf{v}|}{|s\mathbf{v}|}.$$

We define a *motion estimate* to be a triple $(T, \mathbf{v}, \delta)$, consisting of a time interval T, velocity estimate $\mathbf{v}$, and drift bound δ. We say that a motion M *satisfies* this motion estimate if for all time steps in T the drift of M relative to the velocity estimate $\mathbf{v}$ is at most δ.

An equivalent way to describe the drift bound is in terms of a bound on the possible locations of $\mathbf{p}(t_i)$. Given the velocity estimate $\mathbf{v}$ and given any time t, the estimated location of the point after an elapsed time of s is defined to be $\widehat{\mathbf{p}}(t, s) = \mathbf{p}(t) + s\mathbf{v}$. Thus, the estimated location of the point after the time step $[t_{i-1}, t_i]$ of duration s is $\widehat{\mathbf{p}}(t_{i-1}, s)$. (See Fig. 2(a).) Let $B(\mathbf{p}, r)$ denote a Euclidean ball of radius r centered at point $\mathbf{p}$. From the above definition it is easy to verify that

$$\mathbf{p}(t_i) \in B(\widehat{\mathbf{p}}(t_{i-1}, s), s\delta|\mathbf{v}|).$$

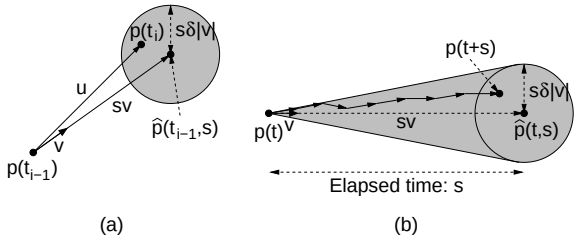


Figure 2. Drift and the motion bound.

If a motion satisfies a given motion estimate, then the following lemma specifies the constraints on its position at any time. It follows by a straightforward application of the triangle inequality.

Lemma 2.1. *Let T be a time interval of duration s starting at time t. If a motion M satisfies a motion estimate $E = (T, \mathbf{v}, \delta)$, then for each time instance $t + s' \in T$, for $0 \leq s' \leq s$, the point $\mathbf{p}(t + s')$ lies within a ball of radius $\delta s'|\mathbf{v}|$ centered at $\widehat{\mathbf{p}}(t, s')$.*

An immediate consequence is that motion lies entirely within a cone-shaped region (see Fig. 2(b)) formed by the union of all these balls, called the *motion bound* for E:

$$\mathrm{MB}(E) = \bigcup_{0 \leq s' \leq s} B(\widehat{\mathbf{p}}(t, s'), \delta s'|\mathbf{v}|).$$

We can now present more formally the manner in which an IM algorithm interacts with the motion processor. First, the algorithm can query the current or recent state of each moving point. In particular, for each moving point and for the current time instance t_i, the MP can answer the following queries:

Position Query: Report the current position of this point, denoted $\mathbf{p}(t_i)$.

Velocity Query: Report the current velocity of this point,[2] denoted $\mathbf{v}(t_i)$.

Maximum Drift Query: Report the maximum drift for this point, relative to its current motion estimate, that has been observed since this point's last event. (Events and motion estimates will be discussed below.)

Observe that for each mobile point, these queries can be maintained and computed in constant time and space.

[1] Our definition of drift has the nice feature of being a dimensionless quantity. One negative aspect of this is that if the motion estimate $\mathbf{v}$ is the zero vector, then the drift is either undefined or infinite. An alternative definition involves replacing $|s\mathbf{v}|$ in the denominator with s. In this way, drift is the absolute rate of deviation from the estimate. If zero motion estimates are to be considered, then this alternative definition may be better. Our results can be adapted to apply with this alternative definition.

[2] In the simplest case, the velocity query may return incremental rate of change, $\mathbf{p}(t_i) - \mathbf{p}(t_{i-1})/(t_i - t_{i-1})$. However, depending on the application, the MP is free to provide a more accurate value. This may be possible, for example, in certain physical simulations. For the sake of our analysis, we only require that, if the motion satisfies a motion estimate generated by the IM algorithm, then the velocities returned from the MP should satisfy the associated drift bounds as well.

The second form of interaction with the MP involves the validation of motions with respect to a given motion estimate. First, we need to discuss certificates. Recall that a *certificate* is a boolean condition on the current point locations. An IM algorithm maintains a set of certificates, which as a group provides a "kinetic proof" of the validity of the current structure [5]. We assume that certificates are testable in constant time. Generally, certificates involve some bounded number of moving points. We discuss the case of single-point certificates for now. (Such certificates would arise, for example, in tracking the motions of some number of independent points through a static environment.) This simplifies the presentation and is sufficient to capture the principal elements of motion estimation. We discuss multiple-point certificates in Section 5.

Each single-point certificate subdivides $\mathbb{R}^d$ implicitly defines a region of space, called a *failure region*, where it fails to hold. Given a motion estimate E, we say that a certificate *encroaches* on E if its failure region intersects the motion bound $\mathrm{MB}(E)$. Clearly, it is important for the MP to know which of the current certificates encroach on the current motion bounds, since the discrete structure may change if the point crosses into such a region.

It is tempting to require that no certificate failure region be allowed to encroach on the current motion bound, for then we can be assured that the discrete structure does not change provided that the motion satisfies this motion estimate. However, this is too strong a restriction. The point is always surrounded by its motion bounds, and so, as it comes closer and closer to a boundary of the failure region, we would be forced to replace its current motion bounds with successively tighter and tighter motion bounds covering successively smaller time intervals. The resulting effect is somewhat the infinite regression of Zeno's paradox, and would only be resolved by the assumption that there is some minimum time step size. (This effect has been observed in incremental collision avoidance systems [17]. See Fig. 3.)

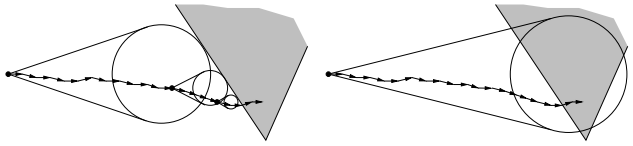


Figure 3. Allowing a constant number of certificate failure boundaries to encroach on motion bounds.

Instead, our approach is to allow a constant number of certificate failure boundaries to encroach on each motion bound. We allow a constant number, rather than just one, since depending upon the application, there may be multiple certificate boundaries in close proximity to some point of space through which the motion passes.

The combination of a motion estimate E and a (constant-sized) set of certificates C is called a *motion unit*. The IM algorithm is required to specify, or *register*, a motion unit for each moving point. This is done in an online manner. Given a motion unit for some point, it is the job of the MP to validate that the point's motion satisfies the motion estimate over the specified time interval. If not, the MP generates an exception for one of the following three events:

Drift violation: The point's actual drift exceeds the drift bound of the current motion estimate.

Safeness violation: The time interval associated with the motion estimate has elapsed.

Certificate violation: One of the the certificates of C no longer holds.

Drift and safeness violations are related, since in both cases we cannot guarantee that the point lies within its motion bounds. We distinguish between them because they provide different information to the IM algorithm. A drift violation suggests that either the drift bound was set too small or the velocity estimate is incorrect. A safeness violation suggests that the velocity estimate is good, but the drift may be too large. The safeness violation is so named because the motion is deemed to be safe from undetected certificate violations as long as it remains within its motion bounds. When the time interval expires this is no longer the case. Note that for each motion unit, these events can be detected in $O(1)$ time per time step given $O(1)$ space.

An IM algorithm operates in an event-driven manner. For each moving point, it *registers* a motion unit for this point to the MP, based on its estimate of the point's future motion and drift and its knowledge of the certificates that involve this point. It then goes to sleep. The MP passively monitors the motions of points, without involving the IM algorithm. If one of the above events occurs, the MP wakes up the IM algorithm, which then performs whatever actions are needed to restore the proper state of the system and motion estimates.

A generic IM algorithm operates as follows. It first queries the MP to determine the initial positions of moving points and constructs the initial structure. It computes and registers the initial motion units for the moving points. (Of course, since there is no motion history at this point, these initial estimates are likely to fail very soon.) The IM algorithm then repeats the following steps until the end of the motion simulation. It sleeps until the next event is generated by the MP. Note that many events may generally occur at the same time instance. Since we assume that the geometric structure to be maintained is independent of the point motion, determining a plausible ordering of these events is not an issue. On being awakened, the IM algorithm queries the MP for the positions and velocities of the points involved. If a certificate violation occurred, the IM algorithm needs to update some subset of the certificates and perhaps the geometric structure as well. Finally, it computes and registers to the MP the new motion units for the involved points. The exact processing depends upon the particulars of the discrete structure and choice of certificates.

Observe that the IM algorithm follows much the same structure as a KDS algorithm with two major differences. First, the IM algorithm needs to use the MP to determine object locations (rather than the flight plans) and needs to generate motion estimates. Second, the IM algorithm relies on signals from the MP to know when kinetic events have occurred, whereas in KDS this is done with a kinetic heap. The basic elements of good KDS design, namely efficiency, locality, compactness, and responsiveness [5] still apply to the design of good IM algorithms. Thus, our framework is not a replacement for KDS, but rather a way of extending the KDS concept to incremental motion.

3 The Online Protocol

Beyond the common elements shared with KDS, the essentially new problem in the design of an IM algorithm is the determination of the motion units. At first, this might to be an impossible task in the absence of either a statistical model of motion or bounds on object's maximum acceleration. Nonetheless, we will present a simple online protocol for generating motion estimates, and we will show in the next section that this protocol is competitive with respect to an algorithm that has full knowledge of the future motion.

The intuition behind our protocol is very simple. We query the MP for the point's current velocity, which becomes the new velocity estimate. The new drift bound is a function of the type of event. If this is a drift event, we double the drift bound. If it is a safeness violation, we halve the drift bound provided that the maximum observed drift is sufficiently small.

Here, we consider the processing for a single-point certificate. (Multiple-point certificates will be discussed in Section 5.) The initial motion estimate can be arbitrary. Otherwise, we are responding to an event that has just occurred. Let t denote the current time, and let δ denote the drift bound from the most recent motion estimate for the violating point. We query the MP to determine the point's current velocity, $\mathbf{v}$, and the maximum drift since the last event, denoted δ^+.

Drift Violation: Set $\delta \leftarrow 2\delta$.
Safeness Violation: If $(\delta^+ < \delta/2)$ then set $\delta \leftarrow \delta/2$.
Certificate Violation: Leave δ unchanged.

Next, the IM algorithm performs whatever processing is needed to update the structure and the associated certificates. Given the sampled velocity $\mathbf{v}$ and the updated drift bound δ, we compute the time interval $T = [t, t + s]$ with maximum duration s subject to the constraint that the resulting motion bound is encroached on by no more certificates than the MP allows. This computation involves sweeping the motion bound forward in time and computing its intersection with the existing certificate failure regions. We ignore the computation time required, since this depends on the particular application. Let C denote the resulting set of certificates. We generate the new motion estimate $E = (T, \mathbf{v}, \delta)$ and register the new motion unit (E, C) with the MP.

4 Competitive Analysis

In this section we present a competitive analysis to justify the efficiency of our online IM protocol. As before, we consider single-point certificates. The obvious point of comparison would be an optimal offline algorithm, which has full knowledge of the point's future motion. This appears to be too demanding, however. The problem is that, in the worst case, there may be virtually no separation between optimum motion bound and an arbitrarily large number of certificates. If our velocity estimate is not absolutely perfect, the online protocol may either suffer an unbounded number safeness events or an unbounded number of drift events, depending on whether the drift is too large or too small.

Our approach instead is to force a small "safety-zone" between the certificates and each of the optimum motion bounds. This is analogous to ϵ-safety requirements that were introduced in the ray shooting algorithm of Mitchell, *et al.* [20], based on the notion of simple cover complexity. In applications where the positions of objects are subject to small measurement errors, such a requirement is quite reasonable.

Given a motion estimate $E = (T, \mathbf{v}, \delta)$, and for $\epsilon > 0$, we define its *ϵ-expanded motion bound*, denoted $\mathrm{MB}_\epsilon(E)$ to be the expansion of $\mathrm{MB}(E)$ by the distance $\epsilon\delta s|\mathbf{v}|$, where $s = |T|$ is the duration of T. Equivalently, it is the Minkowski sum of $\mathrm{MB}(E)$ and a ball of radius $\epsilon\delta s|\mathbf{v}|$. We say that E is ϵ-safe if the number of certificates encroaching on $\mathrm{MB}(E)$ is bounded by a constant. (See Fig. 4 right.) Finally, given a motion M, we define $\mathrm{Opt}_\epsilon(M)$ to be the motion units generated by an optimal offline algorithm for M, in which the motion estimates generated by M are required to be ϵ-safe. We will drop the reference to M when it is clear. We will also abuse notation and let Opt_ϵ refer to the number of MP events generated by this algorithm.

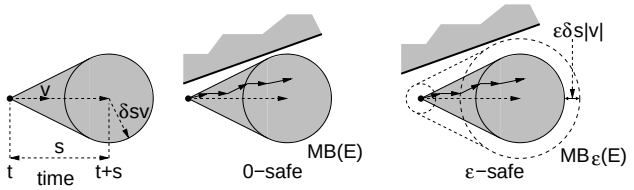


Figure 4. $\mathrm{MB}(E)$ (center) and $\mathrm{MB}_\epsilon(E)$ (right).

As with simple cover complexity, one nice feature of ϵ-safeness is that small increases in ϵ result in small variations in the number of motion bounds. We omit the proof, but it follows by observing that the part of the motion that is covered by some ϵ-safe motion bound $\mathrm{MB}_\epsilon(E)$ can be covered by two 2ϵ-safe motion bounds, each of half the duration but with the same velocity and drift bound.

Lemma 4.1. *For $\epsilon > 0$, $Opt_{2\epsilon} \leq 2Opt_\epsilon$.*

The main result of this section is given below, and the remainder of this section is devoted to its proof.

Theorem 4.1. *For any $\epsilon > 0$, the number of MP events generated by our online IM protocol is larger than Opt_ϵ by at most a factor of*

$$\lg \rho + \frac{16}{(1 - \delta_{\max})\epsilon} + O(1),$$

where $\delta_{\min}$ and $\delta_{\max} < 1$ are the minimum and maximum drift bounds for Opt_ϵ, respectively, and $\rho = \delta_{\max}/\delta_{\min}$.

Before giving the proof, we need to establish a couple of technical lemmas. The first states that, given two motion estimates with sufficiently similar velocities if a motion satisfies one of the estimates, then for all sufficiently large drifts, it satisfies the other as well.

Lemma 4.2. *Let $E^* = (T^*, \mathbf{v}^*, \delta^*)$ and $E = (T, \mathbf{v}, \delta)$ be two motion estimates, where $T \subseteq T^*$. Let $\alpha = |\mathbf{v} - \mathbf{v}^*|/|\mathbf{v}^*|$. If some motion M satisfies E^*, and $\alpha < 1$, and $\delta \geq (\delta^* + \alpha)/(1 - \alpha)$, then M also satisfies E.*

Proof: By the triangle inequality we have $|\mathbf{v}^*| - |\mathbf{v}| \leq |\mathbf{v} - \mathbf{v}^*|$, and so $|\mathbf{v}^*| - |\mathbf{v}| \leq \alpha|\mathbf{v}^*|$, which implies that $|\mathbf{v}| \geq$

$(1-\alpha)|\mathbf{v}^*|$. Consider any time step $[t_0, t_1] \in T$, and let $s = t_1 - t_0$ denote its duration. Since $T \subseteq T^*$, this step also occurs in T^*. The position of the point predicted by E^* is $\mathbf{p}^*(t_0, s) = \mathbf{p}(t_0) + s\mathbf{v}^*$, and the position predicted by E is $\widehat{\mathbf{p}}(t_0, s) = \mathbf{p}(t_0) + s\mathbf{v}$. In order to satisfy the estimate E, it suffices to show that the point's actual position at time t_1 is within a ball of radius $\delta s|v|$ centered at $\widehat{\mathbf{p}}(t_0, s)$.

The distance between the two estimates $\mathbf{p}^*(t_0, s)$ and $\widehat{\mathbf{p}}(t_0, s)$ is $s|\mathbf{v} - \mathbf{v}^*| \leq s\alpha|\mathbf{v}^*|$. That is, $\widehat{\mathbf{p}}(t_0, s)$ lies in a ball of radius $s\alpha|\mathbf{v}^*|$ centered at $\mathbf{p}^*(t_0, s)$. Also, because the motion satisfies E^*, the actual point position $\mathbf{p}(t_1)$ lies in a ball of radius $s\delta^*|\mathbf{v}^*|$ centered at the same point. Thus, the point $\mathbf{p}(t_1)$ is at distance at most $s(\delta^* + \alpha)|\mathbf{v}^*|$ from $\widehat{\mathbf{p}}(t_0, s)$. Given our prior bound on $|\mathbf{v}|$, this distance is bounded above by

$$s\frac{(\delta^* + \alpha)}{1 - \alpha}|\mathbf{v}|,$$

which completes the proof. □

The next technical lemma states that if one motion estimate (from Opt_ϵ) has a drift bound that is less than 1 and is ϵ-safe, then another motion estimate (from the online protocol) that was constructed within this same time interval, and whose velocity is sampled from the interval, whose drift is not too high, and which is 0-safe, will be guaranteed to be able make a certain degree of progress provided that no drift violations interrupt its progress.

Lemma 4.3. *Let M be a motion that satisfies the ϵ-safe motion unit (E^*, C), where $E^* = (T^*, \mathbf{v}^*, \delta^*)$ and $\delta^* < 1$. Consider another motion unit (E, C) where $E = (T, \mathbf{v}, \delta)$ and $T \subseteq T^*$ and $\mathbf{v}$ is sampled from M during T^*. If (E, C) is free of any drift violations and $|T| \leq (\epsilon\delta^*/(2\delta))|T^*|$ then (E, C) is 0-safe.*

Proof: Let $s^* = |T^*|$ denote the duration of T^*. Let t_0 and t_1 denote the starting times of T^* and T, and let t_2 denote the ending time of T. Let $s_1 = t_1 - t_0$ and $s_2 = t_2 - t_0$, and observe that $s = t_2 - t_1$. It suffices to show that as long as $s \leq (\epsilon\delta^*/(2\delta))s^*$, the motion bound ball at time $t_1 + s = t_2$ for E is encroached only by the certificates of C, since this implies that entire motion bound for E starting at time t_1 is encroached on only by the certificates of C.

Let $\widehat{\mathbf{p}}(t_1, s) = \mathbf{p}(t_1) + s\mathbf{v}$ and $\mathbf{p}^*(t_0, s_2) = \mathbf{p}(t_0) + s_2\mathbf{v}^*$ denote the estimated positions of the point at time t_2 based on E and E^*, respectively. The starting point $\mathbf{p}(t_1)$ and the velocity vector $\mathbf{v}$ were both taken from M within T^*, and so following $\mathbf{v}$ for any distance will result in a motion that satisfies E^*. It follows that the distance from $\mathbf{p}^*(t_0, s_2)$ to $\widehat{\mathbf{p}}(t_1, s)$ is bounded above by E^*'s accumulated drift bound of $x_1 = s_2\delta^*|\mathbf{v}^*|$. Another consequence of the fact that $\mathbf{v}$ was sampled within T^* is that $|\mathbf{v} - \mathbf{v}^*|/|\mathbf{v}^*| \leq \delta^*$, from which we obtain $|\mathbf{v}| \leq (1 + \delta^*)|\mathbf{v}^*| \leq 2|\mathbf{v}^*|$ (using the fact that $\delta^* < 1$).

Applying instead E's motion bound from time t_1, the motion bound at time $t_2 = t_1 + s$ is a ball centered at $\widehat{\mathbf{p}}(t_1, s)$ of radius $s\delta|\mathbf{v}|$. By our bound on $|\mathbf{v}|$, this is at most $2s\delta|\mathbf{v}^*|$. By combining the distance x_1 between the estimated positions with this radius, it follows that the motion bound for E at time t_2 lies within distance

$$y_1 \;=\; x_1 + 2s\delta|\mathbf{v}^*| \;\leq\; s_2\delta^*|\mathbf{v}^*| + 2s\delta|\mathbf{v}^*|$$

of $\mathbf{p}^*(t_0, s_2)$.

Now, using the fact that E^* is ϵ-safe we know that no certificates other than those of C can encroach on the expansion of E^*'s motion bound by $\epsilon s^*\delta^*|\mathbf{v}^*|$. Thus, other than C, no certificates lie within distance

$$y_2 \;=\; s_2\delta^*|\mathbf{v}^*| + \epsilon s^*\delta^*|\mathbf{v}^*|$$

of $\mathbf{p}^*(t_0, s_2)$.

It follows that, other than C, no certificates can encroach on E's motion bounds at time $t_1 + s$ provided that y_1 (the maximum distance from $\mathbf{p}^*(t_0, s_2)$ to E's motion bound) does not exceed y_2 (the minimum distance from $\mathbf{p}^*(t_0, s_2)$ to its any certificate that is not in C). This holds if

$$2s\delta|\mathbf{v}^*| \;\leq\; \epsilon s^*\delta^*|\mathbf{v}^*|.$$

It is easy to see that this holds under our hypothesis that $s \leq (\epsilon\delta^*/(2\delta))s^*$, which completes the proof. □

We now present the proof of Theorem 4.1. Consider the execution of Opt_ϵ on any motion sequence M. Let $E^* = (T^*, \mathbf{v}^*, \delta^*)$ be any motion estimate of any motion unit of Opt_ϵ. We shall show that the number of motion events generated by our online protocol during the time interval T^* is at most $\lg \rho + 16/((1 - \delta_{\max})\epsilon) + O(1)$. We begin by observing that since both algorithms are tracking the same motion, both algorithms generate the same certificate events, and so it suffices to consider just the drift and safeness events.

The proof is based on an amortized analysis. Here is an overview of how the analysis works. We define a potential function below. For the ith event (either drift or safeness), let Φ_i denote the value of this function at the time of this event. The actual cost c_i of this event is 1, thus $\sum_i c_i$ is the total number of events during this time interval. The amortized cost a_i of the event is defined to be the sum of the actual cost and the change in potential, that is, $a_i = c_i + \Delta_i$, where $\Delta_i = \Phi_i - \Phi_{i-1}$. If we let Φ_0 and Φ_f denote the initial and final potentials then we use the fact that the sum of the Δ_i's telescope, yielding

$$\sum_i c_i \;=\; \sum_i (a_i - \Delta_i) \;=\; \left(\sum_i a_i\right) - \Phi_f + \Phi_0.$$

We shall show that $a_i \leq 0$ for all i, and since our potential function will be nonnegative, it will follow that the total number of events is at most Φ_0.

We now define the potential function. Let t^* denote the starting time of T^*, and let s^* denote its total duration. Suppose that some event for the online protocol occurs at time $t^* + s$, for $0 \leq s \leq s^*$. The *remainder* of s is defined to be $r(s) = (s^* - s)/s^*$, which ranges from 1 to 0 as time advances through T^*. When a drift or safeness event is generated, let δ denote the value of the drift bound of the most recent motion estimate. Let $\delta_T = 2\delta^*/(1 - \delta^*)$. We define the potential of an event occurring at time $t^* + s$ with prior drift bound δ to be

$$\Phi(s, \delta) = \left\lfloor \lg \max\left(\frac{2\delta_T}{\delta}, \frac{\delta}{\delta_T}\right)\right\rfloor + 2\left\lfloor \frac{8 \cdot r(s)}{(1 - \delta_{\max})\epsilon}\right\rfloor$$

(where lg denote the base-2 logarithm). We call the two terms of this function the *drift term* and the *remainder term*. Observe that the function is nonnegative. (It is 0 if $\delta_T < \delta < 2\delta_T$ and $r(s)$ is sufficiently small.) Intuitively, the first term captures the amount by which the online algorithm's drift bound differs from the optimum and the remainder term

captures the time progress made. Also observe that, since $r(s)$ is a decreasing function of time, the remainder term cannot increase. Ignoring a small additive constant term, the initial potential Φ_0 cannot be larger than

$$\left\lfloor \lg \frac{\delta_{\max}}{\delta_{\min}} \right\rfloor + 2 \left\lfloor \frac{8}{(1-\delta_{\max})\epsilon} \right\rfloor \;\leq\; \lg \rho + \frac{16}{(1-\delta_{\max})\epsilon}.$$

All that remains to be proven is that $a_i \leq 0$ for all i. We consider cases, depending on the type of the current event. First, suppose that the event is a drift event. In this case, because the motion satisfies E^*, we know from Lemma 4.2 (with $\alpha = \delta^*$) that $\delta < \delta_T$ (since otherwise the motion would also satisfy E and no drift violation would occur). Since this is a drift event, the online protocol doubles the value of δ in generating the next motion estimate. Since $2\delta < 2\delta_T$, it follows that (after taking the floor) the first term of the max is dominant, both before and after this drift change. Thus, the potential decreases by at least 1 (and possibly more if a significant amount of time has elapsed). Thus, $a_i = c_i + \Delta_i \leq 1 - 1 = 0$.

Second, suppose that the event had been a safeness event. We consider two subcases. First, if $\delta > 2\delta_T$, then because Opt_ϵ did not encounter a drift event, we know from Lemma 4.2 (with $\alpha = \delta^*$) that the maximum drift with respect to E occurring in the motion over this last time period (denoted δ^+ in the protocol) is at most $\delta_T \leq \delta/2$. By our protocol, the value of δ will be halved by the online protocol. Because $\delta \geq 2\delta_T$, it follows that (after taking the floor) the second term of the max is dominant, both before and after this drift change. Thus, the potential decreases by at least 1 (and again, possibly more if a significant amount of time has elapsed). Thus, $a_i = c_i + \Delta_i \leq 1 - 1 = 0$.

Finally, in the second subcase, we have a safeness violation and $\delta \leq 2\delta_T$. If this is the first event of the time interval, then we ignore it (and absorb its cost into the $O(1)$ term). Otherwise, we know that the velocity $\mathbf{v}$ generating the most recent motion estimate for the online protocol was sampled from within T^*, and no drift violations have occurred, which means we can apply Lemma 4.3. It states that the motion bound for $\mathbf{v}$ and δ is safe for a duration of at least

$$\frac{\epsilon\delta^*}{2\delta} s^* \;\geq\; \frac{\epsilon\delta^*}{4\delta_T} s^* \;\geq\; \frac{\epsilon(1-\delta^*)}{8} s^* \;\geq\; \frac{\epsilon(1-\delta_{\max})}{8} s^*.$$

Since the online protocol selects its motion bound to be of maximum length, subject to the restriction of safeness, we know that the duration of the last protocol interval, $s_i - s_{i-1}$, is greater than or equal to $(\epsilon(1-\delta_{\max})/8)s^*$. Now, let us consider the change of potential from the remainder term. We have

$$\begin{aligned}
&\frac{8r(s_i)}{(1-\delta_{\max})\epsilon} - \frac{8r(s_{i-1})}{(1-\delta_{\max})\epsilon} \\
&\quad= \frac{8(s^* - s_i)}{s^*(1-\delta_{\max})\epsilon} - \frac{8(s^* - s_{i-1})}{s^*(1-\delta_{\max})\epsilon} \\
&\quad= \frac{8(s_{i-1} - s_i)}{s^*\epsilon(1-\delta_{\max})} \\
&\quad\leq -\frac{8s^*\epsilon(1-\delta_{\max})}{8s^*\epsilon(1-\delta_{\max})} \;=\; -1.
\end{aligned}$$

Since the difference is at most -1, it follows that the difference of their floors is at most -1, and hence the remainder term of the potential decreases by at least 2. It is possible that the maximum observed drift δ^+ is arbitrarily small, and hence we may have decreased the value of δ by half. Unlike the previous case, the drift term of the potential may increase by 1 as a result. Thus the net change in the potential is at most $-2 + 1 = -1$. Thus, $a_i = c_i + \Delta_i \leq 1 - 1 = 0$.

Thus for drift and safeness events the amortized cost can only decrease. As we explained earlier, the certificate events are common to both algorithms. Thus, the total cost is bounded by the initial potential, which is at most $\lg \rho + 16/((1-\delta_{\max})\epsilon) + O(1)$. This completes the proof.

Corollary 4.1. *Let $\delta_{\min}$ and $\delta_{\max}$ be the minimum and maximum drift bounds for Opt_ϵ, respectively. If $\delta_{\max}$ is bounded away from 1, and the ratio $\delta_{\max}/\delta_{\min}$ is bounded above by a constant, then the competitive ratio of our online IM protocol relative to Opt_ϵ is $O(1/\epsilon)$.*

5 Multiple-Point Certificates

So far we have concentrated on single-point certificates. Most interesting geometric structures involve certificates that involve a constant number of points (assuming fixed dimension). With single-point certificates it is natural to bundle motion estimates together with certificates because they are both associated with the same point. This is no longer the case with multiple-point certificates. Nonetheless, there is a straightforward way to map our framework to this more general setting.

Recall that the points reside in real d-dimensional space, $\mathbb{R}^d$. A certificate that involves, say k, moving points can be viewed as certificate that involves one moving point in dimension kd, by simply concatenating the coordinates of the position vectors. A certificate involving k points implicitly defines a region of $\mathbb{R}^{kd}$ where the certificate fails to hold. Thus, it is straightforward to generalize motion estimates, motion bounds, and certificate encroachment to this kd-dimensional setting.

We modify the motion bounds for each of the moving points individually. At each time, the motion bound for one of the points is a ball in d-space. The motion bound for the associated k-point certificate is the cartesian product of these k balls. Unlike the single-point certificate case, where we maintain only one motion estimate per point, here we maintain as many different motion estimates per point as the number of certificates in which the point is involved. Thus, the motion estimates for the points belonging to the same certificate all start at the same time. If the IM algorithm has been designed to satisfy the KDS locality property, then each point is only involved in a constant number of certificates at any time, and so it is still the case that the MP can process each point in $O(1)$ time and space.

In order to generalize the online protocol, we deal with points individually. For example, even though a certificate may involve k points, if the ith point undergoes a drift violation, only the drift bound for the ith point is doubled. The competitive analysis is now applied individually to each of the k points. That is, there are k potential functions, one for each point. The various elements of the proof are applied individually to each of the k points. It follows that the final competitive ratio is higher by a factor of k.

References

[1] P. K. Agarwal, L. J. Guibas, et al. Algorithmic issues in modeling motion. *ACM Computing Surveys*, 34:550–572, 2002.

[2] Pankaj K. Agarwal, Jeff Erickson, and Leonidas J. Guibas. Kinetic BSPs for intersecting segments and disjoint triangles. In *Proc. 9th ACM-SIAM Sympos. Discrete Algorithms*, pages 107–116, 1998.

[3] M. J. Atallah. Some dynamic computational geometry problems. *Comput. Math. Appl.*, 11(12):1171–1181, 1985.

[4] J. Basch, J. Erickson, L. J. Guibas, J. Hershberger, and L. Zhang. Kinetic collision detection between two simple polygons. In *Proc. 10th ACM-SIAM Sympos. Discrete Algorithms*, pages 102–111, 1999.

[5] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 747–756, 1997.

[6] A. Czumaj and C. Sohler. Soft kinetic data structures. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 865–872, 2001.

[7] Leonidas Guibas, John Hershberger, Subash Suri, and Li Zhang. Kinetic connectivity for unit disks. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 331–340, 2000.

[8] J. Hershberger. Smooth kinetic maintenance of clusters. In *Proc. 19th Annu. ACM Sympos. Comput. Geom.*, pages 48–57, 2003.

[9] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[10] S. Kahan. A model for data in motion. In *Proc. 23th Annu. ACM Sympos. Theory Comput.*, pages 267–277, 1991.

[11] R. E. Kalman. A new approach to linear filtering and prediction problems. *Trans. of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[12] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. Computing nearest neighbors for moving points and applications to clustering. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages S931–S932, Balitimore, MD, January 1999.

[13] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. The analysis of a simple k-means clustering algorithm. In *Proceedings of the Sixteenth Annual ACM Symposium on Computational Geometry*, pages 100–109, Hong Kong, June 2000.

[14] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Patt. Anal. Mach. Intell.*, 24, 2002. 881–892.

[15] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, New York, NY, 1990.

[16] D. Kirkpatrick, J. Snoeyink, and B. Speckmann. Kinetic collision detection for simple polygons. *International Journal of Computational Geometry and Applications*, 12:3–27, 2002.

[17] M. Lin and D. Manocha. Efficient contact determination in dynamic environments. *Internat. J. Comput. Geom. Appl.*, 7:123–151, 1997.

[18] M. C. Lin and J. F. Canny. Efficient algorithms for incremental distance computation. In *Proc. IEEE Internat. Conf. Robot. Autom.*, volume 2, pages 1008–1014, 1991.

[19] P. S. Maybeck. *Stochastic Models, Estimation, and Control*, volume 1. Academic Press, New York, 1979.

[20] J. S. B. Mitchell, D. M. Mount, and S. Suri. Query-sensitive ray shooting. *International Journal of Computational Geometry and Applications*, 7:317–347, August 1997.