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. The vast majority of theoretical work in this area has been based on the assumption that point motion is continuous and the future motions of the points are known in advance. In practice, however, motion is typically presented incrementally in discrete time steps and the long-term motion of points is not known and may not even be predictable. We consider the problem of maintaining a data structure for storing a set of points under 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. We analyze our algorithms' efficiency by bounding their competitive ratios relative to an optimal algorithm. Our competitive ratio is a function of the doubling dimension of the space and (in the case of the net tree) the height of the net tree.

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) [6]. 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

* 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.

of the points at each time step. For example, this black-box function may be the output of a physics integrator, which determines the current positions of the points based on the numerical solution of a system of differential equations [1,11]. 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 [9].

In this paper we will consider the maintenance of nets and net trees for a set of points. Let P denote a finite set of points in some metric space $\mathcal{M}$, which may be either continuous (as in Euclidean space) or discrete. 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, which 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. A net tree can be viewed as a metric generalization of hierarchical partition trees like quadtrees [13].

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 [3, 5, 7, 10].) 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.* [4] 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., [4,8,12]) 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 heavy restrictions of KDS.

We introduce such a computational model for the online maintenance of geometric structures under incremental black-box motion. Our approach is similar to the observer-tracker model proposed recently by Yi and Zhang [14] in the context of online tracking, and is similar in spirit to the IM-MP model of Mount *et al.* [12]. 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*. The certificates 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 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 providing 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 space.

The rest of the paper is organized as follows. In the next section we present definitions and background information. In Section 3 we present our algorithm for maintaining a net, and in Section 4 we present our algorithm for maintaining a net tree.

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 $\text{dist}\colon \mathcal{M}\times\mathcal{M}\to\mathbb{R}$. (This means that dist is symmetric, positive definite, and satisfies the triangle inequality.) Throughout, we let P be a finite subset of points in some metric space $\mathcal{M}$. 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.

Recall that P is a finite set of points in some metric space $\mathcal{M}$. Given $r\in\mathbb{R}^+$, an *r-net* for P [7] is a subset $X\subseteq P$ such that, for some constant $\gamma\geq 1$,

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

The first constraint is called the *covering constraint*, and the second is called the *packing constraint*. Intuitively, a net defines a simple type of clustering of the point set into balls of radius r, and each point $p\in P$ can 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 representative is not necessarily unique. To simplify notation, henceforth, we assume that $\gamma=1$, but our results apply to any constant γ.

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,

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

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 static point set to be the ratio of the diameter of P and the distance between the closest pair of points in P. Since we will be dealing with points in motion, we assume that we are two values δ and Δ, which provide a

lower and upper bound, respectively, on the distances 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 $|P^{(m)}| = 1$. (More generally, we may replace 2 with any constant that is greater than 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 usage of the term "net tree" is not standard. There have been a number of related data structures, based on a hierarchical collection of nets. Our definition is based on the simplest forms of net tree [4, 10], whose height depends on the aspect ratio. This is in contrast to more sophisticated forms given in [3, 5, 7], which achieve height of $O(\log |P|)$, independent of the aspect ratio.

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. In addition to just moving the points, we will also allow points to be inserted or deleted from the set at any time.

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

Packing Certificate: Given $x \in X$, 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. For the purposes of the results presented here, it suffices to answer such queries approximately to within a constant approximation error (which only affects the constant factors in the analysis). 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 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 3 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 remainder of this section is devoted to proving this theorem.

3.1 Online Algorithm for Maintaining a Slack Net

In this section we present our online algorithm. 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. The builder applies the operations as described in our algorithm below, and returns control to the observer.

Our algorithm maintains not only the points of the slack r-net, which we denote by X, but also 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 incremental update process, we begin with three useful utility operations: *reassignment*, *net-point creation*, and *net-point removal*.

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 the net-point creation operation for 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 the candidate list $\text{cand}(p')$. 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 the reassignment operations on 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, once the observer reports an 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 apply the reassignment operation to p.

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

Assignment-certificate violation(p): Let p be the point involved, and let $x = \text{rep}(p)$ be its representative. Remove x from p's candidate list and apply the reassignment operation to p.

Packing-certificate violation(x): Invoke net-point removal for each point of $X \cap b(x, r)$.

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 we have the following.

Lemma 1. *If no certificates are currently violated, the set X maintained by the above algorithm is a valid $(2, \beta)$-slack r-net.*

3.2 Competitive Analysis for Maintaining Nets

In this section we present a competitive analysis of the computational complexity of the online algorithm presented in the previous section. 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. Our competitive analysis is based on showing 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 (depending on the doubling dimension and associated packing lemma).

First, we 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 are given in the appendix.

Lemma 2. *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.*

Recall that the points of our slack r-net are assumed to arise from a slack $(r/2)$-net. By the above lemma and the definition of (α, β)-slack r-net, we have the following.

Corollary 1. *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, $\beta\lambda^{k+2}$).*

Our choice of $\beta = \lambda^2$ in Theorem 1 is a direct consequence of the following lemma.

Lemma 3. *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

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

The remainder of this section is devoted to showing this.

Our analysis is based on a charging argument, which relates the total number of slack-net operations to the number of slack-net creations and then relates the number of slack-net creations to the number of optimal operations. 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 4. $n_A \;\leq\; \beta\lambda^3 n_C \;+\; \beta\lambda^2 n_I$.

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 5. *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 6. *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 7. *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, $O' = \{o \mid N(o) \cap X'\}$. Then, there exists at least one optimal center $o \in O'$ such that $|N(o) \cap X'| \geq 2$.*

The handling of the packing-certificate violation removes the points of X', and by Lemma 5, 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 changed an assignment. We can therefore charge slack-net point creations to optimal assignments. The following lemma formalizes this intuition.

Lemma 8. *Each net point creation can be uniquely charged an optimal operation.*

We summarize that above analysis to obtain the following bound.

Lemma 9. $n_C \leq (\beta + 2)\ n_A^* + n_C^* + n_D^*$.

The proof of Theorem 1 follows by combining the observations of this section with Lemmas 4 and 9.

4 Incremental Maintenance Algorithm for Net Tree

Recall that a net tree is based on a hierarchy of nets of exponentially increasing radius values. In particular, the points at level i are a (r_i)-net of the points at level $i-1$, where $r_i = 2^i$. Recall that the height of net tree is $h = \lceil \lg \Phi(P) \rceil$, where $\Phi(P)$ is the worst-case aspect ratio of the point set. In this section we present an efficient online algorithm for maintaining a slack net tree for a set of points under incremental motion in a metric space $\mathcal{M}$. 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 $\beta = \lambda^2$ denotes 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.

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 $(\beta\lambda^4 + \beta\lambda^3 + 4)(\beta + 3)h^2 = O(\lambda^8 h^2) = O(h^2)$.*

In the next section we present the algorithm and in the following section we present the competitive analysis of the algorithm.

4.1 Incremental Maintenance Algorithm for Net Tree

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. Since $X^{(i)}$ is a slack r_i-net of $X^{(i-1)}$, it will be convenient to use $P^{(i)}$ as a pseudonym for $X^{(i-1)}$, in order to maintain symmetry with the terminology of the previous section.

The competitive analysis of the previous section (recovery and crowdedness, in particular) was based on the relationship between slack-net points and neighborhoods of the optimal net. However, except at the leaf level, there is no reason to believe that points of $P^{(i)}$ will 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 $4r_i$. We have the following result.

Lemma 10. *Let o denote an optimal net point at level i. Then, for any $x \in N(o)$, $N(o) \subseteq b(x, 4r_i) \cap P$.*

This lemma 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. This observation justifies our choice of $\alpha = 4$ in Theorem 2.

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 3.1 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.

Reassign(p, i): Invoke the net-reassignment operator to p for $X^{(i)}$, but use the level-i net-creation operator (rather than the single-net operator).

Create net point(p, i): Invoke the net-point creation operation for p in $X^{(i)}$, with one difference. The point p is added to the candidate lists of points within distance $4r_i$ (rather than $2r_i$). Invoke point insertion for p at level $i + 1$.

Remove net point(x, i): Invoke the net-point removal operation for x in $X^{(i)}$. Invoke a delete-point operation of x at level $i + 1$.

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

Insert point(p, i): Invoke the point insertion operation for p in $P^{(i)}$, 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 $2r_i$).

Delete point(p, i): Invoke the point deletion operation for p in $P^{(i)}$. If $p \in X^{(i)}$, invoke an net-removal operation for p at level i.

Assignment-certificate violation(p, i): Invoke the assignment-certificate violation processing for p in $P^{(i)}$, but use the level-i reassignment operation (rather than the single-net reassignment operator).

Packing-certificate violation(x, i): Invoke the packing-certificate processing for x in $X^{(i)}$, but use the level-i net-point removal operation (rather than the single-net removal operation).

Note that operations that involve the creation or removal of net points will result in point insertion or deletion, respectively, at the next higher level of the tree. As with single-net operations, each operation may induce addition certificate violations. All these violations are stored in a priority queue, so that

operations are first applied to the lower levels of the tree and then propagate upwards.

Due to space limitations, we have moved the competitive analysis for above the net tree algorithm to the Appendix. The proof involves showing that each of the operations 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. 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.
4. J. Gao, L. J. Guibas, and A. Nguyen. Deformable spanners and applications. *Comput. Geom. Theory Appl.*, 35:2–19, 2006.
5. 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.
6. 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.
7. S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM J. Comput.*, 35:1148–1184, 2006.
8. S. Kahan. A model for data in motion. In *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pages 265–277, 1991.
9. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
10. 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.
11. J. J. Monaghan. Smoothed particle hydrodynamics. In *Reports on Progress in Physics*, volume 68, pages 1703–1759, 2005.
12. 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.
13. H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, San Francisco, 2005.
14. K. Yi and Q. Zhang. Multi-dimensional online tracking. In *Proc. 20th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 1098–1107, 2009.

A Competitive Analysis for Net Tree

We extend the proof of the competitive ratio for maintaining a single net to the case of a net tree. The difference between the proof of net and net tree is that

operations in the net tree can propagate events to other levels of the tree. Similar to the analysis of competitive ratio for a net, our analysis is based on relating the number of slack-net operations to the number of slack net-point creation events. Then, we relate net-point creation to a corresponding optimal operation.

In order to establish Theorem 2, it suffices to show the following, where n denotes the total number of operations executed by our slack-net algorithm and n^* denotes the total number of operations performed by the optimal algorithm.

$$n \;\leq\; (\beta\lambda^4 + \beta\lambda^3 + 4)(\beta + 3)h^2 n^*.$$

As in the single-net analysis, let n_A^*, n_C^*, n_R^*, n_I^*, and n_D^* denote respectively the total number of assignments, net-point creations, net-point removals, point insertions, and point deletions performed by the optimal net-tree algorithm. Let n_A, n_C, n_R, n_I, and n_D denote corresponding quantities for our slack net-tree algorithm.

First, we consider the total number of insertions (n_I) and deletions (n_D). In contrast to the analysis of a single net, the points at level i of the slack net tree may differ from those in the optimal tree, and hence insertions or deletions of points may occur within at various levels within one tree but not the other. Nonetheless, we can show the following bound on the number of these operations.

Lemma 11. $n_I \;\leq\; n_I^* + n_C$ *and* $n_D \;\leq\; n_D^* + n_C$.

The following is the analog to Lemma 4, and its proof is similar.

Lemma 12. $n_A \;\leq\; \beta\lambda^4 n_C + \beta\lambda^3 n_I$.

As in the single-net case, the total number of net-point removals (n_R) is bounded by the total number of net-point creations (n_C). It suffices to bound n_C. As we established in Lemma 5 for the single-net case, any optimal assignment can generate at most one net-point creation. The same applies to each level of the net tree. This leads to the following observation.

Lemma 13. *Each optimal assignment is responsible for the creation of at most h net-points in the slack net.*

Recall that an optimal neighborhood is crowded if it contains two or more points of the slack net. As with Lemmas 6 and 7 in the single-net case, each packing-certificate violation results in a strict reduction in the number of crowded optimal neighborhoods at this level of the tree.

Lemma 14. *After handling a packing-certificate violation, the total number of net points lying within crowded optimal neighborhoods decreases by at least one.*

This shows that, in the absence of optimal assignments, whenever we process a packing-certificate violation, the total number of net points of crowded neighborhoods strictly smaller. Since each crowding event can be charged to an optimal assignment, we have the following.

Corollary 2. *The total number of packing-certificate violations is at most* $h \cdot n_A^*$.

Now, we consider all possible reasons that a net-point creation may occur within the slack net tree. The analysis is essentially the same as in Lemma 8, but there is an increase by a factor of h due to propagation.

Lemma 15. $n_C \;\leq\; (\beta+3)h^2 \cdot n_A^* + n_C^* + h \cdot n_D^*$.

We now summarize the results of this section to obtain the desired competitive ratio.

$$\begin{aligned} n &= n_A + n_C + n_R + n_D + n_I \\ &\leq n_A + 2n_C + n_D + n_I \\ &\leq \left(\beta\lambda^3 n_I^* + (\beta\lambda^4 + \beta\lambda^3)n_C\right) + 2n_C + (n_D^* + n_C) + (n_I^* + n_C) \\ &\leq (\beta\lambda^3+1)n_I^* + (\beta\lambda^4 + \beta\lambda^3 + 4)\left((\beta+3)\,h^2 n_A^* + n_C^* + h \cdot n_D^*\right) + n_D^* \\ &\leq (\beta\lambda^4 + \beta\lambda^3 + 4)(\beta+3)h^2 n^* \end{aligned}$$

This completes the proof of Theorem 2.

B Omitted Proofs

Lemma 2. *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.

Proof. First, we claim that for any p in the metric space, $b(p, r/2)$ contains at most β points of X. To see this, suppose not. Take any $q \in b(q, r/2) \cap X$. Clearly, $b(q, r)$ contains $b(p, r/2)$, and hence would contain more than β points contradicting the fact that it is an (α,β)-slack r-net. In a metric space with doubling constant λ, any ball of radius R can be covered by at most $\lambda^{\lg\lceil R/(r/2)\rceil} = \lambda^{\lg\lceil 2R/r\rceil}$ balls of radius $r/2$. Thus, there are at most $\beta\lambda^{\lg\lceil 2R/r\rceil}$ points in X within any ball of radius R. □

Lemma 3. *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.

Proof. Let b' be a ball concentric with b of radius $2r$. Each ball of Z overlapping b has its center inside b'. Let X be the set of centers of balls in Z. Since all the points of X are separated from each other by distance r, any ball of radius r centered at a point of X contains at most one point of X. (Recall that balls are open.) Thus, by applying Lemma 2 (with $\beta = 1$), there can be at most $\lambda^{\lg\lceil 2\cdot 2r/r\rceil} = \lambda^2$ points of X within a ball of radius $2r$. □

Lemma 4. $n_A \;\leq\; \beta\lambda^3 n_C \;+\; \beta\lambda^2 n_I$.

Proof. Observe that since each reassignment is made to a point in a candidate list, n_A is bounded by the total number of additions to all the candidate lists throughout the course of the algorithm. Such an addition may occur whenever either: (a) a net point is created or (b) a new point is inserted. Whenever a net point x is created, it is added to the candidate lists of all points within distance $2r$. Whenever a point p is inserted it adds all the net points within distance $2r$ of itself to its candidate list. By Corollary 1, the number of such candidate-list additions is at most $\beta\lambda^3$ and $\beta\lambda^2$, respectively. □

Lemma 5. *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.*

Proof. When x is added to X at time t, x is added to the candidate lists of all the points that lie within distance $2r$ of x, which includes all the points that are currently in $N(o)$. During the interval $(t, t']$, no new points are added or removed from $N(o)$, and x is not removed. Thus, x is a valid representative for each $p \in N(o)$. Therefore, (by our reassignment operator) no point of $N(o)$ will be added to X during this time interval. □

Lemma 6. *The number of net points recovered as a result of the processing of a packing-certificate violation is at most β.*

Proof. Whenever a packing-certificate violation occurs at some point x, all the points of X within distance r of x are removed. Consider the set O' of optimal centers whose neighborhoods contain one of these removed net points. Consider the ball of radius r centered at each point of O'. By the properties of a strict net and Lemma 2, $|O'| \leq \beta$. By Lemma 5, at most one slack net point can be created within each of these optimal neighborhoods. Therefore, we obtain the desired bound. □

Lemma 7. *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, $O' = \{o \mid N(o) \cap X'\}$. Then, there exists at least one optimal center $o \in O'$ such that $|N(o) \cap X'| \geq 2$.*

Proof. We can assume that no optimal packing-certificate violation occurs since optimal algorithm can avoid the packing-certificate violation simply by using optimal assignments. Thus, assume that O' is consistent during our operation.

A packing-certificate violation is triggered by at least $\beta + 1$ slack-net points lying within in a ball $b(x, r)$ for some $x \in X$, and $X' = X \cap b(x, r)$. For each $o \in O'$, $N(o) \cap X'$ consists of some subset of points of X' that lie within the ball $b(o, r)$. By Lemma 3, at most β balls from O can have a nonempty intersection with $b(x, r)$. Clearly, therefore, at least one of these subsets contains at least two points of X'. □

Lemma 8. *Each net point creation can be uniquely charged an optimal operation.*

Proof. Consider the creation of a slack-net point x at some time t and let o be the optimal net point such that $x \in N(o)$. We consider two cases. First, there exists $x' \in X$ such that $x, x' \in N(o)$ and $x' \in \text{cand}(x)$ at some time $t' < t$. (This is a case of recovery.) Second, no such x' exists. This can be further divided into two subcases. First, x is the first slack-net point to appear in $N(o)$ after o was added to O. Second, some slack-net points in $N(o)$ had already been added to X, but x had never obtained any of them as its candidates. That is, from the time that x was assigned to $N(o)$ until time t, there had been no slack-net point creations in $N(o)$. More formally, we have the following cases:

1. [Recovery:] Let t' be the latest of the following three events to occur: (1) x is assigned to $N(o)$, (2) x' is assigned to $N(o)$, and (3) x' is added to X. At time t', $x' \in \text{cand}(x)$. Without loss of generality, x' is the last element of $\text{cand}(x) \cap N(o)$. Since x was added to X, we know that x' is not valid at time t. There are three possible reasons. Either:
   (a) x' was assigned to some other optimal center o',
   (b) x' was deleted, or
   (c) x' was removed from X.
2. (a) [Initial Creation:] x is the first slack-net point added to $N(o)$. (Note that this does not imply that there were no slack-net points in $N(o)$, but clearly any such points were not created when they were in $N(o)$.) The creation of x is charged to the creation of o. Clearly, this event can be charged at most once.
   (b) [x Recently Arrived:] Let $t' < t$ be last time such that point $x' \in N(o)$ was added to X and suppose that x was assigned to $N(o)$ at some time $t'' > t'$.
   We observe that the optimal assignment of x generates at most one slack-net point in $N(o)$ (namely, x itself). Before x was added to X, it was not a net point. Thus, x is not charged to any other net creations. Clearly, each such optimal assignment can be charged by at most one such event.

Let us consider Case 1. Observe that each of the possible cases listed above results in the creation of at most one new net point in $N(o)$ by Lemma 5. The cost of the creation of x will be charged to the operation that caused x' to become invalid.

Cases 1(a) and 1(b) can be charged directly to an optimal assignment or optimal deletion operation, respectively. Clearly this optimal operation is charged at most once.

Finally, let us consider Case 1(c). Note that the removal of x' is not directly related to any optimal operation. First, we show that it is charged at most once. After adding x to X, x becomes a candidate of all points of $N(o)$. (That is, x takes a role that x' had performed in representing some of the points of $N(o)$, since x was the last element in $\text{cand}(p) \cap N(o)$ for all $p \in N(o)$.) Thus, x' is charged only once in this way. We observe that a removal operation only happens during the handling of a packing-certificate violation. During the handling of each packing-certificate violation, we can create at most β net points by Lemma 6. Thus,

the total number of net point creations due to Case 1(c) is bounded by β times the total number of packing-certificate violations. Since each packing-certificate violation satisfies the conditions of Lemma 7, we observe that the condition can be made by only optimal assignments (see Cases 1(a) and 2(b)). Thus, the total number of net point creations by Case 1(c) is at most $\beta \cdot n_A^*$. □

Lemma 9. $n_C \leq (\beta + 2)\, n_A^* + n_C^* + n_D^*$.

Proof. Let n_{C1}, n_{C2a}, and n_{C2b} denote the total numbers of net-point creations arising due to cases 1, 2(a), and 2(b), respectively, as given in the proof of Lemma 8. Clearly, n_{C2a} is at most n_C^* and n_{C2b} is at most n_A^*. As observed earlier, for n_{C1}, there are three cases. Each case bounds to the number of optimal operations. Thus,

$$n_{C1} = n_A^* + n_D^* + \beta \cdot n_A^*. \quad \text{(By Lemma 6 and 8)}$$

Therefore, the total number of slack-net point creations is

$$\begin{aligned} n_C &\leq n_{C1} + n_{C2a} + n_{C2b} \\ &\leq (n_A^* + n_D^* + \beta \cdot n_A^*) + n_C^* + n_A^* \\ &\leq (\beta + 2)\, n_A^* + n_C^* + n_D^* \end{aligned}$$

□

Lemma 11. $n_I \leq n_I^* + n_C$ *and* $n_D \leq n_D^* + n_C$.

Proof. Suppose that a point p is inserted at level i of the slack net tree. This implies that p was added to $X^{(i-1)}$. Thus, the number of insertions is at least the number of net point creations. Also, if p is inserted at the leaf level, the optimal algorithm inserts p as well. Thus, the total number of insertions is at most $n_I^* + n_C$.

Similarly, suppose that a point p is deleted from level i. This implies that p is a net point at level $i-1$. If p is deleted from the leaf level, the optimal algorithm deletes p as well. Thus, the total number of deletions is at most $n_D^* + n_C$. □

Lemma 12. $n_A \leq \beta\lambda^4 n_C + \beta\lambda^3 n_I$.

Proof. Because the radius increase to $4r$, n_A is at most $\beta\lambda^4 n_C + \beta\lambda^3 n_I$ (by Lemma 4 and Corollary 1). □

Lemma 13. *Each optimal assignment is responsible for the creation of at most h net-points in the slack net.*

Proof. Let i denote the highest level of our slack-net tree such that $p \in P^{(i)}$. It follows that $p \in X^{(j)}$, for all $j < i$. Since $p \in X^{(j)}$ may be assigned to another optimal net point, an assignment violation may occur at the points having p as their representative at each such level j. Such points may invoke the reassignment operation and generate a new net point as a result. By Lemma 5 there will be at most one such recovery per optimal neighborhood. In addition, at level i, p may be added to $X^{(i)}$. Thus, the total number of net points generated by an optimal assignment in a net tree is at most h. □

Lemma 14. *After handling a packing-certificate violation, the total number of net points lying within crowded optimal neighborhoods decreases by at least one.*

Proof. Let i denote the level at which the packing-certificate violation occurs. By the same procedure as in the single-net case, we invoke the removal operation on all the net points lying within radius r of the affected net point at level i. Recall from Lemma 7 that in order for a packing-certificate violation to occur, there exists at least one neighborhood that has at least two net points involved in the violation. By Lemma 6, the total number of net points of crowded neighborhoods will strictly decrease by at least one at level i.

Let us consider the lower and higher levels of our net tree. This handling operation does not affect any of the levels below i since the net points are removed from $X^{(i)}$ but are still in $P^{(i)}$. At the higher levels, some points will be deleted due to their removal from level i. However, the deletion operation can only reduce the number of net points. Then, during the reassignment operations (as part of the removal operation), some net points may be created by recovery case (1(c)) or creation case (2(a)) of Lemma 8. However, these do not increase the number of net points in crowded neighborhoods. It then invokes the insertion operation on the next higher level. Whenever a point is inserted, the point searches nearby net points first. It becomes a net point only if there is no net points nearby (that is, within distance $4r$). Thus, the insertion operation does not create a net point in already crowded neighborhoods.

Thus, we have the desired bound. □

Lemma 15. $n_C \;\leq\; (\beta+3)h^2 \cdot n_A^* + n_C^* + h \cdot n_D^*$.

Proof. Let us consider case 1(c) first. A net point can be removed only as a result of a packing-certificate violation or the explicit deletion of a point. Suppose that at some level, the jth packing-certificate violation occurs. Then, for some k_j, $\beta + k_j$ distinct net points resulted in this violation. Since the handler removes at most $\beta + k_j$ times h net points, by Lemma 5, at most $(\beta + k_j)h$ net points will be recovered. By Corollary 2 the sum $\sum_j k_j$ is at most $h \cdot n_A^*$. Thus, the total number of net-point creations due to case 1(c) is at most $\sum_j (\beta + k_j)h \;\leq\; (\beta+1)h^2 \cdot n_A^*$.

For case 2(a), the number of net point creations is clearly at most n_C^*. For cases 1(a), 1(b), and 2(b), the total number of net-point creations is at most $h \cdot n_A^*$, $h \cdot n_D^*$, and $h \cdot n_A^*$, respectively, since each operation at level i can propagate to each of the other h levels of the tree. Thus we have

$$\begin{aligned} n_C &\leq (\beta+1)h^2 \cdot n_A^* + n_C^* + h \cdot n_A^* + h \cdot n_D^* + h \cdot n_A^* \\ &\leq (\beta+3)h^2 \cdot n_A^* + n_C^* + h \cdot n_D^*. \end{aligned}$$

□