Maintaining Nets and Net Trees under Incremental Motion*

Minkyoung Cho, David M. Mount, and Eunhui Park

Department of Computer Science
University of Maryland
College Park, Maryland
{minkcho, mount, ehpark}@cs.umd.edu

Abstract. The problem of maintaining geometric structures for points in motion has been well studied over the years. Much theoretical work to date has been based on the assumption that point motion is continuous and predictable, but in practice, motion is typically presented incrementally in discrete time steps and may not be predictable. We consider the problem of maintaining a data structure for a set of points undergoing such incremental motion. We present a simple online model in which two agents cooperate to maintain the structure. One defines the data structure and provides a collection of certificates, which guarantee the structure's correctness. The other checks that the motion over time satisfies these certificates and notifies the first agent of any violations.
We present efficient online algorithms for maintaining both nets and net trees for a point set undergoing incremental motion in a space of constant dimension. We analyze our algorithms' efficiencies by bounding their competitive ratios relative to an optimal algorithm. We prove a constant factor competitive ratio for maintaining a slack form of nets, and our competitive ratio for net trees is proportional to the square of the tree's height.

1 Introduction

Motion is a pervasive concept in geometric computing. The problem of maintaining discrete geometric structures for points in motion has been well studied over the years. The vast majority of theoretical work in this area falls under the category of kinetic data structures (KDS) [7]. KDS is based on the assumption that points move continuously over time, where the motion is specified by algebraic functions of time. This makes it possible to predict the time of future events, and so to predict the precise time in the future at which the structure will undergo its next discrete change. In practice, however, motion is typically presented incrementally over a series of discrete time steps by a *black-box*, that is, a function that specifies the locations of the points at each time step. For example, this black-box function may be the output of a physics integrator, which determines

* This work has been supported by the National Science Foundation under grant CCR-0635099 and the Office of Naval Research under grant N00014-08-1-1015.

the current positions of the points based on the numerical solution of a system of differential equations [1, 12]. Another example arises in the use of Markov-chain Monte-Carlo (MCMC) algorithms such as the Metropolis-Hastings algorithm [2] and related techniques such as simulated annealing [10].

In this paper we consider the maintenance of two well known structures, nets and net trees, for a set of moving points. Let P denote a finite set of points in some (continuous or discrete) metric space $\mathcal{M}$. Given $r > 0$, an *r-net* for P is a subset $X \subseteq P$ such that every point of P lies within distance r of some point X, and no two points of X are closer than r. (We will actually work with a generalization of this definition, which will present in Section 2.) Each point of P can be associated with a covering point of X that lies within distance r. This point is called its *representative*. We can easily derive a tree structure, by building a series of nets with exponentially increasing radius values, and associating each point at level $i-1$ with its representative as parent at level i.

The net tree has a number of advantages over coordinate-based decompositions such as quadtrees. The first is that the net tree is intrinsic to the point set, and thus the structure is invariant under rigid motions of the set. This is an important consideration with kinetic point sets. Another advantage is that the net tree can be defined in general metric spaces, because it is defined purely in terms of distances. A number of papers have been written about improvements to and applications of the above net-tree structure in metric spaces of constant doubling dimension. (See, for example [11, 8, 4, 6].) Note that the net tree is a flexible structure in that there may be many possible choices for the points that form the nets at each level of the tree and the assignment of points to parents.

Although there has been much research on maintaining geometric structures in continuous contexts, such as KDS, there has been comparatively little theoretical work involving efficiency of algorithms for incremental black-box motion. Gao *et al.* [5] observe that their data structure (which is very similar to our net-tree structure) can be updated efficiently in the black-box context, but they do not consider the issue of global efficiency. A major issue here is the computational model within which efficiency is to be evaluated. In the absence of any *a priori* assumptions about the point motions, the time required to update the point locations and verify the correctness of the data structure is already $\Omega(|P|)$. With each time step the points could move to entirely new locations, thus necessitating that the data structure be rebuilt from scratch. This need not always be the case, however. It has been widely observed (see, e.g., [5, 9, 13]) that when the underlying motion is continuous, and/or the time steps are small, the relative point positions are unlikely to change significantly. Hence, the number of discrete structural changes per time step is likely to be small. We desire a computational model that allows us to exploit any underlying continuity in the motion to enhance efficiency, without the strong assumptions of KDS.

We introduce such a computational model for the online maintenance of geometric structures under incremental black-box motion. Our approach is similar other models for incremental motion, such as the observer-tracker model of Yi and Zhang [14] and the IM-MP model of Mount *et al.* [13]. Our model involves

the interaction of two agents, an *observer* and a *builder*. The observer monitors the motions of the points over time, and the builder is responsible for maintaining the data structure. These two agents communicate through a set of boolean conditions, called *certificates*, which effectively "prove" the correctness of the current structure (exactly as they do in KDS). Based on the initial point positions, the builder constructs the initial structure and the initial certificates, and communicates these certificates to the observer. The observer monitors the point motion and, whenever it detects that a certificate has been violated, it informs the builder which certificates have been violated. The builder then queries the new locations the points, updates the data structure, and informs the observer of any updates to the certificate set. An algorithm for maintaining a data structure in this model is essentially a communication protocol between the observer and the builder. The total computational cost of an algorithm is defined to be the communication complexity between these two agents. One advantage of this model is that it divorces low-level motion issues from the principal algorithmic issues involving the design of the net structure itself.

Our main results are efficient online algorithms for maintaining both nets and net trees for a point set undergoing incremental motion. In each case, our algorithm is allowed some additional slackness in the properties of the net to be maintained. For example, while the optimal algorithm is required to maintain all points within distance r of each net point, we allow our algorithm for nets to maintain a covering distance of $2r$ for nets and $4r$ for net trees. (See Section 2 for the exact slackness conditions.) Because our principal motivation is in maintaining net trees under motion, we impose the assumption that the input points to our r-net algorithm arise from an $(r/2)$-net.

We establish the efficiency of our online algorithms by proving an upper bound on the *competitive ratio* on the communication cost of our algorithm, that is, the worst-case ratio between the communication costs of our algorithm (subject to the slackness conditions) and any other algorithm (without the slackness). The exact results are presented in Sections 3 and 4. Assuming that the points are in a space of constant doubling dimension (e.g., Euclidean of constant dimension), we achieve a competitive ratio of $O(1)$ for the maintenance of a net and $O(\log^2 \Phi)$ for the net tree, where Φ is the aspect ratio of the point set (the ratio between the maximum and minimum interpoint distances). Our online algorithm makes no *a priori* assumptions about the motion of the points. The competitive ratio applies even if the optimal algorithm has full knowledge of point motion, and it may even have access to unlimited computational resources. The constant factors hidden by the asymptotic notation grow exponentially in the doubling dimension of the underlying metric space.

2 Preliminaries

We begin with some basic definitions, which will be used throughout the paper. Let $\mathcal{M}$ denote a metric space, with associated distance function dist: $\mathcal{M} \times \mathcal{M} \to \mathbb{R}$. (This means that dist is symmetric, positive definite, and satisfies the triangle

inequality.) Throughout, we let P be a finite subset of points in $\mathcal{M}$. For a point $p \in \mathcal{M}$ and a real $r \in \mathbb{R}^+$, let $b(p,r) = \{q \in \mathcal{M} : \text{dist}(p,q) < r\}$ denote the open ball of radius r centered at p. The *doubling constant* of the metric space is defined to be the minimum value λ such that every ball b in $\mathcal{M}$ can be covered by at most λ balls of at most half the radius. The *doubling dimension* of the metric space is defined as $d = \log_2 \lambda$. Throughout, we assume that $\mathcal{M}$ is a space of constant doubling dimension.

Consider a point set P in $\mathcal{M}$. Given $r \in \mathbb{R}^+$, an *r-net* for P [8] is a subset $X \subseteq P$ such that,

$$\max_{p \in P} \min_{x \in X} \text{dist}(p,x) \;<\; r \qquad \text{and} \qquad \min_{\substack{x,y \in X \\ x \neq y}} \text{dist}(x,y) \;\geq\; r.$$

The first constraint is called the *covering constraint*, and the second is called the *packing constraint*. Each point $p \in P$ may be associated with a *representative* $x \in X$ (denoted by $\text{rep}(p)$) lying within distance r. No two representatives are closer than r. Note that the choice of representative is not necessarily unique.

In order to establish our competitive ratio, we will need to relax the r-net definition slightly. Given constants $\alpha, \beta \geq 1$, an (α,β)*-slack r-net* is a subset $X \subseteq P$ of points such that,

$$\max_{p \in P} \min_{x \in X} \text{dist}(p,x) \;<\; \alpha\, r \qquad \text{and} \qquad \forall x \in X,\; |\{X \cap b(x,r)\}| \;\leq\; \beta.$$

Thus, we allow each point to be farther from the closest net point by a factor of α, and we allow net points to be arbitrarily close to each other, but there cannot be more than β points within distance r of any net point. Clearly, an (α,β)-slack r-net is an (α',β')-slack r-net for $\alpha' \geq \alpha$ and $\beta' \geq \beta$. When we wish to make the distinction clearer, we will use the term *strict r-net* to denote the standard definition, which arises as a special case when $\alpha = \beta = 1$.

Before introducing net trees, we first introduce the concept of the *aspect ratio* (or *spread*) of a kinetic point set P. Consider two positive reals δ and Δ, which denote the minimum and maximum distances, respectively, between any two points of P throughout the course of the motion. We define $\Phi(P)$ to be Δ/δ. By scaling distances, we may assume that $\delta = 1$.

A *net tree* of P is defined as follows. The leaves of the tree consists of the points of P. Note that by our assumption that $\delta = 1$, these form a 1-net of P itself, which we denote by $P^{(0)} = P$. The tree is based on a series of nets, $P^{(1)}, P^{(2)}, \ldots, P^{(m)}$, where $m = \lceil \log_2 \Phi \rceil$, and $P^{(i)}$ is a (2^i)-net for $P^{(i-1)}$. Observe that $|P^{(m)}| = 1$. Recall that, for each $p \in P^{(i-1)}$, there is a point $x \in P^{(i)}$, called its representative, such that $\text{dist}(p,x) \leq 2^i$. We declare this point to be a *parent* of p, which (together with the fact that $|P^{(m)}| = 1$) implies that the resulting structure is a rooted tree. An easy consequence of the packing and covering constraints is that the number of children of any node of this tree is a constant (depending on the doubling constant of the containing metric space). Our definition is based on the simplest forms of net tree [5, 11], in contrast to more sophisticated forms given elsewhere [8, 4, 6].

This definition can be easily generalized to assume that the nets forming each level of the tree are (α, β)-slack nets. We refer to such a tree as an (α, β)-slack net tree. Assuming that α and β are constants, this relaxation will affect only the constant factors in the asymptotic complexity bounds.

Because our ultimate interest is in maintaining net trees under incremental motion, it will be convenient to impose an additional constraint on the points P. In a net tree, the input to the ith level of the tree is a (2^{i-1})-net, from which we are to compute a 2^i net. Thus, in our computation of an r-net, we will assume that the point set P is an $(r/2)$-net.

Recall that we interested in maintaining points under incremental black-box motion. More formally, we assume that the points change locations synchronously at discrete time steps $T = \{0, 1, \ldots, t_{\max}\}$. Given a point $p \in P$ and $t \in T$, we use p to refer to the point in the symbolic sense, and (when time is significant) we use p_t to denote its position at time t.

Let us now consider the certificates used in the maintenance of an r-net. In order to maintain an (α, β)-slack r-net, the observer must be provided enough information to verify that the covering and packing constraints are satisfied. At any time t, let P_t denote the current point set, and let X_t denote the current slack net. We assume the incremental maintenance of any net is based on the following two types of certificates, where the former validates the covering constraint and the latter validates the packing constraint.

Assignment Certificate(p, x): For $p \in P$ and $x \in X$, $\text{rep}(p) = x$, and therefore at each time t, $\text{dist}(p_t, x_t) < \alpha r$.

Packing Certificate(x): For $x \in X$, at each time t we have $|X_t \cap b(x_t, r)| \leq \beta$.

The first condition requires constant time to verify. The second condition involves answering a spherical range counting query. Observe that $O(|P|)$ certificates suffice to maintain the net.

3 Incremental Maintenance of a Slack Net

In this section we present an online algorithm for maintaining an r-net for a set of points undergoing incremental motion, and we provide an analysis of its competitive ratio. Given our metric space $\mathcal{M}$, let $\beta = \beta(\mathcal{M})$ denote the maximum number of balls of radius r that can overlap an arbitrary ball of radius r, such that the centers of these balls are at distance at least r from each other. We shall show in Lemma 1(iii) below that $\beta \leq \lambda^2 = 4^d$, where λ is the doubling constant of $\mathcal{M}$, and d is the doubling dimension of $\mathcal{M}$. The main result of this section is as follows.

Theorem 1. *Consider any metric space $\mathcal{M}$ of constant doubling dimension, and let $\beta = \beta(\mathcal{M})$ be as defined above. There exists an incremental online algorithm, which for any real $r > 0$, maintains a $(2, \beta)$-slack r-net for any point set P under incremental motion in $\mathcal{M}$. Under the assumption that P is a $(2, \beta)$-slack $(r/2)$-net, the algorithm achieves a competitive ratio of at most $(\beta\lambda^3 + 2)(\beta + 2) = O(\lambda^7) = O(1)$.*

The algorithm begins by inputting the initial placements of the points, and it communicates an initial set of certificates to the observer. (We will discuss how this is done below.) Recall that the observer then monitors the point motions over time, until first arriving at a time step t when one or more of these certificates is violated or when a point of P is explicitly inserted or deleted. It then wakes up the builder and informs it of the current event.

Our algorithm maintains the points of the slack r-net, which we denote by X, and the assignment of each point $p \in P$ to its representative $\text{rep}(p) \in X$. For each point $p \in P$, we maintain a subset $\text{cand}(p) \subseteq X$, called the *candidate list*. Before describing the update process, we present the following utility operations.

Reassign(p): If $\text{cand}(p) \neq \emptyset$, repeatedly extract elements from $\text{cand}(p)$ until finding a candidate $x \in X$ that lies within distance $2r$ of p. If such a candidate is found, set $\text{rep}(p) \leftarrow x$, and create a new assignment certificate involving p and x. If no such candidate exists, invoke Create-net-point(p).

Create-net-point(p): We assume the precondition that $\text{cand}(p) = \emptyset$. Add p to the current net X. Set $\text{rep}(p) \leftarrow p$. For each point $p' \in P \setminus \{p\}$ such that $\text{dist}(p', p) < 2r$, add p to $\text{cand}(p')$. Finally, create a packing certificate for p.

Remove-net-point(x): First, x is removed from both X and all the candidate lists that contain it. Remove any packing certificate involving x. For all $p \in P$ such that $\text{rep}(p) = x$, invoke Reassign(p).

Note that the reassignment operation generates one new assignment certificate (for p) and may create one packing certificate if p is added to X. Let us now consider the possible actions of the builder, in response to any event (point insertion or deletion) or a certificate violation (assignment or packing).

Insert-point(p): Set $\text{cand}(p)$ to be the set of net points $x \in X$ such that $\text{dist}(p, x) < 2r$. Then invoke Reassign(p).

Delete-point(p): All certificates involving p are removed. If $p \in X$, then invoke Remove-net-point(p). Finally, remove p from P.

Assignment-certificate-violation(p): Let $x = \text{rep}(p)$. Remove x from p's candidate list and invoke Reassign(p).

Packing-certificate-violation(x): Invoke Remove-net-point(x), for each $x \in X \cap b(x, r)$. (This results in at least $\beta + 1$ removals.)

Observe that the processing of any of the above events results in a constant number of changes to the certificate set, and hence in order to account for the total communication complexity, it suffices to count the number of operations performed.

Initially, X is the empty set, and we start the process off by invoking the insertion operation for each $p \in P$, placing it at its starting location. Observe that after the processing of each assignment- and packing-certificate violation, the condition causing the violation has been eliminated, and therefore it is easy to see that the above protocol maintains a $(2, \beta)$-slack r-net for P.

Recall that P denotes the point set, which is in motion of some finite time period. For any time t, let $N_t(o)$ denote the *optimal neighborhood* consisting

of the points of P that have o assigned as their representative by the optimal algorithm. We will show that each of the operations performed by our algorithm can be charged to some operation of the optimal algorithm, in such a manner that each optimal operation is charged a constant number of times.

We first present some geometric preliminaries, which will be useful later in the analysis. Recall that λ denotes the doubling constant of the metric space. Due to space limitations, the proofs of the lemmas appear in the full version of this paper [3].

Lemma 1. (*i*) *Given any (α, β)-slack r-net X, at most $\beta\lambda^{\lg\lceil 2R/r\rceil}$ points of X can lie within any ball of radius R.*

(*ii*) *Let P be an (α, β)-slack $(r/2)$-net, and let X be an (α, β)-slack r-net for P. Then the number of points of X (respectively, P) that lie within a ball of radius $2^k r$ is at most $\beta\lambda^{k+1}$ (respectively, $\beta\lambda^{k+2}$).*

(*iii*) *Let Z be a set of balls of radius r whose centers are taken from a (strict) r-net. Then any ball b of radius r (not necessarily in Z) can have a nonempty intersection with at most λ^2 balls of Z.*

Given the motion sequence for the point set P, let n denote the total number of operations performed by our online slack-net algorithm, and let n^* denote the total number of operations processed by any correct (e.g., the optimal) algorithm. In order to establish the competitive ratio, it suffices to show

$$n \;\leq\; (\beta\lambda^3 + 2)(\beta + 2)n^*. \tag{1}$$

Let n_A^*, n_C^*, n_R^*, n_I^*, and n_D^*, and denote, respectively, the total number of assignments, net point creations, net-point removals, point insertions, and point deletions performed by the optimal algorithm. Let n_A, n_C, n_R, n_I, and n_D denote corresponding quantities for our slack-net algorithm. Thus, we have $n \;=\; n_A + n_C + n_R + n_I + n_D$.

First, we bound the total number of assignments in terms of the number of point insertions and slack-net creations. Changes in assignment in our algorithm occur as a result of running of the reassignment operator. Since the assignment is made to some point of the candidate list, it suffices to bound the total number of insertions into candidate lists. This occurs when points are inserted and when net points are created.

Lemma 2. $n_A \;\leq\; \beta\lambda^3 n_C \;+\; \beta\lambda^2 n_I$.

Since point insertions and deletions must be handled by any correct algorithm, we have $n_I = n_I^*$ and $n_D = n_D^*$. The total number of net point removals (n_R) cannot exceed the total number of net point creations (n_C). Thus, it suffices to bound n_C, the total number of slack-net point creations.

Before bounding n_C, we make a useful observation. Whenever a net point x is created, it adds itself to the candidate list of all points that lie within distance $2r$. Since (by strictness) the diameter of any optimal neighborhood is at most $2r$, it follows that, in the absence of other events, the creation of net-point x within an optimal neighborhood inhibits the creation of any other net points within this neighborhood. Given $t \leq t'$, let $(t, t']$ denote the interval $[t + 1, t']$.

Lemma 3. *Let o denote an optimal net point. Suppose that no optimal assignments occur to the points of $N(o)$ during the time interval $(t, t']$, $x \in N(o)$ is added to X at time t, and x is not removed from X throughout $(t, t']$. Then, for any time in $(t, t']$ no point $p \in N(o)$ will be added to X.*

This implies that, without optimal assignments, each optimal net point o can have at most one corresponding slack net point $x \in N(o)$. Furthermore, if x is removed from X (e.g., as the result of a packing-certificate violation), only one of the points of $N(o)$ may replace it as a slack-net point. When a net point within an optimal neighborhood is created to replace a removed net point, we call this a *recovery*. Whenever a packing-certification violation occurs, at least $\beta+1$ slack-net points are removed. The following lemma implies that the number of recovered net points is strictly smaller.

Lemma 4. *The number of net points recovered as a result of the processing of a packing-certificate violation is at most β.*

We say that an optimal neighborhood $N(o)$ is *crowded* (at some time t) if $|N_t(o) \cap X_t| \geq 2$. Our next lemma states that whenever a packing-certificate violation occurs, at least two of removed net points lie in the same crowded neighborhood.

Lemma 5. *Consider a packing-certificate violation which occurs in the slack net but not within the optimal net, and let $X' \subseteq X$ denote the net points that have been removed as a result of its handling. Let O' denote a subset of O of overlapping neighborhoods, that is, $O' = \{o \mid N(o) \cap X' \neq \emptyset\}$. Then, there exists $o \in O'$ such that $|N(o) \cap X'| \geq 2$.*

The handling of the packing-certificate violation removes the points of X', and by Lemma 3, this optimal neighborhood will recover at most one net point. Thus, in the absence of other effects (optimal reassignment in particular) the overall crowdedness of the system strictly decreases after processing each packing-certificate violation. Intuitively, crowdedness increases whenever a point of the slack net is moved from one optimal neighborhood to another. But such an event implies that the optimal algorithm has modified an existing assignment. We can therefore charge each slack-net point creation to such an optimal reassignment.

We summarize that above analysis to obtain the following bound.

Lemma 6. $n_C \leq (\beta + 2)\, n_A^* + n_C^* + n_D^*$.

The proof of Theorem 1 follows by combining the observations of this section with Lemmas 2 and 6.

4 Incremental Maintenance Algorithm for Net Tree

The main result of this section is presented below. In contrast to the results of the previous section, the slackness parameter α in the net increases from 2 to 4 and the competitive ratio increases by a factor of $O(\lambda h^2)$. Recall from Section 3 that $\beta = \lambda^2$.

Theorem 2. *Consider any metric space $\mathcal{M}$ of constant doubling dimension. There exists an online algorithm, which maintains a $(4, \beta)$-slack net tree for any point set P under incremental motion in $\mathcal{M}$. The algorithm achieves a competitive ratio of at most $(\beta\lambda^4 + \beta\lambda^3 + 4)(\beta + 3)h^2 = O(\lambda^8 h^2) = O(h^2)$.*

Intuitively, our algorithm for maintaining the net tree is based on applying the algorithm of the previous section to maintain the net defining each level of the tree. Creation and removal of net points at level will result in point insertions and deletions at other levels of the tree. Let $O^{(i)}$ and $X^{(i)}$ denote nets at level i of the tree generated by (strict) optimal algorithm and our slack-net algorithm, respectively. It will be convenient to use $P^{(i)}$ as a pseudonym for $X^{(i-1)}$.

The competitive analysis of the previous section was based on the relationship between slack-net points and neighborhoods of the optimal net. However, except at the leaf level, the points of $P^{(i)}$ need not reside in level i of the optimal net tree. Consider an optimal net point $o \in O^{(i)}$. We define the optimal neighborhood, still denoted by $N(o)$, to be the set of points of P lying in the leaves of the tree that are descended from o. Because of the exponential decrease in the radius values, it is easy to see that the descendants of o lie within distance of o of $2^i + 2^{i-1} + \cdots + 1 < 2^{i+1} = 2r_i$. Thus, the diameter of $N(o)$ is at most $4r_i$. This implies that, if we choose any point $x \in N(o)$ to be in our $(4, \beta)$-slack net, it can be used to cover all the points of the optimal neighborhood.

Let us now consider the operations performed by our algorithm at level i. Each operation is defined in terms of the analogous single-net operation of Section 3 applied to the points at this level of the net tree. In each case the operation may cause events to propagate to higher levels of the tree. We begin by describing a few utility operations. Throughout i denotes the tree level at which the operation is being applied.

Tree-reassign(p, i): Invoke Reassign(p) on $X^{(i)}$, but use Tree-create-net-point at level-i (rather than Create-net-point).

Tree-create-net-point(p, i): Invoke Create-net-point(p) on $X^{(i)}$, with one difference. The point p is added to the candidate lists of points within distance $4r_i$ (rather than $2r_i$). Invoke Tree-insert-point$(p, i + 1)$.

Tree-remove-net-point(x, i): First, invoke Remove-net-point(x) on $X^{(i)}$, and then invoke Tree-delete-point$(x, i + 1)$.

With the aid of these utility operations, we now present the actions taken by the builder in response to the various events.

Tree-insert-point(p, i): Invoke Insert-point(p) on $P^{(i)}$, but with the following change. Set cand(p) to be the set of net points $x \in X^{(i)}$ such that $\text{dist}(p, x) < 4r_i$ (rather than $2r_i$).

Tree-delete-point(p, i): Invoke Delete-point(p) on $P^{(i)}$. If $p \in X^{(i)}$, invoke Tree-remove-net-point(p, i).

Tree-assignment-certificate violation(p, i): Invoke the single-net assignment certificate violation for p on $P^{(i)}$, but use Tree-reassign(p, i) (rather than Reassign(p)).

Tree-packing-certificate-violation(x, i): Invoke the single-net packing certificate violation on $X^{(i)}$, but use Tree-remove-net-point(x, i) (rather than Remove-net-point(x)).

As with single-net operations, each operation may induce addition certificate violations. These violations are stored in a priority queue, and violations are first applied to the lower levels of the tree and then propagate upwards.

Due to space limitations, we have omitted the competitive analysis for above the net tree algorithm. The analysis appears in the full version of the paper [3]. The analysis involves showing that each operation performed by our algorithm can be charged to some operation performed by the optimal algorithm, such that each optimal operation is charged at most $O(h^2)$ times, where h is the height of the tree.

References

1. B. Adams, M. Pauly, R. Keiser, and L. J. Guibas. Adaptively sampled particle fluids. In *ACM SIGGRAPH*, page 48, 2007.
2. S. Chib and E. Greenberg. Understanding the metropolis-hastings algorithm. *The American Statistician*, 49:327–335, 1995.
3. M. Cho, D. M. Mount, and E. Park. Maintaining nets and net trees under incremental motion. Technical Report CS-TR-4941, UMIACS-TR-2009-10, University of Maryland, College Park, 2009.
4. R. Cole and L.-A. Gottlieb. Searching dynamic point sets in spaces with bounded doubling dimension. In *Proc. 38th Annu. ACM Sympos. Theory Comput.*, pages 574–583, 2006.
5. J. Gao, L. J. Guibas, and A. Nguyen. Deformable spanners and applications. *Comput. Geom. Theory Appl.*, 35:2–19, 2006.
6. L.-A. Gottlieb and L. Roditty. An optimal dynamic spanner for doubling metric spaces. In *Proc. 16th Annu. European Sympos. Algorithms*, volume 5193/2008, pages 478–489. Springer, 2008.
7. L. Guibas. Kinetic data structures. In D. Mehta and S. Sahni, editors, *Handbook of Data Structures and App.*, pages 23–1–23–18. Chapman and Hall/CRC, 2004.
8. S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM J. Comput.*, 35:1148–1184, 2006.
9. S. Kahan. A model for data in motion. In *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pages 265–277, 1991.
10. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
11. R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. In *Proc. 15th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 798–807, 2004.
12. J. J. Monaghan. Smoothed particle hydrodynamics. In *Reports on Progress in Physics*, volume 68, pages 1703–1759, 2005.
13. D. M. Mount, N. S. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. A computational framework for incremental motion. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pages 200–209, 2004.
14. K. Yi and Q. Zhang. Multi-dimensional online tracking. In *Proc. 20th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 1098–1107, 2009.